3-2021

Unsupervised Clustering of RF-Fingerprinting Features Derived from Deep Learning Based Recognition Models

Christian T. Potts

Unsupervised Clustering of RF-Fingerprinting Features Derived from Deep Learning Based Recognition Models

THESIS

Christian T. Potts, B.S.E.E., Captain, USAF

AFIT-ENG-MS-21-M-074

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

The views expressed in this document are those of the author and do not reflect the official policy or position of the United States Air Force, the United States Department of Defense or the United States Government. This material is declared a work of the U.S. Government and is not subject to copyright protection in the United States.

AFIT-ENG-MS-21-M-074

Unsupervised Clustering of RF-Fingerprinting Features Derived from Deep Learning Based Recognition Models

THESIS

Presented to the Faculty
Department of Electrical and Computer Engineering
Graduate School of Engineering and Management
Air Force Institute of Technology
Air University
Air Education and Training Command
in Partial Fulfillment of the Requirements for the
Degree of Master of Science in Electrical Engineering

Christian T. Potts, B.S.E.E., B.S.E.E.
Captain, USAF

March 19, 2021

AFIT-ENG-MS-21-M-074

Unsupervised Clustering of RF-Fingerprinting Features Derived from Deep Learning Based Recognition Models

THESIS

Christian T. Potts, B.S.E.E., B.S.E.E.
Captain, USAF

Committee Membership:

James W. Dean, Ph.D
Chair

Michael A. Temple, Ph.D
Member

Gilbert L. Peterson, Ph.D
Member

AFIT-ENG-MS-21-M-074

Abstract

Radio Frequency (RF)-Fingerprinting is focus of machine learning research which aims to characterize wireless communication devices based on their physical hardware characteristics. It is a promising avenue for improving wireless communication security in the Physical Layer (PHY) layer. The bulk of research presented to date in this field is focused on the development of features and classifiers using both traditional supervised machine learning models as well as deep learning. This research aims to expand on existing RF-Fingerprinting work by approaching the problem through the lens of an unsupervised clustering problem. To that end this research proposes a deep learning model and training methodology to extract features from OFDM-based IEEE 802.11a/g preamble waveforms to enhance performance with various clustering algorithms. The model architecture presented takes the form of a convolutional autoencoder with an objective function that combines both autoencoder reconstruction loss as well as triplet loss to learn feature encodings. These features were then clustered using the K-means, DBSCAN, and Mean Shift clustering algorithms.

The models proposed achieved highly effective clustering performance with the K-means and Mean Shift clustering algorithms with average V-measure (V_M) scores of 0.978, 0.822, and 0.901 at SNR = 18db for the K-means, DBSCAN, and Mean Shift clustering algorithms respectively. Additionally the models proposed were able to achieve average V_M scores of 0.789, 0.720, and 0.737 at SNR = 18db for the various clustering methodologies on test datasets containing devices previously unseen to the trained models.

Table of Contents

Page

List of Figures

Figure Page

List of Tables

Unsupervised Clustering of RF-Fingerprinting Features Derived from Deep Learning Based Recognition Models

I. Introduction

It is difficult to overstate the impact that internet communications has had on all aspects of society in the modern age. According to the most recent Cisco Annual Internet Report [2], in North America alone, there is projected to be 5 billion unique devices connected to the internet by 2023. Along with the adoption of the internet as one of the primary means of communication and transference of information, the world has seen the proliferation of the Wireless Local Area Network (WLAN) as a primary medium by which users access the internet. An inherent trade-off of using a wireless communication scheme as opposed to a wired connection is that wireless connections are exposed to more security risk than their wired counterparts in exchange for more flexibility and mobility.

For a user to communicate with a wireless access point, they must transmit an electromagnetic signal outward from the device to be interpreted by the receiver. This also means that all communication signals are potentially visible to any parties geographically collocated with the user and thus security measures must be put in place such that transmitted information is only decipherable to the intended recipients. The realization of this security is typically achieved in the Data Link and Network layers of the internet protocol stack via a combination of encryption techniques as well as device specific passwords and identifying labels. The rapid increase in available computing power and the emergence of quantum computing pose significant threats to such bit level credentials as a means to establish secure wireless connections by enabling previously infeasible means of subverting such protections [3]. Additionally, such methods are susceptible to insider threats and theft of credentials. The intersection of maturating WLAN technologies and machine learning methodologies provides a clear need for Physical Layer (PHY) security in the form of RF-Fingerprinting/Specific Emitter Identification (SEI).

RF-Fingerprinting refers to a specific focus of machine learning research that aims to categorize RF waveforms according to the specific device that transmitted said waveform. Similar to human fingerprints from which the name of the technique is derived, RF-Fingerprinting attempts to capture the expressions of physical uniqueness of a specific RF transmitter's hardware that are present in the waveforms emitted from that device. These unique characteristics arise from minute variations in each devices individual circuit elements that, when aggregated, are significant enough to be distinguished via machine learning methods. The bulk of existing research in this field falls into the broad category of supervised learning which aims to train a model to accurately classify signal observations based on a known device label for each observation. The research presented in this thesis takes the prior research done in this field and applies the concepts therein to the task of unsupervised learning. Unsupervised learning refers to a machine learning domain in which unlabeled data is processed and analyzed in order to discover latent structures present within the pool of data. These structures are then used to express meaningful characteristics of the data such as clusters of similar observations.

1.1 Problem Statement

The specific goal of the research presented within this thesis is to address the following question: Given a data-set of unlabeled RF signal collections, how can a combination of feature extraction methods, RF-Fingerprinting techniques, and clustering algorithms be best used to ascertain meaningful information about the data-set? For the purposes of this research, the term "ascertain meaningful information" will be quantified based on how well the different methodologies are able to group together clusters of observations such that the clusters reflect known device labels for a given data-set. To clarify this point, the data-set being examined has such labels associated with each observation, however, these labels are not to be used in the clustering process but rather are used after clustering is done as a means to measure performance. Feature extraction methods refers to various known methods for reducing the dimensionality of raw input data for use in machine learning tasks. Specific methods will be explained in detail in Chapter III. RF-Fingerprinting techniques refers to the preprocessing techniques that have been researched previously to achieve good results in RF-Fingerprinting/SEI efforts.

1.2 Motivation

Existing research in the domain of RF-fingerprinting has utilized unsupervised learning methods for feature extraction [4]. However it's use in the RF-Fingerprinting literature up to this point exists mainly as a preprocessing step that is ultimately used in a supervised learning problem. There has been significantly less research done in the domain of unsupervised clustering applied to the RF-Fingerprinting problem. Unsupervised learning is a vital arm in the machine learning discipline. In the modern day, the capabilities to collect vast pools of data is ever growing and accurately labeling all observations for use in supervised machine learning tasks is not always feasible. Clustering refers to a category of unsupervised learning research that aims to group together a collection of observations algorithmically according to some measure of similarity such that observations within a given cluster are characteristically similar based on said criteria. Clustering is a useful application of machine learning under circumstances where there aren't intended classes known prior to the machine learning process but there is value in the grouping of similar observations as is the case in problems such as designing recommendation algorithms or market segmentation for targeted advertisements. Clustering is also often used to deal with the issue of large quantities of unlabeled data by developing a clustering model for a data set such that the clusters formed represent some natural grouping that is desirable for a given machine learning problem. Such clustering algorithms are typically evaluated with a small dataset with true labels corresponding to a desired grouping of observations. Using such labels as a reference can give an insight into how the clustering observations will respond to datasets without known labels.

The research presented in this thesis aims to utilize unsupervised deep feature-learning models and various clustering algorithms to ascertain information about the active transmitters (i.e. how many transmitters are present, which transmissions came from the same device, etc.) for a given WLAN. This type of information is typically easy to obtain via aspects of the data link layer such as MAC address. One of the main underlying assumptions and motivations for RF-Fingerprinting, however, is that these types of device identifiers are susceptible to manipulation via techniques such as MAC spoofing and packet sniffing [5] and therefore potentially unreliable. RF-Fingerprinting them aims to recognize impersonation attempts via characteristics present in the PHY layer using machine learning. The motivation behind clustering in RF-Fingerprinting is then to develop a machine learning framework that is able to reliably group together signal transmissions from the same physical device despite not being trained on examples from that specific device. This overcomes a downside of supervised classification problems that typically dominate RF-Fingerprinting research in that the usefulness of models produced are limited to those devices used during the training process. In a scenario where the active emitters in a particular WLAN change frequently, as is often the case, such models need to be re-trained to accommodate new devices which can be very time consuming depending on the type of machine learning model.

1.3 Approach

For any given machine learning problem, there are many degrees of freedom with respect to the design of models and experiments that developing a clear picture of how every decision interacts with each other becomes intractable. Therefore, when conducting machine learning research, it is typical to select a smaller portion of the overall process to evaluate how those aspects of the process can affect the overall goal of the machine learning problem. This thesis focuses on several aspects of the RF-Fingerprint clustering problem, to include:

- Can a single machine learning model be trained to extract features from RF waveforms such that the feature's tend to cluster well across different Signal to Noise Ratios (SNR)?
- Can a machine learning model be trained to extract features from RF waveforms that cluster well using devices not used to train the model?
- How does the number of features extracted by machine learning models affect its ability to effectively cluster observations by emitter?
- What is the most effective clustering algorithm for the feature vectors produced by such a model?

The research in this thesis is conducted first by performing an extensive literature review of the prevailing research in the area of RF-Fingerprinting problems to observe existing methodologies known to perform well. After the literature has been reviewed, a machine learning model is designed in order to address the main research questions presented above. These models take the form of Convolutional Autoencoder style neural networks with the addition of the triplet loss objective function to learn device separable features from WiFi emitters. Once the model design is decided implementations of the model will be created and trained using various configurations of model parameters. These models are then used to generate feature vectors from individual signal observations to be used in various clustering algorithms. The resulting cluster labels are then compared to the true device labels for the signal observations using the external cluster validity measure known as the V_M.

II. Background and Literature Review

This chapter provides an overview of the machine learning concepts and background information necessary to fully describe the research presented within this thesis. Section 2.1 provides a description of WiFi packet and preamble structure. Section 2.2 provides an overview on the topic of deep learning as well as elaborate on specific topics therein to fully contextualize the research presented in this thesis. Section 2.3 provides an explanation of selected clustering methodologies. Lastly, Section 2.4 provides an overview on the topic of RF-fingerprinting and previous research conducted on the topic

2.1 WiFi protocol

WiFi is the colloquial name for the wireless communications protocol described in Institute of Electrical and Electronics Engineers (IEEE) standard 802.11 [1] for WLANs. The standard provides PHY and Medium Access Control (MAC) specifications to enable wireless communications between devices over short distances. The specifics of the specifications presented in IEEE 802.11 vary depending on the specific iteration of the standard being used (i.e. 802.11a, 802.11b, 802.11g, etc.).

Different iterations of the IEEE 802.11 standard have different a different PHY Protocol Data Unit (PPDU) format. All versions prior to IEEE 802.11n (or WiFi 4) use the non-HT PPDU format. The structure of an IEEE 802.11 non-High Throughput (HT) PPDU can be seen in Figure 1. As with most wireless communications protocols, WiFi transmissions begin with a preamble sequence (denoted in Figure 1 as "PLCP Preamble") used to perform frequency and timing adjustments prior to demodulation of the transmitted signal. Preamble signals have been shown to be useful for RF-Fingerprinting tasks in the past due to the fact that the preamble waveform is independent from the data payload and therefore variations in a transmission's preamble can reasonably be assumed to be characteristic of the transmitting device hardware and not bit level information.

IEEE 802.11a/g uses Orthogonal Frequency Division Multiplexing (OFDM) as the primary modulation scheme for RF transmissions. OFDM is a modulation scheme in which transmitted information is split among some number of orthogonally spaced sub-carries each being modulated with some other modulation scheme such as Binary Phase Shift Keying (BPSK), Quadrature Phase Shift Keying (QPSK), or Quadrature Amplitude Modulation (QAM). The underlying modulation scheme for each sub-carrier in the data payload of an 802.11 PPDU varies depending on the intended data rate of the transmission. The Physical Layer Convergence Protocol (PLCP) preamble of a non-HT PPDU consists of ten repetitions of a short (0.8μs) OFDM symbol called the Short Training Field (STF) followed by two repetitions of a long (3.2μs) OFDM symbol called the Long Training Field (LTF).

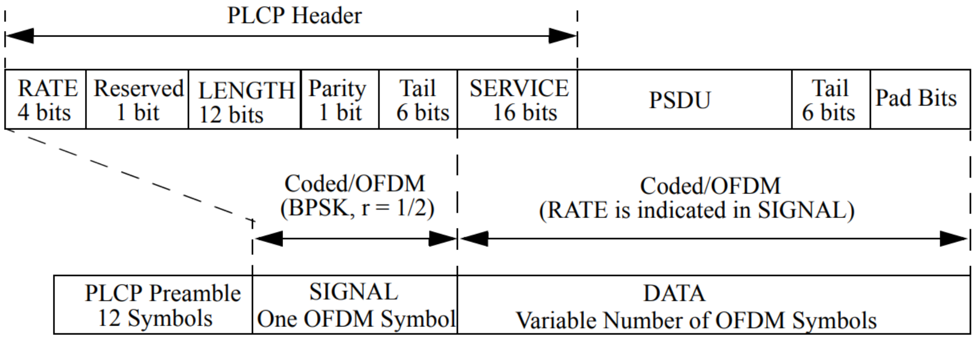


Figure 1: IEEE 802.11 non-HT PPDU Format [1]

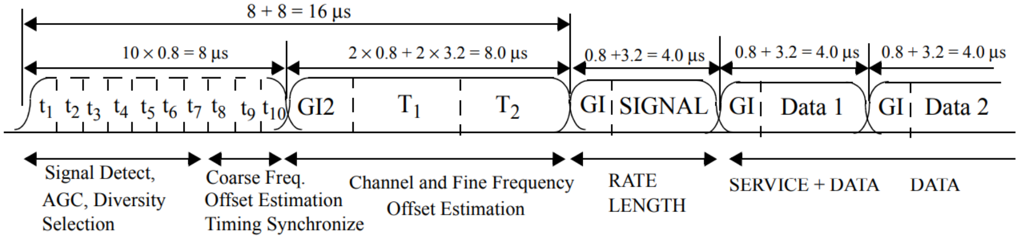

8 + 8 = 16 μs
10 × 0.8 = 8 μs
2 × 0.8 + 2 × 3.2 = 8.0 μs
0.8 +3.2 = 4.0 μs
0.8 + 3.2 = 4.0 μs
0.8 + 3.2 = 4.0 μs
t1 t2 t3 t4 t5 t6 t7 t8 t9 t10
GI2
T1
T2
GI
SIGNAL
GI
Data 1
GI
Data 2
Signal Detect, AGC, Diversity Selection
Coarse Freq. Offset Estimation Timing Synchronize
Channel and Fine Frequency Offset Estimation
RATE LENGTH
SERVICE + DATA
DATA

Figure 2: IEEE 802.11 PLCP Preamble Structure [1]

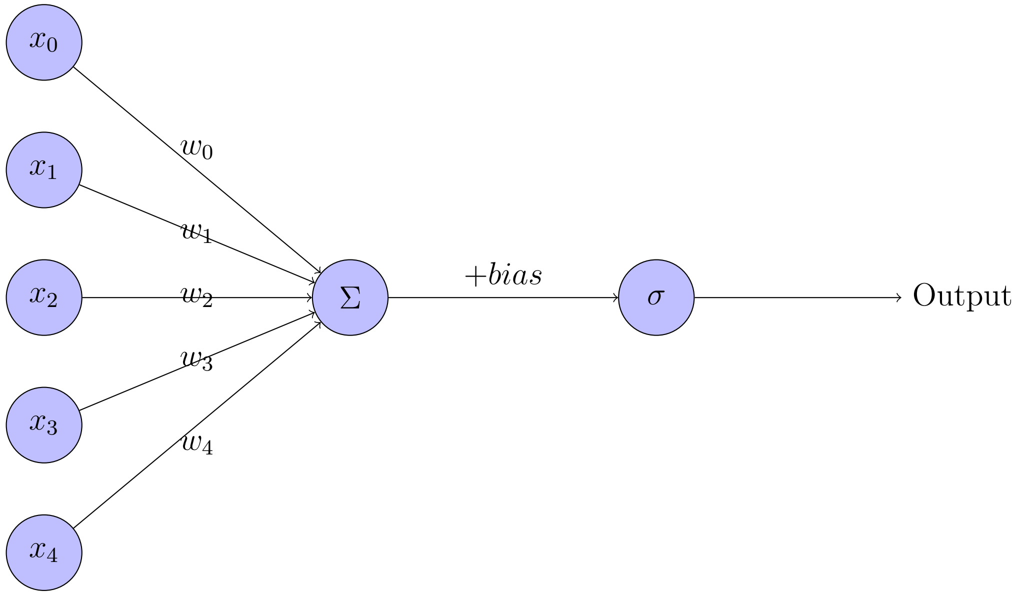
2.2 Deep Learning

Deep learning is a category of machine learning that is characterized by modelling numerous non-linear relationships between independent features of some observed data. Deep learning models are often referred to as artificial neural networks (ANNs) or just simply neural networks due to the inspiration drawn from the biological learning process of the human brain. The basic building block of a neural network is the perceptron leading to another common term for deep learning models, the multi layer perceptron (MLP). The perceptron is intended to be analogous to a neuron within the human nervous system. A single perceptron operates by taking in a weighted sum of some set of feature information connected to the input of the perceptron adding a bias value and applying some activation function to this weighted sum. The output of a single perceptron with input $x_{i=0,\ldots,N}$, weights $w_{i=0,\ldots,N}$, bias b and, activation function σ is:

$$y = \sigma(\sum_{i=0}^{N}[w_i * x_i] + b) \tag{1}$$

The activation function of a perceptron is some nonlinear function, such as the hyperbolic tangent function, which allows a perceptron to represent nonlinear relationships between the input features. The weight and bias values are the trainable parameters of the model that allow the perceptron to learn function mappings from the input values to the desired output. Perceptrons can be combined and organized in

Figure 3: Graphical Depiction of Single Perceptron



countless ways to represent more complicated nonlinear relationships present within high dimensional input data and the architecture of an ANN is defined by how these perceptrons are arranged and connected. Most ANNs are constructed as a sequence of successive layers of perceptrons in which the output of one layer is connected to the input of the next layer.

One of the principal advantages of deep learning models when compared to other machine learning models is the high adaptability of these models to different machine learning problems. The universal approximation theorem presented in [6] states that a feed forward neural network can sufficiently represent any function mapping some set of input measurements to some desired output given that the network has enough hidden units (i.e. perceptrons). Due to this universal nature of ANNs, they have been applied to nearly every conceivable type of machine learning problem to include supervised classification, generation of new data, and unsupervised learning of features.

2.2.1 Dense Layers

The Dense layer is the most basic form of neural network layer. In a dense layer there is an individual connection between each input value to the layer and each perceptron that makes up that layer. The total number of connections that are present in a single dense layer is then the number of inputs n_{in} multiplied by the number of perceptrons in that layer n_{out}. These weights are typically represented as an n_{out} by n_{in} weight matrix W where $w_{i,j}$ represents the weight of the connection between the ith input to the jth perceptron. The output of a dense layer with input x and bias values b can be expressed then as:

$$y = \sigma(xW + b) \tag{2}$$

The high degree of connectivity present within dense neural network layers allow them to learn highly complex non-linear function mappings. This however, comes at the expense of high memory and computational costs which are important practical concerns when it comes to training deep learning models. Additionally, the high number of trainable parameters means that dense neural networks have a higher variance than other neural network layer types. This high variance contributes to a tendency to over-fit to training data meaning that a very large pool of training data is typically required to achieve good results.

2.2.2 Convolutional Neural Networks

The drawbacks of dense network layers become particularly apparent when designing larger, more complicated neural network architectures. As the name would imply, dense layers have a very high number of individual connections. As stated previously, this leads to concerns in regards to the memory and computation time requirements for dense layers. A second limitation of dense layers is that they do not explicitly preserve the intrinsic relationships between certain input features for spatially and temporally organized data such as images, audio waveforms, etc. While these relationships can be captured by dense layers, other layer types are better at prioritizing these characteristics. Convolutional layers address both of these issues and Convolutional Neural Networks (CNN) (referring to neural networks which implement convolutional layers) have become very popular in the realm of modern deep learning research.

