12-2021

Analysis of Deep Learning Methods for Wired Ethernet Physical Layer Security of Operational Technology

Lucas Torlay
ltorlay@clemson.edu

11-2021

Analysis of Deep Learning Methods for Wired Ethernet Physical Layer Security of Operational Technology

Lucas Torlay

Analysis of Deep Learning Methods for Wired Ethernet Physical Layer Security of Operational Technology

---

A Thesis
Presented to
the Graduate School of
Clemson University

In Partial Fulfillment
of the Requirements for the Degree
Master of Science
Electrical Engineering

---

by
Lucas Steven Torlay
December 2021

Accepted by:
Dr. Johan Enslin, Committee Chair
Dr. Bill Suski
Dr. Harlan Russell

Abstract

The cybersecurity of power systems is jeopardized by the threat of spoofing and man-in-the-middle style attacks due to a lack of physical layer device authentication techniques for operational technology (OT) communication networks. OT networks cannot support the active probing cybersecurity methods that are popular in information technology (IT) networks. Furthermore, both active and passive scanning techniques are susceptible to medium access control (MAC) address spoofing when operating at Layer 2 of the Open Systems Interconnection (OSI) model. This thesis aims to analyze the role of deep learning in passively authenticating Ethernet devices by their communication signals. This method operates at the physical layer or Layer 1 of the OSI model. The security model collects signal data from Ethernet device transmissions, applies deep learning to gather distinguishing features from signal data, and uses these features to make an authentication decision on the Ethernet devices. The proposed approach is passive, automatic, and spoof-resistant. The role of deep learning is critical to the security model. This thesis will look at analyzing and improving deep learning at each step of the security model including data processing, model training, model efficiency, transfer learning on new devices, and device authentication.

Dedication

This thesis is dedicated to my parents, Ingrid and Steven Torlay, for their guidance through life, and a year and a half of grad school.

Acknowledgments

I would like to thank my advisors Dr. Johan Enslin, Dr. Harlan Russell, and Dr. Bill Suski for the many meetings and long chats that took place biweekly. They offered necessary criticism and encouraging insights, and were truly generous with their time. You would think they were once students themselves.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

The cybersecurity of power systems is jeopardized by the threat of spoofing and man-in-the-middle style attacks due to a lack of physical layer device authentication techniques for OT communication networks. While cybersecurity defense strategies stay ahead of new attack vectors in IT networks, devices on OT networks contain legacy devices that cannot integrate modern defense strategies. This leaves many OT networks vulnerable.

1.1 Operational Technology Vulnerabilities

Network security strategies usually involve a username/password or digital certificate to authenticate users. Passwords are commonly used by security systems, but they must be complex, regularly maintained, and stored on a deployed device. There have been many noteworthy cases of default passwords allowing intruders into their system [41], [46]. Digital certificates are an effective security strategy on IT networks. However, OT networks contain legacy or low power industrial control system (ICS) devices that are not suited for digital certificates. These devices are often incapable of receiving software updates and do not have the complexity necessary for encryption. The result is a vulnerable OT network.

1.2 Deep Learning in Cybersecurity

Deep learning has become a popular tool in cybersecurity in recent years [40]. Long Short Term Memory (LSTM) networks [21], popular for analyzing data over time, have been utilized for intrusion detection [23], classifying malicious apps [44], and phishing detection [8]. Deep convolutional neural networks (DCNNs), popular for analyzing data in images, have shown success in intrusion detection in internet of things (IoT) networks [43], malware detection [48], and phishing detection [47]. DCNNs deployed for intrusion detection will be the focus of this work.

1.3 MAC Addresses and Spoofing

MAC address is a discrete identifying address assigned to a computer or device. MAC addresses are used in Ethernet, WiFi, and Bluetooth. The data link layer or Layer 2 of the OSI model make use of MAC addresses through the network interface controller (NIC). MAC address spoofing is the process of masking a MAC address as an operator's chosen address which effectively masks the computer's identity from the network infrastructure. This is often done for malicious purposes. Other identifiers such as a device's operating system, software version, or packet delay can be spoofed as well. Spoofing is a major threat to intrusion detection in cybersecurity.

1.4 Statement of Work

This thesis proposes the use of deep learning for intrusion detection in OT networks leveraging physical layer communication data for spoof-resistance.

The rest of this thesis is organized as follows. Chapter 2 will give Background on methods presented. Chapter 3 will discuss the deep learning security model. Chapter 4 will discuss deploying the deep learning security model at the edge. Chapter 5 will discuss the authentication approaches for intrusion devices. Chapter 6 will discuss results from experiments in Chapters 3, 4, and 5. Finally, Chapter 7 will look at conclusions and future work.

Chapter 2

Background

This chapter provides background on current OT security methods and the application of deep learning to cyberphysical systems. Because the proposed approach uses deep learning at the physical layer for OT device security, it is important to establish a level of understanding in both of these areas.

2.1 Operational Technology Security Methods

The vulnerabilities of OT networks and the ICS devices that use them has been a growing concern in recent years. High-profile attacks have driven research into ICS security [13], [2], and garnered national interest [4]. This has led to product development for network access control (NAC) solutions to address device inventory, authentication, and other tasks.

NAC solutions gain information about devices on the network using two approaches: active or passive probing. Active scanning involves sending a request for information to each device on a network and awaiting a response with the desired information. While this approach is effective, it often causes malfunctions in ICS devices [13]. In order to avoid malfunctions many ICS device manufacturers have developed their own protocols to manage active probing [13]. However, this puts an additional burden on network administrators who now must adhere to the specifications of numerous ICS devices. On the other hand, passive probing monitors network traffic without sending out any request packets. The primary advantage of this method is that it causes no disruptions to the ICS devices. This makes it a viable option for OT network security. A disadvantage of passive

probing is that this method is dependent upon the packet traffic and information available. If there is low packet traffic or limited information on received packets, passive scanning might not work. Nonetheless, in order to maintain the proper operation of ICS devices passive probing is a practical approach. Active probing can be enacted on a device by device basis, but it is often still recommended that a degree of passive network monitoring is used for OT networks [3].

The solution to robust security is not as simple as using a hybrid approach. While research has been conducted into using a hybrid of passive and active probing [14], both are still susceptible to spoofing attacks at the data link layer [10]. Recently authors of [43] propose a passive fingerprinting method that operates on Ethernet protocol. The author of [42], has shown passive device management using Modbus protocol. However, for both protocols they do not address the spoofing concerns of the high-level Layer 2 features they gather from transmission data. The proposal in this thesis is to passively probe the physical layer to avoid such spoofing. In the following section, deep learning for passive Ethernet physical layer security will be examined.

2.2 Deep Learning for the Physical Layer

Deep learning is an extension of the machine learning subset of the larger field of artificial intelligence (AI). Machine learning seeks to create decision based algorithms that can learn from data. Deep learning extends this to neural networks which can define the decision criteria itself without input from a user. First made popular in the computer vision space [15], DCNNs have shown to be more effective than traditional neural networks on tasks like classifying digits [24]. As research has developed, DCNNs have also shown success in face recognition [18], speech recognition [6], and even monitoring sensors of self driving cars [9]. The key component of DCNN is the convolutional layer. Figure 2.1 shows the dot product operation of a convolutional filter. The convolutional layer applies a filter (blue) to the input image (gray) and produces a dot product output (red). This filter then strides across the entire image producing a new image on the output. The number of convolutional layers, the number of filters, and the number of pixels to stride are all user-defined parameters. When training a convolutional neural network, updating the filter weights pushes the network to choose the best filters for a set of inputs. Moreover, these convolutional layers can operate in 1-dimension. Figure 2.2 shows how this method can be reduced to a 1-dimensional operation. This allows efficient application of these networks to 1-dimensional data as will be explored in this work. As a whole,

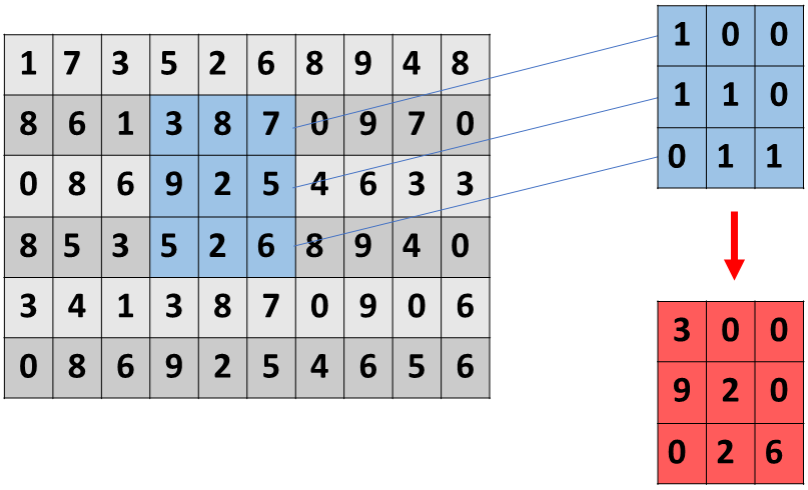

Figure 2.1: Applying a 2D Convolutional Filter

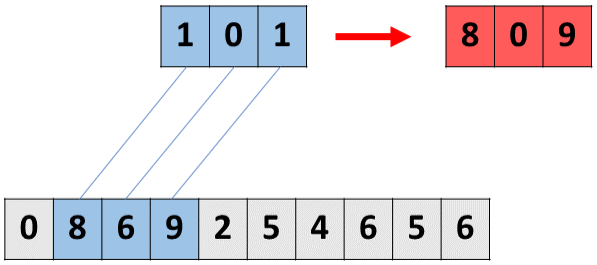

Figure 2.2: Applying a 1D Convolutional Filter

these networks are robust and are integral to the work presented in this thesis.

DCNNs can be leveraged in many different ways in deep learning architectures. There are instances of binary classification, such as classifying between a cat and dog. There are also instances of categorical classification, such as classifying between 10 different animals. Binary classification can be extended to binary detection, such as detecting whether or not it will rain a given day. Anomaly and novelty detection are a subset of binary detection and are key to identifying out of distribution (OOD) data [33]. As a result, novelty detection is key to the OOD detection experiments in later chapters. In context of novelty detection OOD data is data that is outside the data of the classes that a device trained on. Another application of DCNNs is convolutional autoencoders, which can be used to compress and reconstruct an image [27]. This use case will be explored in later chapters. Finally, DCNNs can be deployed with generative adversarial networks (GAN) which seek

to reproduce new data from a given distribution. GANs are often used as a data generation tactic when there is limited data available. This too will be investigated in later chapters.

The DCNN can be focused to address physical layer security. There are instances of deep learning navigating the physical layer for radar detection [7], cell phone detection [36], and even heartbeat detection using radio reflections [16]. The goal of this work is to investigate the capacity of deep learning when applied to physical layer security of Ethernet OT devices. As a result, these trained models will focus on the discrete hardware components that cause perturbations on a transmitted signal. Authors in [34] and [35], look at how perturbations are caused by the digital to analog converter (DAC), power amplifier, and radio frequency (RF) oscillator of a transmitting device. These imprints also exist in the physical layer transmissions of wired Ethernet devices. The DCNN in this paper will exploit such imprints on Ethernet transmissions to distinguish between transmitting devices.

Chapter 3

System Pipeline

The pipeline for applying deep learning to physical layer Ethernet security is described in this chapter. The three sections that will be discussed are data collection, the deep learning model, and authentication.

3.1 Data Collection

Data collection is necessary for any application of deep learning. For the purposes of this work, physical layer samples must be collected from each of the devices on a network. This process is done by digitally sampling voltages from the transmit (TX) pins of each device of interest. The Siglent SDS2352X-E Super Phosphor Oscilloscope was used for digital data sampling, which has a 350MHz bandwidth and a sampling rate of 1M samples/second. As shown in Figure 3.1, the Control/Storage Device pings a Device Under Test to generate a response packet. The Signal-Split printed circuit board (PCB) provides inline access to the Device Under Test TX+/- pins. This differential data is digitally sampled, truncated, and forwarded by the Oscilloscope to the Control/Storage Device. This process is repeated 10,000 times for each device. A sample is 5.7 µs long and results in approximately 11,400 samples.

Clemson University Energy Innovation Center (EIC) is where the majority of the data was collected that is used in this thesis. The Clemson EIC provided a simulation and testing bed that was integral for collecting data in an OT network setting from relevant device manufacturers (i.e. Schweitzer Engineering Laboratories, Woodward, and Allen Bradley). These collections made up

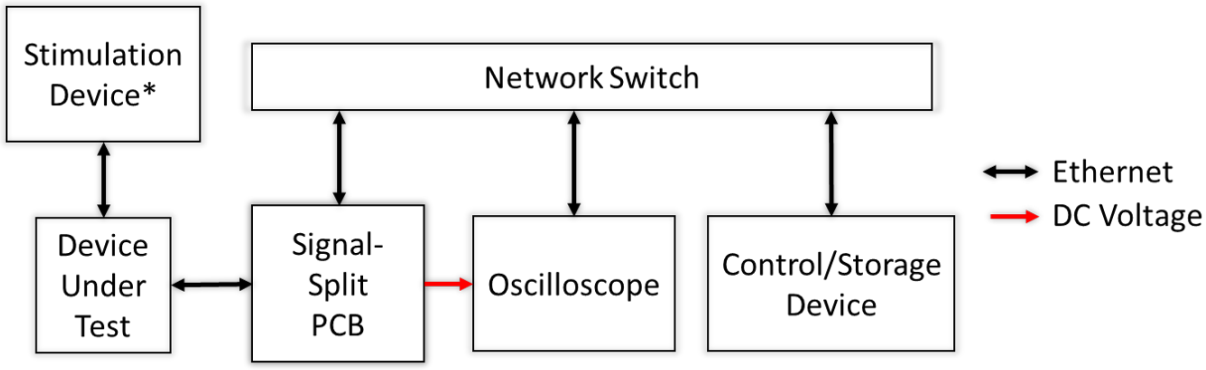

Figure 3.1: The components of data acquisition

the Clemson EIC dataset. Other data was collected using off the shelf network switches (i.e. Cisco, LinkSys, and D-Link). These collections made up the AVS526 dataset. The two Ethernet protocols that were employed were 10BaseT and 100BaseTX. Figure 3.2 and 3.3 show a comparison of these two protocols, 10BaseT and 100BaseTX respectively, sampled from an Allen Bradley programmable logic controller (PLC). Verifying results on gigabit Ethernet and other protocols is left to future work.

