NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

DETECTION OF SYNTHETIC ANOMALIES ON AN EXPERIMENTALLY GENERATED 5G DATA SET USING CONVOLUTIONAL NEURAL NETWORKS

by

Ashley E. Edmond

September 2022

Thesis Advisor: Preetha Thulasiraman
Second Reader: Chad A. Bollmann

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			*Form Approved OMB No. 0704-0188*

1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** September 2022	**3. REPORT TYPE AND DATES COVERED** Master's thesis

4. TITLE AND SUBTITLE DETECTION OF SYNTHETIC ANOMALIES ON AN EXPERIMENTALLY GENERATED 5G DATA SET USING CONVOLUTIONAL NEURAL NETWORKS	**5. FUNDING NUMBERS** RMQ80
6. AUTHOR(S) Ashley E. Edmond	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000	**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) ONR	**10. SPONSORING / MONITORING AGENCY REPORT NUMBER**

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release. Distribution is unlimited.	**12b. DISTRIBUTION CODE** A

13. ABSTRACT (maximum 200 words)

The research microgrid currently deployed at Marine Corps Air Station, Miramar, is leveraging Verizon's Non-Standalone (NSA) 5G communications network to provide connectivity between dispersed energy assets and the energy and water operations center (EWOC). Due to its anchor to the Verizon 4G/LTE core, the NSA network does not provide technological avenues for cyber anomaly detection. In this research, we developed a traffic anomaly detection model using supervised machine learning for the energy communication infrastructure at Miramar. We developed a preliminary cyber anomaly detection platform using a convolutional neural network (CNN). We experimentally generated a benign 5G data set using the AT&T 5G cellular tower at the NPS SLAMR facility. We injected synthetic anomalies within the data set to test the CNN and its effectiveness at classifying packets as anomalous or benign. Data sets with varying amounts of anomalous data, ranging from 10% to 50%, were created. Accuracy, precision, and recall were used as performance metrics. Our experiments, conducted with Python and TensorFlow, showed that while the CNN did not perform its best on the data sets generated, it has the potential to work well with a more balanced data set that is large enough to host more anomalous traffic.

14. SUBJECT TERMS anomaly, 5G, networks, energy, communications, convolution, neural	**15. NUMBER OF PAGES** 93
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	**18. SECURITY CLASSIFICATION OF THIS PAGE**	**19. SECURITY CLASSIFICATION OF ABSTRACT**	**20. LIMITATION OF ABSTRACT**
Unclassified	Unclassified	Unclassified	UU

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89) Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release. Distribution is unlimited.

DETECTION OF SYNTHETIC ANOMALIES ON AN EXPERIMENTALLY GENERATED 5G DATA SET USING CONVOLUTIONAL NEURAL NETWORKS

Ashley E. Edmond
Lieutenant, United States Navy
BS, United States Naval Academy, 2015

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
September 2022

Approved by: Preetha Thulasiraman
Advisor

Chad A. Bollmann
Second Reader

Douglas J. Fouts
Chair, Department of Electrical and Computer Engineering

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

The research microgrid currently deployed at Marine Corps Air Station, Miramar, is leveraging Verizon's Non-Standalone (NSA) 5G communications network to provide connectivity between dispersed energy assets and the energy and water operations center (EWOC). Due to its anchor to the Verizon 4G/LTE core, the NSA network does not provide technological avenues for cyber anomaly detection. In this research, we developed a traffic anomaly detection model using supervised machine learning for the energy communication infrastructure at Miramar. We developed a preliminary cyber anomaly detection platform using a convolutional neural network (CNN). We experimentally generated a benign 5G data set using the AT&T 5G cellular tower at the NPS SLAMR facility. We injected synthetic anomalies within the data set to test the CNN and its effectiveness at classifying packets as anomalous or benign. Data sets with varying amounts of anomalous data, ranging from 10% to 50%, were created. Accuracy, precision, and recall were used as performance metrics. Our experiments, conducted with Python and TensorFlow, showed that while the CNN did not perform its best on the data sets generated, it has the potential to work well with a more balanced data set that is large enough to host more anomalous traffic.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ACRONYMS AND ABBREVIATIONS

3GPP	Third Generation Partnership Project
5GLL	5G Living Labs
AI	Artificial Intelligence
BACNET	Building Automation and Control network
CNN	Convolutional Neural Network
DER	Distributed Energy Resources
DNN	Deep Neural Network
DOD	Department of Defense
DoE	Department of Energy
EMS	Energy Management System
EPC	Evolved Packet Core
EWOC	Energy and Water Operations
ICS	Industrial Control Systems
IoT	Internet of Things
IPEM	Intelligent Power and Energy Management
MCAS	Marine Corps Air Station
MEC	Mult-Access Edge Computing
mmWave	Millimeter wavelength
NFV	Network Function Virtualization
NPS	Naval Postgraduate School
NR	New Radio
NSA	Non-Standalone
OFDM	Orthogonal Frequency-Division Multiplexing
OS	Operating System
PCAP	Packet Capture
PV	photovoltaic
RAN	Radio Access Network
ReLU	Rectified Linear Unit
ResNet	Residual Network
RRU	Remote Radio Unit

SA	Standalone
SDN	Software Defined Network
SLAMR	Sea Land Air Military Research
TSNE	T-Distributed Stochastic Neighbor Embedding
UWB	Ultra-Wideband

ACKNOWLEDGMENTS

I find myself in constant awe of my professors and peers here at the Naval Postgraduate School. I never saw myself obtaining a master's degree in a technical field after undergraduate school, but I am forever grateful that I was given the opportunity. I have met many people during my time who have had a major impact on my professional and personal life. I would like to thank Dr. Preetha Thulasiraman for guiding me on this journey. I would also like to thank the NPS AT&T team and TMGCore team for supporting my data collection; without this support my results would not be possible. Thanks goes to my family for giving me moral support when studying got tough. Last, I would like to give a special thanks to my fiancé, whom I met my first quarter here and who continues to give me constant love, support, and encouragement.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

The U.S. electrical grid is a complex system that is at risk from various threats. In 2011, "the Department of Defense (DOD) partnered with the Department of Energy (DoE) and the National Renewable Energy Lab to develop renewable energy technology to cut costs, provide energy security, and comply with DOD mandates" [1]. The main recommendation that came from these government organizations was the implementation of microgrid technology at military bases [1]. "A microgrid is a local energy grid with control capability, which means it can disconnect from the traditional grid and operate autonomously" [2]. The benefits of the microgrid are that it provides resiliency, incorporates renewable energy, and provides redundancy to important facilities if the power grid fails. [3].

A. MICROGRID AT MIRAMAR

In May 2016, the electric companies Black & Veatch and Schneider Electric were selected to design and build the microgrid at Marine Corps Air Station (MCAS) Miramar in San Diego, CA [3]. The Miramar microgrid is the first of its kind on U.S. military bases and is the most energy-forward defense installation in the nation [3], [4]. In the event of a utility grid outage, the companies designed the microgrid to generate power for multiple facilities using energy resources [3]. The microgrid at Miramar is operated directly out of the Energy and Water Operations Center (EWOC). The EWOC is the main control hub for all energy control systems and activities [3]. It manages and operates distributed energy resources (DER) including but not limited to photovoltaic (PV) array inverters and backup generators located across the base, as well as the building-level research microgrid. The building level research microgrid is the EWOC environment for testing and validating potential new systems before being added to the larger installation-wide microgrid at the base.

Non-critical control and management of the building-level research microgrid is performed by an Intelligent Power and Energy Management (IPEM) integrated microgrid controller, referred to as the Energy Management System (EMS). The EMS was installed

by Raytheon in 2015 but did not have full operational status until February 2022. The EMS is housed in the EWOC facility. The Miramar microgrid was completed in March 2021 [4].

B. ENERGY COMMUNICATIONS NETWORK AT MIRAMAR

The various DERs that the EWOC currently manages and controls are dispersed across the Miramar base. However, the EWOC does not have remote monitoring capabilities of its PV inverters and backup generators, which limits visibility of DER operational status on a continuous basis. In August 2021, the EWOC partnered with U.S. Ignite, a national non-profit, to build and implement an energy communications infrastructure that would allow MCAS Miramar to support smart technology using Internet of Things (IoT) devices. EWOC and U.S. Ignite aim to leverage wireless communications to connect specific DERs to the EMS located in the EWOC. Specifically, the EWOC and U.S. Ignite proposed the use of 5G communications [5].