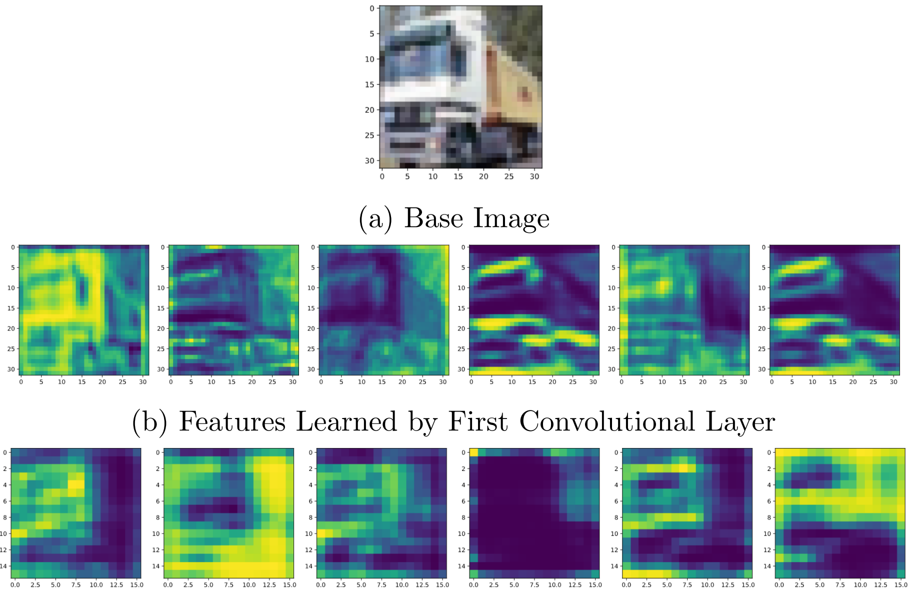
Convolutional layers apply the concept of kernels and the convolution operation (denoted by the * operator shown in Equation (3)) between matrices that is common within image processing applications to deep learning models. A convolutional operation between two dimensional matrices X and Y both with dimensions m by n is defined as follows [7]:

$$X * Y = \sum_{m} \sum_{n} X(m,n)Y(m,n) \tag{3}$$

Kernels (also referred to as filters) are matrices that when convolved with an image, produce some type of desirable representation of that image. The total convolution process for an image involves applying the kernel to each possible position on the image where the two completely overlap. Kernels used in image processing are typically explicitly designed in such a way as to apply specific transformations such as highlighting horizontal and vertical edges or applying specific blurring effects to an image. The effect of a kernel is determined by the individual values that make up the kernel matrix. In a CNN layer these kernel matrix values are the trainable parameters of the network taking the place of connections between individual perceptrons as seen in dense layers. Using this concept of trainable kernels, convolutional layers are able to learn features that are more applicable to data in which spatial relationships between features are important. In order to visualize the features learned by a CNN, a basic CNN classifier was trained on the CIFAR 10 dataset described in [8]. The output of the kernel convolutions for the first two layers of this layer on a single input image is shown in Figure 4.

Convolutional layers are also able to learn structures across multiple "channels" of data. Channels in the context of CNNs refers to different pieces of information that can describe a single point in a given spatial or temporal position. Common examples of channels for two dimensional images would be RGB color channels as well as depth. The presence of different channels for input data means that input data will always have one more dimension than the dimension of the convolution operation (i.e. dimensions for a two dimensional convolution input would be height × width × number of channels). At the output of a convolutional layer, channels refer to the distinct output of each of the different kernels applied to the image.

The kernel based structure of convolutional layers introduces new parameters control how the convolution operation is applied to the input data. The primary convo-



(a) Base Image

(b) Features Learned by First Convolutional Layer

(c) Features Learned by Second Convolutional Layer

Figure 4: Visualization of feature representations at different layers of CNN

lution specific parameters include:

- **Number of Kernels** - The number of unique kernels convolved with the input features.
- **Kernel Size** - The dimensions of each kernel matrix
- **Padding** - The amount of additional data appended to the borders of the input data.
- **Stride** - The amount that the sliding kernel is shifted after each convolution operation.

A single convolutional layer can have any number of individual kernel matrices. More kernels in a layer equates to more unique spatial characteristics that can be potentially represented by the layer's output. Each individual kernel matrix is convolved with the layer's input and the output for each kernel makes up one channel of the layers output.

Kernel size refers to the dimensions of each kernel matrix in a convolutional layer. Optimal kernel size varies depending on the nature of the problem. Larger kernels are generally better at capturing large simple patterns whereas smaller kernels are more suited to distinguishing finer intricate details. Smaller odd numbered kernel sizes are generally preferred for most machine learning problems.

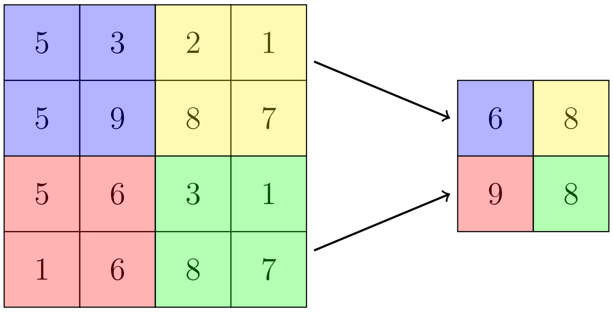
A large kernel size results in fewer total positions in which the kernel and the input data will totally overlap. The result is that the output of a convolutional layer will always result in a reduction of dimension in its output for kernels of size > 1. Padding is a way to preserve the dimensionality between input and output of a convolutional layer by appending data to the borders of the input before performing convolution. The most common choice is zero padding in which zeroes are added to the borders however other types of padding also exist.

Stride refers to how far the kernel is shifted after each convolution operation. The stride typically defaults to one. Higher strides can be used to achieve a down-sampling type of effect.

2.2.3 Max Pooling

Pooling layers are a type of neural network layer that are incredibly common in the design of CNN architectures. The use of pooling layers is so prevalent in CNN architectures that their use is often simply assumed when stating that a convolutional layer is present within a network. The use of pooling layers is so universally associated with convolutional layers because the effects of the pooling process aligns so closely with the common motivations for choosing a convolutional architecture.

Pooling is a kind of down-sampling process that is applied to some input. Pooling involves a sliding window being applied to the input of the layer and for each position of the pooling window over the input a single output value is created similar to the convolution process. As the process of pooling is so similar to the convolution process, many of the same parameters are applied to pooling layers such as stride, kernel size, and padding. While the output from a convolutional layer is the result of the convolution operation between the windowed input and the kernel, the output of a pooling layer can be any function applied to the windowed input values. The most common of these is the maximum value of the window but others include the



5	3	2	1
5	9	8	7
5	6	3	1
1	6	8	7

6	8
9	8

Figure 5: Max Pooling Operation

minimum value or the average value.

One of the main benefits of pooling layers is the size reduction that the down-sampling effect provides. This results in lower memory and computational burdens which is also one of the benefits for using a CNN. Additionally, pooling layers provide a degree of translational in-variance to the network [7] meaning there is a reduction in the change to the network output in response to minor translations of the input data. This is often helpful as it preserves the general organization of features in the input while being robust to more variable datasets.

2.2.4 Batch Normalization Layer

Normalization of data is a fundamental concept within the field of machine learning. It is an important pre-processing step often used before training of virtually every type of machine learning model. Normalization counteracts a problem within machine learning in which differences and scale and variability between different input features causes certain features to dominate and others to become negligible in affecting model output. Batch normalization is a related concept specific to neural networks, originally introduced in [9], that performs normalization steps at different points within the model.

The training process of neural networks operates by descending the gradient of the cost function and updating each trainable parameter within the network at the same time. This results in a phenomenon known as covariate shift in which changes to early layers in the network characteristically change the layer's output distribution and make changes in the later portions of the network ineffective. Batch normalization is a means of addressing the problem of covariate shift by normalizing the output of layers within the network over each minibatch being trained on. In addition to the normalization applied to the output layer, additional scaling and shifting parameters γ

and β are learned by the batch normalization layer to preserve the full scope of possible representations learned by the incoming layer. The effect of this is a significant reduction in training time, regularization of the model, and a lowered sensitivity to network hyper-parameters such as learning rate.

The batch normalized output y for a minibatch output $X = \{x_{1 \ldots m}\}$ of some layer is then achieved as such:

- Calculate the mean of the minibatch output:

$$\mu = \frac{1}{m} \sum_{i=1}^{m} x_i \tag{4}$$

- Calculate the variance of the minibatch output:

$$\sigma^2 = \frac{1}{m} \sum_{i=1}^{m} (x_i - \mu)^2 \tag{5}$$

- Normalize the minibatch output:

$$\hat{x_i} = \frac{x_i - \mu}{\sqrt{\sigma^2 + \epsilon}} \tag{6}$$

- Apply scaling and shifting parameters:

$$y_i = \hat{x_i} \gamma + \beta \tag{7}$$

2.2.5 Autoencoders

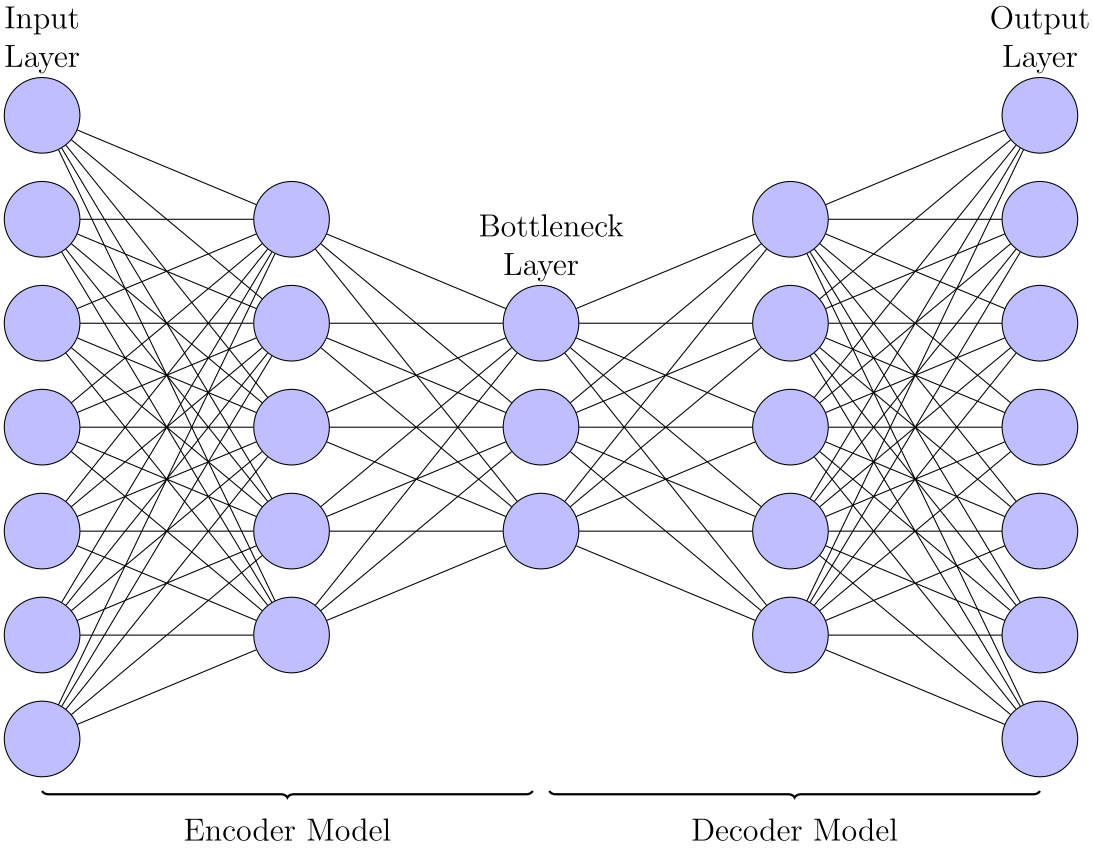
A key benefit of deep learning models when compared to traditional machine learning models is their ability to learn useful hierarchical feature representations from high dimensional data during the training process. This feature learning is a natural occurrence of the neural network training process for many supervised learning problems and will typical manifest itself within the hidden layers of the network. Latent variable models, by contrast, make this ability of neural networks to learn feature representations from high dimensional data the primary focus of the machine learning task.

Autoencoders are one of the most common types of latent variable models due to their intuitive nature and ease of construction. A general description of an autoencoder neural network would be a symmetrical feed forward neural network in which the input features and output features have the same dimensionality. In addition, there are some number of hidden layers between the input and the output however the number and type of these hidden layers are chosen based on the nature of the training data and the machine learning task. Feature extraction and dimensionality reduction are very popular applications of autoencoders [10, 11] however this type of neural network architecture has also commonly been used for other tasks such as removing noise from input data [12].

An autoencoder can be thought of as two separate neural network models trained in tandem with one another. These two sub-models are referred to as the encoder model and decoder model. The encoder model consists of all layers from the input to the center-most layer of the autoencoder which is typically referred to as the bottleneck because it is often the layer with the smallest dimension. The decoder model is made up of all layers from the bottleneck to the output. The distinction between encoder and decoder models is visualized in Figure 6. The output of the encoder model is a smaller dimensional representation of the input data and this output is typically referred to as the latent encoding of the input represented by the variable z.

Autoencoders are able to learn to produce these encodings of high dimensional

Figure 6: Graphical Depiction of an Autoencoder Model



input data through the process of gradient decent via back-propagation as with any other neural network architecture. Autoencoders are trained using an objective function that minimizes the dissimilarity between the input (x) and output (y) of the model. A common choice for this objective function is the Mean Squared Error (MSE). With N being the dimensionality of the output, the MSE loss for an autoencoder model is defined as [7]

$$MSE(x, y) = \frac{1}{N} \sum_{i=1}^{N} (y_i - x_i)^2 \tag{8}$$

2.2.6 Triplet Loss

Triplet loss is an objective function detailed in [13] used to train neural network models specifically to maximize distance between output feature vectors for inputs of different classes while simultaneously minimizing distance between outputs for inputs of the same class. The triplet loss function is not calculated on a single training datapoint along with some desired output value as is the case with many neural network objective functions. Triplet loss is calculated using the outputs of a group of three different training points called a triplet. Each triplet consists of an anchor point (a), a positive point (p) being another datapoint from the same class as a, and a negative point (n) coming from a different class than a and p. Letting $f(x)$ be the function representing passing an input x through some neural network, the triplet loss is calculated as follows [13]:

$$\mathcal{L}(a, p, n) = ||f(a) - f(p)||^2 - ||f(a) - f(n)||^2 + \alpha \tag{9}$$

The α term in the triplet loss function is a tunable hyper-parameter affecting the enforced margin between positive and negative classes.

2.3 Clustering

Clustering refers to a fundamental unsupervised machine learning task in which the goal is to find natural groupings of observations within a dataset in order to obtain meaningful insights into the nature of the data. Clustering has been widely studied and some of its most notable applications can be seen in the form of market segmentation algorithms to provide effective targeted advertising[14] as well as recommendation algorithms for e-commerce services [15].

Clustering algorithms come in many different varieties that utilize different mathematic principles to produce groupings. Different clustering algorithms typically rely on a different set of assumptions of the overall structure of the data and properties of clusters therein meaning that there is not an objectively superior algorithm for all clustering problems or datasets.

2.3.1 K-means Clustering

The K-means clustering algorithm is one of the most widely used and fundamental methodologies used within the domain of unsupervised clustering problems[16]. The algorithm is fairly computationally inexpensive and simple to implement hence its widespread use for clustering based research. Much like many clustering methods, K-means uses some distance/similarity metric (most often Euclidean distance) between datapoints as a means by which to produce clusters. Cluster assignments are given to datapoints based on closest proximity of said datapoint to the centroid (geometric average of all points within a cluster) of all available clusters. At the beginning of the K-means clustering process, all datapoints are randomly assigned to one of the K possible clusters and the centroids for each cluster are calculated. Cluster assignments are updated based on the closest centroid to each datapoint and then centroids are recalculated. This process continues until cluster assignments no longer change or some maximum number of iterations has been reached.

One weakness of the K-means clustering algorithm that makes the method unsuitable for certain clustering problems is that it requires a specified number of desired resulting clusters. This is not a problem for certain clustering problems in which there is a pre-conceived notion of the expected clusters within a data-set however, this is not always the case for clustering problems. Alternatively, number of clusters can be determined without known class labels via visual inspection of the data as well as other procedural methods. One other drawback to the K-means algorithm is that the

Algorithm 1 K-means Clustering

```
function K-means(data,$k$)
    Rand(x,y): choose random integer between x and y
    CalcCentroids(x): Calculate cluster centroids from dataset x
    AssignLabel(x,c): Assign point x cluster label of closest centroid in c
    for i = 0,. . . ,data.size do
        data[i].label $\leftarrow$ Rand$(1, k)$
    end for
    centroids $\leftarrow$ CalcCentroids($data$)
    oldCentroids $\leftarrow$ null
    while Centroids $\neq$ oldCentroids do
        oldCentroids $\leftarrow$ centroids
        for i = 0,. . . ,data.size do
            data[i].label $\leftarrow$ AssignLabel(data[i],centroids)
        end for
        centroids $\leftarrow$ CalcCentroids(data)
    end while
    return data.labels
end function
```

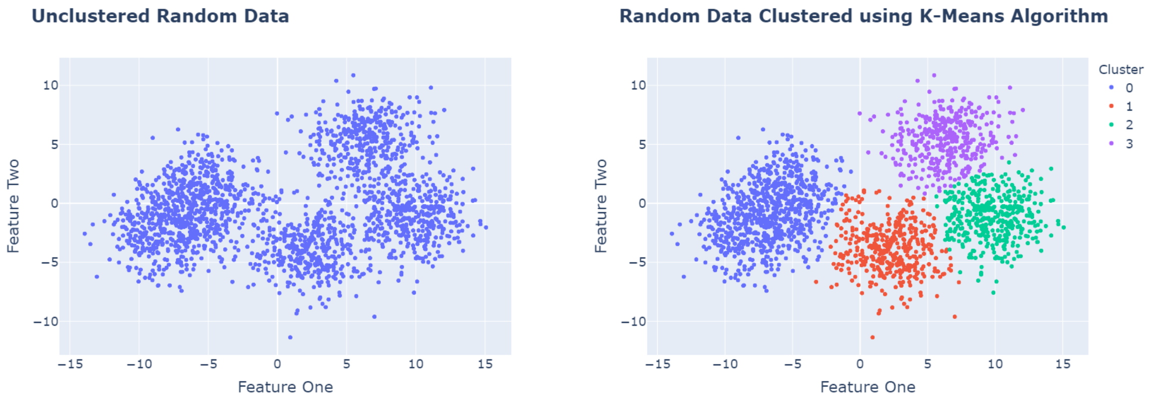


Figure 7: Visualization of K-Means clustering

cluster assignments produced are not deterministic. Due to the stochastic nature of the initial cluster assignments, running the K-means algorithm multiple times on the same dataset is not guaranteed to produce the same cluster assignments. There has been a significant amount of research into more informed cluster initialization procedures to improve K-means performance. Some of the more notable methodologies have been summarized in [17]. Lastly, the underlying assumption of the K-means algorithm is that clusters within a dataset tend to take the form of a distinct number of gaussian distributions. If the data doesn't comform to such a structure, the efficacy of the K-means algorithm is limited.

2.3.2 DBSCAN

Density Based Spatial Clustering of applications with Noise (DBSCAN) is a clustering algorithm described in [18] that operates in a fundamentally different way to the K-means algorithm. Similarly to K-means, however, it also uses a distance metric such as euclidean distance with which to make cluster assignments. DBSCAN is based on the premise that clusters can be defined as groups of densely packed datapoints that are well separated by areas with a low density of datapoints.

The DBSCAN algorithm begins with the definition of two hyper-parameters: ϵ and min_{pts}. The ϵ term defines how close two points need to be together to fall within the "ϵ-neighborhood" of each other. The min_{pts} term refers to how many datapoints need to be in the ϵ-neighborhood of some other point x for x to be a "core point". The DBSCAN algorithm defines three different categories for points within a dataset:

- **Core Points** - Points that have $\geq min_{pts}$ datapoints within their neighborhood
- **Border Points** - Points that do not meet the criteria of a core point but are have a distance $\leq \epsilon$ (i.e. is in the ϵ-neighborhood) from a core point
- **Noise Point** - A point that neither meets the criteria of a core point nor is reachable from a core point. These points are assigned no cluster label

The DBSCAN algorithm for producing cluster assignments is fully described in Algorithm 2.