3.2 Deep Learning Model

The DCNN is the part of the pipeline responsible for fingerprinting each device. Deep learning models were first made popular in the computer vision space [19] and then extended to the RF signal space [30]. This thesis analyzes transmissions in the time domain, however there are on going research debates over whether the time or frequency domain is more suited for deep learning techniques [49], [29]. One major difference between image data and time domain signal data is that image data is 2-dimensional while time domain signal data is 1-dimensional. Two model architectures for the DCNN were considered for the experiments in this thesis, the ResNet [30] and the MCNet [20]. The MCNet proved to be more efficient while achieving competitive accuracy with the ResNet. Table I compares the attributes and performance of each model. Presentation of accuracy comparison will be given in the Testing Results chapter.

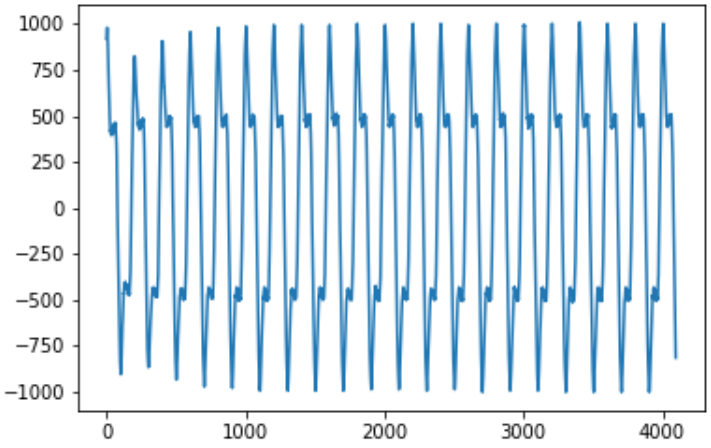

Figure 3.2: Voltage (mV) per sample plot of Allen Bradley PLC in 10BaseT (N=4096)

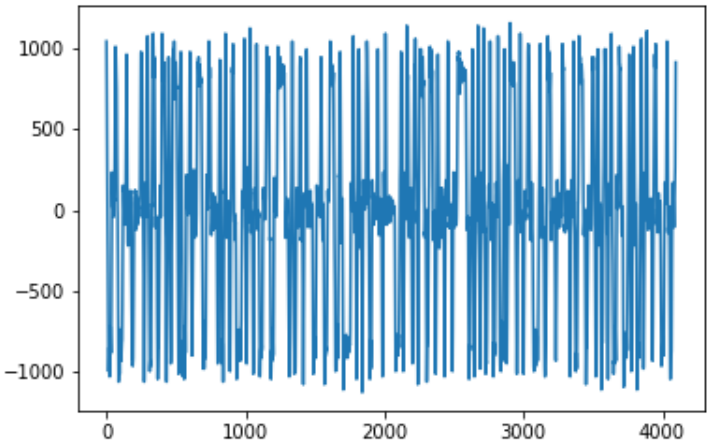

Figure 3.3: Voltage (mV) per sample plot of Allen Bradley PLC in 100BaseTX (N=4096)

	ResNet	***MCNet***
Latency (1 sample)	900ms	290ms
Throughput (256 samples)	150 samples per sec	750 samples per sec
Model File Size (.h5)	211MB	1.96MB
Model Training Parameters	2M	150k

Table 3.1: MCNet and ResNet comparison tested on Nvidia Jetson Xavier

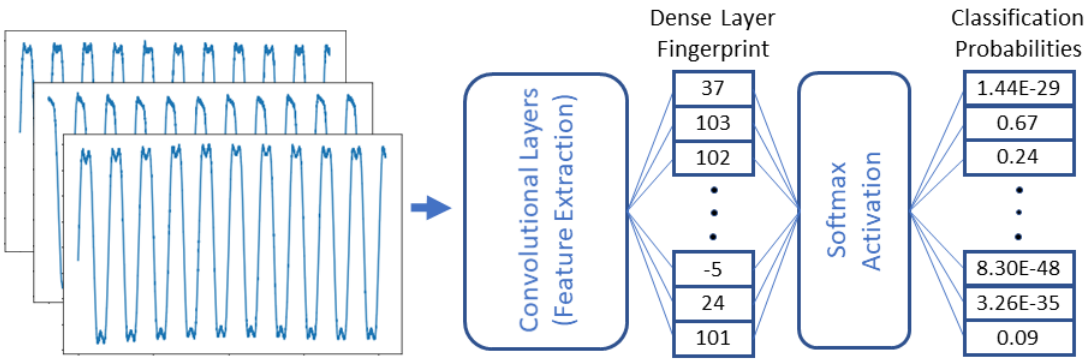

Figure 3.4: Components of DCNN for Classification

3.2.1 Training for Classification

Training the DCNN for classification involves using N sample signal examples taken from D devices. N is the sample size in range [256, 4096] (by powers of 2) of the example extracted at a random index from the initial 11,400 sample collection. The chosen value of N samples is mentioned in all results provided. Figure 3.4 shows a high level overview of the classification DCNN. The data, a 10BaseT protocol device sample, passes through the convolutional layers, which perform many layers of feature extraction. These features are then passed through the dense layer with a filter the same size as the number of classes or devices. A SoftMax activation is applied to the output of this dense layer and a classification is made. For Figure 3.4 this classification would have a probability of 0.67.

The convolutional layers involve a complex combination of max pooling and skip connections. For both DCNNs a categorical crossentropy loss function and Adam optimizer [22] were used, and each architecture was implemented using the TensorFlow python library [5].

There are two reasons why a classification model is useful. One, it provides evidence that a group of physical layer devices can be distinguished by a DCNN. Two, it acts as a feature extractor and compresses representations of devices into lower dimensional space that can be used as a fingerprint. The fingerprint is the output of the final fully connected (dense) layer as shown in Figure 3.4. A unique fingerprint is generated for each device by passing signal examples through the DCNN and removing the SoftMax activation. In the following experiments this fingerprint is the D dimensional output from the dense layer of the DCNN.

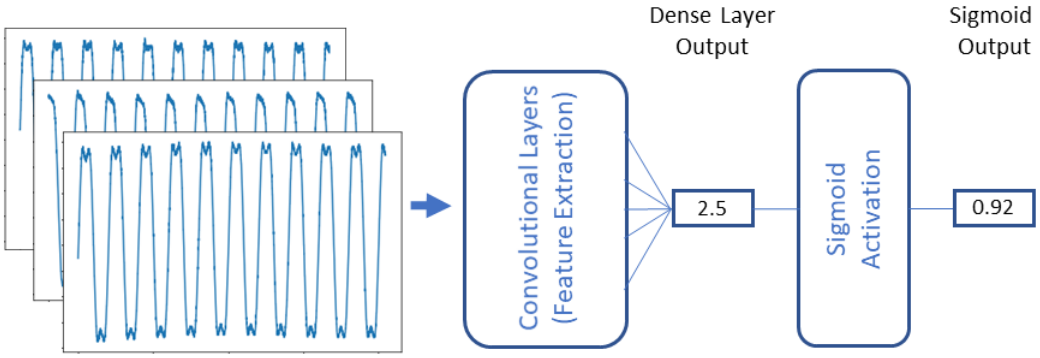

Figure 3.5: Components of DCNN for Detection

3.2.2 Training for ID/OOD Detection

Training the DCNN for in-distribution (ID) and OOD detection involves inputting the signal data the same way as with classification. However, the data is now separated by ID devices (positive or 1) and OOD devices (negative or 0). In a network security scenario, the data from approved or registered devices would be labeled positives. Then either mock intruder devices would be put on the network or some artificial data would be used for the negatives. The DCNN can then be trained in a similar manner as classification, but now with a 1 filter dense layer, binary crossentropy loss, and Sigmoid activation. This new set up is depicted with only one output node in Figure 3.5. From the value of the dense layer output the Sigmoid activation suggests it is more likely an ID device (positive or 1) with value 0.92. The benefits and details in using this model will be discussed in the next section.

3.3 Authentication

The robust authentication of a device is one of the open ended challenges presented in this thesis. There are two primary ways that this problem is approached in this thesis. One, a DCNN creates a fingerprint from the device samples and then machine learning methods are used on this fingerprint to authenticate the device. Two, a DCNN directly evaluates the device sample to then authenticate the device.

3.3.1 Fingerprint Authentication Method

The fingerprint generation method was the first approach that was made to authenticate the devices. Both of the authentication setups are an application of novelty detection. Novelty detection is when a model attempts to identify novel pieces of data given access to preexisting data [17]. In the fingerprint authentication setup, novel data are the fingerprints from an intrusion device and preexisting data would be the registered fingerprints.

The machine learning model that is best suited for novelty detection is the one-class support vector machine (OC-SVM) [17]. The OC-SVM can be implemented using the scikit-learn python library [31].

The OC-SVM proved to have trouble properly authenticating the fingerprints for both registered and intrusion devices. This issue will be discussed in the Authentication Challenge subsection.

3.3.2 Direct Authentication Method

The direct authentication method does not require a fingerprint. Instead, the authentication prediction is made directly from the DCNN as shown in Figure 3.5. The benefit of using this model is simplifying the model training process. The creation of fingerprints and training of a OC-SVM is now unnecessary. Another advantage of using this approach is it allows the model to be used alongside a GAN more easily.

3.3.3 Authentication Challenge

The issue with improperly authenticating persists with both authentication methods. The problem is likely related to the fact that the there is limited exposure to the true distribution of intrusion (or OOD) devices during training in either method. In theory, there are a known number of registered devices and a much larger unknown number of intrusion devices. Furthermore, there is limited knowledge of the ways in which these unknown intrusion devices might differ from the registered devices. The challenge is creating a robust model in these circumstances. A GAN or other data generation or augmentation method appear to be the best option to do this.

Chapter 4

Security at the Edge

While deep learning is an effective tool posed to redefine security and the internet of things (IoT), it is limited by the computational complexity it requires [11]. This has led to a movement in recent years towards TinyML enabling this complex software to run on low power efficient devices [37]. For the purposes of this paper, efficient neural network deployment is key to fit the power, speed, heat, and size requirements of a rack mounted security device. This chapter aims to explain different approaches to maximizing efficiency of the security model, while maintaining the ability to update the model to new devices or other variables.

4.1 Model Parameters and Input Size

The number of model parameters is usually what is referred to when reporting the size of a neural network model. The model parameters have a direct impact on the performance of a neural network model [12]. For instance, you can see the performance and memory difference of ResNet and MCNet in Table 4.1, which made a big impact on the decision to use MCNet in this thesis. Not only do model parameters effect inference speed, they also effect memory requirements and training time.

The relationship between input size and model parameters is an important item. The input size is measured in samples (N). Table 4.1 specifies the relationship using MCNet as the DCNN. Its clear that the parameters have an exponential relationship with the samples. A minimum input size is important because the parameters can get large quickly. Model accuracy related to input size is

Input Size	*Parameters*
256 samples	120k
512 samples	123k
1024 samples	129k
2048 samples	142k
4096 samples	166k
8192 samples	215k

Table 4.1: ID/OOD Detection DCNN: Model Parameters vs. Input Size

	Before TensorRT	*After TensorRT*
Latency (1 sample)	5500ms	1700ms
Throughput (256 samples)	45 samples per sec	140 samples per sec
Model File Size (SavedModel)	2.8MB	3.3MB
Model Accuracy (256 samples)	98%	99%

Table 4.2: Comparison of Before and After TensorRT Tested on Nvidia Jetson Xavier

another important topic that will be discussed in results.

4.2 Deployment with TensorRT Conversion

Deploying a neural network with low size, weight, and power (SWaP) is a challenge for most applications. Neural networks require graphics processing units (GPUs) for optimal speed and processing. GPUs are hardware that has not existed in low SWaP prior to the advent of deep learning. There are many devices on the market that meet this need of efficient computation, but this paper will focus on the Nvidia Jetson Xavier. The Xavier is a 32GB random-access memory (RAM) device with 512-core Volta GPU and 8-core ARM 64-bit CPU. The advantages of this device are the size (105mm x 105mm x 65mm), the PCIe connections, and the Gigabit ethernet. Most importantly, all Nvidia specific software is supported by the large developer community.

The Xavier is compatible with the TensorRT software, which is a software geared towards making neural networks more efficient. TensorRT claims to optimize a neural network for deployment in a myriad of ways: weight precision calibration (FP32 $\Leftrightarrow$ FP16 $\Leftrightarrow$ INT8), layer and tensor fusion, kernel auto-tuning, and dynamic tensor memory. For more information on how this is done visit TensorRT documentation [1].

Applying TensorRT to the MCNet DCNN was successful at speeding up inference time of the model when deployed on a Xavier. The results of this optimization can be seen in Table 4.2.