U.S. Ignite has a 5G living lab (5GLL) program at MCAS Miramar that is funded through NIWC-PAC and connected to the Verizon 4G LTE/5G Ultra-Wideband (UWB) network. This is a technology pilot program meant to develop applications of 5G that support the DOD mission. The Verizon commercial network (through the 5GLL effort) currently provides ubiquitous 4G LTE connectivity across the base. There are also two dozen small cell nodes that have been installed [5]. The small cell nodes operate in the millimeter wavelength (mmWave) spectrum to provide areas of ultra-high bandwidth and low latency across the base. PV inverters and backup generators located at various buildings on the base are within range of one or more of these small cell nodes. By leveraging 5G, the EWOC aims to have seamless connectivity to remotely monitor and control DERs (PV arrays and backup generators) across the base, improving the ability to assess the situation and respond accordingly during outages and other events [6]. Figure 1 provides a suggested architecture for EMS integration with PV arrays and back generators over Verizon's 4G LTE and 5G networks.

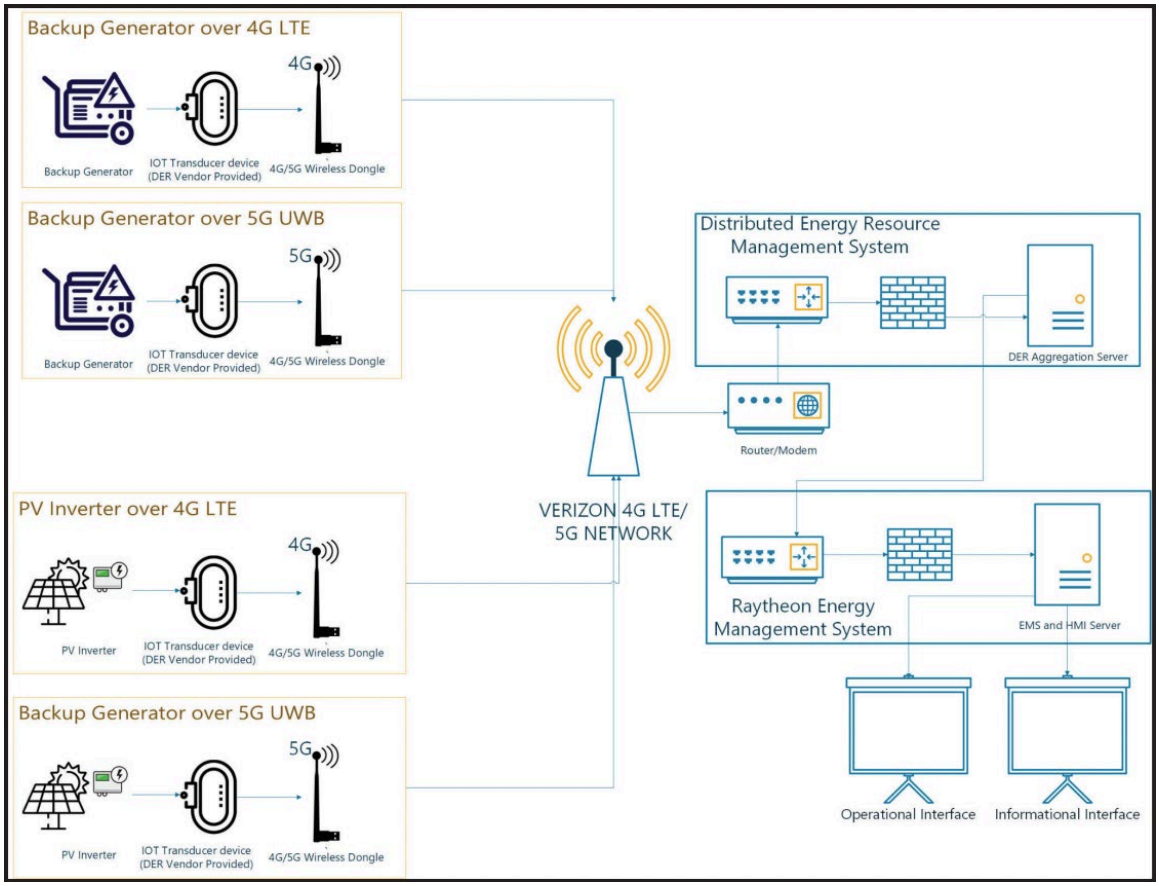

Figure 1. Suggested Research Microgrid 4G LTE/5G Control of PV and Backup Generators. Source: [6].

The wireless infrastructure discussed above is currently in the phase of implementation and testing at Miramar. At present, the PV inverters and backup generators to be included in the network have been identified, but connectivity between the DERs and the EMS has yet to be established.

C. MOTIVATION AND CONTRIBUTION

The Verizon 4G LTE/5G network currently deployed at Miramar is a commercial, non-standalone (NSA) network as seen in Figure 1. In the NSA architecture, the control signaling of the 5G radio network is anchored to the 4G core (further discussion of the 5G NSA architecture will be provided in Chapter II). The use of a commercial network exposes Miramar to a range of cyber threats. While Verizon has a robust cybersecurity framework that deals with the standard issues of encryption, authentication and availability, the

Verizon NSA network architecture does not provide technological avenues for cyber anomaly detection.

The objective of this thesis is to develop a traffic anomaly detection model using supervised machine learning (ML) for the energy communications infrastructure at Miramar. The idea is to produce a mathematical model that would allow us to classify packets/data traffic as anomalous or benign based on specific data features. As stated above, the energy communications network at Miramar is still under implementation. Therefore, our goal in this thesis is to provide a proof of concept for the use of supervised ML to efficiently detect anomalies on 5G data. It is imperative that the cyber analytical framework does not substantially decrease resiliency of the microgrid.

In this thesis, we develop a preliminary cyber anomaly detection platform using supervised ML. Specifically, we use a Convolutional Neural Network (CNN), which is a type of deep learning model used for classification tasks. We chose the CNN because of its ability to train itself using a labeled dataset and subsequently classify data that passes through is benign or anomalous. Incorporating anomaly detection for the messages transmitted between the DERs and EMS over the NSA 5G network will help operators reduce false alarm rates, limit power consumption, and potentially detect cyber threats.

Since 5G network data has not yet begun to flow at Miramar, the dataset that we used to train our algorithm was generated using an AT&T 5G cellular tower deployed at the Naval Postgraduate School (NPS) Sea Land Air Military Research (SLAMR) facility. We leveraged the NPS Cooperative Research and Development Agreement (CRADA) with AT&T and TMGCore to generate a benign 5G data set. The dataset was then manipulated to include anomalies to train and test the accuracy of the CNN. The research presented in this thesis is foundational and provides a starting point to use ML models for anomaly detection on the Miramar energy communications network.

The contributions of this work are as follows:

- The first experimental collection of 5G network data using the AT&T network infrastructure at NPS.

- Manipulation of the 5G data set through the addition of synthetic anomalies. Creation of five different datasets that incorporate different percentages of anomalies within the data (10%, 25%, 30%, 40%, and 50%). We use these datasets to train the CNN.

- Use of a preprocessing technique to transform the network flow data into data that can be used as input to the CNN.

- Comparison of the different datasets to show the differences of how the CNN performs and identify any performance tradeoffs.

This thesis supports funded research through the Office of Naval Research and their NextStep program to study cyber anomaly detection on 5G supported energy communications networks.

D. THESIS ORGANIZATION

The remainder of this thesis is organized as follows: In Chapter II, we provide an overview of the CNN model and discuss the differences between 5G standalone (SA) vs. 5G NSA architectures, as they relate to cyber security. We also provide a thorough literature review. Chapter III describes the experimental process of obtaining the dataset used for training the CNN. We also explain how the CNN is used with the data set and how anomalies were implemented. Chapter IV describes and analyzes the results obtained from the experiments. Chapter V concludes the thesis with recommendations for future research opportunities.

THIS PAGE INTENTIONALLY LEFT BLANK

II. BACKGROUND

In this chapter, we provide a basic overview of a CNN model. We also discuss the 5G architecture of a SA vs. NSA network to highlight the significance of the MCAS Miramar communication architecture in terms of cybersecurity. Lastly, we provide a literature review that highlights the related work in this area, specifically the use of CNNs to detect cyber anomalies.

A. OVERVIEW OF CNN MODEL

The CNN model is used in this research to classify 5G wireless data. CNNs are commonly applied in imagery data analysis to find in finding patterns in images. Figure 2 illustrates the typical architecture of the CNN, which is separated into two main networks which are the Feature Extraction Network and the Classifier Network [7].