One of the principal benefits of the DBSCAN algorithm is its ability to handle more complex cluster geometries within feature space. Many clustering algorithms

Algorithm 2 DBSCAN Algorithm

```
function DBSCAN(data,$\epsilon$, $min_{pts}$)
    GetNeighbors(x,$\epsilon$): return set of points $\leq \epsilon$ from x
    CurrentLabel ← 0
    for i = 1,. . . , data.size do
        point ← data[i]
        if point.label = unclassified then
            if GetNeighbors(point,$\epsilon$).size $\geq min_{pts}$ then
                ExpandClusters(point,CurrentLabel)
                CurrentLabel += 1
            end if
        else
            point.label = noise
        end if
    end for
    return data.labels
end function
```

Algorithm 3 Expand Cluster: Helper function for main DBSCAN algorithm

```
function ExpandClusters(point,CurrentLabel,$\epsilon$, $min_{pts}$)
    InCluster ← GetNeighbors(point,$\epsilon$)
    while InCluster.empty == False do
        CurrentPoint ← InCluster.front
        InCluster.delete(CurrentPoint)
        if CurrentPoint.label == unclassified then
            CurrentPoint.label ← CurrentLabel
            Result ← GetNeighbors(CurrentPoint,$\epsilon$)
            if Result.size $\geq min_{pts}$ then
                InCluster ← InCluster ∪ Result
            end if
        end if
    end while
end function
```

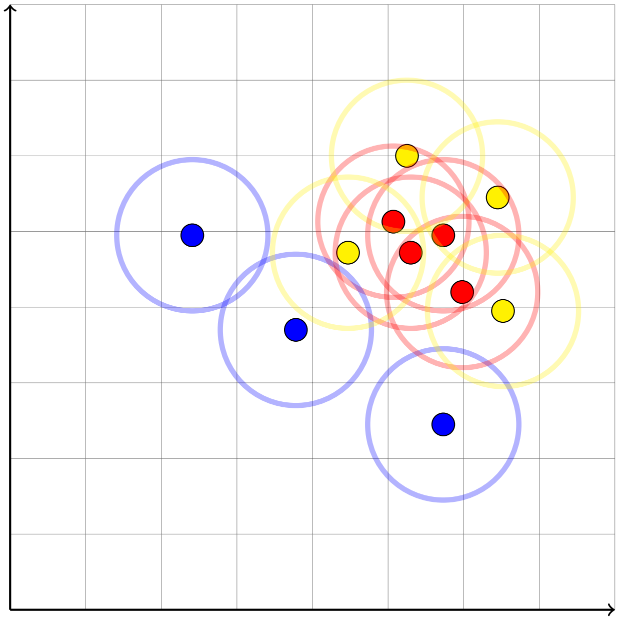


Figure 8: Visualization of DBSCAN clustering with $\min_{pts} = 3$ $\epsilon = 1$. Red points represent core points of cluster. Yellow points represent border points. Blue points represent outliers/noise points

such as K-means operates on strong assumptions about the shape and distribution of clusters within a dataset. DBSCAN however does not have this assumption about the geometry of clusters. DBSCAN is also useful as it does not rely on a specified number of resulting clusters. The DBSCAN algorithm will produce as many clusters as there are densely packed regions of datapoints. This makes the algorithm more suitable to problems in which there is less known about the dataset and the potential clusters within. Lastly, a notable property of the DBSCAN algorithm is that not all datapoints are guaranteed to be assigned to any cluster at all. This may or not be a downside to the algorithm depending on the nature of the intended clustering problem. A notable property of the DBSCAN algorithm is that the effectiveness of the clustering is highly sensitive to the ϵ value chosen. Choosing a poor value for this parameter can result in extreme situations in which either all observations are grouped into a single cluster or all observations are labeled as outliers/noise points.

2.3.3 Mean Shift Clustering

Mean Shift is a clustering algorithm proposed in [19] that is similar in approach to the K-means algorithm relying on iterative calculation of cluster centroids. This algorithm however also addresses one of the primary downsides to the K-means algorithm, the need to specify a number of desired clusters.

The Mean Shift algorithm assigns cluster labels to points within a dataset by ascending the gradient of the density of the dataset in space to identify the different modes of the dataset. It does this process by going through each observation of the dataset and iteratively moving in the direction of its mean shift vector until it converges/ stops changing(i.e. the mean shift vector approaches zero). All data points converge to a mode of the dataset and points which converge to the same (or very nearly the same) point are given the same cluster label. The mean shift vector $v(p)$ for a given position p in n-dimensional space with datapoints $X = x_{i=1...n}$ within a specified distance/bandwidth bw of p is calculated as shown in Equation (10). The distance bw is the main tunable parameter for the mean shift algorithm and effects the number of resulting clusters of the algorithm.

$$v(p) = \frac{\Sigma_{x_i \epsilon X}(x_i - p)x_i}{\Sigma_{x_i \epsilon X}(x_i - p)} \tag{10}$$

The Mean Shift algorithm has some similarity to the DBSCAN clustering algorithm in that clusters are assigned according to a datapoint's relationship to some area of high density within the overall dataset. The Mean Shift algorithm doesn't have the assumption that clusters are well separated by areas of low density that DBSCAN does which results in labeling datapoints as outliers. This makes mean shift more appropriate in feature spaces in which there is a more gradual transition between regions of high density.

Algorithm 4 Mean Shift Clustering

```
function MeanShift(data,$bw$, $max\_iter$)
    ShiftVector(x,$bw$): Returns meanshift vector for point x and bandwidth $bw$
    for i = 0,...,data.size do
        Center ← data[i].position
        while V != 0 and n < $max\_iter$ do
            V ← ShiftVector(Center,$bw$)
            Center ← Center + V
            $max\_iter+ = 1$
        end while
        data[i].clusterCenter ← Center
    end for
    Prune all cluster centers retaining local maxima
    Each datapoint with the same cluster center is given the same cluster label
    return data.labels
end function
```

2.4 RF-Fingerprinting

RF-Fingerprinting refers to the task of mathematically characterizing RF waveforms transmitted by wireless communications devices typically for the purposes of machine learning research. The key assumption of RF-fingerprinting research is that there are unavoidable slight variations in RF hardware that, in aggregate, manifest in the physical transmissions from that device. A detailed mathematical analysis of how these variations can manifest themselves within an RF waveform can be seen in [20]. The goal of the bulk of RF-fingerprinting research is to leverage modern machine learning techniques to improve the security of wireless communications systems [21] however, the field has also been applied to other tasks such as counterfeit detection for embedded circuit devices [22]. The existing research into the domain of RF-Fingerprinting covers a broad scope of known machine learning methodologies. The existing RF fingerprinting literature can be generally divided into two major categories: those utilizing manual feature engineering with traditional machine learning models [23] and those utilizing deep learning [24].

The use of manually defined feature information within machine learning research is a common practice as it leverages the domain knowledge of experts within a given field. An experienced researcher within the realm of RF-Fingerprinting will have a good intuition of which features are most applicable to a given machine learning task and knowledge of how different aspects of the collection environments, communications protocol, etc. may affect such features. Radio Frequency Distinct Native Attribute (RF-DNA) fingerprinting is one example of a feature extraction methodology used for RF-fingerprinting research that is primarily researched at the Air Force Institute of Technology (AFIT)[25]. This methodology operates by dividing signal observations into a number of equally sized subsections and calculating statistical measurements on the instantaneous amplitude, frequency, and phase of these subsections. These statistical measurements often consist of skewness, variance, and kurtosis and these measurements constitute the feature vectors used for machine learning problems. The RF-DNA process is an effective example of traditional RF-fingerprinting techniques which are typically performed by calculating some specified measurements from RF signals deemed to have discriminate value to machine learning models. These features are often put through feature selection and projection steps before being used in different types of classification models such as Multiple Discriminant Analysis / Maximum Likelihood (MDA/ML) [26], Random Forrest (RndF) [27], and Learning Vector Quantization (LVQ) [28].

Deep learning has been applied to RF-fingerprinting to forego the process of manually extracting feature information from RF waveforms by leveraging deep learning's ability to learn features from complex non-linear information present in raw data. A variety of different deep learning techniques have been applied to the RF-fingerprinting problem with some commonalities present among the majority of the existing literature. A common design choice in deep learning based RF-fingerprinting is the incorporation of convolutional layers in some capacity in neural nets used. The network architecture proposed in [29] is a good example of the application of CNNs to RF-fingerprinting in such a way that is intended to be protocol agnostic. The network proposed in [30] by contrast, utilizes convolutional layers in conjunction with Long Short Term Memory (LSTM) layers, a popular newtwork layer used in Recurrent Nerual Networks (RNN) for time series data to perform binary same vs. not same classification for ZigBee emitters. Yu et al. propose in [31], a RF-fingerprinting classification model which combines a traditional classification network with a convolutional autoencoder and observed that the presence of autoencoder reconstruction loss was able to achieve better performance compared a typical CNN for the same classification problem.

III. Methodology

This chapter provides a description of the design and implementation of the experiments performed to evaluate the research questions proposed in Chapter I for this research. In summary, these experiments are designed to evaluate if an RF-fingerprinting model can be used to extract feature vectors from waveforms that form clusters using various clustering algorithms that are highly representative of the device of origin for said waveforms. The resulting models are evaluated on various test data sets to determine how well models can perform in different scenarios. The different datasets evaluate how models perform on both devices used in the training process as well as devices unseen to the model. Finally different algorithms are used to compare the clustering performance and determine the most effective algorithm for these feature vectors.

First, the signal collection procedures as well as the descriptions of the training and evaluation data-sets will be discussed. The pre-processing and data preparation steps will then be presented. Next, the various feature extraction model architectures and training processes will be provided. Finally the clustering methodologies used will be explained as well as the metrics used to evaluate resulting performance of the overall clustering process.

3.1 Signal Collection Experimental Setup

The data being used to perform the experiments presented in this thesis consist of two separate datasets containing characteristically different signal collections. The first dataset, that will henceforth be referred to as the **lab** dataset, consists of IEEE 802.11a/g WiFi transmissions from 19 different emitters of the same manufacturer and model number. Each of the signal observations in this dataset were recorded in a controlled laboratory environment and contain bit-wise identical data (to include identical MAC addresses). The purpose of this dataset is to act as the ideal RF-Fingerprinting scenario in which the only possible differences between classes must be those characteristic to the device hardware itself. The second dataset,is the **wild** dataset that consists of IEEE 802.11a/g signal collections observed in various public locations both indoor and outdoor. The **wild** dataset contains observations from $>$ 53k different devices and the number of signal observations per device varies. Each observation in this dataset is given a label associated with the specific transmitting device as well as the manufacturer of the transmitter both based on the MAC address associated with the observation. The purpose of this dataset is to provide a dataset representative of a real collection scenario to evaluate the performance of RF-Fingerprinting models on.

The hardware configuration used in the signal capture process for both datasets can be seen in Figure 9. The primary hardware used to perform the signal capture and

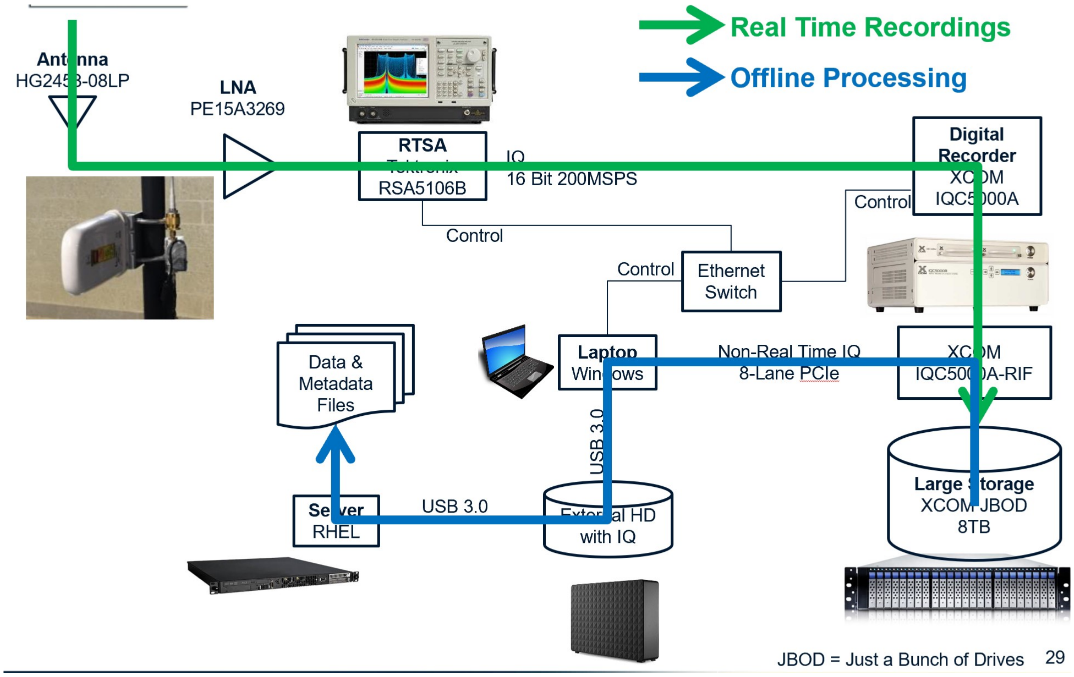


Figure 9: Signal Capture Hardware Diagram

recordings are the Tektronix 5016B Real Time Spectrum Analyser (RTSA) and the XCOM IQC5000A RF signal recorder. The RTSA samples the incoming waveforms at a frequency of $F_{samp} = 200$MHz and a capture bandwidth of $BW_{capture} = 165$MHz. Durring capture, 1.25μs of recording is appended to the front and back of the detected signal to capture transient behavior for devices. After the signals are collected and recorded, they are processed and saved in the sigmf file format. Sigmf is a JSON based file format created by the GNU Radio Foundation to facilitate and standardize the recordings of RF data. Each sigmf file is composed of a meta-data file containing important information about the signal collections (i.e. sampling frequency, recording hardware, collection bandwidth, etc.) as well as a data file containing the raw binary of the signal collections. In this dataset, the RF collections are recorded as 16-bit quadrature samples meaning each sample consists of a 16 bit integer representing the I channel value followed by a 16 bit integer representing the Q channel value.

3.2 Signal Pre-processing

The signal processing steps for all data used in these experiments consists of obtaining the baseband signal for each observation, isolating the preamble for each observation, and converting the observation into the PyTorch tensor data structure that can be used to train and evaluate neural networks. The preamble is chosen as the signal Region of Interest (ROI) as it contains no coded information about the transmitting device or the data payload of the transmission. The decision to utilize the preamble portion of each transmission is inspired by the RF-DNA RF-Fingerprinting technique with likewise calculates feature information from this portion of a collected signal. The motivation behind this is to limit machine learning models from latching onto discriminatory information present in the signal that is not resultant of the physical characteristics of the emitter hardware. For example, the encoded data or

MAC addresses in signal transmissions can be used by machine learning models to discriminate signals without consideration for the hardware characteristics which is contrary to the aim of RF-Fingerprinting.

For all training and testing collections, the first step of pre-processing is down-converting the signal to baseband frequency. The metadata file for each signal contains the center frequency (f_{cent}) of the acquisition as well as the upper (f_u) and lower (f_l) edge frequencies of the channel being transmitted on for each observation. The signal is then down-converted by centering the specified frequency channel at f = 0.

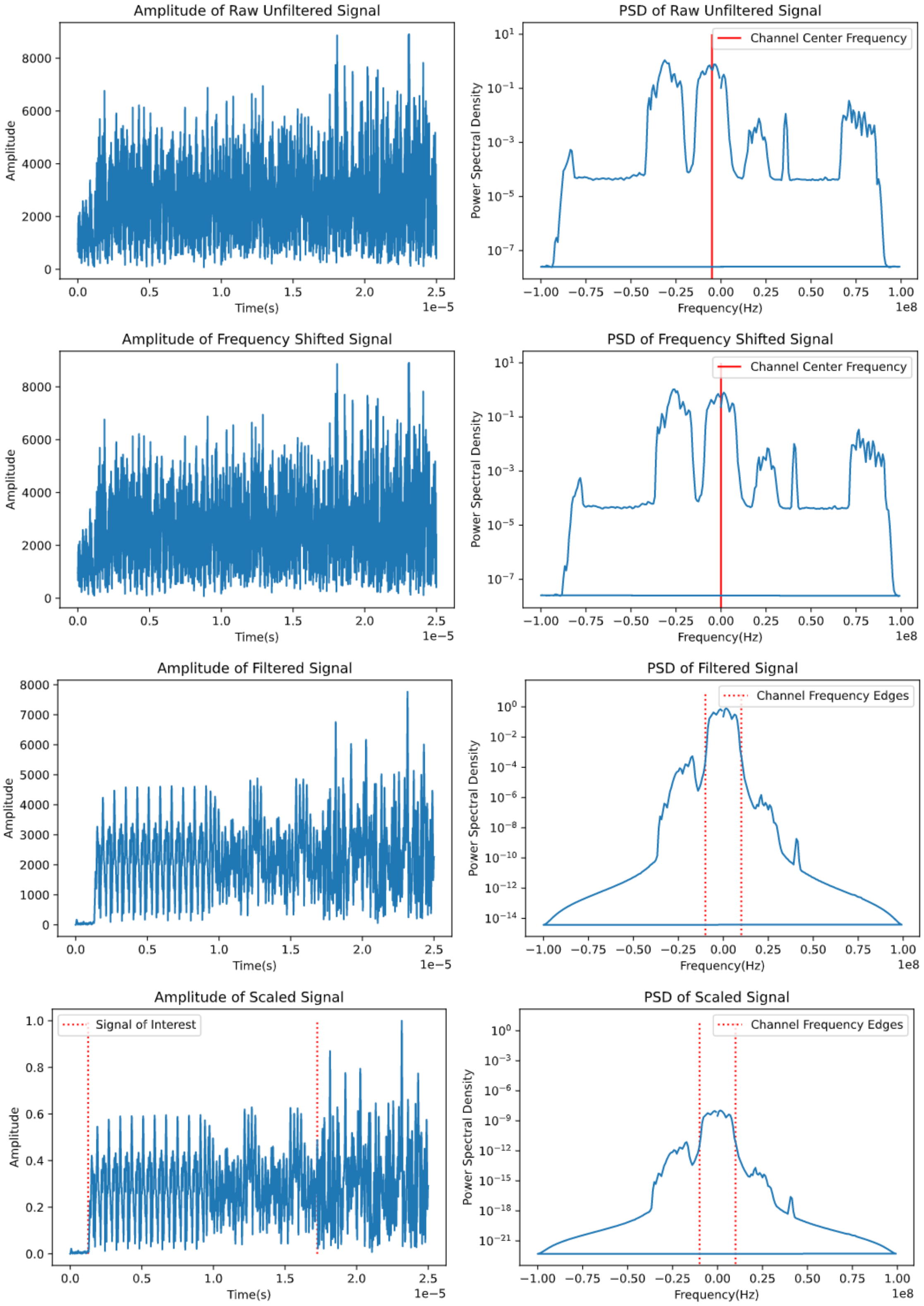
The base-band conversion process performed here is based on the edge frequencies of the WiFi channel being used by a given transmission. This does not correct for minor frequency variations that can result from the transmission process. Performing fine frequency and phase adjustment can be done via comparison to a generated ideal preamble sequence as is the intended purpose for this portion of the signal. This is a crucial step in performing software based demodulation of communications signals however, further research is needed to determine the impact of such fine adjustments on performance of the RF-Fingerprinting models presented in this thesis.

After conversion of signals to base-band frequency, filtering is applied to remove frequency content from the signal not belonging to the frequency channel of interest. According to the WiFi standard, a WiFi transmission channel has a bandwidth of $BW_{channel} = 20MHz$. A baseband signal centered at $0Hz$ can then have it's frequency content isolated by applying a Low Pass Filter (LPF) with a cutoff frequency $f_{cutoff} = 10MHz$. This filter is implemented here as a fifth order Butterworth filter using the scipy.signal python library.

As described in Section 2.1 the non-HT preamble sequence consists of 10 repetitions of a short training symbol followed by 2 repetitions of a long training symbol for a total duration of 16μs. At a sampling rate of $F_{samp} = 200$MSps, the preamble ROI for each signal has a size of 3200 samples. Since there are 1.25μs appended before the signal for each collection, the preamble can then be isolated by selecting samples [250,3450] for each observation. Once the preamble is isolated for each observation, the signal is normalized by dividing by its maximum amplitude resulting in signals whose amplitude ranges from zero to one.