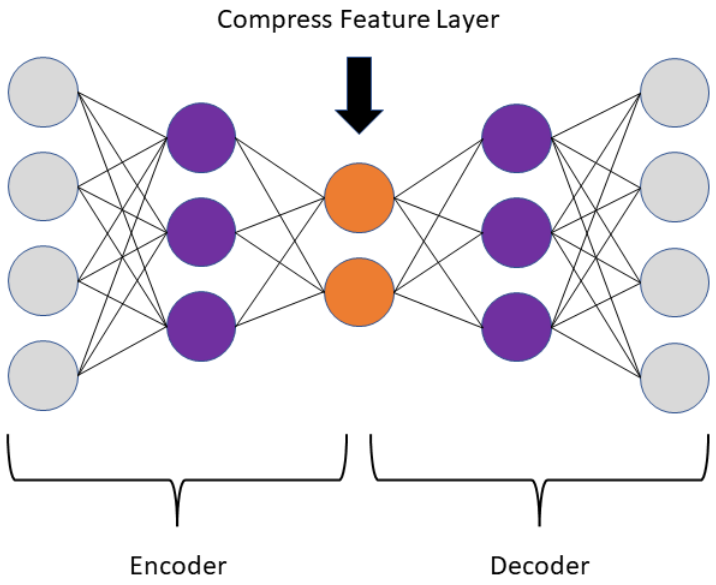

Figure 4.1: An example diagram of an autoencoder

4.3 AI for Matched Filter Generation

Because of the computational expense of DCNNs, there is a motivation to eliminate such neural networks from the deployment pipeline. Matched filter detection is a long standing detection approach that can serve as a replacement. Matched filter detection is a process of correlating a signal of interest with a time delayed version of itself or a template. A device specific signal template would allow the ability to distinguish between different devices on the network. The task becomes creating an effective template for this detection using AI.

Matched filter generation with deep learning was an idea that was inspired by Google's Deep Dream [28]. Deep learning generative models have been documented using a neural network trained on images to generate new images. This generative approach could possibly be extended to training on device signals to generate signal templates for a matched filter.

There are two approaches that are made in this work to create a matched filter signal template. In this section, these two approaches will be presented.

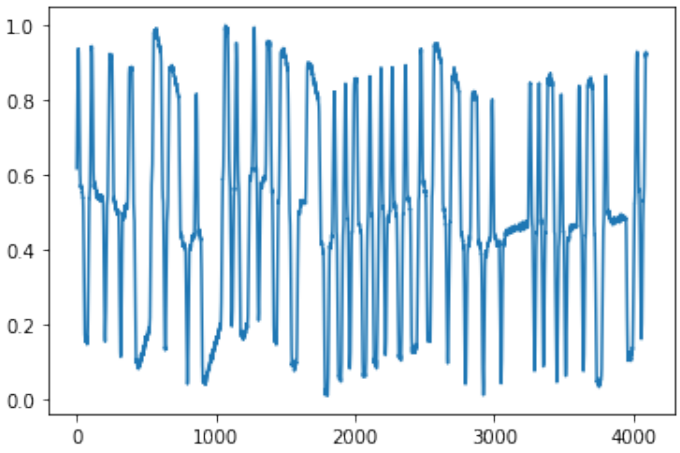

Figure 4.2: Input into autoencoder. 100BaseTX Jetson Nano transmission. ($N = 4096$)

4.3.1 Autodecoder Template Generation

The matched filter generation method with deep learning involves a generative model similar to an autoencoder. Autoencoders are traditionally used to compress or encode a signal into low dimensional subspace and then decompress or decode the signal back into the original space. An example of an autoencoder can be seen in Figure 4.1.

A convolutional autoencoder was extended to device signal compression to verify its feasibility. The device signal was a 100BaseTX signal of sample size $N = 4096$. The encoder network consisted of two 32 filter convolutional layers each with tanh activation and max pooling with pool size 2. Each max pool layer reduced the initial signal by a factor of 2. The last layer of the encoder was a 1 filter convolutional layer with tanh activation. The compressed layer output was then a vector of size 1024. The decoder portion was the same as the encoder, but it replaced instances of maxpooling with upsampling layers. The autoencoder is trained with mean squared error loss and root mean square propagation (RMSprop) optimization [38]. The results of this autoencoder were verified by the signals in Figure 4.2 and 4.3. Note that both signals are normalized to positive unit variance. Furthermore, the reconstructed signal appears to contain no high frequency noise or perturbations. This is a reason for autoencoders to be used for denoising or smoothing purposes in images [25].

The use of an autoencoder for signal compression and reconstruction has been verified. To extend this to signal template reconstruction, the idea is to create a reconstruction of an image

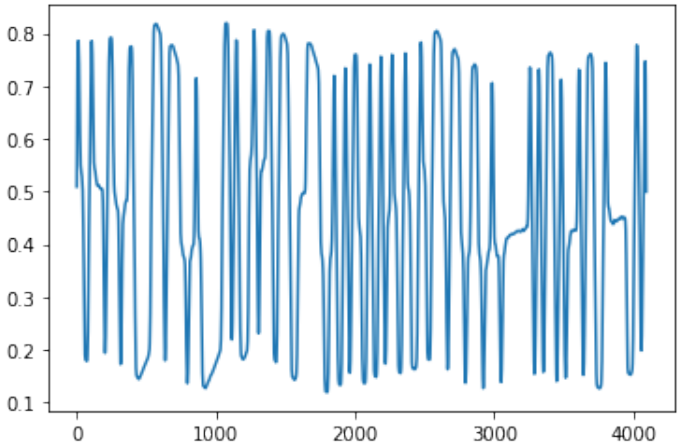

Figure 4.3: Output of autoencoder. Reconstructed 100BaseTX Jetson Nano transmission. ($N = 4096$)

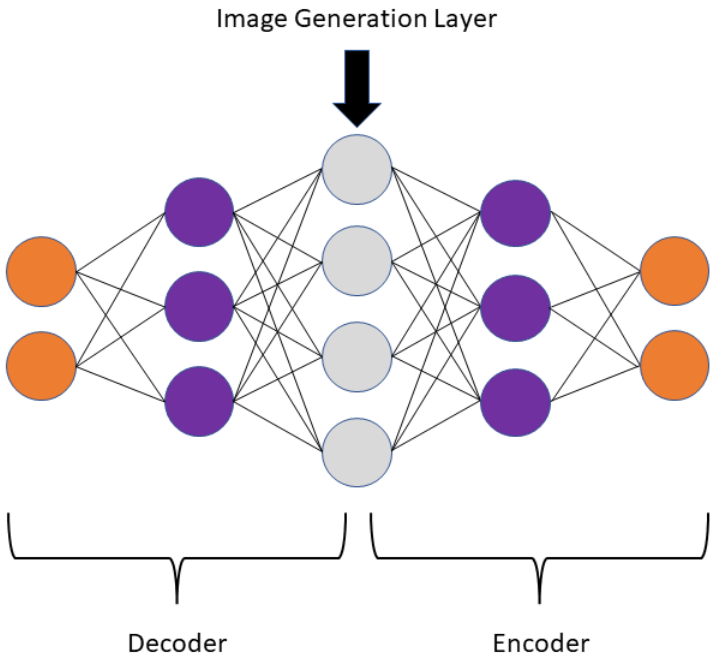

Figure 4.4: An example diagram of an autodecoder

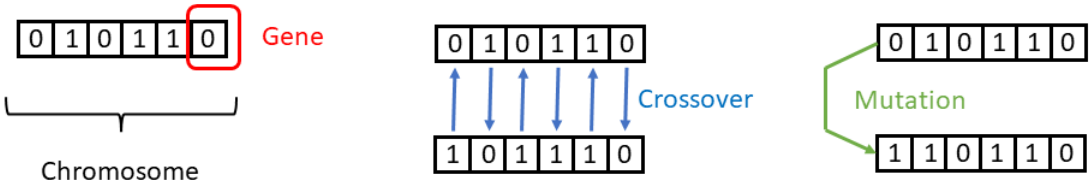

Figure 4.5: Genetic binary chromosome and its operations

that optimizes the DCNN classifier. In short, the goal is to create an autodecoder that takes a compressed signal, expands it out, and compresses back to be classified again. An example of an autodecoder can be seen in Figure 4.4. An autoencoder tries to reproduce image space $\hat{x}$ to be x, while the depicted autodecoder tries to reproduce compressed space $\hat{y}$ to be y.

As mentioned, the idea is to create a reconstruction of an image that optimizes the DCNN classifier. This will require an untrained decoder at the front and a frozen, trained DCNN as the encoder at the back of the network. The only trainable parameters will be part of the decoder. The loss and optimization are the same as what is used in the autoencoder.

The results of this model were not desirable. Reconstructing a signal that optimizes a DCNN classifier is possible and was successful. However, the reconstruction did not result in a template that could be effectively used as a part of a matched filter bank. In an effort to find another way forward, an evolutionary algorithm was explored.

4.3.2 Evolutionary Algorithm Template Generation

This section explores the use of an evolutionary algorithm. An evolutionary algorithm is an algorithm inspired by Charles Darwin's theory of evolution. The specific evolutionary algorithm chosen was a genetic algorithm. A genetic algorithm takes an input vector, usually binary, and performs crossover and mutations. This is similar to how genetic crossovers and mutations take place in a chromosome in the natural world. Figure 4.5 depicts a chromosome along with the crossover and mutation operations. Crossover involves exchanging gene (or bit) values between two mating chromosomes. Mutation involves sporadically flipping a gene (or bit) to the opposite value after the mating process. The entire life cycle of a genetic algorithm is shown in Figure 4.6.

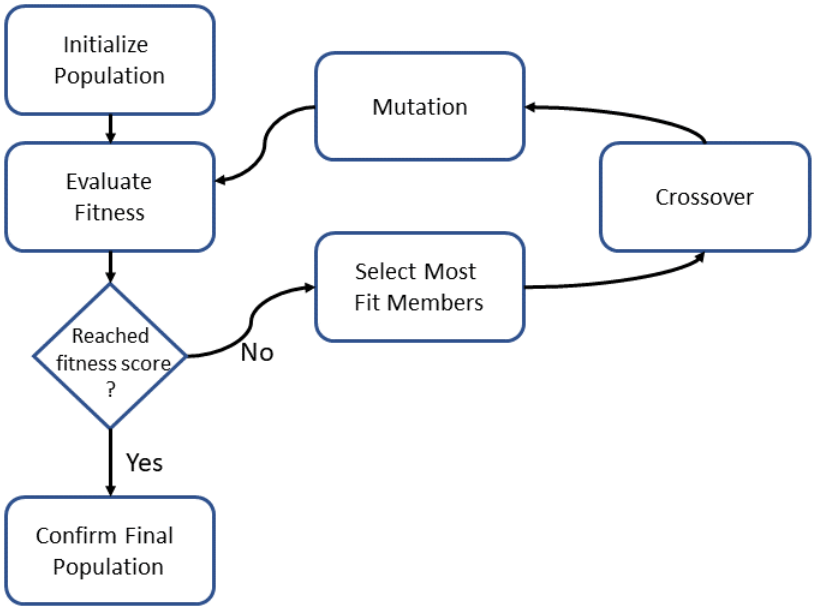

Figure 4.6: The proposed genetic algorithm

4.3.2.1 The Initial Population

The algorithm is a mutli-step process beginning with population initialization. For the purposes of this thesis, the population was initialized as a "perfect" square wave preamble as shown in Figure 4.7. The goal is then to perform enough operations to get the preamble to resemble a signal template for a 10BaseT device, which would look similar to the Dlink switch 10BaseT preamble sample in Figure 4.8. Notice both plots are normalized to unit variance over a 5.7 second window.

4.3.2.2 The Genetic Operations

Genetic algorithms are traditionally used on binary sequences, but in this case the algorithm is reworked to operate on the floating point preamble. The crossover and mutation operations must be adapted to floating point operations. A series of floating-point values behaves differently than a series of bits when introduced to genetic operations. This means that the crossover process cannot flip between either a one or zero and the mutation process cannot involve flipping a bit. Furthermore, a signal template should not have significant discontinuities in the floating-point array

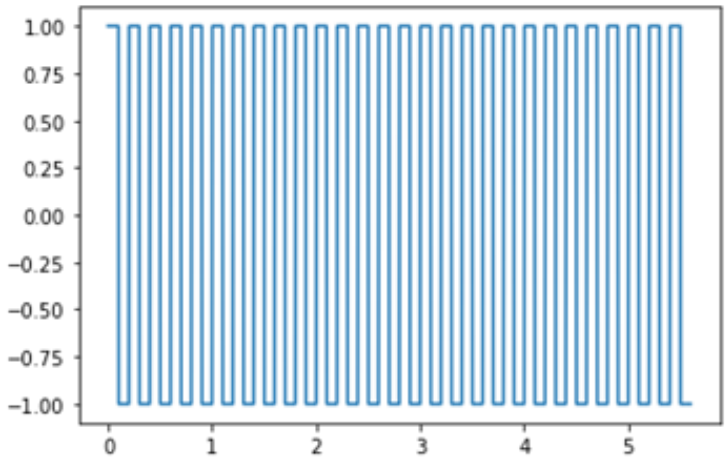

Figure 4.7: Perfect preamble for initial population (5.7s)

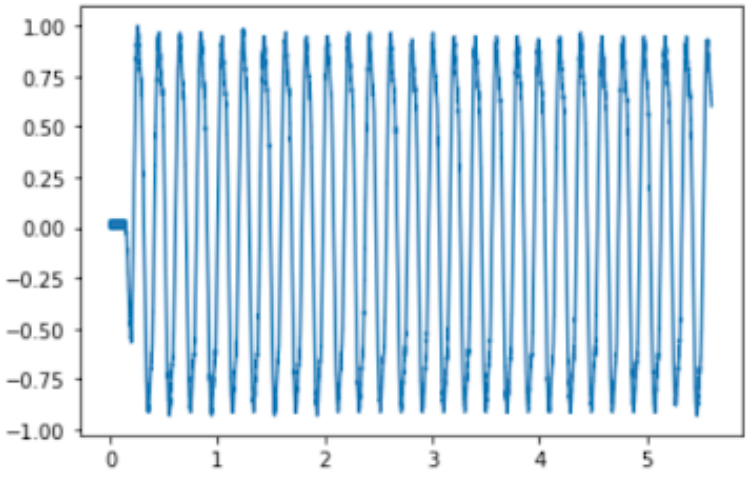

Figure 4.8: DLink preamble sample for comparison (5.7s)

which must be considered in population initialization and genetic operations. Investigating research into genetic algorithms for artificial reverberations [26], information on applying genetic algorithms within these parameters was gathered. The primary takeaway was adjusting the uniform crossover equation. Given individual parent templates x_1 and x_2 and a random uniform (0, 1] weight w the new crossover equation is: $x = x_1 * (w) + x_2 * (1 - w)$. Repeating the crossover equation twice with new w values would result in two new child templates to carry on for the next population. This proved to be a way to prevent the discontinuities caused by uniform crossover. Mutation consisted of mutating the entire N sample example with a set mutation probability. The mutation occurred by adding random gaussian noise using the mean and standard deviation of 1000 real 10BaseT preambles. Other methods of mutation were tried involving changing the amplitude of random samples limited by a variable alpha. To help with discontinuities, the perfect square wave was adjusted at initialization such that it had values at -0.5, 0, and 0.5 at each transition. Finally, a variable alpha was used to limit discontinuities of genetic operations that had too large of a change between two consecutive samples. Alpha controlled the amount of difference at a discontinuity.