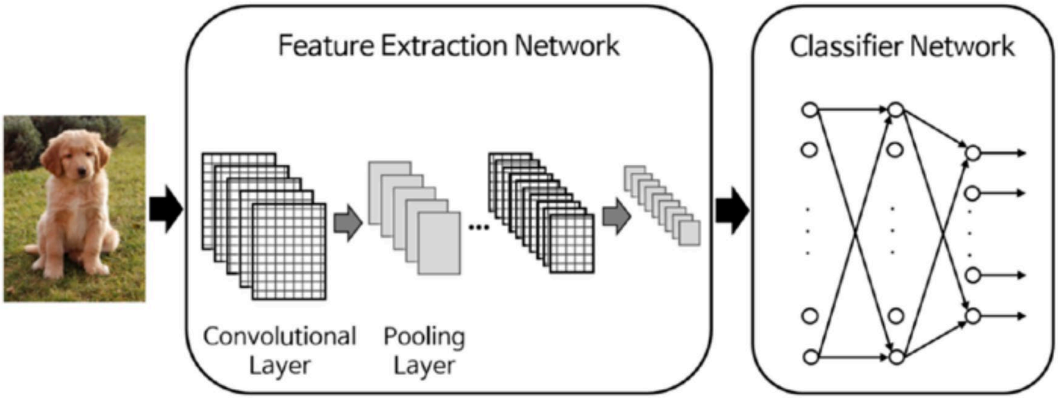

Figure 2. Typical architecture of CNN consisting of a feature extraction network and a classifier network. Source: [7].

An image is provided as input into the feature extraction network which is comprised of several convolution layers and pooling layers. The convolution layer generates new images called feature maps that emphasizes unique features from the original image by passing the image through trained convolution filters. The feature map that is created is then passed through an activation function [7]. An activation function is

added to the convolution layer to increase the non-linearity of the output. CNNs commonly use the Rectified Linear Unit (ReLU) activation function in the convolution layer [8]. ReLU activation functions are in place to resolve the vanishing gradient problem [9].

The vanishing gradient occurs when hidden layers that are close to the input layer are not properly trained due to backpropagation adjustments not reaching earlier hidden layers in a neural network. Backpropagation in a neural network is when errors are back propagated from the output nodes to the input nodes [7]. We will not be going into detail about backpropagation in this thesis. Further reading on backpropagation can be found in [7].

The ReLU function and its derivative are defined in Eq. (2.1) and Eq. (2.2), respectively. Figure 3 shows the ReLU function and its derivative. ReLU activation output is either zero, if the input x is negative, or the output equals x, if x is positive [9].

$$g(x) = max(0,x) = \begin{cases} x, & \text{if } x \geq 0, \\ 0, & \text{if } x < 0, \end{cases} \tag{2.1}$$

$$g'(x) = \begin{cases} 1, & \text{if } x \geq 0, \\ 0, & \text{if } x < 0, \end{cases} \tag{2.2}$$

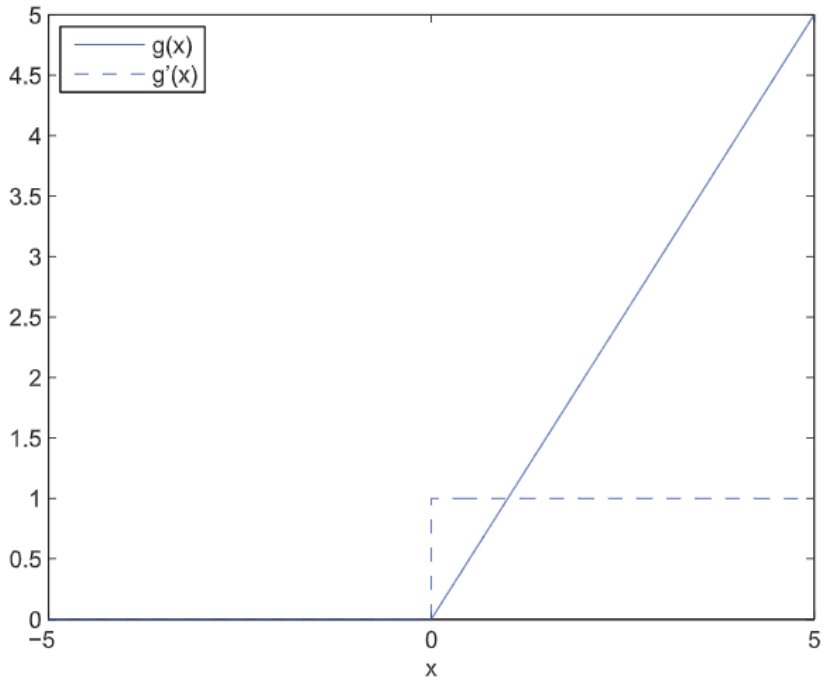

Figure 3. The Graphic Depiction of the ReLU Function and Its Derivative. Source: [9].

After the convolution layer, "the pooling layer reduces the size of the image, as it combines neighboring pixels of a certain area of the image into a single representative value" [7]. Figure 4 shows the representative value from pooling which can either be set as the mean or maximum of the selected pixels [7]. In Figure 4, the pixels are divided into four quadrants which create a 2x2 square of pixels. Mean pooling adds the value of the pixels in each quadrant and divides by the number of pixels in the quadrant. This results in one value for each quadrant, creating a 2x2 pixel product. The same steps are taken for maximum (max) pooling, but instead of taking the mean of each quadrant, the maximum value is taken to create the final 2x2 pixel product. The pooling process is beneficial for relieving the computational load and preventing overfitting [7].

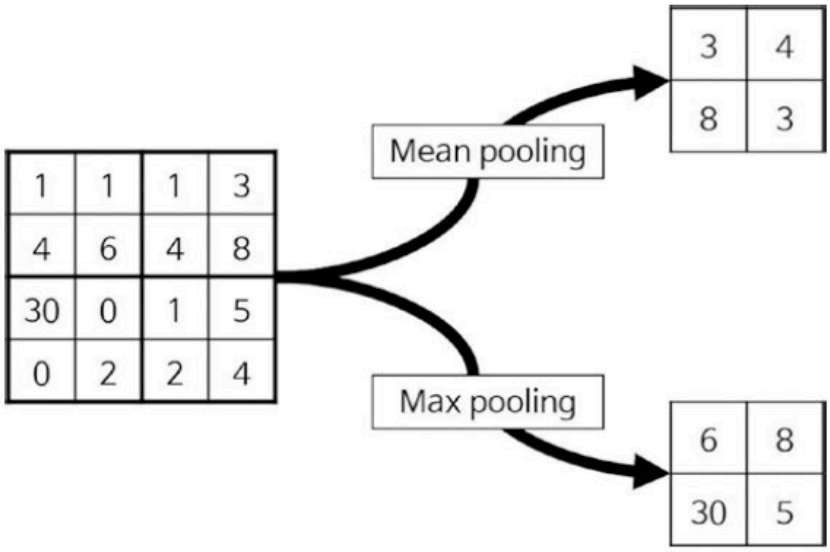

Figure 4. Depiction of the pooling layer. Source: [7].

Overfitting happens when models learn the specifics of a training data set too well. This causes the model to not be able to generalize to new datasets and is usually due to the limits of training data [10]. An overfit model will not perform as well to data it has not seen before. Furthermore, "complex models such as deep neural networks (DNN) can detect subtle patterns in the data but if the training set is too noisy or too small then the model is likely to detect patterns in the noise itself" [11]. Common solutions to overfitting include simplifying the model by selecting fewer parameters, gathering more training data, and reducing the noise in the training data.

The way we test how a model will generalize is by splitting the data into two sets, which are the training set and the test set. We train our model on the training set and we test using the test set [11]. Generalization error is the error rate from when we evaluate our model on the test data. If the training error is low but the generalization error is high, the model is overfitting the training data [11]. Regularization techniques reduce overfitting by constraining the model to make it simpler. Common regularization techniques for neural networks include early stopping, dropout, and data augmentation. Early stopping interrupts training when its performance on the validation set starts dropping, which in turn prevents overfitting [11]. Dropout regularization can potentially add up to 2% accuracy on state-of-

the-art neural networks, which significantly reduces the error rate [11]. During all training steps, every input neuron has a chance, with probability *p*, that it will not be considered during the current training step but could potentially be considered for future training steps (i.e., dropped); a commonly used value for *p* is 50% [11]. After training, the neurons are no longer dropped and there is a less chance of overfitting. Lastly, "data augmentation consists of generating new training instances from existing ones, artificially boosting the size of the training set" [11]. Performing this technique reduces overfitting [11]. Once pooling is completed, the output of the feature extraction network is the input to the classification network. This network contains fully connected layers that use the features given to classify the image into different classes based on a training dataset [8].