At this point, the observations are converted into a Pytorch tensor with dimensions $N_{obs} \times N_{channels} \times N_{samples}$. The N_{obs} corresponds to the total number of signal observations in a given training or testing dataset. This value is different for different training and testing sets. The $N_{channels}$ dimension corresponds to the number of channels used for one dimensional convolution operations. In this case two channels are used one corresponding to the I component of a sample and the other corresponding to the Q component. The $N_{samples}$ dimension is the total number of quadrature samples for a given observation which in this case is 3200. The overall size of a dataset tensor would then be $N_{obs} \times 2 \times 3200$.

Figure 10: Visualization of the total signal preprocessing procedure for a single observation



3.3 Datasets

Although all observations being used in the performance of the experiments detailed in this thesis are pulled from the aforementioned **lab** and **wild** datasets, the observations from these datasets are gathered into smaller datasets used for different purposes in the training and evaluation process of models. All datasets are used for one of three main purposes; training, validation, and testing. Training datasets consist of observations used to train models. Validation datasets consist of a small selection of observations not present within a training dataset to evaluate model performance during the training process to aid in the fine tuning and design of models. It is generally considered bad practice in machine learning to report performance on a validation set as it is explicitly used in the design of the model. Test datasets consist of observations present in neither validation or training sets. These consists of observations not used in any way in the model construction or training process and thus represent an objective measure of performance.

For this research, a single training and validation dataset is created to create the various models to be evaluated. There are however multiple testing sets created to evaluate model performance on various RF-Fingerprinting scenarios. The specific details of the various datasets can be seen in Table 1. Estimates for the average estimated SNR per device and for the total datasets can be seen in Appendix B.

The three different testing datasets created with the same number of devices (N_D) are intended to evaluate different aspects of the models performance. The Lab Test dataset is the most representative of the data used to train the model and thus performance on this dataset is somewhat analogous to an n-class classification problem. The Unseen test dataset is meant to evaluate how a model's performance generalizes to observations from devices previously unseen to the network in a collection environment identical to the training data. The Wild Test Set is meant to evaluate how well

Table 1: **Dataset Descriptions**

Name	N_D	Obs/Dev	Description	Dev #
Training	9	5000	Observations from 9 different devices taken from the lab dataset	0-8
Validation	9	100	Different observations from the same 9 devices in the Training set	0-8
Lab Test Set	9	1000	Different observations from the same 9 devices in the Training set	0-8
Unseen Test Set	9	1000	Observations from 9 devices not present in the Training set taken from the lab dataset	9-17
Wild Test Set	9	1000	Observations taken from 9 devices selected from wild dataset	18-26

the model generalizes to observations from devices that are characteristically different from the Training set. In this test set both the collection environments are different from the training set as well as the model and manufacturer of devices being tested. This test set provides the most significant measure of a model's ability to generalize to a more realistic collection environment.

3.4 Deep Learning Models

The design of neural network architectures to achieve some machine learning task is a significantly nuanced and complicated discipline. There are many different types of neural network structures such as autoencoders, Generative Adversarial Networks (GAN), and RNNs as well as combinations thereof which are all suited to different problem domains. Additionally, the specific architecture (i.e. number and size of layers, parameters of layers, etc.) can all be tuned to great impact on overall model performance. Due to the long training times typically required to produce a deep learning model, there is a small number total number of model permutations that are feasible to train and evaluate in a timely fashion. Therefore a typical choice is to adopt an existing model architecture for a related problem area and make modifications to suit the specific problem being researched.

3.4.1 Model Architecture

When choosing a model architecture, it is important to consider both the objective of the machine learning problem being performed as well as the nature of the data being used as the input to the network. Each of these will inform the design choices made in the network architecture. When evaluating these considerations for the problem being proposed in this thesis, this can be done as such:

- Machine Learning Objectives:
    - Extract feature information from input data
    - Ensure that features extracted are well seperated based on class
- Input Data Properties:
    - Temporally organized data
    - Each individual time point contains multiple pieces of information (I/Q value, magnitude, phase, etc.)
    - Each observation is composed of the same sequence of data symbols (WiFI preamble sequence)
    - There is only minor variation between observations of different classes

Observing the machine learning objectives will lead toward a choice for overall style of neural network being used. For this problem the primary function of the neural network model is to extract feature information. Extraction of feature information is an implicit aspect in many deep learning models however the extraction of features is made most explicit as the objective of Autoencoder neural networks. Therefore an autoencoder style model is the natural choice for this problem. Autoencoder models do not however, address the second machine learning objective as there is no guarantee that the encoded output is well seperated by class, particularly for the input data being used which has only subtle variations between classes. This aspect of the machine learning objective will be accomplished via use of the triplet loss function described in Section 2.2.6. The specific implementation of the triplet loss will be described in Section 3.4.2.

The characteristics of the model input data will determine the specific architecture implementation details. In particular, the fact that the input data being used in this case is temporally organized will guide the architecture decisions. Three of the most common types of neural network layers for most applications include dense layers, convolutional layers, and recurrent layers. The general use case for these three kinds of layers can be described as such:

- **Dense Layers** - A collection of related feature information without any natural ordering
- **Convolutional Layers** - Data that has an ordering such that the characteristic of groups nearby features contains important information about input data
- **Recurrent Layers** - Data which is defined by the sequence of and relation between a set of discrete possible values

The two of these layers that are the most appropriate for the type of data being used in these models would be convolutional and recurrent layers as they are both commonly used for temporally organized data. For this research however, convolutional layers were deemed to be the more appropriate option. Recurrent layers perform well in instances where the ordering of distinct shapes/symbols within a waveform is of primary importance. The input data being used in this case consists of only preamble signals which is guaranteed to have the same sequence of OFDM symbols. This research is more focused on the variations in the shape of a waveform that are characterstic of physical device hardware. To compare the two approaches, a recurrent network can be thought of as analogous to identifying a speaker based on their diction. A convolutional type network can by contrast be thought of as identifying a speaker based on the tone of voice while speaking an identical phrase.

With the choice of a convolutional autoencoder model made, the next step is to define the makeup of a single convolutional block in this network. The choice of a one dimensional convolutional layer is made using PyTorch's Conv1d layer as the signal varies primarily with respect to one dimension that being time. Another possible choice in this case is to use a two dimensional convolutional layer with one dimension corresponding to time and the other corresponding to the I and Q channel. The I and Q channel were instead represented as multiple channels of the one dimensional convolution to make explicit that each I/Q value is associated with a single point in time. Each convolutional layer was given a stride and dilation factor equal to 1 and the kernel size l_k, padding p, and number of filters N_f parameters were varied at different depths of the model. The kernel size k for the first layer is chosen to be 9 and is decreased by two for each successive layer of the model. The number of filters for the first convolutional layer is set to 10 and is increased by 5 for each successive layer. Each convolutional layer is then given a zero padding value p equal to:

$$p = \frac{k-1}{2} \tag{11}$$

The padding is given this value to preserve the dimension from input to output of the convolutional layer. The choice of increasing the number of filters and decreasing kernel size is consistent with proven effective CNN architectures such as AlexNet [32] and LeNet-5 [33]. The Rectified Linear Unit (ReLU) activation function is then applied to the output of each convolutional layer to introduce non-linearity. The

ReLU function is defined as such:

$$ReLU(x) = max(0, x) \tag{12}$$

Next the output is fed through a batch normalization layer as described in section 2.2.4. The batch normalization layers are implemented using PyTorch's BatchNorm1d layer. The final component of a convolutional block in this model is a maximum pooling layer with a window size of 2 implemented with PyTorch's MaxPool1d layer.

After four convolutional blocks as described above, the resulting tensor is flattened and fed through a single dense layer with a linear activation function in order to achieve the desired latent dimension z_{dim} for the feature encoding. This layer is referred to as the bottleneck layer and its output is the encoding of the input data. The encoding is then fed through another dense layer with the ReLU activation function applied and reshaped to the same dimension as the output of the fourth convolutional block. The output is finally fed through four more convolutional blocks in reverse order to achieve the original input dimension. The overall model structure can be seen in Table 2. This model architecture will be referred to as the ClusterAE model.

3.4.2 Training Methodology

With a fully realized model architecture conceived the final step is to clearly define the training process. With deep learning models the key decisions to be made here include choice of loss function, optimization algorithm, number of epochs (full passes through the training data), and batch size (size of chunks the training data is divided into for each gradient step). Of these considerations, the most significant is choice of loss function. The loss function determines how progress toward the objective of the

Table 2: **ClusterAE Model Architecture**

Layer Type	l_k	N_f	p	**Output Dim**	**Activation**
Convolutional Block 1					
Conv1d	9	10	4	10 x 3200	ReLU
BatchNorm1d	-	-	-	10 x 3200	None
MaxPool1d	2	-	0	10 x 1600	None
Convolutional Block 2					
Conv1d	7	15	3	15 x 1600	ReLU
BatchNorm1d	-	-	-	15 x 1600	None
MaxPool1d	2	-	0	15 x 800	None
Convolutional Block 3					
Conv1d	5	20	2	20 x 800	ReLU
BatchNorm1d	-	-	-	20 x 800	None
MaxPool1d	2	-	0	20 x 400	None
Convolutional Block 4					
Conv1d	3	25	1	25 x 400	ReLU
BatchNorm1d	-	-	-	25 x 400	None
MaxPool1d	2	-	0	25 x 200	None
Latent Dimension Conversion					
Name	**Input Size**	**-**	**-**	**Output Dim**	**Activation**
Flatten	-	-	-	5000	None
Dense	5000	-	-	z_{dim}	None
Dense	z_{dim}	-	-	5000	ReLU
Reshape	5000	-	-	25 x 200	None
Layer Type	l_k	N_f	p	**Output Dim**	**Activation**
Convolutional Block 5					
BatchNorm1d	-	-	-	25 x 200	None
Upsample	2	-	0	25 x 400	None
Conv1d	3	20	1	20 x 400	ReLU
Convolutional Block 6					
BatchNorm1d	-	-	-	20 x 400	None
Upsample	2	-	0	20 x 800	None
Conv1d	5	15	2	15 x 800	ReLU
Convolutional Block 7					
BatchNorm1d	-	-	-	15 x 800	None
Upsample	2	-	0	15 x 1600	None
Conv1d	7	10	3	10 x 1600	ReLU
Convolutional Block 8					
BatchNorm1d	-	-	-	10 x 1600	None
Upsample	2	-	0	10 x 3200	None
Conv1d	9	2	4	2 x 3200	None

model is quantified and all trainable parameters in the model are trained to achieve a better result for this metric.

3.4.2.1 Loss Function

The loss function used to train this model consists of two components corresponding to the two main objectives of the machine learning model described in Section 3.4.1. The first component of the loss is the MSE between the input and output of the model (see Section 2.2.5 for details) and is the most common choice of loss function for an autoencoder. This portion of the loss will henceforth be referred to as Autoencoder loss. The objective of the Autoencoder loss is to ensure that features learned in the bottleneck layer of the network contain sufficient feature information to fully represent the input data. The second component of the loss function is the triplet loss function described in Section 2.2.6. The triplet loss function is applied to the output of the encoding portion of the model. This portion of the loss function is intended to ensure that the learned features have good separation between observations from different devices in the encoding layer of the model. This means that any one training example must be composed of three separate signal observations: an anchor observation a, a different observation from the same device p, and an observation from a different device p. The two components of the loss function are then given a weighting value λ that allows different levels of emphasis to be placed on the two loss components. Let the output of the encoding layer of the model for an input x be x_e and the output of the total model be $\hat{x}$. The overall loss function for a single training triplet a, p, n is then defined as:

$$\mathcal{L}(a, p, n) = (1 - \lambda) * MSE(a, \hat{a}) + \lambda * TripletLoss(a_e, p_e, n_e) \tag{13}$$

3.4.2.2 Triplet Generation

With the incorporation of triplet loss the process for how triplet pairs are made must be defined. There are multiple methods for making informed triplet pairs from training data detailed in [13]. These methods include choosing the valid triplet groups within the training set that perform the worst on the triplet loss function. Such methods for generating triplet pairs help to increase the speed at which models converge to acceptable performance. These methods however can take a large computation time and can decrease the overall time it takes to train models for large datasets. For this reason triplet pairs are generated without the use of a metric of triplet performance. See Algorithm 5 for the triplet generation process. New triplet pairs are generated at the beginning of each training epoch. This is intended to prevent overfitting of the model as it is being constantly exposed to new triplet pairs for the entirety of the training process.

Algorithm 5 Triplet Generation process

```
SAME(X): all other observations in the same class as x
DIFF(X): all observations from all other classes than x
RANDOMCHOICE(X): chose an element from x at random
for i = 0, ...N do
    a ← dataset[i]
    p ← randomchoice(Same(x))
    n ← randomchoice(Diff(x))
    triplets[i] ← [a,p,n]
end for
return triplets
```

3.4.2.3 Optimization

The next important decision in the definition of the training process is the choice of optimization algorithm. In general, neural networks are trained by iteratively updating each of its trainable parameters(weights, biases, etc.) in accordance with the gradient of its loss function. At each update step, the parameters of the model are updated with the gradient calculated via the back-propagation algorithm multiplied by the learning rate (a scaling factor applied to each gradient update step). The optimization algorithm is a way of controlling the learning rate for each step to alleviate certain problems that arise for non convex loss functions. This can take the form of a pre-defined learning rate schedule for each training epoch as well as methods that continuously re-calculate the learning rate based on previous gradient steps. For this model the Adam optimization algorithm [34] was chosen as it is a common choice within modern deep learning research and has shown to be effective for many deep learning models. The initial learning rate for the Adam optimizer was chosen to be $lr = .001$ as it was empirically shown to work well in preliminary experimentation for this model architecture.

The last step to fully define the training process is to decide on the number of training epochs as well as the batch size. There are multiple ways to decide the number of epochs to use when training a neural network model. A common choice is to use early stopping criteria to halt the training of a model after the loss on the training and validation sets indicate that the model is no longer improving after new passes through the training data. This is an effective method to ensure that models don't become over-fit to the training data. For this research, however, the choice was made to have a consistent number of training epochs over the multiple models trained to remove it as a variable that might affect comparative performance between models. The choice was made to halt training after $N_E = 300$ epochs after preliminary experimentation showed that this was a point at which all models stagnate in terms of training and validation loss. Finally, the batch size was chosen to be $N_B = 50$ examples per batch. Multiple factors can affect choice of batch size when training a neural network model not least of which being the available computational resources required to store large batches in memory. In general large batches mean that there is a shorter total time to make a complete pass through the training data. Larger batch sizes do however tend to result in a slower overall convergence of the model and thus smaller batches are often preferred.

3.4.2.4 Data Augmentation

Data augmentation is a common practice in machine learning which involves introducing distortions/modifications to training data to increase the resulting model's ability to generalize to new unseen data. For this research, the application of Additive White Gaussian Noise (AWGN) to the training data prior to training was used for data augmentation. The use of AWGN was chosen as it is a common practice within RF-Fingerprinting research to simulate noisy channel effects. At the beginning of each training epoch, each observation within the clean training data was augmented with AWGN to achieve an SNR value chosen from a uniform distribution between 15db and the SNR of the unmodified signal.The SNR for each observation is calculated by comparing the average power of the preamble signal and the noise appended to the beginning of each observation. The SNR calculation and AWGN formation procedure for a single observation is as follows:

- Calculate the average power of the preamble signal $x_{signal}[t]$ and the 1.25μs of noise appended to the beginning $x_{noise}[t]$:

$$P_{noise} = \frac{1}{N}\sum_{t=0}^{N}|x_{noise}[t]|^2 \tag{14}$$

$$P_{signal} = (\frac{1}{N}\sum_{t=0}^{N}|x_{signal}[t]|^2) - P_{noise} \tag{15}$$

- Calculate SNR for observation in db

$$SNR = 10\log_{10}(\frac{P_{signal}}{P_{noise}}) \tag{16}$$

- Choose desired SNR value SNR_{new} from uniform distribution between 15db and calculated SNR of observation

- Calculate additional noise power P_{added} required to achieve desired SNR

$$P_{added} = \frac{P_{signal}}{10^{\frac{SNR_{new}}{10}}} - P_{noise} \tag{17}$$

- Generate AWGN signal a[t] = $\mathcal{N}$(0,1)+$\mathcal{N}$(0,1)i

- Scale a[t] to achieve P_{added} and add to original signal

$$x_{AWGN}[t] = x[t] + a[t]\sqrt{P_{added}} \tag{18}$$

The overall training process for each network model is described in Algorithm 6. Each model is trained using the Training Set of observations described in Table 1.

Algorithm 6 Model Training Process

```
function TrainModel(data,$N_B$,$N_E$,$\lambda$)
    AE(x): Output of autoencoder model for input x
    E(x): Encoding of autoencoder model for input x
    for i = 0,. . . , $N_E$ do
        tensor ← AddNoise(data)
        triplets ← GenerateTriplets(tensor)
        for j = 0,. . . ,tensor.size/$N_B$ do
            $a, p, n$ ← triplets[j*$N_B$:(j+1)*$N_B$]
            $\hat{a}$ ← AE($a$)
            $a_e, p_e, n_e$ ← E($a$), E($p$), E($a$)
            loss ← $(1 - \lambda)$*MSE($a, \hat{a}$)+$\lambda$*TripletLoss($a_e, p_e, n_e$)
            Calculate loss gradient
            Update network parameters using Adam optimizer algorithm
        end for
    end for
end function
```

3.5 Clustering Methods

After all models are trained, the performance of each model is evaluated on the different clustering methodologies detailed in Section 2.3. First the observations in the test set being evaluated are fed through the encoding portion of the model being tested. Then the clustering method being tested is performed on the resulting feature vectors to obtain cluster assignments for each observation within the test set. Finally the cluster assignments are compared to the true device labels using the metrics described in Section 3.5.4.

3.5.1 K-means Clustering

The implementation of the K-means clustering algorithm is fairly straight forward in this case as the true number of devices is known which constitutes the only major parameter involved with this algorithm. The K-means clustering assignments are created using the module provided in the sklearn python package using the parameters $n_{clusters} = 9$ because there are nine devices present in each test dataset and max_{iter}

= 1000.

3.5.2 DBSCAN Clustering

The implementation of DBSCAN clustering is not as intuitive as that of the K-means clustering algorithm. The ϵ and min_{pts} parameters can drastically effect the quality of the resulting cluster assignments and values for these parameters that are effective for one dataset do not necessarily translate to others. Factors such as the number of features being used to cluster observations as well as the total number of observations in a given cluster can effect clustering performance. This calls for the need of a more procedural method for selecting the parameters being used to reasonably compare performance across different models. Shubert et al. provide an analysis of practical concerns regarding the DBSCAN clustering algorithm in [35]. The analysis provides a heuristic that will be used to select parameters for these experiments as described below.

The min_pts parameter for each model under consideration is chosen to be equal to the twice the dimension of the latent space z_{dim}. The choice of the ϵ parameter does not have as exact of a method for calculating an appropriate value but there is a helpful heuristic to determine range of effective values. The heuristic operates by first calculating the distance from every observation in the dataset of interest to its kth neighbor where k = min_pts. Next all calculated distances are sorted in descending order and are plotted. The distance value at the most dramatic inflection point of the resulting graph (or the knee of the curve) corresponds to an effective choice for ϵ for the given dataset. The "knee" of a curve does not have a single mathematical definition and thus the method being used to determine the knee point will be that described in [36]. The DBSCAN module from sklearn is used to generate clustering assignments using ϵ and min_pts parameters chosen in this way for each model and dataset. The ϵ parameter calculation is performed directly after obtaining features from the trained models.

3.5.3 Mean Shift Clustering

The Mean Shift clustering algorithm has only one major parameter affecting performance that being the bandwidth. The foundational paper on the Mean Shift algorithm [19] provides heuristics to determine an appropriate value for this parameter. These methods however, are often ineffective for higher dimensional data or require sweeping over a large range of bandwidth values and observing the resulting clustering performance which is computationally expensive. The bandwidth parameter used in the mean shift algorithm is qualitatively similar to the ϵ parameter used in the DBSCAN algorithm and thus the bandwidth parameter will be determined using the same procedure for calculating ϵ described in Section 3.5.2. The mean shift module from sklearn is used with the bandwidth parameter chosen in this way to generate mean shift clustering assignments. The bandwidth parameter calculation is performed directly after obtaining features from the trained models. Preliminary experimentation showed behavior of the Mean Shift clustering algorithm where many (≥ 100) small clusters begin to form at low noise levels. These small clusters in general have roughly uniform distributions among true device label and thus provide no significant clustering benefit. Due to this behavior, the choice was made to prune clusters with size < 50 observations with the observations in such clusters being labeled instead as outlier points.