4.3.2.3 The Fitness Function

The fitness function follows a similar process to the autodecoder. The fitness function is the DCNN. A template that is fit is one that maximizes the DCNN output probability that a template classifies as a desired device. As an example, if the DLink switch is device 0, the objective is to maximize the DCNN classifier Softmax output for device 0. A threshold set for the Softmax output for device 0 is the fitness score.

The problem with the autodecoder persisted with the evolutionary algorithm. In this case, the DCNN acting as the fitness function was optimized. However, this signal was noisy and did not work as a matched filter template. As a result, solving this matched filter problem is left to future work.

4.4 Filter and Decimation Testing

Following the input size tests, additional bandwidth tests were done focusing on efficiency from a data perspective. The motivation for bandwidth testing is to establish the minimum sampling rate that collection equipment can have while still maintaining accurate predictions with the DCNN

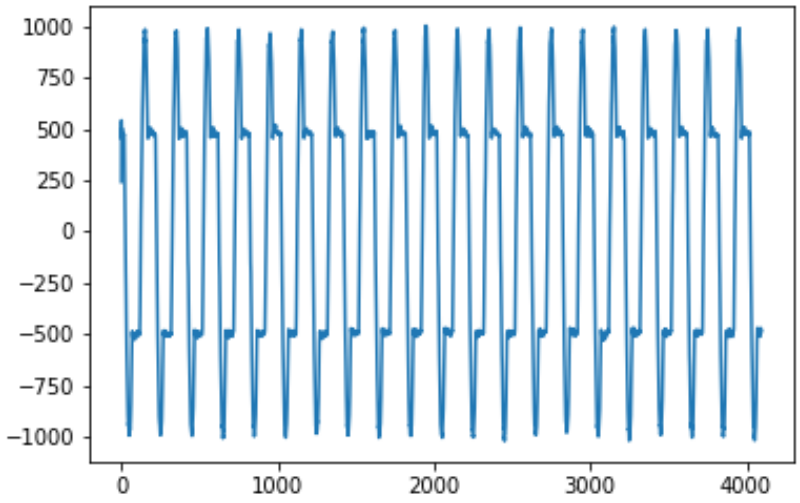

Figure 4.9: Prosoft preamble transmission at cutoff frequency 350Hz (N=4096)

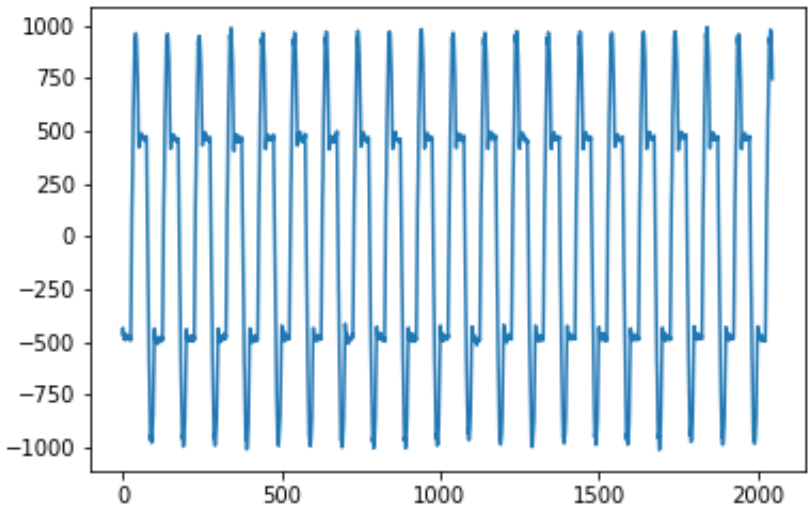

Figure 4.10: Prosoft preamble transmission at a decimation of 2 (N=4096)

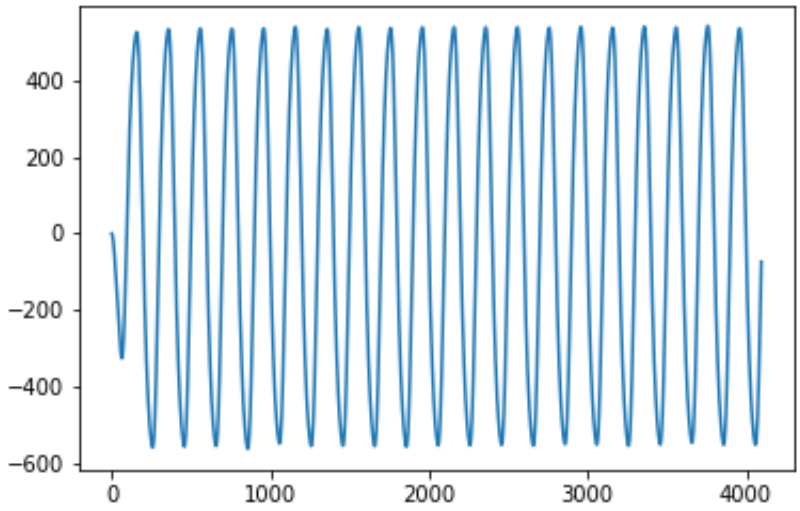

Figure 4.11: Prosoft preamble transmission at cutoff frequency 10Hz (N=4096)

classifier. A smaller minimum sampling rate can mean smaller and cheaper sampling equipment for the system pipeline. In addition, separate filtering tests are done to help understand what the DCNN learned about our signals.

The first experiment with bandwidth was to perform decimation and filtering on the input data. The decimation and filtering experiments decimated the signal example and applied an anti-aliasing filter using python's SciPy library [45]. The anti-aliasing filter used was an order 8 Chebyshev type 1 filter. As an example, a signal example with 4096 samples decimated by 2 would then be 2048 samples. Figures 4.9 and 4.10 show an example of a Prosoft device preamble with 10BaseT protocol size 4096 samples and then decimated to 2048 samples. They appear similar. Note neither signal is normalized and is presented in milliVolts.

Another advantage of bandwidth testing on input signal data is that it brings insight to the frequencies that the distinguishing features take place. While deep learning is often a black box, knowing the significant frequency information of a device signal is useful in understanding what allows the DCNN classifier to learn. The filtering process employed was simple. Incoming data was filtered with an order 2 low pass Butterworth filter again using SciPy. In the results section, it is clear at what frequencies the DCNN makes its decisions by monitoring the training curves. Here we

can see the difference in the appearance of a Prosoft device preamble at 350MHz cutoff frequency and 10MHz bandwidth in Figures 4.9 and 4.11 respectively. The results will show that there is a significant difference in training accuracy for these two signals.

4.5 Transfer Learning

Transfer learning is an important topic for edge deployment. In this section, efficiency from a training perspective will be considered. When operating at the edge, training a new DCNN model from scratch is an inefficient way of adding a device to the registered pool. Adding a device in this way requires many hours of model training time. Transfer learning offers a solution to this problem.

Transfer learning is traditionally deployed when adapting a model to a new data domain. For instance training a dog breed classifier on photos of dogs on a sunny day, would likely not perform well on photos of dogs in the evening with worse lighting. In such a case, transfer learning on breeds of dogs in this new lighting would allow accurate classification without training from scratch. Transfer learning on a DCNN works by freezing all but the final fully connected layers of the model. Then retraining on the new dataset only updating the final layer weights. The result is a model that can adapt to the new training domain in a few epochs.

The experiment conducted with transfer learning sets up a scenario where a proposed new device is transfer learned updating the DCNN model. This use of transfer learning is different than what is traditionally used. Nevertheless, the pre-trained convolutional layers allow this model to adapt effectively. In the results, the success of the transfer learning experiments will be demonstrated.

Chapter 5

OOD Detection for Authentication

This chapter discusses the role of OOD detection for authentication of OT devices. The experiments discussed in this chapter examine the role that ID and OOD data plays on the success of an OOD detector.

5.1 OOD Devices and The Authentication Problem

OOD devices present a considerable challenge when authenticating devices on a OT network. In context of this thesis, OOD devices represent the pool of unknown intruder devices. The challenge is that the DCNN used for authentication contains no direct knowledge about this pool of devices. The only knowledge about whether a transmitting device is an intruder is given by the registered (label 1) and known intruder devices (label 0). As a result the authentication is highly dependent on the pool of devices chosen for registered and known intruder datasets.

The pool of registered devices should not be eliminated in an authentication scenario. However, the pool of known intruders can be manipulated such that it makes the DCNN the most capable of accurate OOD detection. The experiment introduced in the results examines whether adding a larger pool of known intruders increases the likelihood of detecting an unknown intruder.

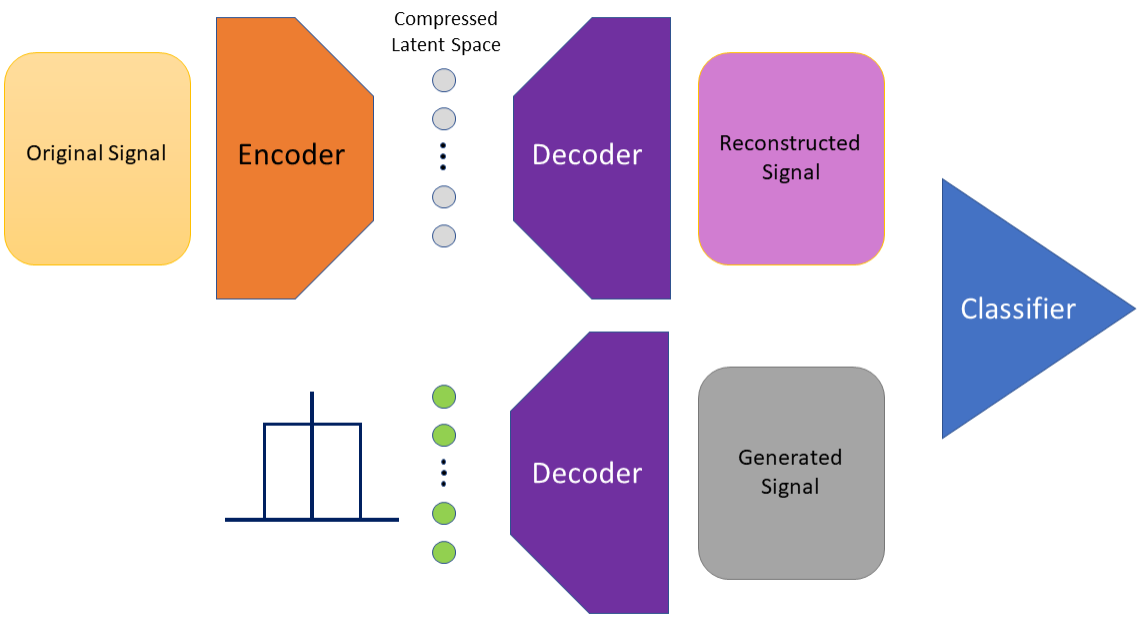

Figure 5.1: Architecture of Generative Autoencoder

5.2 Data Generation for OOD Detection

This section explores data generation for OOD detection. Two deep learning architectures, the autoencoder and the GAN, will be considered in an effort to generate accurate data. Its important to note that not all data is good to generate. For the purposes of this work, data must be generated to resemble the registered devices. Assuming the generation is not perfect, this would allow a classifier to train on data representing the boundary conditions of a registered device.

5.3 The Generative Autoencoder

Autoencoders are used to compress data in a latent dimension and then reconstruct it to the original shape. In doing so the decoder portion of an autoencoder creates an understanding of images in compressed latent space. The goal of the generative autoencoder experiments is to leverage this understanding by taking random points in the latent space to generate new data.

The architecture of this generative autoencoder is shown in Figure 5.1. There are three steps

to training with the generative autoencoder. The first step is to train the autoencoder with mean squared error loss and RMSprop optimization. When a low loss is reached, the second step is to generate data. The generation process involves generating a random vector representing the latent space and constructing this vector into a signal using the decoder. In the third step, the classifier, or the DCNN, then trains on the reconstructed data as registered device data and the generated data as intruder data.

The results of this experiment are not presented in the following section. Although the autoencoder was capable of generating data that looked somewhat similar, it failed to improve the OOD detection of the DCNN on intruder data.

5.4 The One Class Generative Adversarial Network

This section looks at the One Class Generative Adversarial Network (OCGAN) for OOD detection data generation [32]. The researchers behind the OCGAN try to address the novelty detection problem by generating data that closely resembles the data from the one known class. The one known class in the experiment presented is the pool of registered devices. Figure 5.2 lays out the architecture of the OCGAN.