In Chapter III, we will provide further detail on the type of CNN used for this research and provide justification in experimental design specifics.

B. 5G NETWORK ARCHITECTURE: NSA vs. SA

The Third Generation Partnership Project (3GPP) standardized the 5G network which includes both SA New Radio (NR) and NSA NR [12]. NR is a new radio access technology based on orthogonal frequency-division multiplexing (OFDM). The 5G NR uses two main frequency ranges. The first includes 6 GHz frequency bands and below. The second includes bands in the mmWave range, which includes 20–60 GHz. The mmWave range enables 5G UWB networks [13].

The NSA and SA architectures are essentially different deployment modes of 5G. The MCAS Miramar energy communication network is based on the 5G NSA NR network. Explaining the distinctions between the two will provide clarity to the level of security vulnerability.

1. 5G NSA NR networks

The NSA architecture is based on dual connectivity between 4G and 5G networks. Essentially, the NSA is a 5G radio access network (RAN) that operates on the 4G LTE core, known as the Evolved Packet Core (EPC). More specifically, the NSA deployment requires that the 5G NR control plane be anchored to the 4G LTE EPC. This anchoring can

be seen on the left side of Figure 5. This dual connectivity was standardized in 3GPP in 2017 [12]. The NSA architecture is the first step to transition to 5G SA architecture and it is an important step because it creates a 5G network on an already stable and existing 4G LTE infrastructure [14]. This is important for MCAS Miramar since it alleviates the financial pressure of replacing the full 4G/LTE network already in place and enables leveraging the functioning existing network.

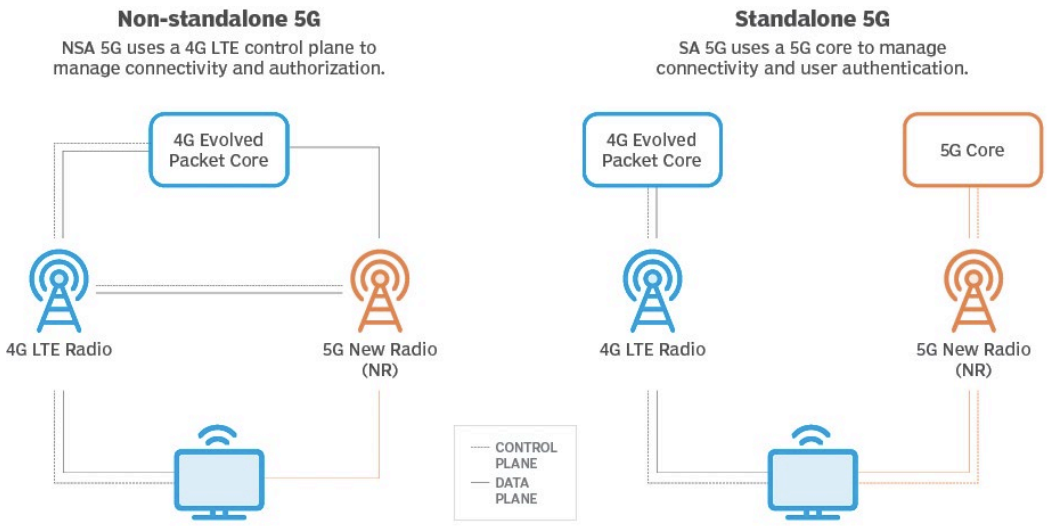

Figure 5. Depiction of 5G NSA vs. 5G SA. Source: [20].

However, the NSA also has its disadvantages. The 5G NSA cannot provide for certain capabilities in an SA 5G network. Several features and functions available in the 5G core network (5GC) are not available in the NSA network [14]. For example, the NSA does not enable low latency [15]. Another disadvantage is that the NSA requires a higher level of energy consumption [16]. In addition, the 5G SA takes advantage of key technologies such as Software Defined Networks (SDN) and Network Function Virtualization (NFV). The use of these functions entails a change in network infrastructure, elements, and various control components which are not compatible with the NSA architecture [17]. SDN and NFV will be explained further in the next section.

In terms of security, the 5G NSA network inherits all the vulnerabilities of the LTE network. Furthermore, common solutions to alleviate cyber threats, such as ML and artificial intelligence, are inherently unavailable in the NSA network.

2. 5G SA NR networks

The right side of Figure 5 shows that the SA architecture is a new 5G system based solely on the deployment of a 5G network that consists of NR and the 5GC [17]. 5G SA networks are designed to address the limitations of previous cellular network generations and provide more robust security [17]. The security comes in the form of new features and services such as cell densification, Multi-Access Edge Computing (MEC), network slicing, virtualization of the Radio Access Network (RAN) and 5GC. Cell densification serves "large numbers of users and new techniques such as beamforming to direct the wireless communication channel at users and reduce interference" [18]. The MEC is a cloud paradigm that pushes applications closer to the edge of the network to improve latency and enable high data rates in real time [17] [18]. Network slicing creates "multiple virtual networks that provide different quality of service levels over shared physical infrastructure" [18]. The virtualization of the RAN and 5GC dynamically scales the network functions [18]. Two virtual functions in the network are SDN and NFV, which are implemented for network splicing. "SDN abstracts the network control level from data transmission devices, allowing implementation in software" [17]. NFV virtualizes the network functions that serve as switches and storage devices, which improves efficiency and quality of service. Both functions make deployment of the network simpler, faster, and more flexible [17], [18], [19]. NFV and SDN enable slicing and the extension of additional tools into 5GC and RAN such as ML. ML increases network service efficiency by enabling dynamic allocation through network slicing. The ML inclusion in the 5G SA system helps in real-time source administration of the virtual network functions and the RAN [19].

3. 5G NSA Network at Miramar

The 5G NSA network at MCAS Miramar is limited when it comes to functions that are provided with a 5G SA network. The network will not have the implemented features such as the SDN or NFV. This means there are no ML solutions provided with the

implementation of the 5G NSA network. To address this limitation, this work will incorporate ML solutions to predict traffic flow anomalies on the Miramar 5G NSA network. Providing the network with an anomaly detection model will be crucial to maintaining a secure network.

C. LITERATURE REVIEW

To be well versed in utilizing ML on a wireless network, related work from other researchers must be reviewed. This section discusses selected relevant literature in the fields of anomaly detection, cyber security, ML, and wireless networks or systems.

In [21], the authors use a CNN for anomaly detection for 5G networks. Their idea behind detecting anomalies on a 5G network is to pair software defined security (SDS) with the CNN to provide end-to-end protection of the network. Unable to obtain 5G anomalous data, they use open source benign and malicious NetFlow data to test their image generator that converts network flows to images to input into the CNN. NetFlow data is flow records of IP traffic that moves between network devices. The results showed up to a 96% accuracy rate of the CNN detecting anomalies [21]. Related, [22] lists other preprocessing techniques for processing input data and converting it for the CNN. Two popular techniques mentioned are the normalization of input data which makes the data standardized and able to feed into the CNN and Vector-to-Matrix which converts the data to form a matrix [22].

In [23], the authors introduces a layered approach to threat modeling on a 5G network. Threat modeling is defined as an approach to identify security requirements, potential threats, vulnerabilities, and prioritizing remediation methods. The authors detailed different attack vectors affecting each layer of the network stack. They also categorized the attacks in terms of type, impact, affected components, affected layer, and entry point [23].

In [24], the authors present a methodology that generates anomaly detection datasets in Industrial Control Systems (ICS) to evaluate cyber anomalies using machine and deep learning using a four-step approach. They successfully created the Electra dataset and employed different supervised and semi-supervised machine learning models

including Random Forest, Support Vector Machine, Deep Learning, and Dense Neural Networks, showing they can be used in an industrial environment [24].

In [25], the authors generate a 5G dataset inspired from 3GPP specifications for a 5G network. To test their ML model, the authors injected anomalies into the dataset such that they represented unexpected increases in data traffic in a designated remote radio unit (RRU) cell. The author represented these traffic/data spikes as either a 1 or 0 [25]. In this thesis, we use a similar approach in creating our anomalous data. We will further discuss the approach of defining anomalous packets in Chapter III.

As the research shows, there are different methods of developing preprocessing techniques for CNN inputs as well as different ways to create anomalous data and categorizing 5G threats. In the next chapter, we will discuss how we obtained the necessary 5G data to train our CNN and how the data set was manipulated to create anomalies. We will also detail the preprocessing technique used to input the data into our CNN model.