3.5.4 Evaluation Metrics

The quantification of performance of a clustering algorithm is not as straightforward a process as a typical supervised classification problem. Despite prior knowledge of the known labels of the data being clustered, the differences in the qualitative nature of clusters being produced by different algorithms makes objective evaluation more complex. Evaluation is further complicated in the case of clustering algorithms without a specified number of cluster labels as the number of clusters produced isn't guaranteed to equal the true number of classes. Therefore, multiple evaluation metrics will be used to evaluate clustering performance for these experiments.

The metrics being used for this research are those proposed in [37]. The evaluation metric proposed is called V-measure (V_M) and is calculated by taking the weighted harmonic mean of two other metrics called homogeneity and completeness. Homogeneity(h) and completeness(c) measures are calculated based on the conditional entropy of the dataset between the known device labels and the clustering labels generated. Let N be the number of datapoints in a dataset, C be the set of true device labels, K be the set of cluster assignment labels, and $a_{c,k}$ be the number of datapoints from the cth device in the kth cluster. The exact mathematical definition of homogeneity and completeness is then defined as follows [37]:

$$h = \begin{cases} 1 & \text{if } H(C,K) = 0 \\ 1 - \frac{H(C|K)}{H(C)} & \text{else} \end{cases} \tag{19}$$

$$H(C|K) = -\sum_{k=1}^{|K|} \sum_{c=1}^{|C|} \frac{a_{c,k}}{N} \log \frac{a_{c,k}}{\sum_{c=1}^{|C|} a_{c,k}} \tag{20}$$

$$H(C) = -\sum_{c=1}^{|C|} \frac{\sum_{k=1}^{|K|} a_{c,k}}{N} \log \frac{\sum_{k=1}^{|K|} a_{c,k}}{N} \tag{21}$$

$$c = \begin{cases} 1 & \text{if } H(K,C) = 0 \\ 1 - \frac{H(K|C)}{H(K)} & \text{else} \end{cases} \tag{22}$$

$$H(K|C) = -\sum_{c=1}^{|C|}\sum_{k=1}^{|K|}\frac{a_{c,k}}{N}\log\frac{a_{c,k}}{\sum_{k=1}^{|K|}a_{c,k}} \tag{23}$$

$$H(K) = -\sum_{k=1}^{|K|}\frac{\sum_{c=1}^{|C|}a_{c,k}}{N}\log\frac{\sum_{c=1}^{|C|}a_{c,k}}{N} \tag{24}$$

The two h and c metrics are symmetric and range from [0,1] with a value of one being the best possible score and a value of zero representing a clustering functionally equivalent to random chance. Homogeneity is a measurement of the degree to which observations within a given clustering assignment belong to the same true class label. Completeness conversely is a measure of the degree to which observations within the same class label are assigned to the same cluster. Two extreme possibilities exist for clustering results those being

- All observations are given a different cluster assignments such that there is a single observation in each cluster. In this case Homogeneity $h = 1$ and Completeness is very poor

- All observations are assigned to the same cluster. In this case Completeness $c = 1$ and Homogeneity is very poor.

While the two measures tend to respond inversely to one another in such extreme circumstances, the two measures both evaluate to be close to one in cases in which the clustering very nearly matches the true device labels. The V_M is then a way of giving a single performance metric to directly measure clustering performance based on the two desirable clustering properties of homogeneity and completeness. Unlike performance measures typically used to evaluate supervised classification problems, V_M doesn't have a simple interpretation such as the percentage of observations assigned to the correct cluster. This is because there is not necessarily a “correct” cluster associated with every class label especially in clustering results where the number of unique cluster assignments does not match the number of unique class labels. V_M can be interpreted as the degree of certainty with which one set of labels (cluster assignments or true device labels) can be predicted given knowledge of the other set. A value $V_M = 1$ then means that given either the cluster assignments or the class labels, the other set can be predicted with 100% certainty. A value $V_M = 0$ means that the knowledge of one set of labels gives no benefit at all at predicting the other set of labels. The definition of the V_M for a given clustering assignment is given by [37]:

$$V = \frac{(1 + \beta) * h * c}{(\beta * h) + c} \tag{25}$$

The β parameter is a way of adjusting the V_M score to place more emphasis on either the homogeneity of the completeness. When $\beta > 1$ more emphasisis put on completeness score and a value $\beta < 1$ puts more emphasis on homogeneity. For this research, a $\beta = 1$ parameter value was chosen.

3.6 Experiments

The experiments presented within this thesis will evaluate how variations in the parameters of the ClusterAE model architecture defined above affect the resulting model's clustering performance. The novel design decision of the ClusterAE architecture is the use of the objective function that combines the autoencoder reconstruction loss with triplet loss applied to the encoding layer. The case in which the λ hyperparameter is chosen to be $\lambda = 0$ is then simply a standard convolutional autoencoder (referred to here as a "pure" autoencoder). The clustering results achieved by these pure autoencoder models are presented in Section 4.1 separate from the remaining ClusterAE models incorporating a combined loss function. The two model parameters under consideration include the λ parameter of the loss function and the size of the latent dimension z_{dim} of the model. The λ parameter affects the relative emphasis of the two components of the loss function for the model and the variation of this parameter evaluates how the interaction between the loss components affects clustering performance. The z_{dim} parameter controls the total number of features that are extracted from the raw data. Variations in this parameter provides insight into the number of features required to characterize the differences between emitters. In order to evaluate the effect of these two parameters, four different λ and three different z_{dim} values are chosen and twelve different model are trained for each possible combination of the values for the two parameters. The values evaluated for the two parameters are:

- $\lambda = \{.25,\ .50,\ .75,\ 1.00\}$
- $z_{dim} = \{32,\ 64,\ 128\}$

These experiments are intended to evaluate the ClusterAE architecture as a means of recognizing features from RF-waveforms such that the feature vectors can be clustered in such a way that the clusters are highly representative of the signals device of origin. In order to provide a point of comparison for the performance of the ClusterAE models proposed in this thesis, the Principal Component Analysis (PCA) dimensionality reduction method is performed on the raw quadrature samples as an alternate feature extraction method. Three different PCA implementations were done on the Lab Test Set with a number of output features feats = [32,64,128] to correspond to the three values chosen for the z_{dim} parameter in the ClusterAE models. PCA dimensionality reduction is performed using the PCA module provided in the sklearn python package. Additionally pure autoencoder models (ClusterAE models with $\lambda = 0$) were also evaluated to compare how the features learned by a normal convolutional autoencoder compare to those learned via the ClusterAE model with a combined objective function. The procedure for producing each individual V_M calculation is depicted graphically in Figure 11 and is as follows:

- Select a test set to evaluate from [Lab Test Set, Unseen Test Set, Wild Test Set] at desired simulated SNR level SNR = [-3,0,...,18]

- Generate feature vectors from waveforms using one of the following methods [ClusterAE model, Pure Autoencoder Model, PCA]

- Choose clustering algorithm from [K-means, DBSCAN, Mean Shift] to perform clustering on feature vectors to produce cluster assignment labels for each observation

- Calculate V_M using cluster assignment labels and true device labels for the dataset

For the Lab Test Set and Unseen Test sets, AWGN was applied across the range of SNR = [-3db,18db] in increments of 3db. A single copy of both the Lab Test Set and Unseen Test Set was created at each of the simulated noise values for model evaluation. The simulated SNR values are generated using the same methodology as described in Section 3.4.2.4. This is a common practice within RF-Fingerprinting research as it demonstrates how fingerprinting models respond to noisy environments.

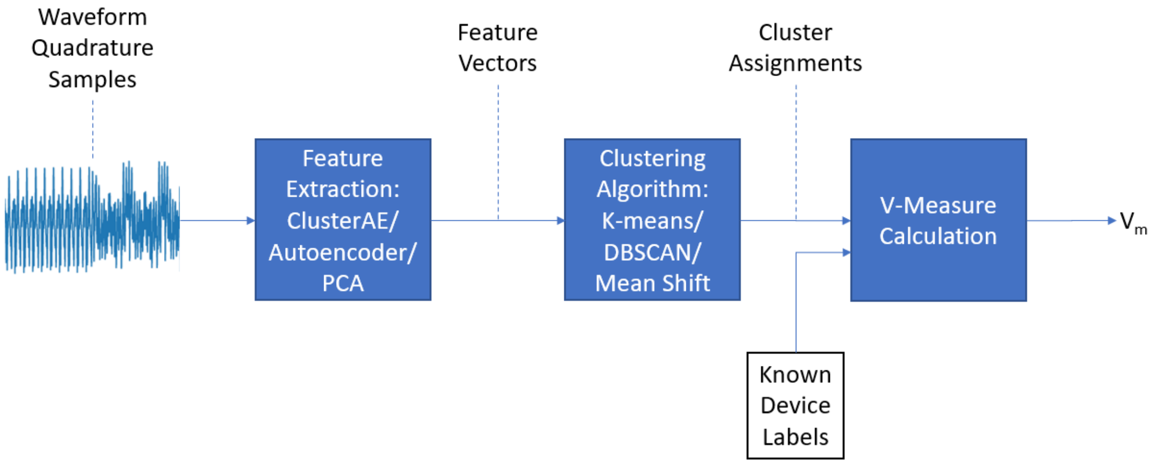


Figure 11: Block Diagram of Cluster Generation and Evaluation Procedure

This was not done for the Wild Test Set as the measure SNR values of the observations in this test set was much more variable and thus achieving a range of usable SNR values is infeasible.

IV. Results and Analysis

This section presents the results of the RF-Fingerprint clustering experiments as described in Chapter III on various datasets of IEEE 802.11a/g transmissions. The RF-fingerprint feature vectors for each observation were generated using the encoder portion of the ClusterAE models described in Section 3.4.1. These feature vectors are then used as the input to various clustering algorithms and cluster assignments are returned for each observation. These cluster assignments are then compared to the known device labels and a measure of performance is calculated using the V-measure (V_M) metric described in Section 3.5.4.

The presentation of the results in this section is organized as follows. First, conventional dimensionality reduction methods are used to produce feature vectors for clustering and the results of the clustering performance on these feature vectors will be presented as a performance baseline for comparison to the ClusterAE models. Next, a model hyper-parameter sweep is performed in order to evaluate how the size of the latent dimension (z_{dim}) of the models and the λ parameter of the objective function effect clustering performance. The performance of the three clustering algorithms described in Section 3.5 on the feature vectors produced by the ClusterAE models is then presented and compared. Finally, a comparison of model performance on the Lab Test Set, Unseen Test Set, and Wild Test Set is presented in order to evaluate how ClusterAE models generalize to new devices.

In situations where different clustering algorithms are not being directly compared, the K-means clustering algorithm is used to compare model performance. This is chosen because the number of clusters produced is controlled with this algorithm and thus the results have more predictable behavior compared to the other two algorithms evaluated. Additionally, in situations where the different test sets are not being compared the Lab Test Set is used as the input dataset.

4.1 Comparison to Conventional Dimensionality Reduction

As there has been minimal work presented on the clustering of RF-fingerprinting feature vectors there is not a foundation of typical expected performance to compare the results of this thesis to. Previously researched RF-fingerprinting techniques not based in deep learning typically rely on some form of feature projection based on knowledge of the device labels (e.g. Multiple Discriminant Analysis (MDA)) in order to produce feature vectors suitable for machine learning models. A core goal of this research is to develop an RF-Fingerprinting methodology that produces feature vectors without an assumption of a set of known devices and thus these previous RF-fingerprinting methods do not provide a suitable comparison to the methodology presented here. Additionally, previous deep learning based fingerprinting methods require a lengthy design and training process that is outside of the scope of this research. For that reason, feature vectors produced by the conventional dimensionality reduction methods Principal Component Analysis (PCA) and pure autoencoder models (ClusterAE models without the addition of triplet loss) are used as a basis of comparison for the models proposed in this thesis.

The clustering results of the PCA derived feature vectors can be seen in Figure 12. The results show a performance score well below $V_M = 0.01$ across all SNR regardless of the number of output features used. These results indicate that such linear projections on raw quadrature sample data is not sufficient to perform clustering on these signal observations.

The clustering results of the pure Autoencoder derived features can be seen in Figure 13. As with the PCA derived features, the pure autoencoder derived features also achieved performance well below $V_M = 0.01$ for all SNRs evaluated. These results show that the features learned by a convolutional autoencoder model trained with reconstruction loss alone are not sufficient to produce suitable feature vectors for

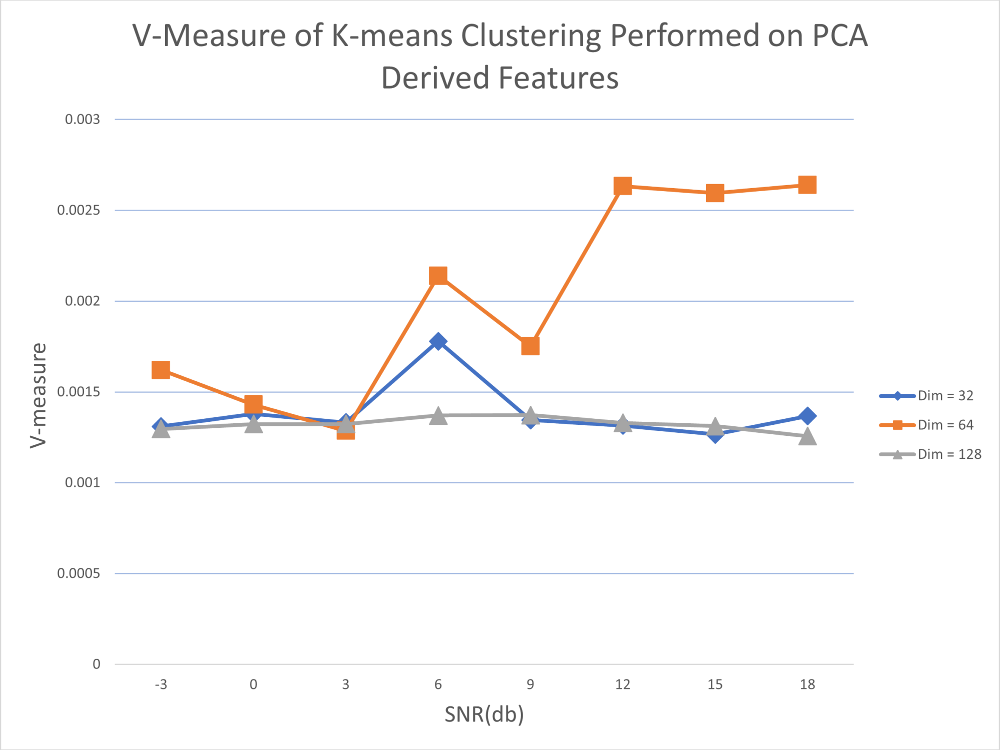


Figure 12: V_M Results of K-means Clustering Assignments on Feautre Sets Produced by PCA Dimensionality Reduction of Raw Signal Data

clustering applications.

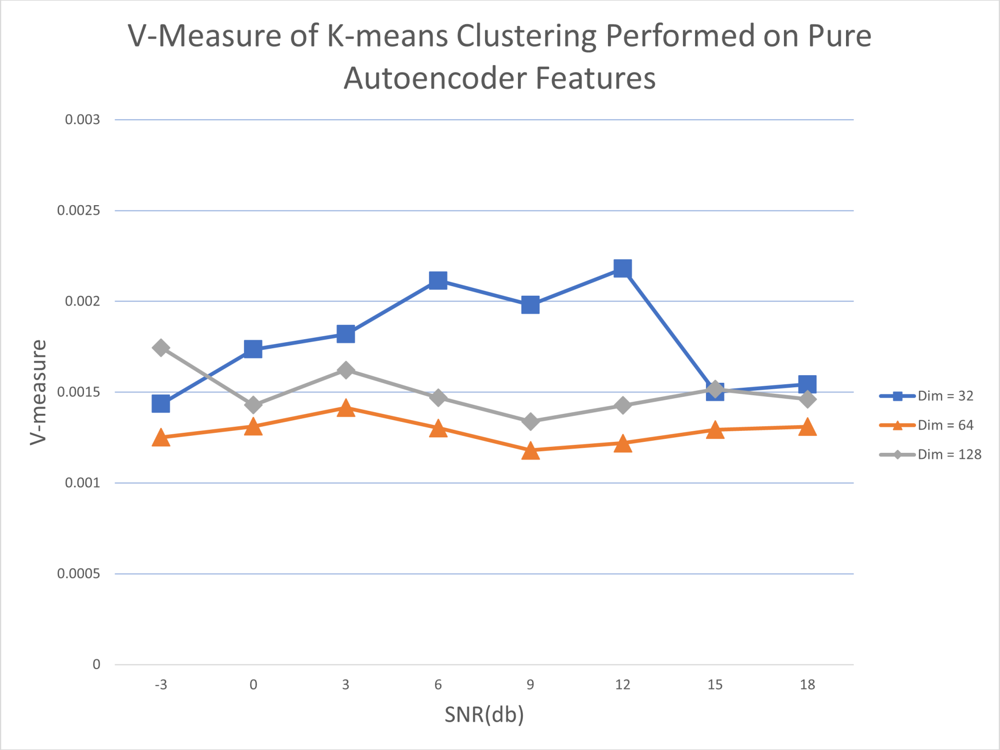


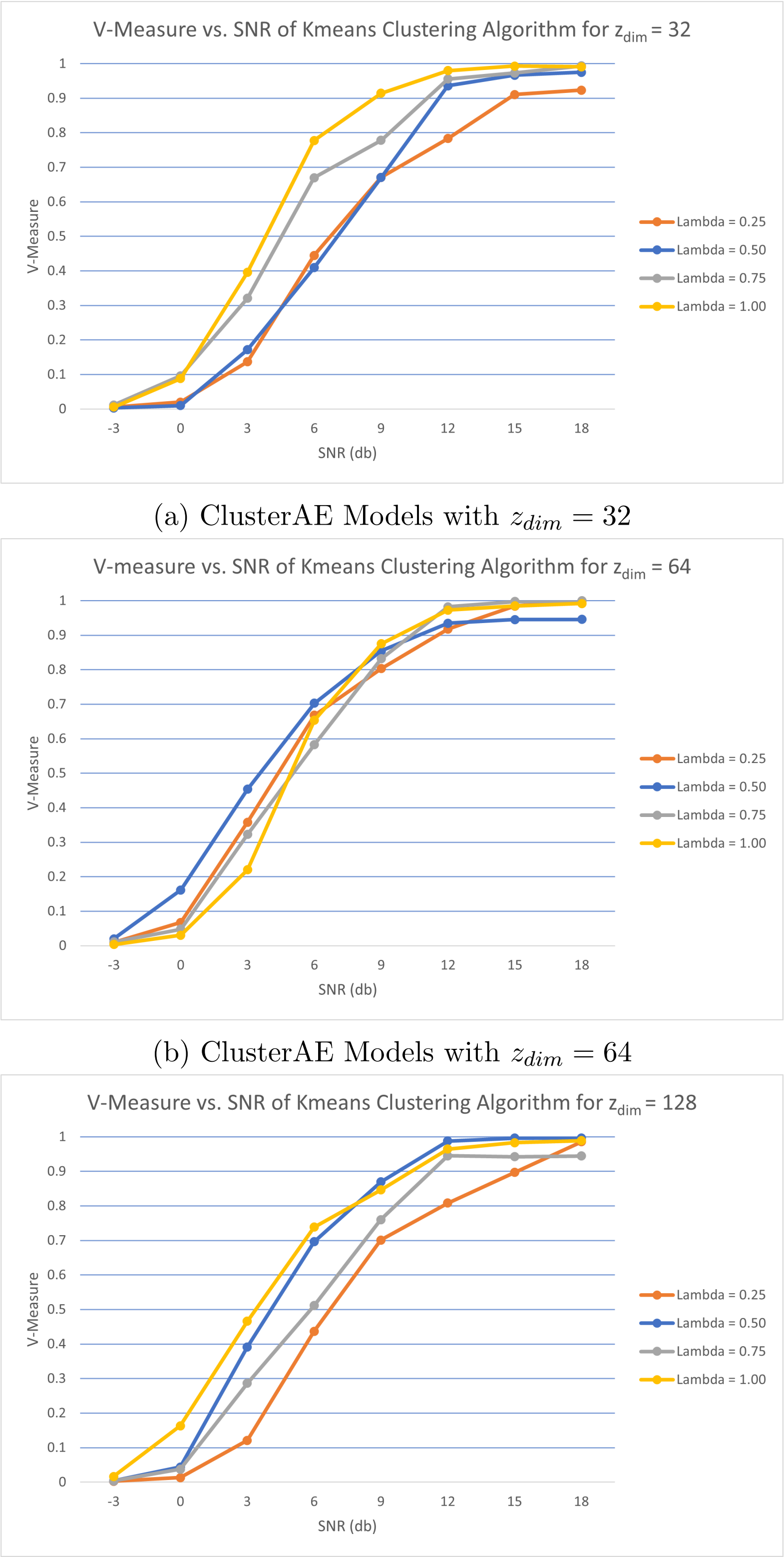
Figure 13: V_M Results of K-means Clustering Assignments on Feautre Sets Produced by Pure Autoencoder Encoding of Raw Signal Data

The results here show that neither linear projections such as PCA nor deep learning based dimensionality reductions such as autoencoder models are sufficient to produce feature vectors suitable for use in unsupervised clustering algorithms. The following results presented in this chapter demonstrates how the models described in Section 3.4.1 are able to produce effective feature vectors for unsupervised clustering through the incorporation of triplet loss into the training objective function.