The architecture is complex, but it is simpler if thought of in three parts: the generator, the discriminator, and the classifier. The generator is made up of the encoder and decoder, while the discriminator is made up of the latent discriminator and the visual discriminator. Adversarial training takes place between the generator and the discriminator. Adversarial training is best described as a tug of war between neural networks. The generator attempts to make the most convincing fake data possible, while the discriminator seeks to distinguish this fake data from real data. From Figure 5.2 it is apparent the visual discriminator seeks to distinguish real from fake by looking at the generated signals. The latent discriminator seeks to distinguish real from fake by looking at the latent space. The classifier is independent of the adversarial training.

As the OCGAN trains, there will inevitably be a winner of this tug of war. In the experiments conducted it was often the discriminator. This is likely because the generator struggled to properly mimic the one class registered data. However, experiments were still conducted to evaluate whether the OCGAN can generate data that improves the classifiers OOD detection performance.

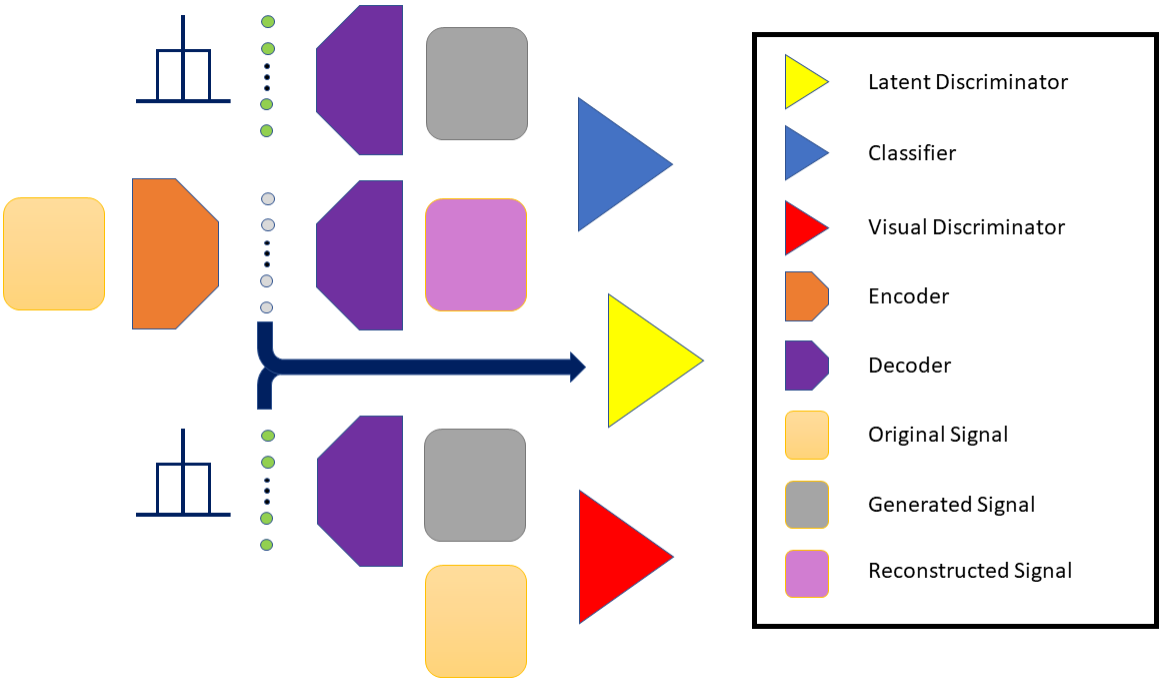

Figure 5.2: Architecture of One Class Generative Adversarial Network

Chapter 6

Results

In this chapter results will be presented and discussed in the order of each experiment that was discussed in the preceding chapters. Results are only given from experiments that were successful or provide useful information.

6.1 Classification Results

The classification results are available for ResNet and MCNet models. The two protocols evaluated are 10BaseT and 100BaseTX. These setups were evaluated with the two datasets provided below.

6.1.1 Clemson EIC Dataset

The Clemson EIC dataset consists of data from 15 devices collected on the testing and simulation bed in the Clemson EIC. There were several devices that made up this dataset, each providing a monitoring function for the simulated power system. A DCNN that can effectively classify these devices is able to gather distinguishing information that can be later used for authentication.

The resulting confusion matrix of applying the ResNet and MCNet DCNNs for classification of 10BaseT preamble transmissions are shown in Figures 6.1 and 6.2. To save space, Table 6.1 provides the device names. The results from the ResNet are not as high as the training data suggested. It was projected to be on par with the MCNet. This is possibly due to overfitting on the training data. Overfitting is when a model learns its training data too well, and as a result can

generalize to unseen data [39]. The confusion matrix for MCNet tells another story. It appears this model learned the training data well and is now highly effective at distinguishing the devices in the dataset. However, the confusion matrix shows the MCNet still struggled to distinguish certain pairs of devices. One such pair is devices 7 and 13. Device 7 misclassified examples from device 13 up to 59 times. Table 6.1 reveals that these two devices are the same Woodward 3000XT device model. This means that although the DCNNs are capable of distinguishing two devices of the same device model, they are still not perfectly distinct.

Device Number	**Device Name**
Device 0	Allen Bradley
Device 1	Prosoft
Device 2	SEL651R_A
Device 3	SEL735
Device 4	SEL751
Device 5	Woodward_EZG_3500XT_A
Device 6	Woodward_EZG_3500XT_B
Device 7	Woodward_LCD_3000XT_A
Device 8	SEL451
Device 9	SEL651R_B
Device 10	SEL2440
Device 11	Woodward_EZG_3400XT_A
Device 12	Woodward_EZG_3400XT_B
Device 13	Woodward_LCD_3000XT_B
Device 14	SMA_EDMMUS10

Table 6.1: Names of Devices in Clemson EIC Dataset

6.1.2 AVS526 Dataset

The AVS526 dataset is a larger dataset with 27 devices collected in a custom local network scenario. This collection setup aimed to test the robustness of a DCNN for Ethernet device classification.

The resulting confusion matrix of applying the ResNet and MCNet DCNNs for classification of 100BaseTX transmissions are shown in Figures 6.3 and 6.4. As done previously, Table 6.2 provides the device and port names. For this dataset the ResNet and MCNet both have high accuracy. This means that trained appropriately, these DCNN models are robust to several devices even when some are the same device models or different ports on the same device. The scale with which these DCNN models classify, suggests that they can collect distinguishable information even on larger networks.

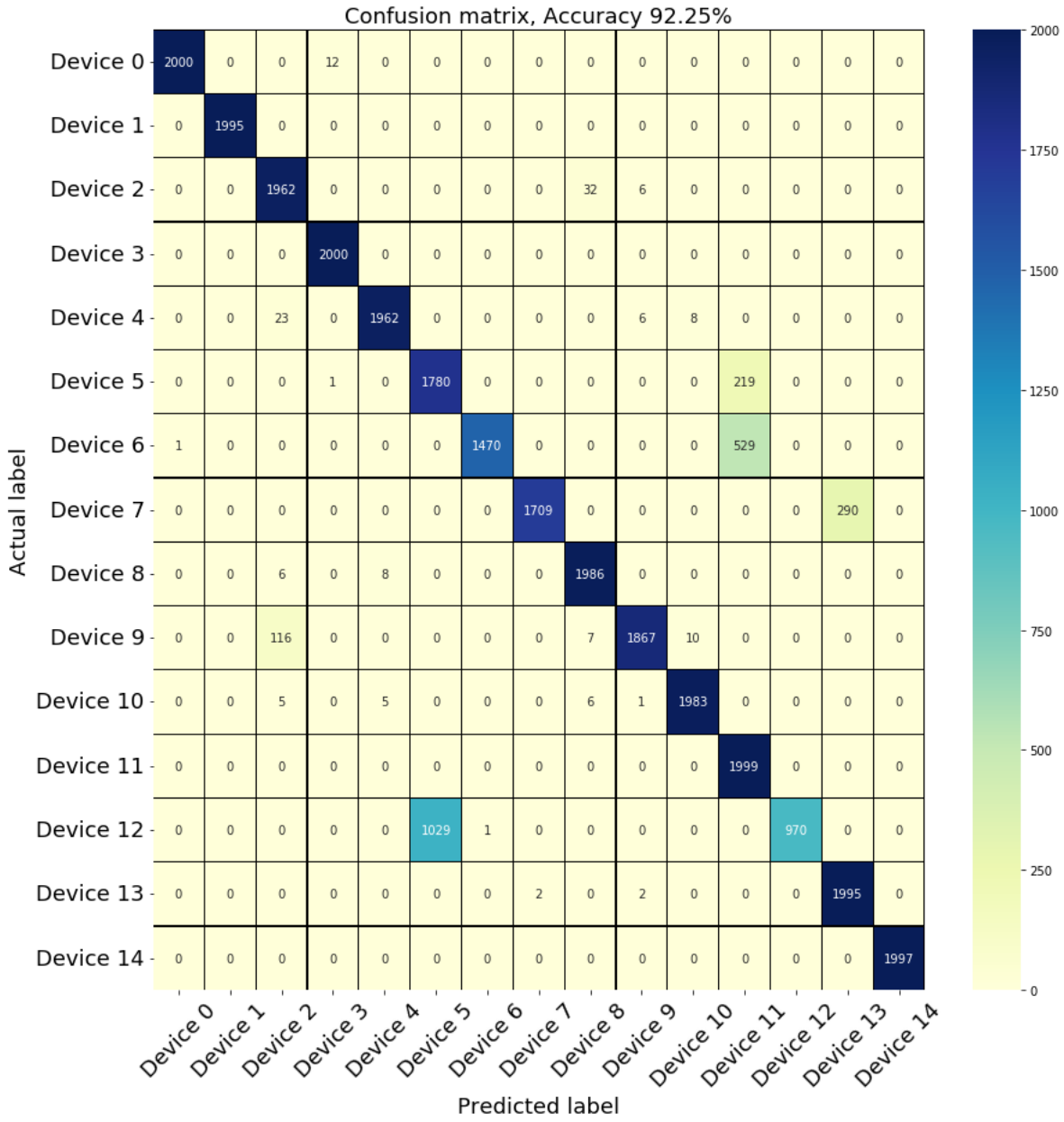

Figure 6.1: ResNet confusion matrix evaluated on 10BaseT Clemson EIC devices (N=4096)

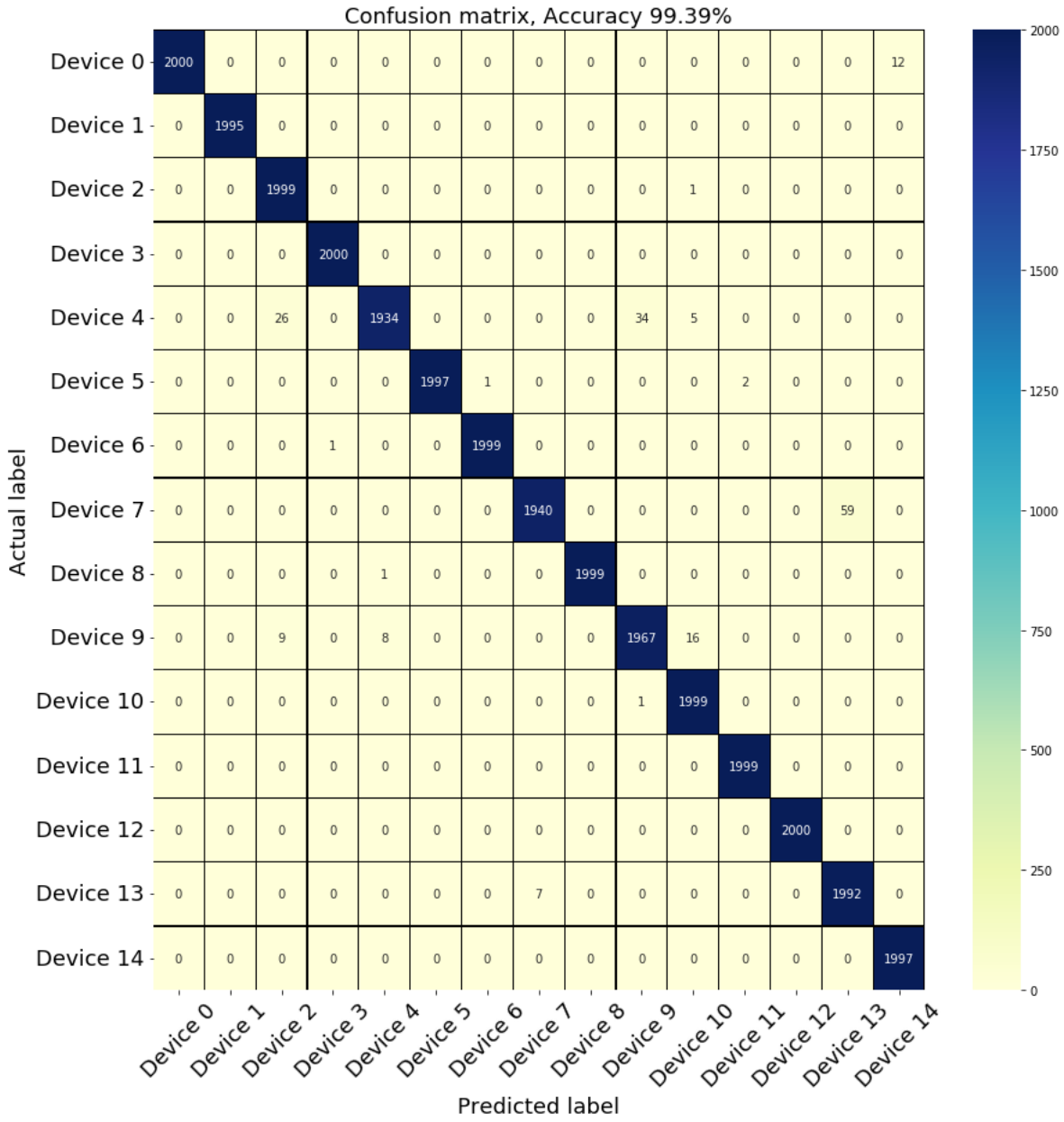

Figure 6.2: MCNet confusion matrix evaluated on 10BaseT Clemson EIC devices (N=4096)

The preponderance of errors occurred between devices 20 and 25 for both DCNN models. In this case, rather than the same device models causing the errors, the errors came from two ports on the same DLink device.