THIS PAGE INTENTIONALLY LEFT BLANK

III. EXPERIMENTAL DESIGN

In this chapter, we discuss the process of obtaining the 5G benign dataset collected from the AT&T 5G cellular tower at the NPS SLAMR facility. We then describe how we created our own anomalous dataset from the benign dataset. We conclude the chapter by explaining the preprocessing technique used to input into the CNN followed by the design of the CNN model.

A. DATASET

As was discussed in Chapter I, 5G data flow on the Miramar network has not yet begun. To obtain a dataset on which to train our algorithm, we developed a process to collect data on the newly established NPS 5G network, serviced by AT&T. We were able to manipulate the collected data collected to produce our own anomalous data. To begin this experimental collection at the SLAMR facility, we required the 5G cellular tower to be operating at 5G+ speeds. To measure the speed of the upload and download times, we used the speed test created by Ookla [26].

1. Data Collection Process

The first step in the data collection process was to establish that our data be in packet capture (PCAP) form. The reason for the PCAP form is because the CNN can potentially be placed to monitor and collect the wireless network traffic flowing between the DER and the EMS. We then positioned ourselves within range of the NPS 5G cellular tower to gain wireless connection to our devices. A Linux based laptop was then directly connected to a Netgear Nighthawk router via ethernet cord, and a Mac operating system (OS) laptop was connected to the NPS 5G wireless network. The Netgear Nighthawk we used is a 5G/5G+ mobile hotspot device configured to run the wireless gateway from the cellular tower to the TMG Core Edgebox 4.5 located on the NPS campus. The TMG Core Edgebox is a high-density, two-phase liquid immersion cooling data center with the purpose of handling high performance computing [27]. The Edgebox can store the wireless network traffic being transferred from the NPS cellular tower located at the SLAMR facility while monitoring and decrypting the traffic in real time.

We simulated PCAP files being transferred from the Linux OS laptop that is connected to the Nighthawk to the datacenter. In conjunction with the file transfer, the Mac OS laptop was monitoring the traffic flowing into a designated node in the datacenter. A variety of PCAPs were chosen to try and replicate the potential wireless dataflow that will occur at MCAS Miramar. The open source PCAPs we used were ICS normal traffic [28], MODBUS ICS traffic with various cyber-attacks/anomalies [29], and Building Automation and Control network (BACNET) traffic [30]. We separated the PCAP files into three folders that represented the different open-source sites and ranged in file sizes. The folder names, associated sizes, and content are as follows: p1 with a size of 209.2 MB and contains ICS normal traffic, p2 with a size of 4.48 MB and contains MODBUS ICS traffic with various cyber-attacks/anomalies, and p3 with a size of 479 KB and contains BACNET traffic. The original intention of collecting these specific PCAPs was to find a way to replay the content over the NPS 5G network to simulate the recorded anomalies and cyber-attacks. The attempt at finding a way to replay the PCAPs failed due to specific clients and services in the PCAP trace not being available on the 5G network at the time of experimentation. As a result, we used the *rsync* command in the Linux OS terminal which allows the transfer of folders from local directories to remote directories. In our set up the local directory is the Linux OS and the remote directory is the Mac OS logged into the datacenter. It is important to note that the *rsync* command does not read the content of the folders/files being transferred. It transfers all the data from one place to another, meaning the PCAPs sent will not reflect their content. Any folder/file could have been used for this transfer and produce the same results.

Once *rsync* is executed and progress of the data transfer is tracked, to physically capture the traffic coming into the datacenter, we used the network program analyzer *tshark*. The *tshark* command allows us to capture specific traffic coming into the designated node on the EdgeBox. We repeated the transfers of all three folders multiple times to collect a large amount of data to train our CNN. Figure 6 shows the experimental data path from the Nighthawk to the EdgeBox. This is the data path used for data collection. The simulated generator labeled in the figure represents the Linux OS laptop that stored p1, p2, and p3.

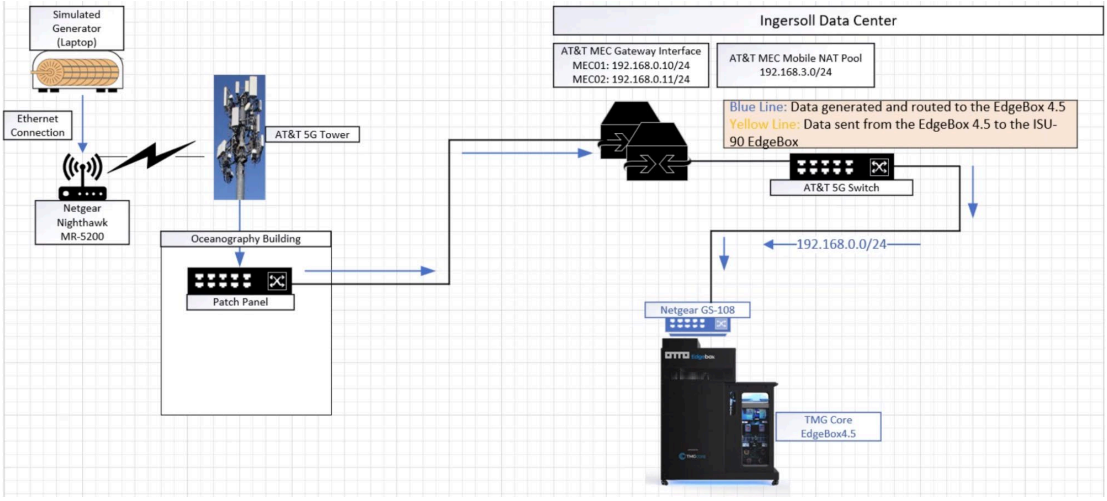

Figure 6. Experimental Data Path from Nighthawk to EdgeBox

2. Creating the Anomalous Dataset

The raw data collected was in PCAP form. Reading the PCAP into memory with *tshark* allowed for the extraction of the fields (features) within the PCAP. For each packet in the PCAP, the features extracted were the source IP, destination IP, protocol, TCP acknowledgment number, TCP sequence number, TCP destination port number, TCP source port number, TCP flags, TCP length, TCP time delta, TCP checksum, TCP header length, TCP PDU size, TCP time relative, TCP window size, IP length, IP identification number, and IP header length. These 18 features were chosen based on how likely these same features would be present in the data capture at the the MCAS Miramar 5G network. The extracted fields can then be written to a convenient file format, such as a csv file. The command used to generate the features with *tshark* is shown in Figure 7.

```
(base) C:\Users\hatha>tshark -r C:\Users\hatha\OneDrive\Documents\AshThesisFolder\pcap_1\pcap_one\pw1 -E header=y -E separator=, -T fields -e ip.src -e ip.dst -e ip.proto -e tcp.ack -e
tcp.seq -e tcp.port -e tcp.srcport -e tcp.flags -e tcp.len -e tcp.time_delta -e tcp.checksum -e tcp.hdr_len -e tcp.pdu.size -e tcp.time_relative -e tcp.window_size -e ip.len -e ip.id -e
ip.hdr_len > C:\Users\hatha\OneDrive\Documents\AshThesisFolder\csv\pw7.csv
```

Figure 7. Feature Generator Command

Once the benign data features were collected and written to a csv file, we then needed to ensure the data was in a form amenable to ML classification techniques. In the literature, we found a simple method to achieve this goal. In [25], the author simulated anomalous data by adding an additional column to the features and labeled each row with either a 1 or a 0 (where a 1 represents anomalous data and 0 represents non-anomalous data). This is a reasonable approach for our data labeling due to the fact that we currently do not know what anomalies will be on the MCAS Miramar network or what patterns they will display across the network. To implement this approach, the author used the original data from their own selected features to generate a csv file and added a column of either a 1 or 0 [25]. The method we used to generate our dataset involves a similar idea from [25], but instead of adding a 1 or 0 to a csv, we used a script that takes advantage of the Python library to read in the data from the *tshark* features collected and label the data directly in memory with either a 1 or 0. Since there are so many unknowns with the MCAS Miramar network, the labels 1 or 0 were randomly assigned throughout each prospective dataset. After labeling, the data can then be directly passed to further ML pre-processing modules. This approach allows us to dynamically generate datasets with various percentages of anomalous labels. Datasets with 10%, 25%, 30%, 40%, and 50% anomalous data were generated with this approach.

B. CNN DESIGN

Our CNN design is comprised of a preprocessing technique that will allow an image input into the CNN and the image classification model from the TensorFlow implementation of Residual Network (ResNet) 50.