4.2 Model Hyper-parameter Evaluation

This section provides an examination of how two key model hyper-parameters effect the clustering performance of feature vectors generated by the ClusterAE models described in Section 3.6. The first parameter that is being evaluated is the size of the latent dimension z_{dim} of the models. The value of this parameter controls the total number of features being used to represent each observation. The second parameter being evaluated is the λ parameter of the objective function used to train the various models. This parameter controls the relative weighting between the reconstruction loss and triplet loss portion of the objective function. A value of $\lambda = 0$ indicates the use of only autoencoder reconstruction loss (as seen in Section 4.1) and a value $\lambda = 1$ indicates the use of only triplet loss to train the model. A value in between these two varies the amount of emphasis placed on each loss component. The results in this section evaluates clustering performance on the feature vectors produced by the various models from the Lab Test Set over a range of simulated SNRs.

The results shown in Figure 14 show how the clustering performance of the ClusterAE models is affected by variations in the λ parameter at various sizes of latent dimension. The first observation to be made from these results is that all models evaluated were able to achieve a $V_M \geq 0.9$ at highest simulated SNR = 18db. A V_M score in this range indicates that the cluster assignment labels produced for these



(a) ClusterAE Models with $z_{dim} = 32$

(b) ClusterAE Models with $z_{dim} = 64$

(c) ClusterAE Models with $z_{dim} = 128$

Figure 14: V_M Results of K-means Clustering Performed on ClusterAE Model Feature Vectors for Hyper-parameter values $z_{dim} = \{32,64,128\}$ and $\lambda = \{0.25,0.50,0.75,1.00\}$

feature vectors allow for the prediction of true device labels with a high degree of certainty. This suggests that the application of triplet loss in the ClusterAE architecture provides a high degree of class separability in RF-fingerprint feature vectors compared to the use of solely autoencoder reconstruction loss as seen in Figure 13. Additionally, those models trained using only triplet loss training (i.e. $\lambda = 1.00$) were able to achieve comparable performance to those models using a combination of the two loss metrics at all values of z_{dim} evaluated. This suggests that the use of triplet loss on the output feature vectors of the model was the more important of the two components of the objective function for enabling feature vectors to perform well in clustering applications.

When observing the clustering results of the various models in response to the λ parameter, the response varies with the size of the latent dimension of the model z_{dim}. At the lowest latent dimension size $z_{dim} = 32$, the model performance continues to decrease as the λ parameter decreases therefore placing more emphasis on autoencoder reconstruction loss compared to triplet loss. At the intermediate latent dimension size $z_{dim} = 64$, the models with more emphasis placed on triplet loss again outperform those placing more emphasis on autoencoder reconstruction loss at the highest simulated SNR values. This relationship reverses at the lower simulated SNRs in which the models with lower λ values are the highest performers. This suggests that the addition of autoencoder loss provides benefit to a model's ability to learn feature vectors that are robust to noisy environments at this value of $z_{dim} = 64$. Lastly at the largest latent dimension evaluated $z_{dim} = 128$, the model with $\lambda = 0.50$ placing equal emphasis on the two loss components performs the best at the highest simulated SNRs and is overtaken in performance by the triplet loss only model at lower SNRs. The $\lambda = 0.50$ model still outperforms the other two models incorporating a combination of the two loss components across all simulated SNRs evaluated. Again the results for this value of $z_{dim} = 128$ show that there is some performance benefit added by the combination of the two loss components.

In summary the results of this section show that the incorporation of the triplet loss in the objective function used to train the ClusterAE model architecture is able to produce feature vectors from IEEE 802.11a/g waveforms that perform well in unsupervised clustering applications. Additionally, the results showed that, while the use of only triplet loss to train these models provided effective clustering results, the combination of triplet loss and autoencoder reconstruction loss provides performance benefit for models with latent dimensions $z_{dim} \geq 64$. The performance characteristics in relation to these model hyper-parameters remains consistent over the other clustering algorithms evaluated in this research and the performance curves for all models on the various clustering algorithms can be seen in Appendix A. In order to fully characterize the performance benefit of this combination of objective functions, a higher fidelity sweep across these two model hyper-parameters is necessary.

4.3 Comparison of Clustering Algorithms

This section presents a comparison of the results achieved by various clustering algorithms on the feature vectors produced by the ClusterAE models described in Section 3.6. For the sake of visual clarity, Figure 15 shows the results of the various clustering algorithms on the best performing model from Section 4.2 with hyper-parameter values $z_{dim} = 32$ and $\lambda = 1.00$. The discussion of the results in this section will however addresses the performance of models using all combinations of hyper-parameters to observe general trends associated with the various clustering algorithms. The individual performance of all models on these clustering algorithms can be seen in Appendix A. The results in this section evaluates clustering performance on the feature vectors produced by the various models from the Lab Test Set over a range of simulated SNRs.

The results shown in Figure 15 show that overall the highest performing clustering algorithm of those evaluated was the K-means algorithm. This was expected to be the case as this is the only clustering algorithm of those evaluated in which the number of clusters to be produced is specified prior to performing clustering. As this parameter is set to be equal to the number of devices present in the test set $N_D = 9$, this clustering algorithm is provided a significant degree of knowledge of the dataset not provided to the other clustering algorithms. Overall, this clustering algorithm performed very well on the Lab Test Set with an average $V_M = 0.978$ across all models at the highest simulated SNR = 18db. Additionally, this clustering algorithm displayed clustering performance that was asymptotic with relation to simulated SNR with V_M levelling out at SNR ≈ 12db for all models. These results indicate that the cluster assignments

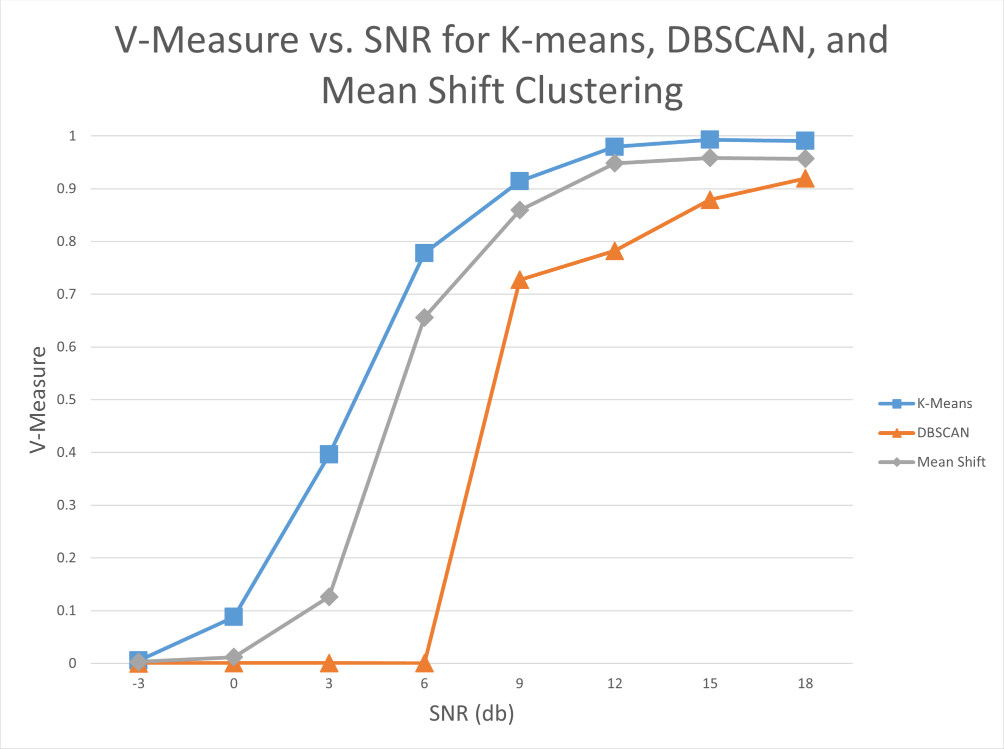


Figure 15: Comparison of V-measure score of K-means, DBSCAN, and Mean Shift clustering on ClusterAE Feature Vectors w/ hyper-parameters $z_{dim} = 32$ $\lambda = 1.00$

produced at the highest simulated SNRs are nearly exactly correlated with the true device labels. As the simulated SNR decreases, the set of true device labels within any given cluster approaches a roughly uniform distribution. This indicates that the cluster assignments provide no statistical advantage in predicting true device label and therefore the V_M approaches $V_M \approx 0$ at the lowest simulated SNR for all models.

The results of the DBSCAN clustering algorithm show that this was overall the worst performing clustering methodology for the feature vectors produced by the ClusterAE models. The average V_M achieved across all models evaluated at the highest simulated SNR = 18db was $V_M = 0.822$. This is an overall drop in average V_M of 0.156 at SNR = 18db between from the K-means clustering algorithm. A drop in performance was to be expected from this algorithm as there is no information given to the algorithm about the number of clusters present within the dataset. Additionally this algorithm exhibited a behavior in which the V_M drops off quickly to $V_M \approx 0$ at SNR = 6-9db for all models evaluated. As the simulated SNR decreases, the clusters produced by the DBSCAN algorithm begin to combine together such that each cluster contains nearly all observations from a set of multiple devices. When the simulated SNR becomes low enough, all clusters combine into a single cluster containing nearly all observations in the dataset with a small set of observations being labeled as outliers. In this scenario, the cluster labels provide no statistical advantage at predicting true device label and thus the V_M approaches 0. This behavior is likely largely due to the way in which the ϵ parameter is chosen for each clustering result. As the simulated SNR becomes lower, the ϵ parameter increases exponentially as show in Figure 16 which gives the algorithm a tendency to group all observations into a single cluster.

Finally the results of the Mean Shift algorithm showed that it was the better performing clustering method of the methods evaluated that did not have a specified

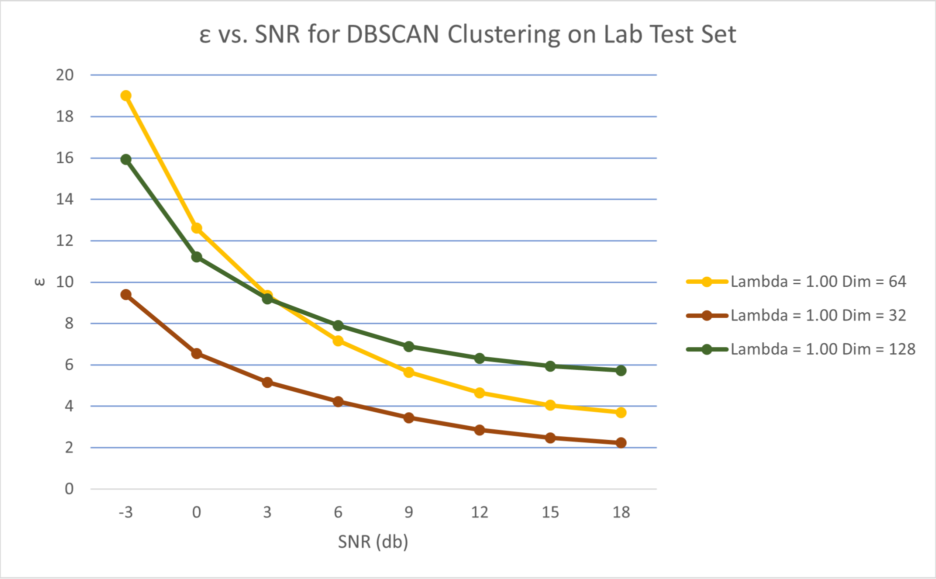


Figure 16: Calculated DBSCAN ϵ Parameter as a Function of Simulated SNR

number of clusters. The average model performance was found to be $V_M = 0.901$ across all models at simulated SNR = 18db. This results shows that the Mean Shift clustering algorithm was able to produce clusters highly representative of device of origin on these feature vectors without knowledge of the number of devices within the test set. Additionally this clustering methodology displayed a more gradual performance response to simulated SNR compared to the DBSCAN algorithm. As the simulated SNR was decreased, much like the DBSCAN algorithm, the clusters formed by the Mean Shift algorithm began to coalesce into a single cluster containing the majority of observations in the dataset. this process however occurred more gradually in the Mean Shift algorithm compared to the DBSCAN algorithm. Additionally, as the simulated SNR was decreased, many small clusters with $\approx$ 10-50 observations per cluster began to from using the Mean Shift algorithm. The distribution of true device labels within these clusters was roughly uniform and therefore had a limited impact on V_M calculated for this algorithm. Lastly, the phenomenon observed with the calculated ϵ parameter shown in Figure 16 likely had a similar effect on the clustering performance observed with Mean Shift clustering as the bandwidth parameter used for this method used the same calculated values.

4.4 Model Generalization to New Devices

This section presents the results of the trained ClusterAE models clustering performance on the Lab Test Set, Unseen Test Set, and Wild Test Sets described in Section 3.3. The clustering results of the feature vectors produced by these models on the various test sets is intended to evaluate how well they tend to remain class separable for devices previously unseen to the models. The Lab Test Set contains observations from the same set of devices used to train the ClusterAE models (not the same observations in the Training Set but new observations from the same set of devices). The Unseen Test Set contains observations from nine of the remaining devices in the **lab** dataset described in Section 3.1. The results on this dataset represent how robust the features learned by the ClusterAE are to new devices of the same model recorded under identical environmental conditions. The Wild Test Set contains observations from nine devices selected from the **wild** dataset described in Section 3.1. The results on this dataset represent how robust the features learned by the ClusterAE models are to new models of devices and varied propagation environments. The results presented in this section for the Lab Test Set and Unseen Test Set were performed at a simulated SNR = 18db. The results of the Wild Test Set were performed without any AWGN applied. The results across all simulated SNRs for the Unseen Test Set can be seen in Appendix A. The results on each of the test sets was performed on a new instance of the clustering algorithm without knowledge of the cluster assignments produced for the other test sets.

The results show in Figure 17, Figure 18, and Figure 19 show an overall decrease in clustering performance across all models from the Lab Test Set to the Unseen Test

Set. The average performance across all models observed on the Unseen Test Set is 0.789, 0.720, and 0.737 for the K-means, DBSCAN, and Mean Shift algorithms respectively. This equates to a decrease in average performance across all models at the simulated SNR = 18db in V_M of 0.189, 0.102, and 0.164. The overall performance on the Unseen Test Set is less variable than that observed on the Lab Test Set and thus the largest decrease in performance equates to the clustering methodology with the best performance on the Lab Test Set. Despite the overall decrease in performance observed on the results on the Unseen Test Set, the V_M achieved on this test set at a simulated SNR = 18db ranges [0.635, 0.925] across all models and clustering algorithms. These results indicate that there is still a strong correlation between the cluster assignments produced and the true device labels and therefore the ClusterAE models do tend to generalize well to new devices of the same model transmitting under similar conditions.

The results shown in Figure 17, Figure 18, and Figure 19 show a large decrease in performance of all models on the Wild Test Set compared to the other two test sets evaluated. The K-means clustering algorithm achieved the best performance on the Wild Test Set of all clustering algorithms evaluated as is to be expected as this algorithm was shown to perform the best overall in Section 4.3. The Wild Test Set results on the K-means clustering algorithm showed an average $V_M = 0.237$ with a range of [0.096,0.412] across all models. While these results are generally poor when compared to the other test sets evaluated, these results still show a significant degree of statistical correlation between cluster assignments especially when taking into account that the significant differences of this dataset. The Wild Test Set results on the DBSCAN algorithm show overall very poor performance with an average V_M across all models of 0.024. These results show that the feature vectors produced by the ClusterAE models on the Wild Test Set are entirely ineffective for DBSCAN clustering. Finally the Wild Test Set results using the Mean Shift Clustering algorithm again show poor performance with an average $V_M = 0.092$ across all models. The exception to this overall poor performance was the model with hyper-parameter values $z_{dim} = 32$ $\lambda = 1.00$ with a $V_M = 0.316$ comparable to the better performing models' performance on the K-means clustering algorithm. This indicates that the Mean Shift algorithm can achieve comparable performance to the K-means algorithm on the Wild Test Set given a proper choice of model hyper-parameters.

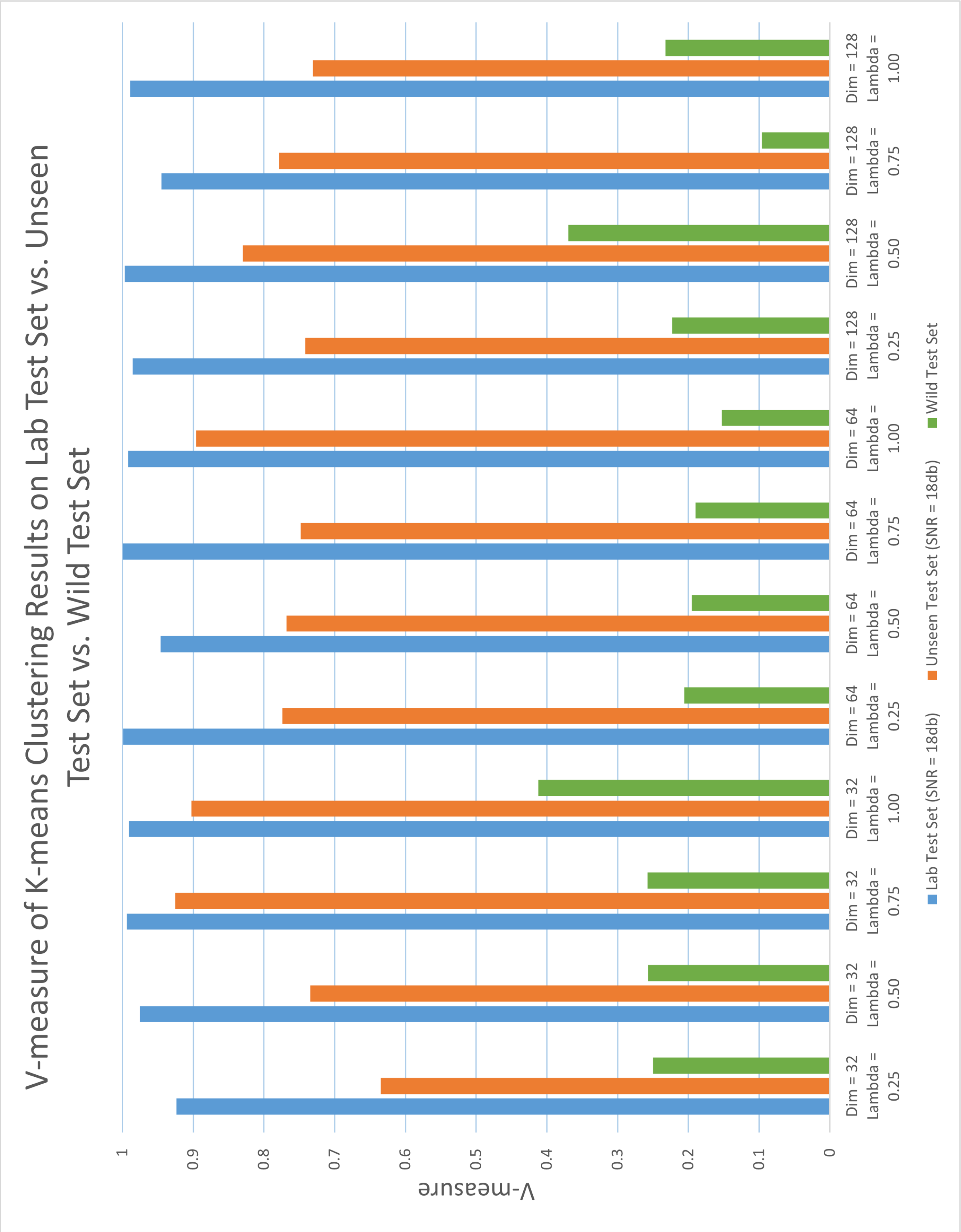


Figure 17: Comparison of K-means Clustering Results on Lab Test Set vs. Unseen Test Set vs. Wild Test Set