Device Number	Device Name
Device 0	Xavier_1A
Device 1	Cisco_Port2
Device 2	Cisco_Port7
Device 3	Siglent_1A
Device 4	Siglent_1B
Device 5	Cisco_Port3
Device 6	Cisco_Port8
Device 7	Pluggable_USB_A
Device 8	Cisco_Port4
Device 9	Cisco_Port9
Device 10	Pluggable_USB_B
Device 11	Cisco_Port10
Device 12	Cisco_Port5
Device 13	RPI_CCard_A
Device 14	RPI_CCard_B
Device 15	Linksys_Port1
Device 16	Xavier_2
Device 17	Linksys_INet
Device 18	Siglent_2
Device 19	DLink_Port1
Device 20	DLink_Port2
Device 21	Cisco_Port6
Device 22	DLink_Port3
Device 23	DLink_Port4
Device 24	DLink_Port5
Device 25	Cisco_Port1
Device 26	Xavier_1B

Table 6.2: Names of Devices in AVS526 Dataset

6.2 Input Size Testing Results

The input size tests were conducted in order to determine the smallest input size necessary to maintain a high accuracy for the DCNN classifiers. The MCNet was the only model used in these experiments. As with the classification results, the two scenarios analyzed were the Clemson EIC dataset with 10BaseT protocol and the AVS526 dataset with 100BaseTX.

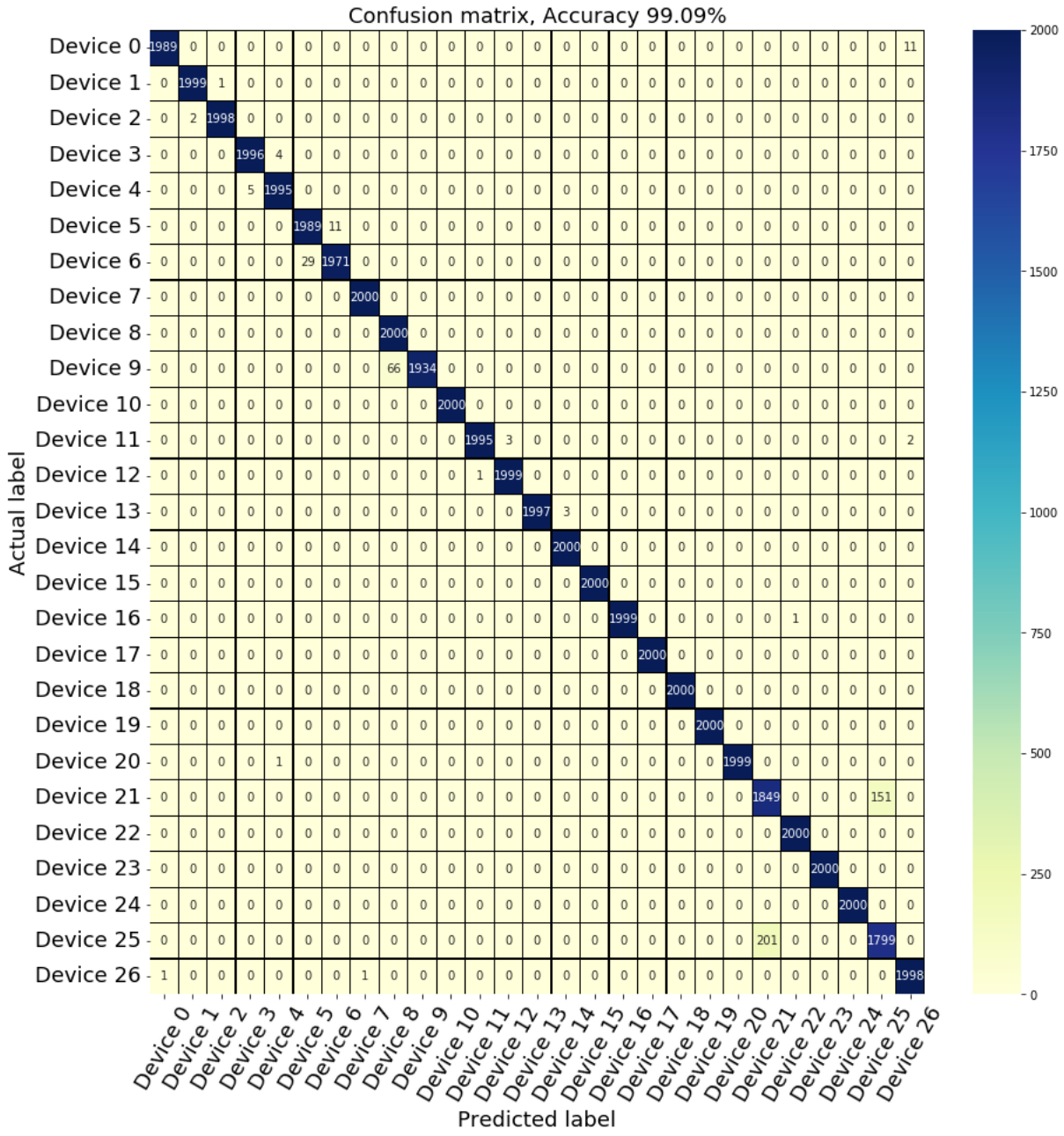

Figure 6.3: ResNet confusion matrix evaluated on 100BaseTX AVS526 devices (N=4096)

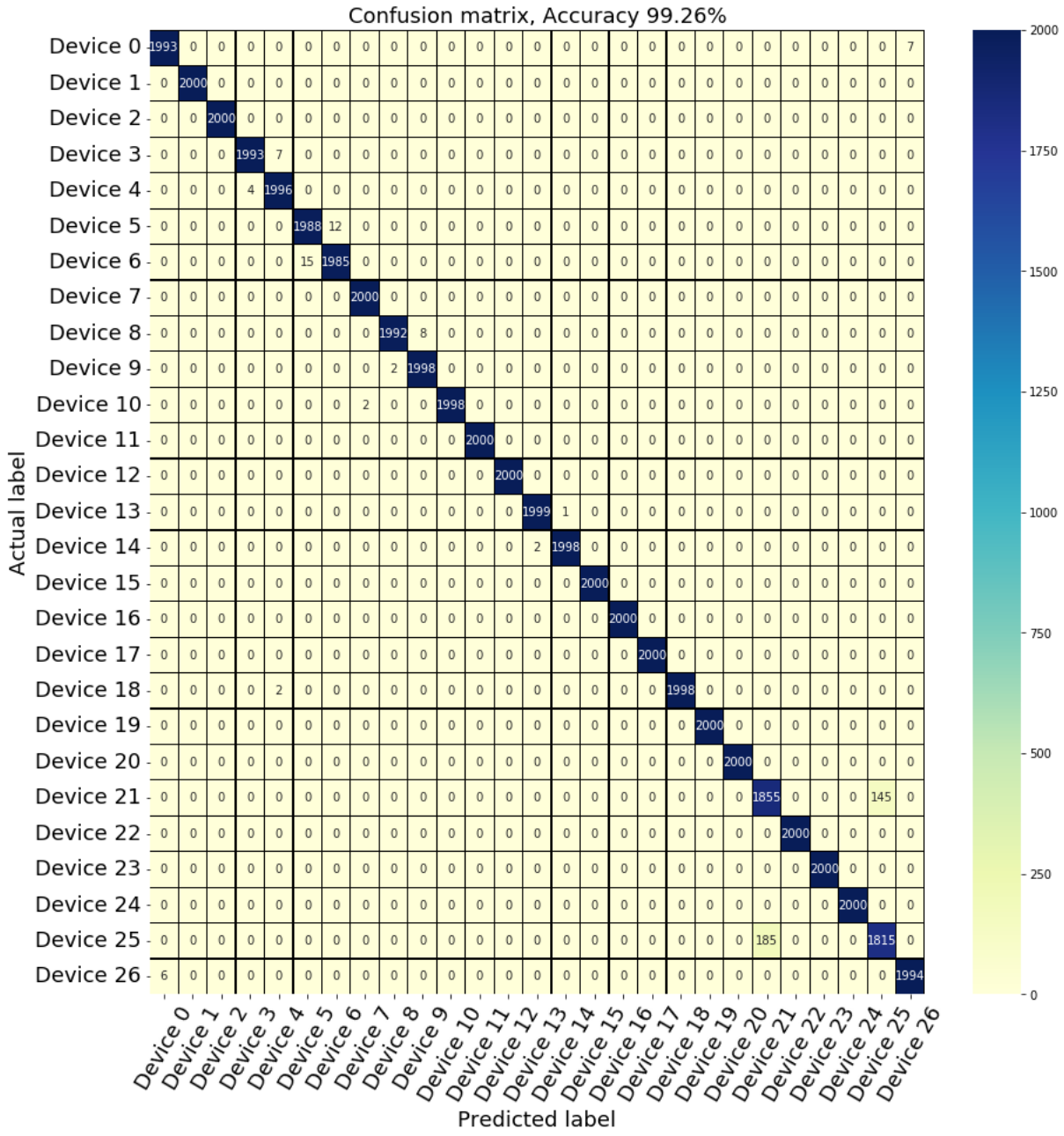

Figure 6.4: MCNet confusion matrix evaluated on 10BaseT AVS526 devices (N=4096)

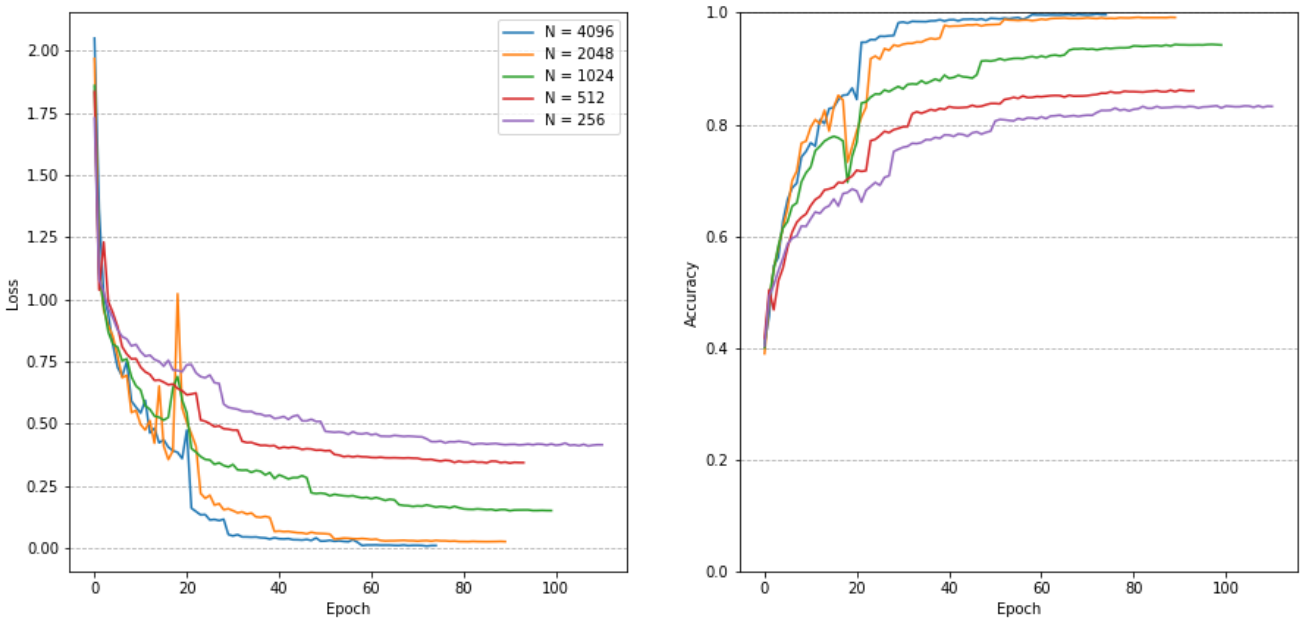

Figure 6.5: MCNet loss and accuracy curves for each N from 10BaseT Clemson EIC dataset