1. Preprocessing Technique Methodology

The preprocessing technique of the CNN is in place to create a way to provide an input to the CNN. In [31], the authors used a correlation matrix, referred to as an SC matrix

in [31], to convert NetFlow data into images. The NetFlow data collected in [31] was converted into a csv file, where every row in the NetFlow sample file is called a record. The author then used a correlation matrix to evaluate the comparisons among the features selected in the NetFlow data. In a SC matrix, all features are given a numeric value and are placed in a sequential order, which gives the matrix the ability to compare correlations between features with decimals ranging from -1 to 1 [31]. Figure 8 shows the correlation matrix described in [31].

The original NetFlow features are represented at the top of Figure 8 as the orange blocks where x_0 is the first feature, x_1 is the second feature and so on, with x_n reflecting the last feature. For every feature, the correlation between itself and every other feature is calculated and is denoted as the anchor feature. The anchor feature is the orange block located in the SC matrices box at the center of each 3 by 3 matrix in Figure 8. An example of how the 3 by 3 matrix is derived in Figure 8 is noting x_{01} (white block) is the calculated correlation between feature 0 (the anchor feature) and feature 1. The white blocks surrounding the anchor feature are called correlation values. The SC matrices are comprised of each individual feature along with the correlation values to create a 3 by 3 matrix. The top eight features as well as the anchor feature correlation values are arranged in a 3 by 3 matrix, with the anchor feature having a value of 1 and placed in the center of the matrix. Once all 3 by 3 matrices are computed, they are then concatenated together to create a 3 by 3(n) matrix, where n represents the number of features. The values in that matrix are then computed as pixel values in red, green, blue (RGB) color map using a simple python color mapping function that is included in the matplotlib library in Python. The resulting image is a 3 by 3(n) size pixel image for each NetFlow record, which is an adequate input for a CNN.

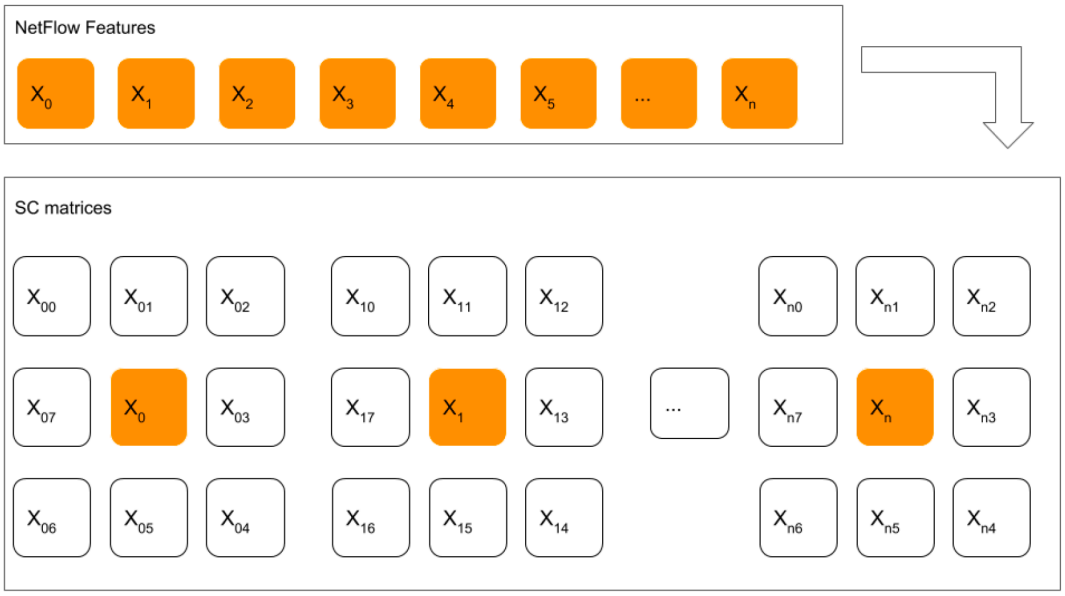

Figure 8. An example of correlation matrix as discussed in [31]. Source: [31].

2. Preprocessing Technique with Image Results

We used the same preprocessing technique as in [31], but in our case we used individual packets vice the NetFlow data. For all the packet captures that we obtained at the NPS 5G site, we merged the p1, p2, and p3 PCAPs into one large capture. Then, for each packet within the combined capture we transformed the packet into an image utilizing the methodology in [31]. A minimum of nine features is needed for the method in [31] to work. In our work we used 18 features. The 3 by 3 matrices were computed and were concatenated together which yielded a 3 by 54 (3 by 3 x 18) matrix. The values in the 3 by 54 matrix were used as pixel values in a RGB color map using the same python library method previously discussed, resulting in an image of size 3 by 54 pixels for each sample. The resulting images, shown in Figure 9, are images generated from two anomalous packets and two benign packets. The two anomalous packets have different image patterns from the two benign images. However, when compared together, both anomalous images exhibit similar image patterns, similarly when comparing both benign images.

The images were then sorted into directories we created which were labeled anomalous or non-anomalous (benign). We created a directory tree structure for every

percentage of anomalous dataset tested. This tree structure is shown in Figure 10. Each directory is comprised of training, validation, and testing subdirectories. For each of the training, validation, and testing subdirectories, there are two additional subdirectories labelled anomalous and benign. The anomalous and benign images are sorted into the appropriately labeled subdirectories. Each image data is then split among the directories as follows: 60% of the data will be used to train, 20% of the data will be used to validate, and 20% of the data will be used to test. The reason for splitting the dataset is to prevent overfitting of the CNN model. The training dataset is for the CNN to learn the parameters of the difference between anomalous and benign images. The validation dataset evaluates the model after each epoch of training on data that was not used during the initial training. The test dataset is the final dataset that has never been used in training and is used to evaluate the final version of the CNN model.

Each directory was then passed into a dataset generator from the Tensorflow Library, where it pulls labels for each dataset from the name of the subdirectory the images were located. The Tensorflow dataset generator performs various processing methods on the images, including flipping various images, shearing various images, and using a processing method optimized for images destined for use in a ResNet50 model. The dataset generators output the processed images in tensor format with their associated label, and are ready for use in either training, validation, or testing of a ResNet50 model. Tensor format is a represented matrix or array that hosts all the data [32].

The code for the image generator can be found in Appendix A. The code for the training, validating, and testing datasets can be found in Appendix B.

Figure 9. Generated Images of Anomalous vs. Benign data. Both anomalous images exhibit similar image patterns, similar to that of both benign images.

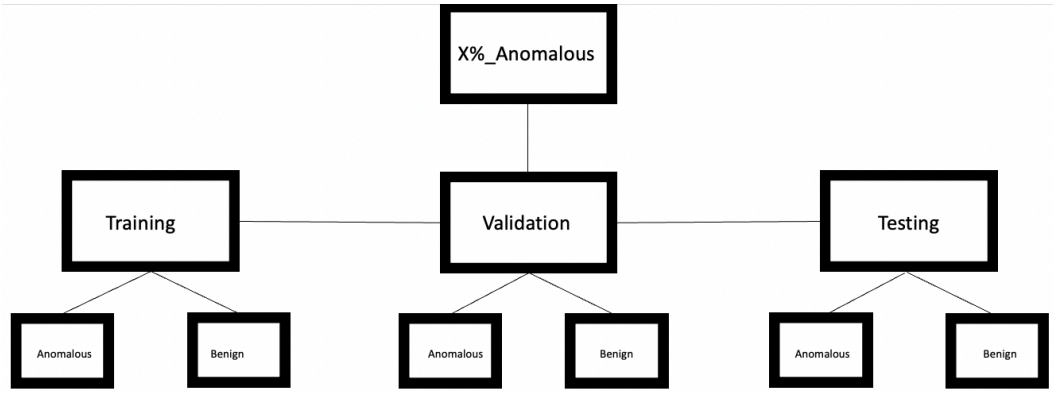

Figure 10. Directory tree structure layout for every percentage of anomalous dataset tested.

3. TensorFlow's ResNet 50

The programming language we used to implement our CNN is Python. More specifically, we used the free open-source library TensorFlow. "TensorFlow is an end-to-end open-source platform for ML that has tools, libraries, and community resources that lets researchers push the state-of-the-art in ML and developers easily build and deploy ML powered applications" [32]. In the open-source library, over 300 image classification models are available [33]. In this research, we use ResNet 50, which is an established image classification model for basic image processing that includes 50 layers, which includes convolution layers and the fully connected layer. The ResNet models use skip connections which means the input image feeding into a layer is added to the output of the layer; this is known as residual learning [11]. Residual learning makes it feasible to train on deep networks without losing any information during training and boosts the performance of the system [34]. The residual connection advantages of the ResNet 50 architecture are that the

skip connections keep the gained knowledge during training and speeds up the training timeline of the CNN model by increasing the size of the network [34].