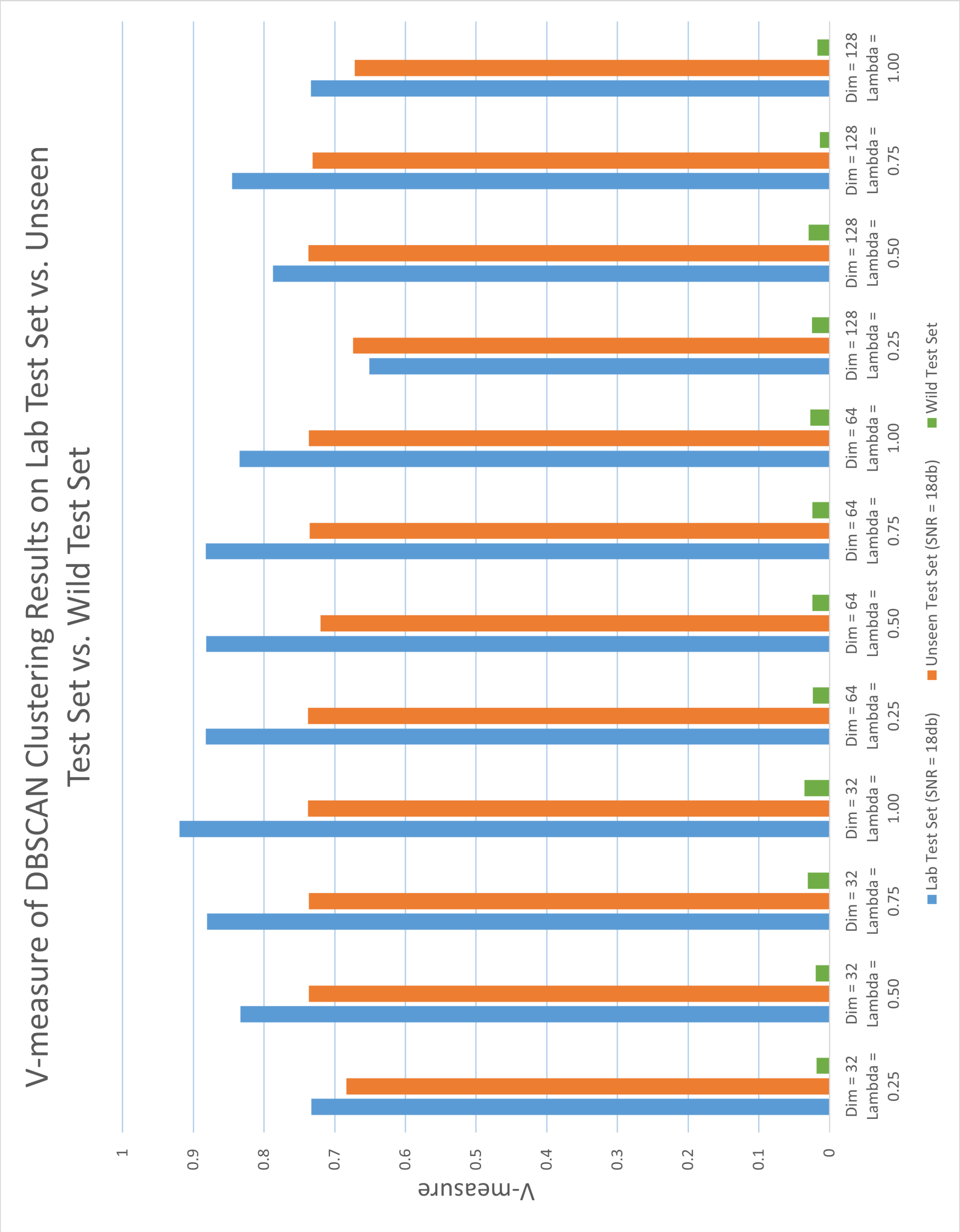


Figure 18: Comparison of DBSCAN Clustering Results on Lab Test Set vs. Unseen Test Set vs. Wild Test Set

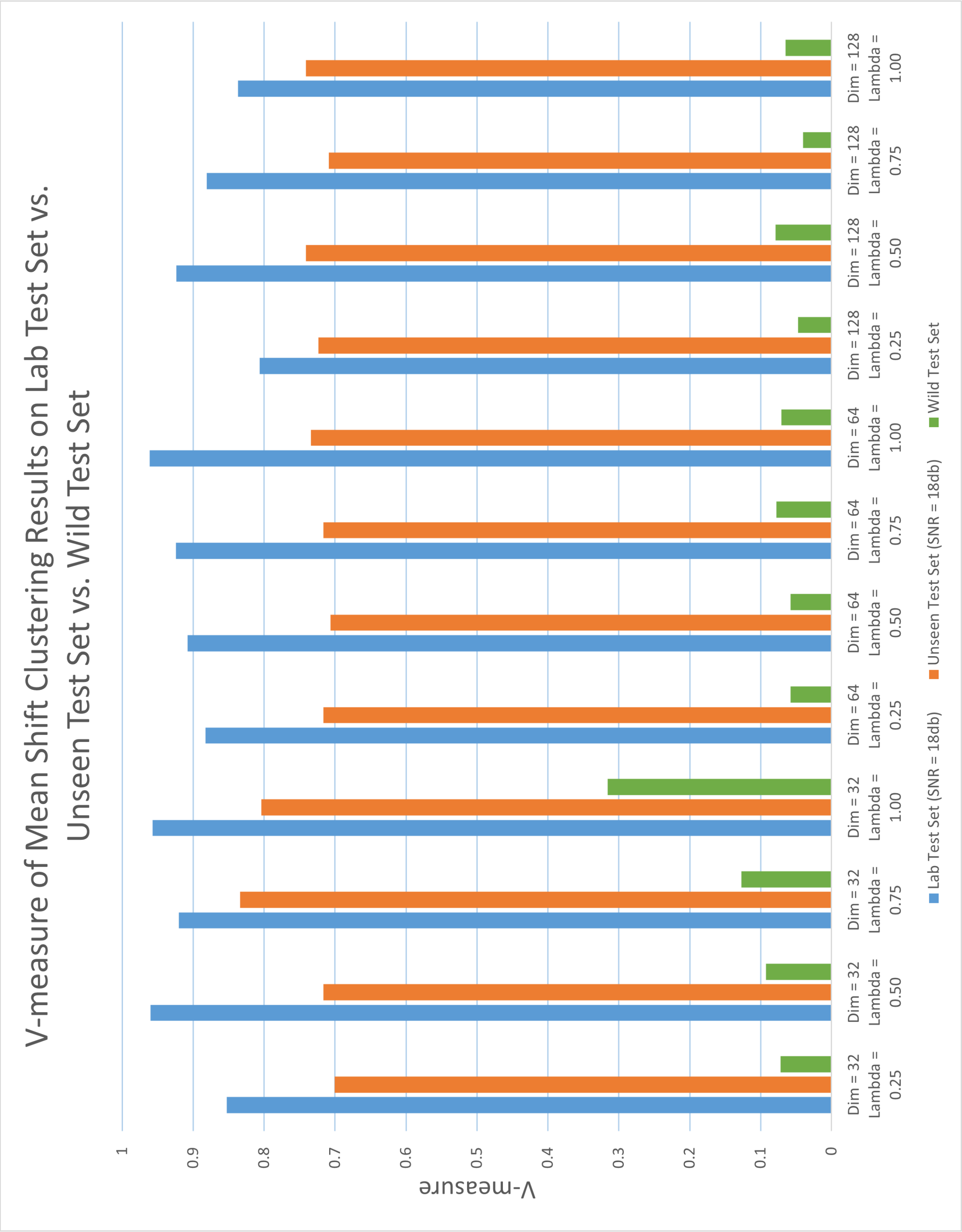


Figure 19: Comparison of DBSCAN Clustering Results on Lab Test Set vs. Unseen Test Set vs. Wild Test Set

4.5 Summary

Overall the results presented in this section demonstrate that the ClusterAE RF-Fingerprinting model proposed is an effective means of extracting feature vectors from IEEE 802.11a/g for use in unsupervised clustering algorithms. In particular this model architecture is shown to succeed in producing effective feature vectors for clustering where conventional dimensionality reduction methods such as PCA and pure convolutional autoencoder models fail to do so as shown in Section 4.1. It is also shown that using a loss function which combines autoencoder reconstruction loss and triplet loss provides a performance benefit over solely the use of triplet loss to train the ClusterAE architecture when the size of the latent dimension is ≥ 64. Additionally, the results show that the K-means clustering algorithm is more effective than the Mean Shift clustering algorithm for the feature vectors produced by these models which is in turn more effective than the DBSCAN algorithm. Finally it showed that the features learned by the ClusterAE model architecture tend to generalize well to new devices of the same model as those used to train the model while struggling to achieve good clustering performance on observations from different device models and different environmental conditions.

V. Conclusions

5.1 Research Summary

The research presented within this thesis examined the machine learning problem of RF-Fingerprinting/SEI through the lens of an unsupervised clustering problem. The application of modern machine learning techniques to identify the originating device for wireless communications remains a promising avenue to bolstering security in the PHY layer. Clustering research provides a useful counterpoint to the more prevalent research in supervised classification problems by showing how learned fingerprint features generalize to new devices. Much of the RF-Fingerprinting literature is focused on developing a classifier that can effectively determine device of origin from a pre-defined set of devices used in the training process. Such classifiers have limited usefulness in a communication scenario with a dynamic set of client devices. This research effort was intended to address this problem by presenting a method to produce fingerprint features that effectively group together observations from devices previously unseen in the machine learning process.

The methodology used to generate these fingerprint feature sets was based on a convolutional autoencoder deep learning model. The model was trained using a novel objective function utilizing a combination of autoencoder reconstruction loss as well as triplet loss to learn characteristic features from RF waveforms that tend to separate well based on device. The models were trained using a range of relative weightings between the two components of the objective function to evaluate the effect of its two loss components. All models were trained on IEEE 802.11a/g preamble signals from 9 different emitters collected in a controlled laboratory environment grouped into randomly generated triplet pairs. The models were then all evaluated by the clustering performance of its generated features on various clustering methods over a range of

SNR values. Three different test sets were used to evaluate model performance:

- Lab Test Set: Test set of reserved transmissions from devices used in the training process
- Unseen Test Set: Test set of laboratory collected transmissions from a set of devices unseen to trained models
- Wild Test Set: Test set of transmissions collected in public areas in various physical environments.

Three different clustering methodologies were used to evaluate performance of each models' learned features: K-means, DBSCAN, and Mean Shift Clustering. Accuracy of each clustering algorithm was evaluated using the homogeneity, completeness, and V_M metrics using the produced cluster assignments and the known true device labels.

5.2 Research Findings

The findings of this research, as presented in Chapter IV, can broadly be grouped into three primary categories: effect of model hyper-parameters on model performance, effectiveness of different clustering algorithms on generated feature sets, and ability for models to generalize to unseen devices.

5.2.1 Effect of Model Hyper-parameters

The results of this research show a significant interaction between the latent dimension z_{dim} and objective function weighting λ model hyper-parameters. The results show that placing additional emphasis on autoencoder reconstruction loss tends to provide a performance benefit at higher latent dimensions. The performance benefit of autoencoder reconstruction loss does not appear to be present at the lower dimension $z_{dim} = 32$ in which the models which place the least emphasis on reconstruction loss $\lambda = 0.75, 1.00$ are the top performers. This relationship between z_{dim} and λ is seen across all test datasets and clustering algorithms evaluated. These results suggest that the features learned via autoencoder reconstruction loss provide meaningful benefit to a models ability to cluster. Additionally the results suggests a threshold at which latent dimension becomes to low to produce an effective autoencoder reconstruction and thus this loss components begins to act mostly as noise in the objective function.

5.2.2 Comparison of Clustering Algorithms

The results presented here show that K-means was the highest performing clustering methodology of those evaluated. This is an expected result as it is the only clustering algorithm of those evaluated that is given the prior knowledge of a known number of clusters within the dataset. The K-means clustering algorithm was able to achieve an average V_M across all models > 0.9 for SNR ≥ 12 on the Lab Test Set. The worst performing clustering algorithm for this research was found to be DBSCAN with a maximum average V_M of 0.822 with a steep drop off in average V_M of < 0.01 at SNR = 6db as all observations collapse into a single cluster. The Mean Shift algorithm was found to be the better performing of the clustering algorithms without a provided number of clusters. The Mean Shift algorithm achieved an average V_M of > 0.9 at SNR = 18db with a more gradual decline in performance in comparison to DBSCAN clustering. Neither of the clustering methodologies without a provided number of clusters was able to accurately identify the true number of clusters present within the dataset however the Mean Shift clustering algorithm performed better in this regard.

5.2.3 Performance on Unseen Devices

Clustering performance for all models was found to be best on the Lab Test Set which is to be expected as this is the test set containing the devices used to train the various models. The average V_M score on the Lab Test Set at SNR = 18db for all models evaluated was found to be 0.978, 0.822, and 0.901 for the K-means, DBSCAN, and Mean Shift clustering algorithms respectively. While the V_M score metric is not as intuitively interpretable as the percent classification accuracy metric used for supervised classification, a V_M of greater than 0.9 can reasonably be considered a very effective clustering. These average V_M scores drop to 0.789, 0.720, and 0.737 on the Unseen Test set. This equates to a drop in V_M of 0.1-0.2 for the different clustering methodologies between seen and unseen devices. The V_M on the Unseen Test set remains above 0.7 for all clustering methodologies on completely unseen devices indicating the cluster labels still provide significant statistical advantage in predicting true device labels. The results on the Wild Test Set across all clustering methodologies and models was found to be generally lacking with an average V_M across all models of 0.237, 0.024, and 0.092 for K-means, DBSCAN, and Mean Shift clustering respectively. The results show that the features learned from the laboratory collected data were not sufficient to effectively cluster transmissions collected in an uncontrolled environment. Despite the generally poor performance on this dataset, several models were able to achieve V_M scores of > 0.3 indicating some degree of statistical advantage in clustering assignments for certain models. This methodology may then be useful as a pre-training step to be used in a transfer learning process.

5.3 Future Research

The results shown here indicate that there is merit to the application of the novel combined objective function presented to RF-Fingerprinting applications. Additionally there is further work to be done on the unsupervised clustering problem for RF-Fingerprinting/SEI. Some avenues of future research to expand on the work presented here include the following:

- Evaluating the effects of more in depth pre-processing steps on this methodology: The pre-processing steps used for this research do not perform fine frequency and phase adjustment as would typically be the case in signal demodulation. The inclusion of these signal processing steps may have significant impact on clustering performance particularly on the wild collected data

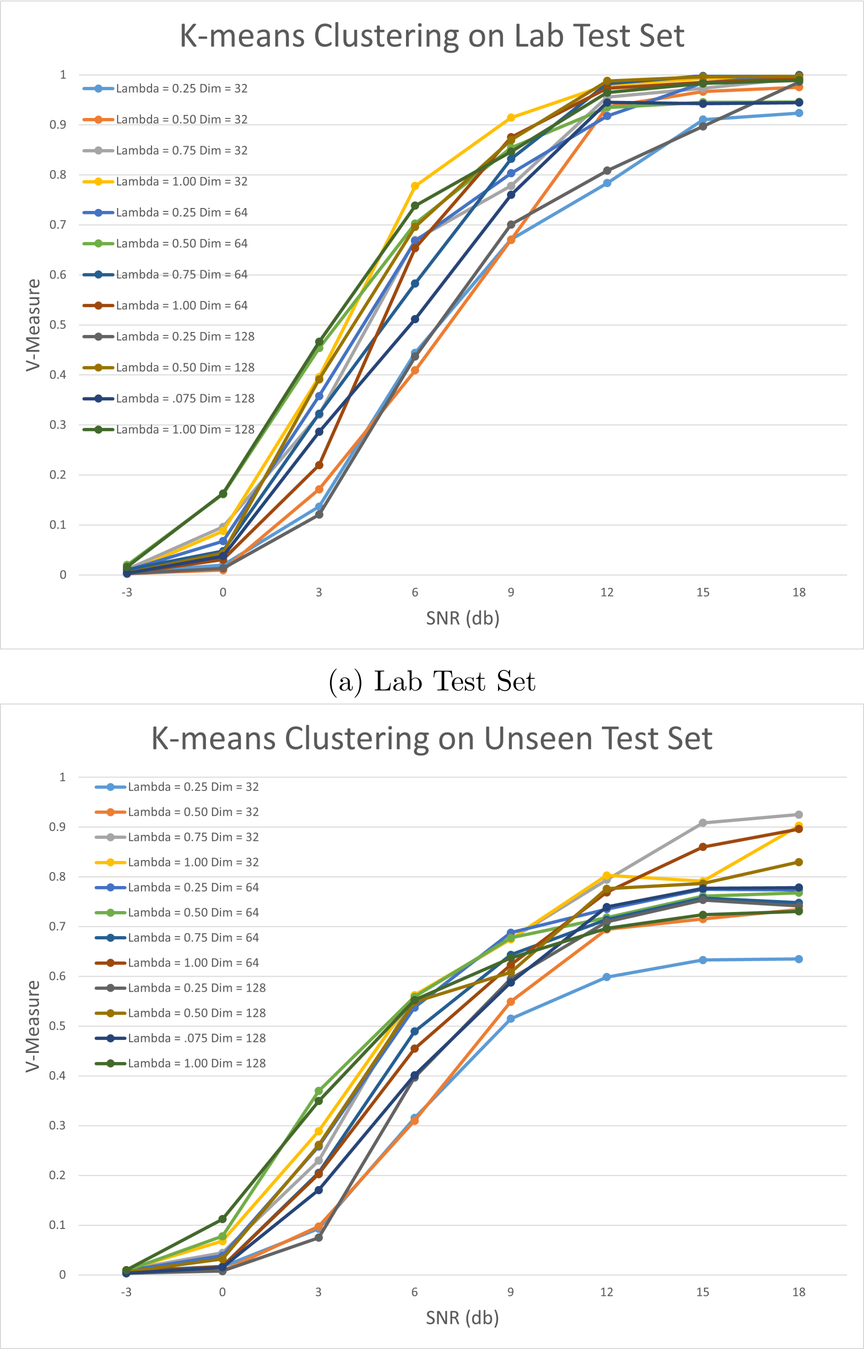
- More exhaustive analysis of the effects of model hyper-parameters: Due to timing considerations in the training of neural networks, only a small selection of values for the λ and z_{dim} parameters were evaluated. A higher fidelity sweep over these parameters may provide a clearer picture of how the two parameters interact.

- Dimensionality Reduction Analysis of produced fingerprint features: The number of features was controlled in these experiments by the latent dimension of the models before training, however a dimensionality reduction step may be able to retain the features learned by a larger latent dimension without having dimension effect clustering performance across models.

- Evaluating other clustering algorithms: A clustering algorithm that was not presented in this thesis may be able to achieve better performance on the feature sets produced by these models. Additionally, there may be a more effective method for selecting hyper-parameters for the DBSCAN and Mean Shift clustering algorithms than those used here.

- Diversity of Training Set: The dataset used to train models for these experiments consisted of WiFi emitter devices of identical make and model collected under ideal laboratory conditions. This likely has a significant impact on the types of differences the learned features encapsulated between devices. A training dataset that contained a range of device models may result in improved ability to generalize to new devices. Additionally, incorporation of several channel models instead of solely AWGN for data augmentation may result in better clustering performance on unseen datasets.