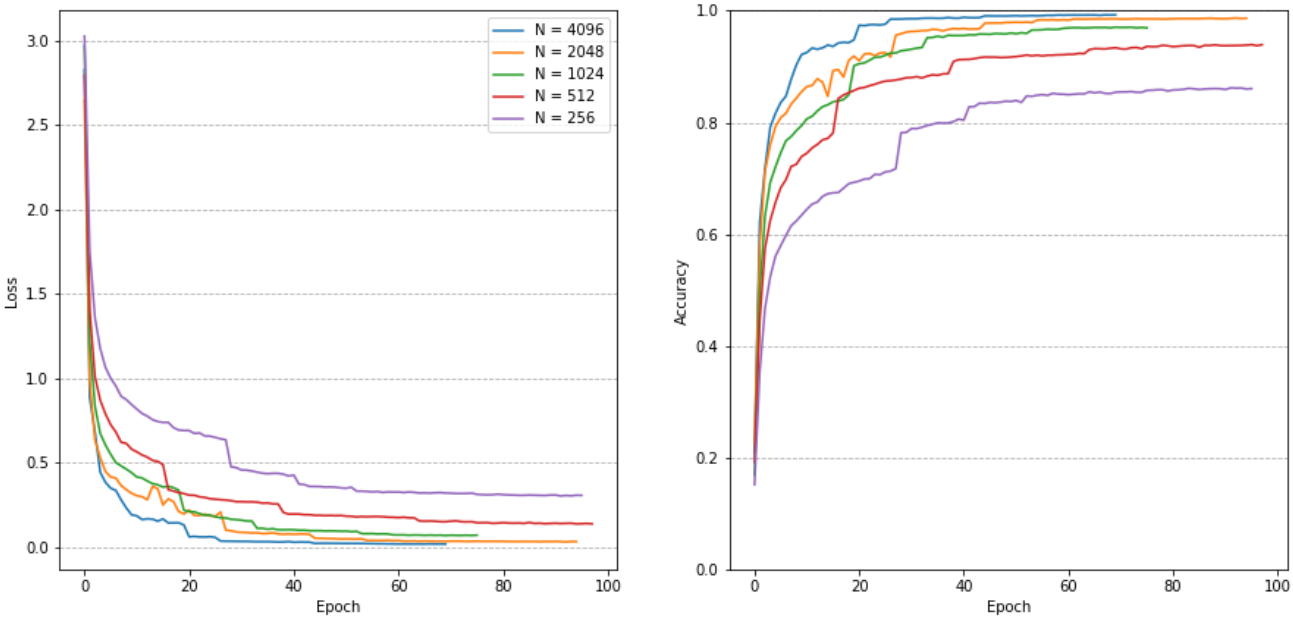

Figure 6.6: MCNet loss and accuracy curves for each N from 100BaseTX AVS526 dataset

6.2.1 Clemson EIC Dataset

The MCNet Clemson EIC dataset training accuracy and loss curves are shown in Figure 6.5. From the loss curves its clear that the more samples the better performance. Diminishing accuracy returns take place with increasing input samples. Values of N that are 1024 and greater achieve at least 90% accuracy. The results suggest that choosing an input size N that is at least 2048 will achieve high accuracy with sufficient training.

6.2.2 AVS526 Dataset

The AVS526 dataset results tell a similar story in Figure 6.6. The curves for this dataset are more compact. Values of N that are 512 and greater achieve at least 90%. This is less samples than is required for the 10BaseT data. Also in this case, the results suggest choosing an input size N that is at least 1024 will achieve high accuracy with sufficient training. Furthermore, the results show input size may be more sensitive for 10BaseT than for 100BaseTX. This would have to be investigated further.

6.3 Filter and Decimation Testing Results

The results in this section cover anti-aliasing decimation and low pass filter testing. The purpose of anti-aliasing filter decimation is to determine the necessary sampling rate for the data collection. The purpose of the low pass filter is to help understand what the DCNN model is learning by filtering out frequency information.

6.3.1 Anti-Aliasing Decimation Test

For this test an anti-aliasing decimation filter was applied to the input data of the MCNet. Figure 6.7 depicts the accuracy and loss over training epochs. For the decimation, an order 8 Chebyshev type 1 filter was used. The decimation rates for a 4096 sample input were chosen to be 2, 4, and 8. Referencing back to Figure 4.10, its apparent that for smaller decimation rates the input signal looks similar. Furthermore, a signal sampled at 4096 samples and decimated by 2, and a signal just sampled at 2048 contain different information. Figure 6.7 shows that it takes a decimation of at least 8 to make a significant impact on the training. This information is consequential for sampling

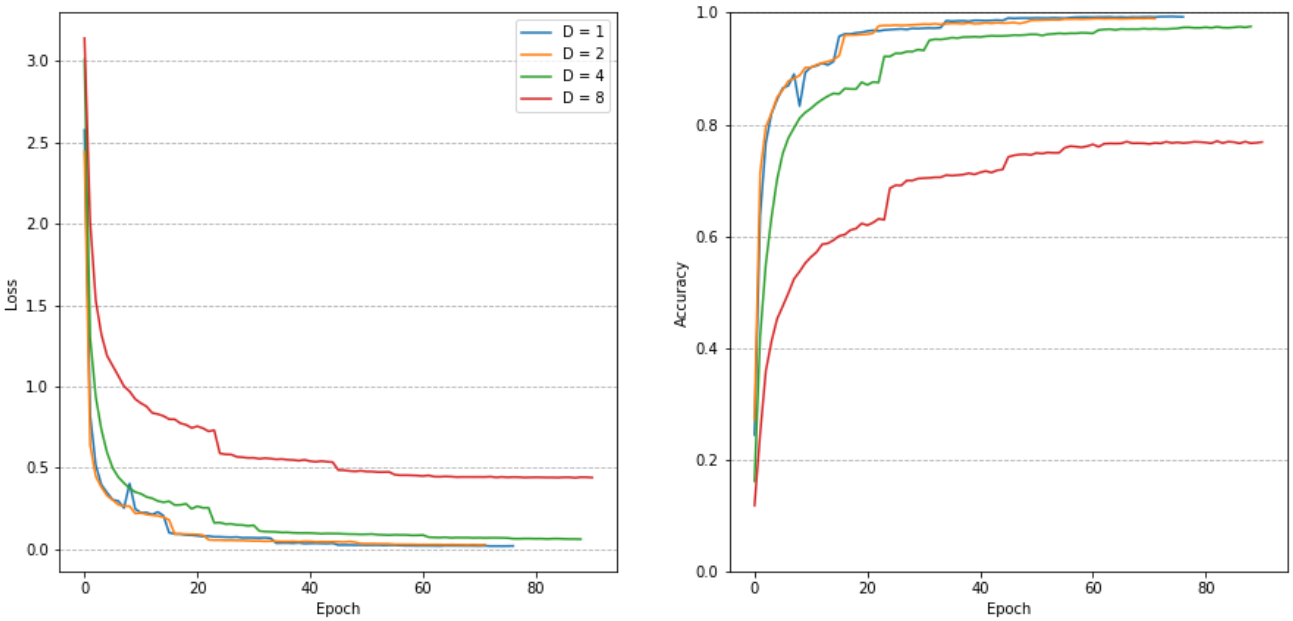

Figure 6.7: MCNet loss and accuracy curves for each decimation rate ($N = 4096$) from 100BaseTX AVS526 dataset

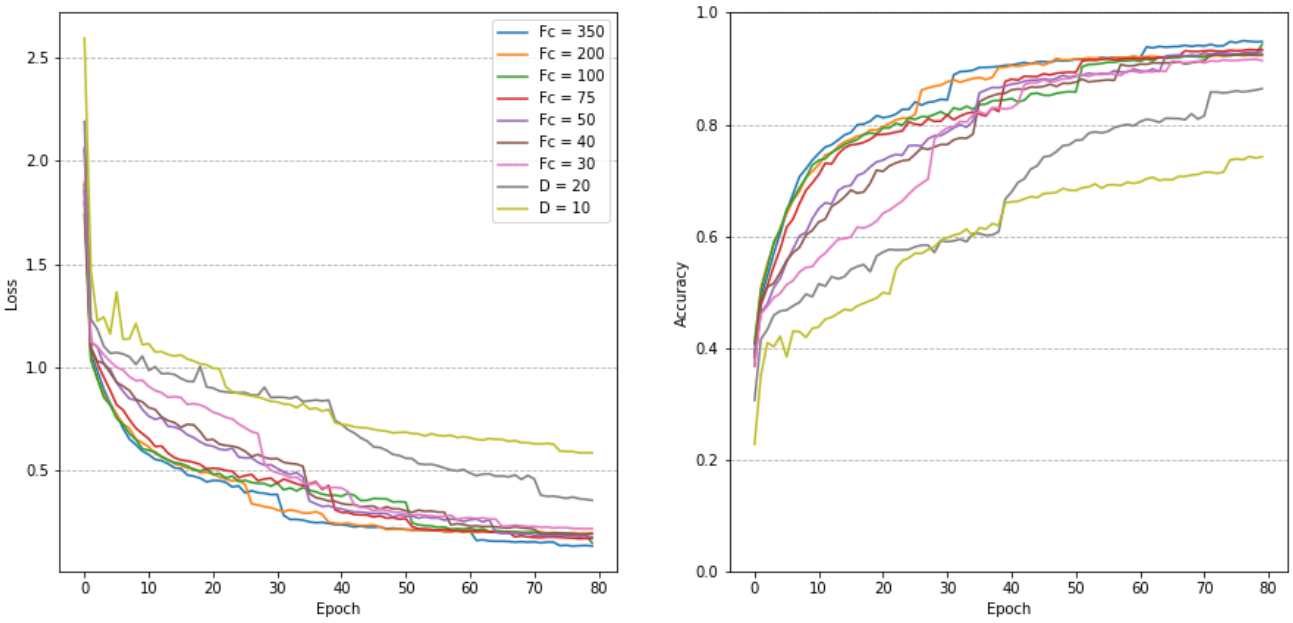

Figure 6.8: MCNet loss and accuracy curves for each cutoff frequency ($N = 512$) from 10BaseT Clemson EIC dataset

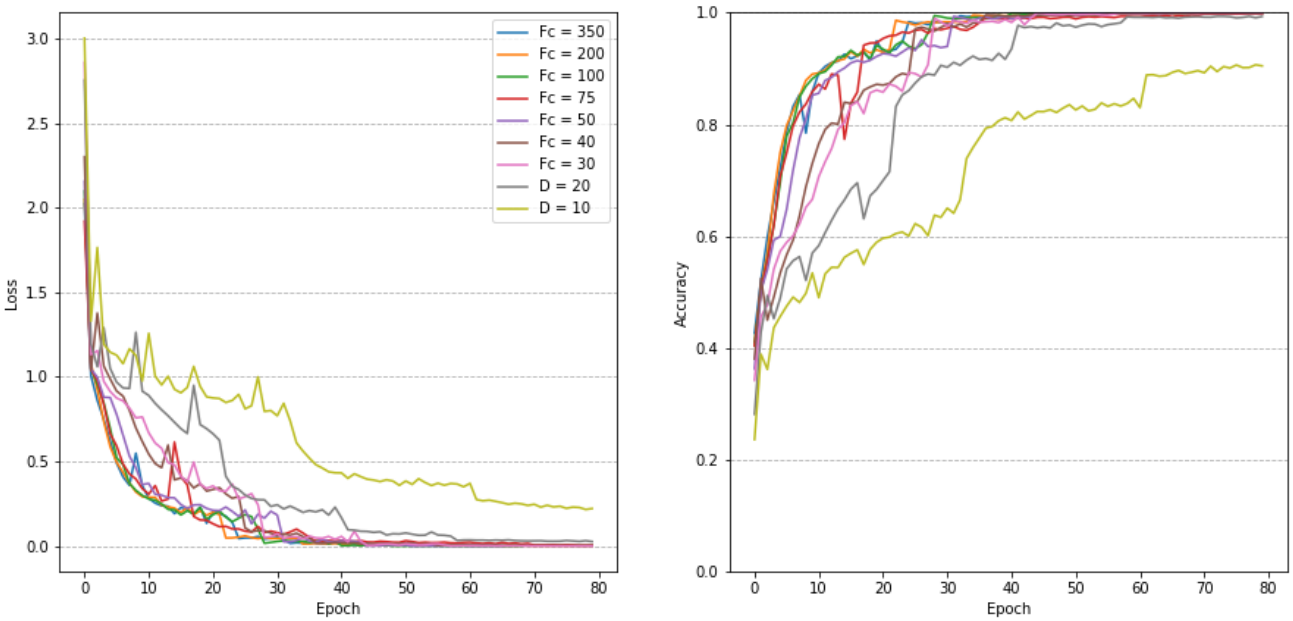

Figure 6.9: MCNet loss and accuracy curves for each cutoff frequency ($N = 4096$) from 10BaseT Clemson EIC dataset

device decisions. The results suggest the DCNN model can work well with a lower sampling rate than is used with the oscilloscope.

6.3.2 Low Pass Filter Test

The low pass filter test was applied to the input data of the MCNet. The filter used was an order 2 Butterworth low pass filter. The cutoff frequencies chosen were 350, 200, 100, 75, 50, 40, 30, 20, and 10MHz. Figures 6.8 and 6.9 examine the accuracy and loss over training epochs for the MCNet with 512 samples and 4096 samples respectively. Both sample sizes were chosen because they show the difference performance when filtering.

It is apparent from the curves that the lower the cutoff frequency the lower the training accuracy. For Figure 6.8, it is evident that there is a drop off in accuracy for frequencies of 20 and 10MHz. For Figure 6.9, this drop off is only distinct at 10MHz. These curves indicate that there is important frequency information between 10-30MHz. This is evidence that a transmission component that perturbs the signal at these frequencies allows the DCNN to distinguish devices to some degree.

There are two additional takeaways from the results presented. The first is that there is still crucial frequency information below 10MHz for both sample sizes. Looking back to Figures 4.9 and 4.11, its obvious there is a large difference in a signal filtered at 350MHz and one at 10MHz with the latter looking more like a sine wave. Yet, this sine wave carries useful information for the DCNN. This information may operate at a much lower frequency than what was examined. The second takeaway is that the smaller the input sample size the less adaptable to frequency information. The 512 sample input MCNet is hindered further by the 20MHz cutoff than the 4096 sample input MCNet. This hints that there may be a relationship between the lower MHz frequency information and a larger input sample size. Lower frequency periodic information would likely require more samples to understand. For both takeaways, more evidence is necessary to make a conclusion. This is left for future work.