Figure 11 depicts a skip connection and residual training. The right half of Figure 11 depicts the example of residual learning. The objective of training a neural network is to make it model a target function, which is represented in Figure 11 as $h(x)$ [11]. When an input of x is added to the output, the network is forced to model a new function labeled $f(x)$, which enables the skip connection [11].

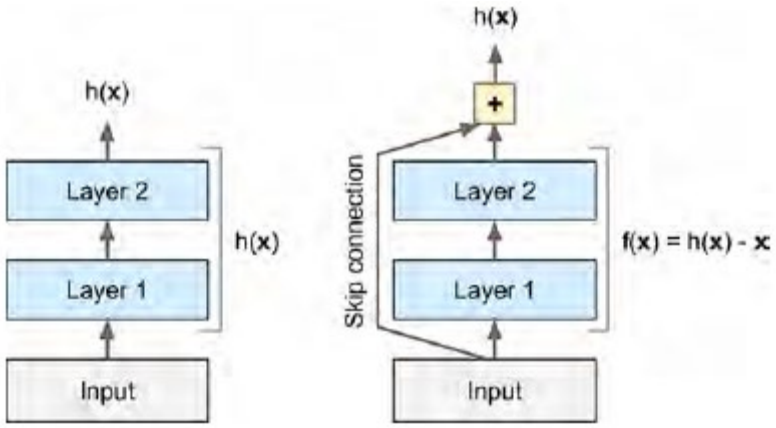

Figure 11. Example of Residual Learning and skip connections. Source: [11].

Each ResNet 50 mode has residual units that include three convolutional layers, with batch normalization and ReLU activation functions, which means that the connection skips three consecutive blocks. This is shown in Figure 12 using the colors peach, red, blue, and yellow. Batch normalization is a procedure used in conjunction with an activation function that standardizes a collection of inputs to the model [11]. Within the residual block, the three convolution layers are stacked with the following dimensions: 1x1, 3x3, and 1x1. The purpose of these convolution layers is to reduce the dimensions and calculate the feature map. The pooling layers do not occur within the residual layer, but outside of the residual block, which is depicted in Figure 12.

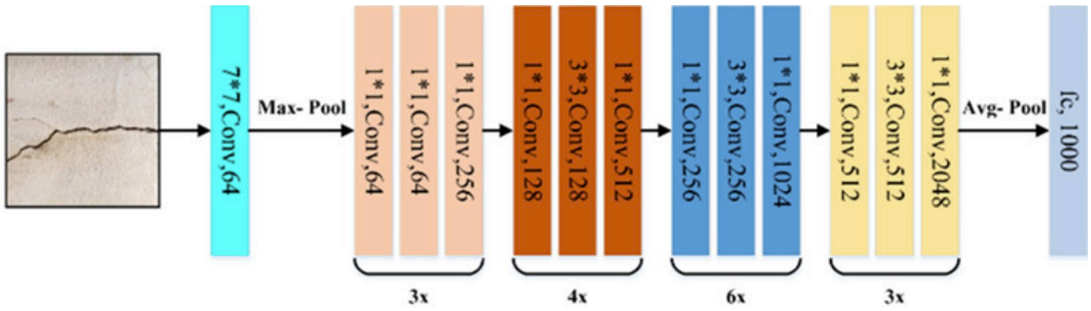

Figure 12. ResNet 50 Architecture. Source: [35].

4. Characteristics of the CNN Model

Figure 13 shows our CNN model, which include TensorFlow's pre-trained ResNet 50 model. In row two of Figure 13, labeled input_2, a four-dimensional array is always given as input to a CNN. The input shape includes batch size (the number of images), height, width, and depth (RGB layers). The input layer has a shape of 224x224 pixels and the 3 RGB channels. The 224x224 pixel size is required since the pre-trained ResNet 50 model was built with a 224x224 input layer shape. The batch size is labeled 'none' due to the network not knowing the batch size in advance [36]. Our data preprocessing modules ensure the images used for training/validation/testing of the model will have a 224x224x3 shape which is needed to feed into the input layer of the model. Immediately following the input layer, in row three of Figure 13, labeled resnet50, a pre-trained ResNet 50 model is added to our architecture. The pre-trained model weights have already been optimized for image classification tasks. The pre-trained model is trained on images from the ImageNet database. The output of the last convolutional block of ResNet 50 layer is a 7x7x2048-dimensional array used as an input for the fully connected (dense) layer, which will be used to classify the images in our dataset. Regarding the functionality of each of the layers in our model, the 'none' in the ResNet 50 layer refers to random initialization, which means random numbers are used to initialize the weights. To combat overfitting, a global average pooling layer, in row four of Figure 13, is added after the ResNet 50 layers, followed by a dropout layer (shown in row five of Figure 13). The pooling layer serves to combat model overfitting by reducing the dimensionality to two dimensions and the number of trainable parameters of the model. Dropout provides for model regularization. Lastly, the dense

layer, in row six of Figure 13, is added to the architecture. The fully connected layer is used for classification of two classes, anomalous or non-anomalous. The code for the CNN model implementation can be found in Appendices C, D, and E.

```
Number of layers in the base model:  175
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         [(None, 224, 224, 3)]     0
_________________________________________________________________
resnet50 (Functional)        (None, 7, 7, 2048)        23587712
_________________________________________________________________
global_average_pooling2d (Gl (None, 2048)              0
_________________________________________________________________
dropout (Dropout)            (None, 2048)              0
_________________________________________________________________
dense (Dense)                (None, 2)                 4098
=================================================================
Total params: 23,591,810
Trainable params: 4,098
Non-trainable params: 23,587,712
```

Figure 13. Characteristics of the CNN model developed in this thesis.

5. Summary of Training Process

In summary, our experimental design allowed the capturing of benign PCAPs on the NPS 5G network. Utilizing the benign data, we manipulated it to create several different percentages of anomalous datasets while making the data an agreeable image input for the CNN by using a correlation matrix preprocessing technique. For each packet, 18 features were extracted with *tshark* and used to create the correlation matrix. The correlation values within the matrix were then used as pixel values and an image was created. Every packet resulted in a single image. Based on the percentages of anomalies we inputted, those images were then sorted into a directory structure, which were then passed into TensorFlow's image generator that produced training, validating, and testing datasets. These datasets were then used for training and testing the CNN model. The model is

designed to take batches of images from the datasets to push through the CNN model to be classified into either benign or anomalous labels.

IV. RESULTS AND ANALYSIS

This chapter discusses the results and analysis of the experiments conducted on the 10%, 25%, 30%, 40%, and 50% anomalous datasets that were created. The chapter begins by explaining the metrics used to visualize performance of the CNN model, then compares all graphical representations with the discussion of how effective the CNN is for this work. The chapter concludes with providing recommendations for deployment on the energy communications network at Miramar.

A. GRAPHICAL REPRESENTATIONS AND METRICS FOR MEASURING CNN PERFORMANCE

Three different graphical representations were used on the datasets. The first visual used was the confusion matrix, the second a learning curve plot, and finally a t-distributed stochastic neighbor embedding (TSNE) plot.

1. Confusion Matrix

The confusion matrix is a binary classification method that is used to show model performance [37]. The matrix cross classifies predicted and actual values to evaluate the quality of the classifier [37]. Figure 14 depicts how a general confusion matrix is set up with the following terminology: true positive (TP), false negative (FN), false positive (FP), and true negative (TN). TP means the model correctly predicted a positive value and a TN means the model correctly predicted a negative value [37], [38]. FP means the model incorrectly predicted a positive value and a FN means the model incorrectly predicted a negative value [37], [38]. The positives will be categorized as anomalous and the negatives as benign.

		Predicted Label	
True Label	Positives (Anomalous)	TP True Positive	FN False Negative
	Negatives (Benign)	FP False Positive	TN True Negative
		Positives (Anomalous)	Negatives (Benign)

Figure 14. Example layout of a confusion matrix and its associated labels.