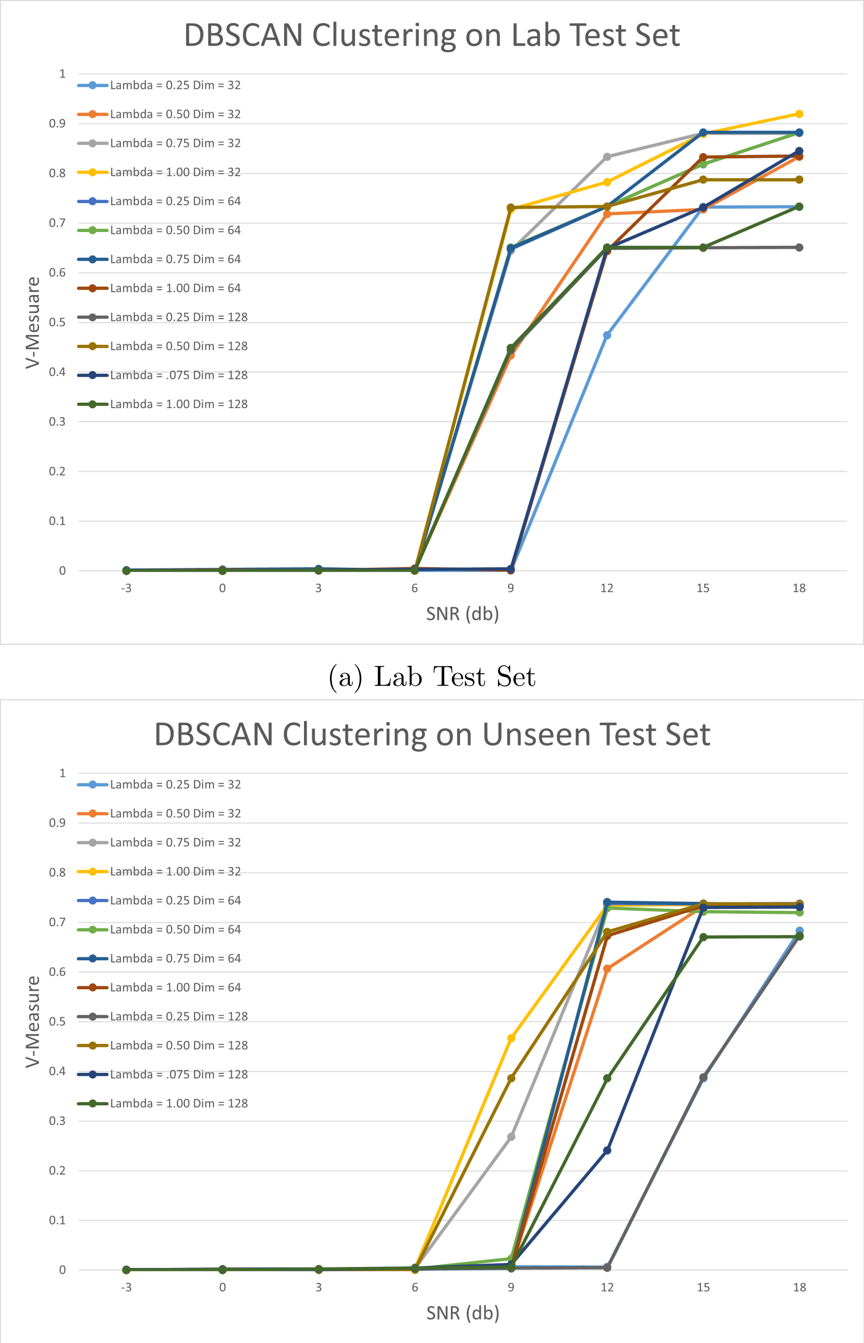
Appendix A. K-Means, DBSCAN, and Mean Shift Clustering Results on Lab Test Set and Unseen Test Set



(a) Lab Test Set

(b) Unseen Test Set

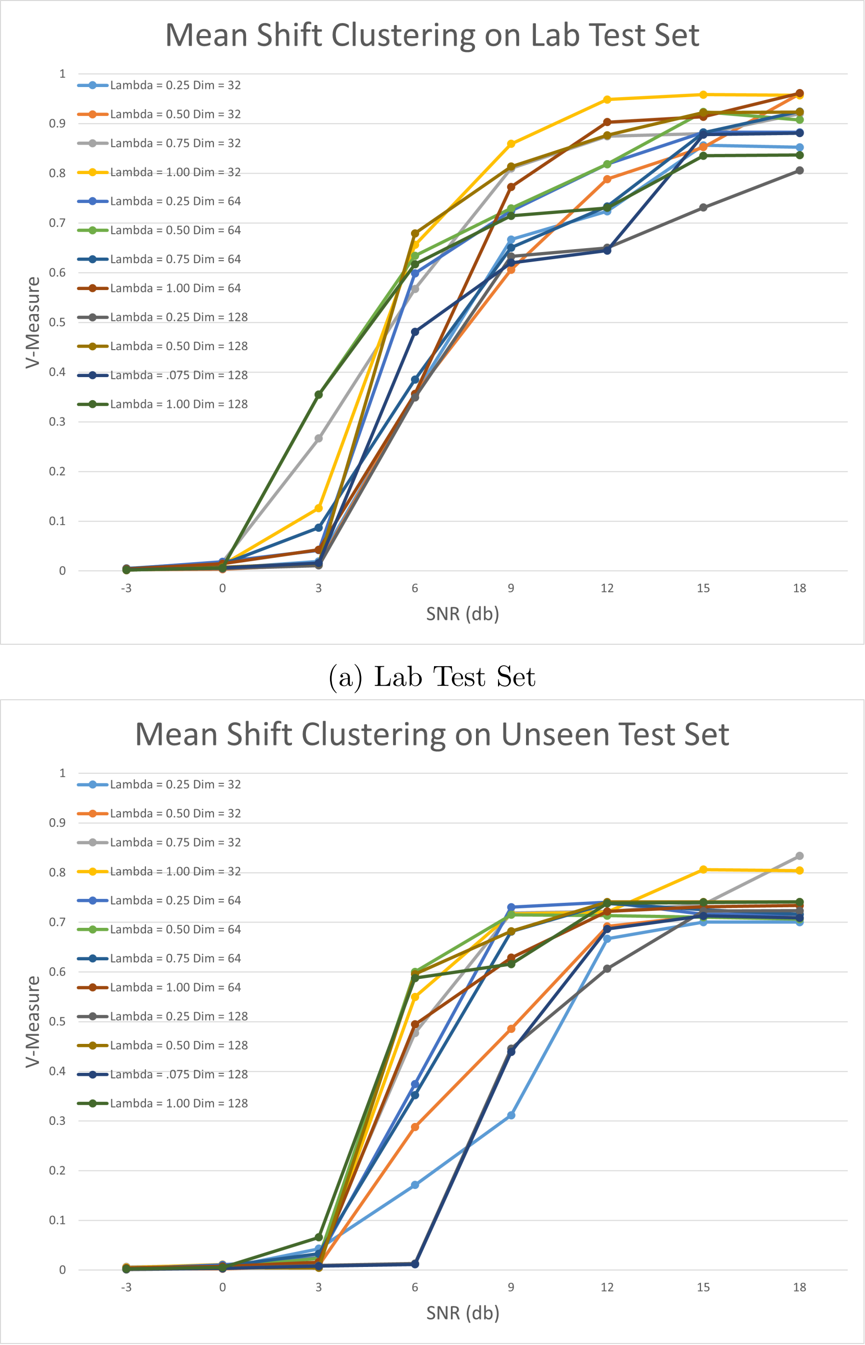
Figure 20: V_M vs. SNR for K-means Clustering



(a) Lab Test Set

(b) Unseen Test Set

Figure 21: V_M vs. SNR for DBSCAN Clustering



(a) Lab Test Set

(b) Unseen Test Set

Figure 22: V_M vs. SNR for Mean Shift Clustering

Appendix B. SNR Estimates for all Test Sets

Table 3: SNR Measurements by Device for Training Set

Device Number	Min SNR(db)	Max SNR(db)	Avg SNR(db)
0	48.38	56.24	51.94
1	44.54	56.33	49.87
2	48.66	57.22	52.24
3	48.42	57.37	52.16
4	48.42	57.80	52.00
5	48.26	56.94	52.03
6	48.64	57.43	52.02
7	48.52	58.29	52.15
8	48.75	56.66	52.11
Total	44.54	58.29	51.84

Table 4: SNR Estimates by Device for Lab Test Set

Device Number	Min SNR(db)	Max SNR(db)	Avg SNR(db)
0	48.79	56.21	51.63
1	49.06	56.38	51.94
2	49.19	56.66	52.45
3	48.71	56.89	52.15
4	48.52	56.68	51.99
5	48.11	56.93	52.03
6	49.17	57.94	52.83
7	49.40	56.48	52.09
8	49.11	55.95	51.73
Total	48.11	57.94	52.09

Table 5: SNR Estimates by Device for Unseen Test Set

Device Number	Min SNR(db)	Max SNR(db)	Avg SNR(db)
9	49.25	56.58	51.85
10	44.99	54.05	50.45
11	48.84	57.10	52.13
12	49.22	56.06	52.07
13	48.98	56.19	51.79
14	48.92	57.38	51.91
15	48.77	56.01	51.86
16	48.93	57.16	52.04
17	49.38	55.83	52.10
Total	44.99	57.38	51.80

Table 6: SNR Measurements by Device for Wild Test Set

Device Number	Min SNR(db)	Max SNR(db)	Avg SNR(db)
18	24.58	43.06	33.19
19	29.60	35.04	31.18
20	26.30	38.75	31.97
21	25.11	32.06	27.02
22	24.32	46.97	30.46
23	26.91	33.01	28.39
24	31.07	38.39	33.00
25	20.03	30.59	22.88
26	28.52	33.64	29.73
Total	20.03	46.97	29.76

Bibliography

1. IEEE Standard for Telecommunications and Information Exchange Between Systems - LAN/MAN Specific Requirements - Part 11: Wireless Medium Access Control (MAC) and physical layer (PHY) specifications: High Speed Physical Layer in the 5 GHz band. *IEEE Std 802.11a-1999*, pages 1–102, 1999.

2. Cisco. Cisco Annual Internet Report (2018–2023). *Cisco*, pages 1–41, 2020. `https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html`.

3. Vasileios Mavroeidis, Kamer Vishi, Mateusz D. Zych, and Audun Jøsang. The Impact of Quantum Computing on Present Cryptography. *International Journal of Advanced Computer Science and Applications*, 9(3):405–414, 2018.

4. Enrico Mattei, Cass Dalton, Andrew Draganov, Brent Marin, Michael Tinston, Greg Harrison, Bob Smarrelli, and Marc Harlacher. Feature learning for Enhanced Security in the Internet of Things. *GlobalSIP 2019 - 7th IEEE Global Conference on Signal and Information Processing, Proceedings*, 2019.

5. Nathalie Domingo, Bryan Pearson, and Yier Jin. Exploitations of Wireless Interfaces via Network Scanning. *2017 International Conference on Computing, Networking and Communications, ICNC 2017*, pages 937–941, 2017.

6. Kurt Hornik. Approximation Capabilities of Multilayer Neural Network. *Neural Networks*, 4(1991):251–257, 1991.

7. Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

8. Alex Krizhevsky. Learning Multiple Layers of Features from Tiny Images. *University of Toronto*, 05 2012.

9. Sergey Ioffe and Christian Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *32nd International Conference on Machine Learning, ICML 2015*, 1:448–456, 2015.

10. Reyhan Kevser Keser and Behcet Ugur Toreyin. Autoencoder Based Dimensionality Reduction of Feature Vectors for Object Recognition. *Proceedings - 15th International Conference on Signal Image Technology and Internet Based Systems, SISITS 2019*, pages 577–584, 2019.

11. Quentin Fournier and Daniel Aloise. Empirical Comparison between Autoencoders and Traditional Dimensionality Reduction Methods. *Proceedings - IEEE 2nd International Conference on Artificial Intelligence and Knowledge Engineering, AIKE 2019*, pages 211–214, 2019.

12. Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre Antoine Manzagol. Stacked Denoising Autoencoders: Learning Useful Representations in a Deep Network with a Local Denoising Criterion. *Journal of Machine Learning Research*, 11:3371–3408, 2010.

13. Florian Schroff, Dmitry Kalenichenko, and James Philbin. FaceNet: A Unified Embedding for Face Recognition and Clustering. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 07-12-June:815–823, 2015.

14. T. Kansal, S. Bahuguna, V. Singh, and T. Choudhury. Customer Segmentation using K-means Clustering. In *2018 International Conference on Computational Techniques, Electronics and Mechanical Systems (CTEMS)*, pages 135–139, 2018.

15. J. Mai, Y. Fan, and Y. Shen. A Neural Networks-Based Clustering Collaborative Filtering Algorithm in E-Commerce Recommendation System. In *2009 International Conference on Web Information Systems and Mining*, pages 616–619, 2009.

16. Arpita Nagpal, Arnan Jatain, and Deepti Gaur. Review Based on Data Clustering Algorithms. *2013 IEEE Conference on Information and Communication Technologies, ICT 2013*, (April):298–303, 2013.

17. Chiheb Trabelsi, Olexa Bilaniuk, Ying Zhang, Dmitriy Serdyuk, Sandeep Subramanian, João Felipe Santos, Soroush Mehri, Negar Rostamzadeh, Yoshua Bengio, and Christopher J. Pal. Deep Complex Networks. *6th International Conference on Learning Representations, ICLR 2018 - Conference Track Proceedings*, (2016):1–19, 2018.

18. Anant Ram, Jalal Sunita, Anand Jalal, and Kumar Manoj. A Density Based Algorithm for Discovering Density Varied Clusters in Large Spatial Databases. *International Journal of Computer Applications*, 3, 06 2010.

19. Dorin Comaniciu and Peter Meer. Mean Shift: A Robust Approach Toward Feature Space Analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(5):603–619, 2002.

20. Wang Xu, Wu, Daneshmand, Liu. Theoretical Performance Analysis of Radio Frequency Fingerprinting Under Receiver Distortions. *Wireless Communications and Mobile Computing*, (February 2015):421–430, 2015.

21. Oktay Ureten and Nur Serinken. Wireless Security Through RF Fingerprinting. *Canadian Journal of Electrical and Computer Engineering*, 32(1):1–8, 2007.

22. William E. Cobb, Eric W. Garcia, Michael A. Temple, Rusty O. Baldwin, and Yong C. Kim. Physical Layer Identification of Embedded Devices Using RF-DNA Fingerprinting. *Proceedings - IEEE Military Communications Conference MILCOM*, pages 2168–2173, 2010.

23. N. Soltanieh, Y. Norouzi, Y. Yang, and N. C. Karmakar. A Review of Radio Frequency Fingerprinting Techniques. *IEEE Journal of Radio Frequency Identification*, 4(3):222–233, 2020.

24. Tong Jian, Bruno Costa Rendon, Emmanual Ojuba, Nasim Soltani, Zifeng Wang, and Kunal Sankhe. Deep Learning for RF Fingerprinting: a Massive Experimental Study. *IEEE Internet of Things Magazine*, (March):50–57, 2020.

25. Mathew W Lukacs, Angela J. Zeqolari, Peter J. Collins, and Michael A. Temple. "RF-DNA" Fingerprinting for Antenna Classification. *IEEE Antennas and Wireless Propagation Letters*, 14:1455–1458, 2015.

26. Christopher Talbot, Michael Temple, Timothy Carbino, and J. Betances. Detecting rogue attacks on commercial wireless Insteon home automation systems. *Computers Security*, 74, 10 2017.

27. J. Lopez, N. C. Liefer, C. R. Busho, and M. A. Temple. Enhancing critical infrastructure and key resources (cikr) level-0 physical process security using field device distinct native attribute features. *IEEE Transactions on Information Forensics and Security*, 13(5):1215–1229, 2018.

28. D. R. Reising, M. A. Temple, and J. A. Jackson. Authorized and Rogue Device Discrimination Using Dimensionally Reduced RF-DNA Fingerprints. *IEEE Transactions on Information Forensics and Security*, 10(6):1180–1192, 2015.

29. Timothy J. O'Shea, Johnathan Corgan, and T. Charles Clancy. Convolutional Radio Modulation Recognition Networks. *Communications in Computer and Information Science*, 629:213–226, 2016.

30. Kevin Merchant and Bryan Nousain. Enhanced RF Fingerprinting for IoT Devices with Recurrent Neural Networks. *Proceedings - IEEE Military Communications Conference MILCOM*, 2019-Novem:590–597, 2019.

31. Jiabao Yu, Aiqun Hu, Fen Zhou, Yuexiu Xing, Yi Yu, Guyue Li, and Linning Peng. Radio Frequency Fingerprint Identification Based on Denoising Autoencoders. *International Conference on Wireless and Mobile Computing, Networking and Communications*, 2019-Octob, 2019.

32. Yicheng Zhang, Jipeng Gao, and Haolin Zhou. Breeds Classification with Deep Convolutional Neural Network. *ACM International Conference Proceeding Series*, pages 145–151, 2020.

33. Yann Lecun, Le'on Bottou, Yoshua Bengio, and Parick Haffner. Gradient-Based Learning Applied to Document Recognition. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 1998.

34. Diederik P. Kingma and Jimmy Lei Ba. Adam: A Method for Stochastic Optimization. *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, pages 1–15, 2015.

35. Erich Schubert, Jörg Sander, Martin Ester, Hans Peter Kriegel, and Xiaowei Xu. DBSCAN Revisited, Revisited. *ACM Transactions on Database Systems*, 42(3):1–21, 2017.

36. Ville Satopää, Jeannie Albrecht, David Irwin, and Barath Raghavan. Finding a "kneedle" in a Haystack: Detecting Knee Points in System Behavior. *Proceedings*

- *International Conference on Distributed Computing Systems*, pages 166–171, 2011.

37. Andrew Rosenberg and Julia Hirschberg. V-Measure : A Conditional Entropy-Based External Cluster Evaluation Measure. *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, (June):410–420, 2007.

Acronyms

AFIT Air Force Institute of Technology. 28

AWGN Additive White Gaussian Noise. 46

BPSK Binary Phase Shift Keying. 8

CNN Convolutional Neural Networks. 12

DBSCAN Density Based Spatial Clustering of applications with Noise. 23

GAN Generative Adversarial Networks. 37

HT High Throughput. 7

IEEE Institute of Electrical and Electronics Engineers. 7

LPF Low Pass Filter. 33

LSTM Long Short Term Memory. 29

LTF Long Training Field. 8

LVQ Learning Vector Quantization. 28

MAC Medium Access Control. 7

MDA Multiple Discriminant Analysis. 58

MDA/ML Multiple Discriminant Analysis / Maximum Likelihood. 28

MSE Mean Squared Error. 19

OFDM Orthogonal Frequency Division Multiplexing. 8

PCA Principal Component Analysis. 54, 58

PHY Physical Layer. iv, 2

PLCP Physical Layer Convergence Protocol. 8

PPDU PHY Protocol Data Unit. 7

QAM Quadrature Amplitude Modulation. 8

QPSK Quadrature Phase Shift Keying. 8

ReLU Rectified Linear Unit. 40

RF Radio Frequency. iv

RF-DNA Radio Frequency Distinct Native Attribute. 28

RndF Random Forrest. 28

RNN Recurrent Nerual Networks. 29

ROI Region of Interest. 32

RTSA Real Time Spectrum Analyser. 32

SEI Specific Emitter Identification. 2

SNR Signal to Noise Ratios. 5

STF Short Training Field. 8

V-measure. iv, 51, 57

WLAN Wireless Local Area Network. 1

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704–0188

The public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704–0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202–4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number. **PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.**

1. REPORT DATE *(DD–MM–YYYY)* 25–03–2021	**2. REPORT TYPE** Master's Thesis	**3. DATES COVERED** *(From — To)* Sept 2019 — Mar 2021

4. TITLE AND SUBTITLE

Unsupervised Clustering of RF-Fingerprinting Features Derived from Deep Learning Based Recognition Models

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Potts, Christian T, Capt

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Air Force Institute of Technology
Graduate School of Engineering and Management (AFIT/EN)
2950 Hobson Way
WPAFB OH 45433-7765

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT-ENG-MS-21-M-074

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Intentionally Left Blank

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

DISTRIBUTION STATEMENT A:
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

RF-Fingerprinting is focus of machine learning research which aims to characterize wireless communication devices based on their physical hardware characteristics. It is a promising avenue for improving wireless communication security in the PHY layer. The bulk of research presented to date in this field is focused on the development of features and classifiers using both traditional supervised machine learning models as well as deep learning. This research aims to expand on existing RF-Fingerprinting work by approaching the problem through the lens of an unsupervised clustering problem. To that end this research proposes a deep learning model and training methodology to extract features from IEEE 802.11a/g preamble waveforms to enhance performance with various clustering algorithms. The model architecture presented takes the form of convolutional autoencoder with an objective function that combines both autoencoder reconstruction loss as well as triplet loss to learn feature encodings. These features were then clustered using the K-means, DBSCAN, and Mean Shift clustering algorithms.

15. SUBJECT TERMS

RF-Fingerprinting, Machine Learning, Deep Learning, Neural Networks, Wireless Communications

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON** LtCol James W. Dean, ENG
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**			**19b. TELEPHONE NUMBER** *(include area code)*
U	U	U	UU	104	(937) 255-3636; James.Dean@afit.edu