6.3.3 Transfer Learning Results

The transfer learning experiment setup was simple. The test used the 27 device AVS526 dataset. The input data was divided into three sets of devices. The first set is one device representing

	Before Transfer Learning	*After Transfer Learning*
Registered Decision	69.24%	99.98%
Intruder Decision	30.76%	99.97%

Table 6.3: Prediction Accuracy of MCNet on New Device Before and After Transfer Learning (Avg. 5 runs)

the registered devices. One device is used for simplicity. The second set is one device representing the device to add to the network. The third set is a group of 5 devices representing the known intruder devices. Each group of devices will be chosen at random from the dataset. The results are averaged over 5 runs with new random sets each time. The DCNN model used was a MCNet trained for direct authentication as is described in Section 3.3.2. This model was trained initially on the first and third sets of data. Transfer learning for a new class involves freezing all but the last dense layer for the DCNN. Then retraining the last layer of the model with the second set added to the first. This then makes a new registered device. If instead, the goal was to add the new device to a list of intruders the second set would be added to the third.

For the purposes of this experiment, its important to note that a new device will be considered registered or intruding by the DCNN before ever knowing said device. This can be thought of as the prior decision before transfer learning. The first column of Table 6.3 explains how the new device was decided for the 5 runs. The new device is predicted to be registered 69.24% of the time before training, and the other times it was considered an intruder. So the data indicates a bias towards being registered. The second column indicates how the device performed once being transfer learned for 1 epoch. Whether the new data was added to the registered pool or the known intruder pool it performed well.

The primary advantage of transfer learning is the significant reduction in adding a new device to a DCNN model. Transfer learning for 1 epoch takes about 40 seconds, while training a model from scratch takes at least 1.5 hours.

6.4 OOD Detection Testing

The OOD detection experiment was motivated by the question of whether increasing the known intruder devices would increase the likelihood of detecting the unknown, or OOD, intruder devices. The test used the 27 device AVS526 dataset. The input data was separated into three sets

	OOD Detection Accuracy
5 Known Intruders	62.40%
10 Known Intruders	50.14%
15 Known Intruders	48.26%

Table 6.4: OOD Detection Accuracy with Varying Number of Known Intruders (Avg. 15 runs)

of devices. The first set is one device representing the registered devices. The second set is a group of three different sizes, 5, 10 and 15 devices, representing the known intruder devices. The third set is a holdout group of 5 devices representing the unknown intruder devices. Each group of devices was chosen at random from the dataset. The results were averaged over 15 runs with new random sets each time. The direct authentication MCNet model was used as is described in Section 3.3.2. This model was trained on the first and second device sets. In each run, the DCNN was trained three times for each of the three sizes, 5, 10 and 15, of the known intruder devices.

The results of the prediction on the holdout set with the three known intruder sizes is shown in Table 6.4. The results did not meet expectations. It turns out the the more known intruder devices, the less accurate OOD detection. This is relationship is counter-intuitive. It would seem, the more devices in a category the more the decision space shrinks. Further testing would have to be done to support this conclusion.

6.5 OOD Data Generation Testing

The OOD data generation experiment utilized the OCGAN. The architecture of the OCGAN allows for the training of the GAN and the classifier in parallel. The implementation of the OCGAN followed its paper closely, but the informative negative mining was left out. Informative negative mining offered a way of improving the model by training the latent space and it was determined this was not necessary. Training the OCGAN took place in four steps: pre-training the autoencoder, training the discriminators, training the generator, and training the classifier.

The results of the experiment are presented in Table 6.5. The table reveals that training the DCNN with OCGAN generated data actually reduces the OOD detection performance. The results were not ideal, but they follow a similar trend to OOD detection testing. The trend suggests that generating more OOD data can cause reduced OOD detection accuracy. Because the OCGAN research is verified it is more likely that the performance degradation is related to the way the

	OOD Detection Accuracy
Before OCGAN	60.8%
After OCGAN	45.1%

Table 6.5: OOD Detection Accuracy with OCGAN Data Generation over 20000 epochs (Avg. 3 runs)

OCGAN was implemented. One such reason is the lack of a effective means of navigating the latent space. Generating data for novelty detection is only useful if it closely resembles known data. Its possible that the generator was not able to generate data of high fidelity. As a result the useless data served to confuse the DCNN rather than improve it. Regardless of the reason behind unsuccessful OOD detection, finding a means of generating and evaluating data for this purpose is left for future work.

Chapter 7

Conclusions and Discussion

Physical layer OT Ethernet device authentication using deep learning can redefine security for power systems. While legacy and low power ICS devices fail to operate in tandem with IT network active scanning technologies, its key to look to passive approaches. Additionally, spoofing attacks can be prevented by authenticating devices at the physical layer. It is shown that discrete physical components within a OT network device impose distinguishable perturbations on Ethernet transmissions. Neural networks, such as the DCNN, are able to leverage these distinguishable perturbations for device authentication. A system pipeline for the deep learning solution involves Ethernet device transmission data collection, training a DCNN on this data, and authenticating with the DCNN. It is shown that although distinguishing between known devices on a network is successful, the challenge of robust detection of unknown devices is a substantial task.

Future work should explore a reliable means of making an authentication decision on devices that the DCNN has not yet seen. Data generation for novelty detection with GANs is a promising approach. Another possible approach is metric learning using the DCNN fingerprint. Research into the specific hardware components that cause distinguishing perturbations may help with this data generation process. If successful, a natural extension would be to find applications of physical layer authentication in the wireless space.

Bibliography

[1] Nvidia accelerated computing tensorrt documentation. `https://docs.nvidia.com/deeplearning/tensorrt/developer-guide/index.html`. Accessed: 2021-11-3.

[2] History of industrial control system cyber incidents, December 2018.

[3] Top 5 ics security best practices, October 2020.

[4] Stop malicious cyber activity against connected operational technology, April 2021.

[5] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[6] Ossama Abdel-Hamid, Abdel-rahman Mohamed, Hui Jiang, Li Deng, Gerald Penn, and Dong Yu. Convolutional neural networks for speech recognition. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 22(10):1533–1545, 2014.

[7] Fahad Jibrin Abdu, Yixiong Zhang, Maozhong Fu, Yuhan Li, and Zhenmiao Deng. Application of deep learning on millimeter-wave radar signals: A review. *Sensors*, 21(6):1951, 2021.

[8] Moruf Adebowale, Khin Lwin, and Alamgir Hossain. Intelligent phishing detection scheme algorithms using deep learning. *Journal of Enterprise Information Management*, ahead-of-print, 05 2020.

[9] Akhil Agnihotri, Prathamesh Saraf, and Kriti Rajesh Bapnad. A convolutional neural network approach towards self-driving cars. *CoRR*, abs/1909.03854, 2019.

[10] Seonggwan Ahn, Thummin Lee, and Keecheon Kim. A study on improving security of ics through honeypot and arp spoofing. In *2019 International Conference on Information and Communication Technology Convergence (ICTC)*, pages 964–967, 2019.

[11] Chunlei Chen, Peng Zhang, Huixiang Zhang, Jiangyan Dai, Yugen Yi, Huihui Zhang, and Yonghui Zhang. Deep learning on computational-resource-limited platforms: A survey. *Mobile Information Systems*, 2020:1–19, 03 2020.

[12] Yu Cheng, Duo Wang, Pan Zhou, and Tao Zhang. A survey of model compression and acceleration for deep neural networks. *CoRR*, abs/1710.09282, 2017.

[13] Kyle Coffey, Richard Smith, Leandros Maglaras, and Helge Janicke. Vulnerability analysis of network scanning on scada systems. *Sec. and Commun. Netw.*, 2018:5, March 2018.

[14] TW Edgar, S Niddodi, TR Rice, WJ Hofer, GE Seppala, KM Arthur-Durett, M Engels, and DO Manz. Safer and optimised vulnerability scanning for operational technology through integrated and automated passive monitoring and active scanning. *Journal of Information Warfare*, 18(4):125–155, 2019.

[15] Frank Emmert-Streib, Zhen Yang, Han Feng, Shailesh Tripathi, and Matthias Dehmer. An introductory review of deep learning for prediction models with big data. *Frontiers in Artificial Intelligence*, 3:4, 2020.

[16] Sabitha Gauni, Mani Megalai, Krishnan Kalimuthu, and M Anjanasree. Detection of breathing and heartrate using a simple vital radio system. *International Journal of Engineering Technology*, 7:311, 03 2018.

[17] C. Geng, S. Huang, and S. Chen. Recent advances in open set recognition: A survey. *IEEE Transactions on Pattern Analysis Machine Intelligence*, 43(10):3614–3631, oct 2021.

[18] Shanshan Guo, Shiyu Chen, and Yanjie Li. Face recognition based on convolutional neural network and support vector machine. In *2016 IEEE International Conference on Information and Automation (ICIA)*, pages 1787–1792, 2016.

[19] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[20] Thien Huynh-The, Cam-Hao Hua, Quoc-Viet Pham, and Dong-Seong Kim. Mcnet: An efficient cnn architecture for robust automatic modulation classification. *IEEE Communications Letters*, 24(4):811–815, 2020.

[21] Changhui Jiang, Yuwei Chen, Shuai Chen, Yuming Bo, Wei Li, Wenxin Tian, and Jun Guo. A mixed deep recurrent neural network for mems gyroscope noise suppressing. *Electronics*, 8(2):181, 2019.

[22] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[23] Thi-Thu-Huong Le, Jihyun Kim, and Howon Kim. An effective intrusion detection classifier using long short-term memory with gradient descent optimization. pages 1–6, 02 2017.

[24] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[25] Woong-Hee Lee, Mustafa Ozger, Ursula Challita, and Ki Won Sung. Noise learning based denoising autoencoder. *CoRR*, abs/2101.07937, 2021.

[26] Edward Ly and Julián Villegas. Generating artificial reverberation via genetic algorithms for real-time applications. *Entropy*, 22:1309, 11 2020.

[27] Ilja Manakov, Markus Rohm, and Volker Tresp. Walking the tightrope: An investigation of the convolutional autoencoder bottleneck. *CoRR*, abs/1911.07460, 2019.

[28] Alexander Mordvintsev, Chris Olah, and Mike Tyka. Inceptionism: Going deeper into neural networks, Jun 2015.

[29] Soha A. Nossier, Julie Wall, Mansour Moniri, Cornelius Glackin, and Nigel Cannings. A comparative study of time and frequency domain approaches to deep learning based speech enhancement. In *2020 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2020.

[30] Timothy James O'Shea, Tamoghna Roy, and T. Charles Clancy. Over-the-air deep learning based radio signal classification. *IEEE Journal of Selected Topics in Signal Processing*, 12(1):168–179, 2018.

[31] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[32] Pramuditha Perera, Ramesh Nallapati, and Bing Xiang. OCGAN: one-class novelty detection using gans with constrained latent representations. *CoRR*, abs/1903.08550, 2019.

[33] Marco AF Pimentel, David A Clifton, Lei Clifton, and Lionel Tarassenko. A review of novelty detection. *Signal Processing*, 99:215–249, 2014.

[34] Adam C. Polak, Sepideh Dolatshahi, and Dennis L. Goeckel. Identifying wireless users via transmitter imperfections. *IEEE Journal on Selected Areas in Communications*, 29(7):1469–1479, 2011.

[35] Adam C. Polak and Dennis L. Goeckel. Wireless device identification based on rf oscillator imperfections. In *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 2679–2683, 2014.

[36] Poonam Rajput, Subhrajit Nag, and Sparsh Mittal. Detecting usage of mobile phones using deep learning technique. 07 2020.

[37] Vijay Janapa Reddi, Brian Plancher, Susan Kennedy, Laurence Moroney, Pete Warden, Anant Agarwal, Colby R. Banbury, Massimo Banzi, Matthew Bennett, Benjamin Brown, Sharad Chitlangia, Radhika Ghosal, Sarah Grafman, Rupert Jaeger, Srivatsan Krishnan, Maximilian Lam, Daniel Leiker, Cara Mann, Mark Mazumder, Dominic Pajak, Dhilan Ramaprasad, J. Evan Smith, Matthew Stewart, and Dustin Tingley. Widening access to applied machine learning with tinyml. *CoRR*, abs/2106.04008, 2021.

[38] Sebastian Ruder. An overview of gradient descent optimization algorithms. *CoRR*, abs/1609.04747, 2016.

[39] Shaeke Salman and Xiuwen Liu. Overfitting mechanism and avoidance in deep neural networks. *CoRR*, abs/1901.06566, 2019.

[40] Iqbal H. Sarker. Deep cybersecurity: A comprehensive overview from neural network and deep learning perspective. *SN Comput. Sci.*, 2:154, 2021.

[41] Tara Seals. The solarwinds perfect storm: Default password, access sales and more, 2020.

[42] Abhinav Singh. Distributed intrusion detection system for modbus protocol. 2020.

[43] Bambang Susilo and Riri Sari. Intrusion detection in iot networks using deep learning algorithm. *Information*, 11:279, 05 2020.

[44] R. Vinayakumar, K. P. Soman, and Prabaharan Poornachandran. Deep android malware detection and classification. *2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, pages 1677–1683, 2017.

[45] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020.

[46] Zack Whittaker. Uk cell giant ee left a critical code system exposed with a default password, May 2018.

[47] Xi Xiao, Dianyan Zhang, Guangwu Hu, Yong Jiang, and Shutao Xia. Cnn–mhsa: A convolutional neural network and multi-head self-attention combined approach for detecting phishing websites. *Neural Networks*, 125:303–312, 2020.

[48] Jinpei Yan, Yong Qi, and Qifan Rao. Detecting malware with an ensemble method based on deep neural network. *Security and Communication Networks*, 2018:1–16, 03 2018.

[49] Jing Zhang, Jing Tian, Yang Cao, Yuxiang Yang, Xiaobin Xu, and Chenglin Wen. Fine-grained ECG classification based on deep CNN and online decision fusion. *CoRR*, abs/1901.06469, 2019.