To measure performance, the following metrics were used: recall, precision, F1 score, and accuracy. Recall is the ratio of TP values predicted to the sum of TP values and the total number of FN numbers. Recall measures the predicted positive rate [38], [39]. Recall is shown in Eq (4.1) [38]. Precision is the ratio of TP values to the sum of TP values and FP values. Precision measures the percentage of how often the model classifies positive predictions as a TP [39]. Precision is shown in Eq (4.2) [38]. The F1 score is the ratio of the product of precision and recall to the sum of precision and recall, multiplied by two [38]. The F1 score combines precision and recall, which compares the performance of the classifiers. The F1 score is shown in Eq (4.3) [38]. Accuracy is the ratio of the number of correct predictions to the total number of predictions [39]. Accuracy measures the number of predictions the model classified correctly and is shown in Eq (4.4) [39].

$$Recall = \frac{tp}{tp + fn} \tag{4.1}$$

$$Precision = \frac{tp}{tp + fp} \tag{4.2}$$

$$F1\ score = 2(\frac{(Precision)(Recall)}{Precision + Recall}) \tag{4.3}$$

$$Accuracy = \frac{\#\ of\ \ correct\ predictions}{\#\ of\ \ total\ predictions} \tag{4.4}$$

2. Learning Curve Plot

The learning curve plot is a visual technique to observe how well the model does with the effects of adding training data. The learning curve plots both the training and validation loss in conjunction with the training set size [40]. The y-axis of a learning curve is the loss value, which evaluates how the model is learning; higher numbers indicate more learning. The x-axis represents epochs. In a learning curve the risk of overfitting increases if too small of a dataset it used, meaning the training loss will be low and the validation loss will be high [40].

3. TSNE Plot

A TSNE plot is a non-linear embedding algorithm that preserves points within clusters [40]. The TSNE reduces dimensionality, which allows the separation of data in clusters. It is used for visualization purposes and is separated by colors [11]. The TSNE plot interprets the feature maps from different levels of a neural network and makes the classification viewable to the human eye. The TSNE plot generated from our model differentiates between what the model classifies as benign data and anomalous data.

B. DISCUSSION OF FINDINGS

The following section analyzes the results of running the 10%, 25%, 30%, 40%, and 50% anomalous datasets through the CNN model. These results represent how well the CNN performed with the created datasets. The FN rates are of great importance in the evaluation of the model; a FN indicates that the model predicted a packet was benign when it was actually anomalous.

1. 10% Anomalous Dataset

As previously mentioned, the positive label is represented as anomalous data and the negative label is represented as benign data. The learning curve in Figure 15 shows a typical validation loss vs. training loss trend during model training. After the first few

epochs, training loss was consistently lower than the validation loss. The learning curve did not indicate that severe overfitting was occurring, apart from a random spike in validation loss at epoch 4. The validation loss curve was still trending downward with a very slight slope at the end of training. Further epochs would have likely resulted in severe overfitting due to the validation loss curve leveling out.

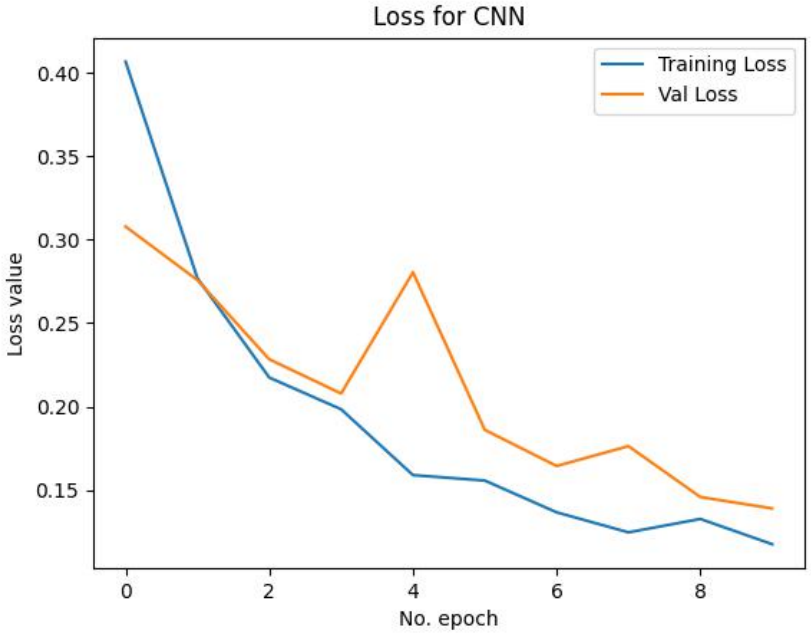

Figure 15. A typical learning curve of the 10% anomalous dataset. No severe overfitting occurred apart from a random spike at epoch 4.

The confusion matrix in Figure 16 shows that 96% of anomalous packets were incorrectly classified as benign. For the benign packets, 97% of the packets were correctly classified as benign.

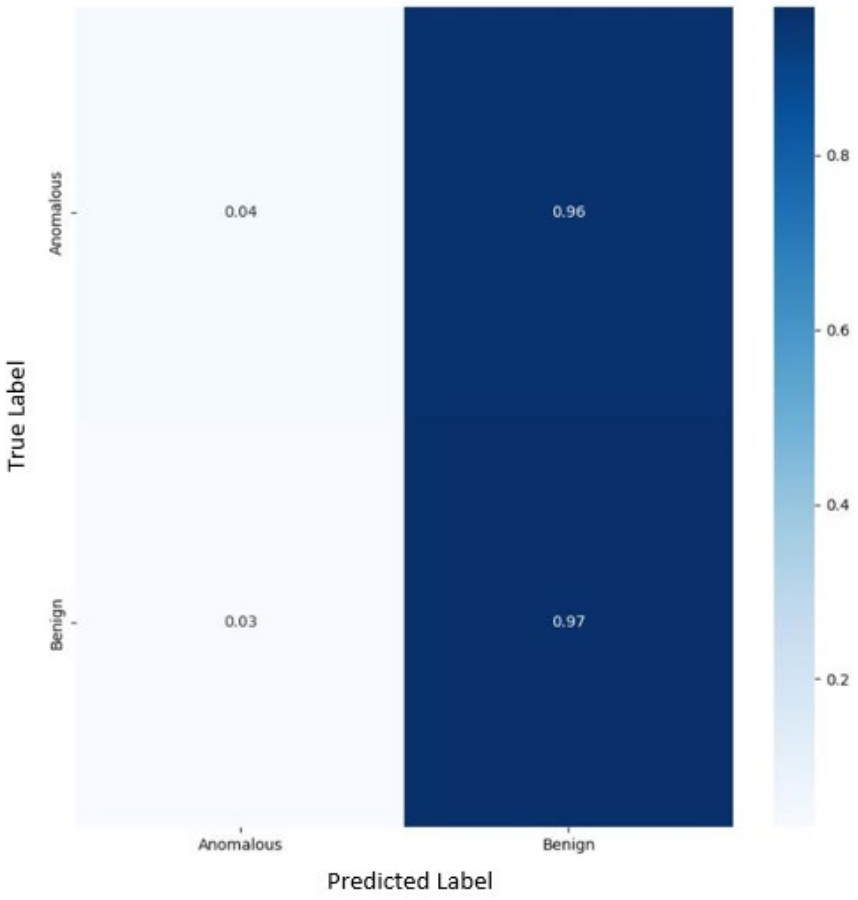

Figure 16. Confusion Matrix with Anomalous Data at 10%; 96% of anomalous data was incorrectly labeled.

The performance metrics in Table 1 indicate that the model is 87% accurate at identifying the correctly predicted labels. Despite the relatively high accuracy percentage (87%), and high TN rate (97%) achieved by the model, the model did not perform well. The rate of FNs was 96%, meaning that for all the anomalous packets processed by the model, only 4% of the anomalous packets were classified as anomalous. The high TN rate and high accuracy score achieved by the model is an artifact of the distribution of anomalous packets in the training dataset. The bulk of the training data were packets labelled as benign; thus, the model was heavily skewed towards classifying most data as benign and was not provided enough anomalous packets to learn the discerning patterns present in anomalous packet data.

Table 1. Performance metrics on the 10% anomalous dataset: precision, recall, f1-score, and accuracy. CNN model is 87% accurate at identifying correctly predicted labels. Model did not perform well due to FN rate of 96%.

	Precision	Recall	F1-score
Anomalous	.111	.038	.057
Benign	.900	.966	.932
Accuracy			.873

As expected from the data above, the 10% anomalous TSNE plot in Figure 17 shows a large cluster and a few smaller clusters of benign data. The anomalous data is sporadically spread throughout the plot. Most of the benign data is represented in a large ball meaning the model was able to classify the benign data more easily than the anomalous data. The anomalous data was spread out and not shown clustered together meaning the model had a hard time classifying the anomalous data.

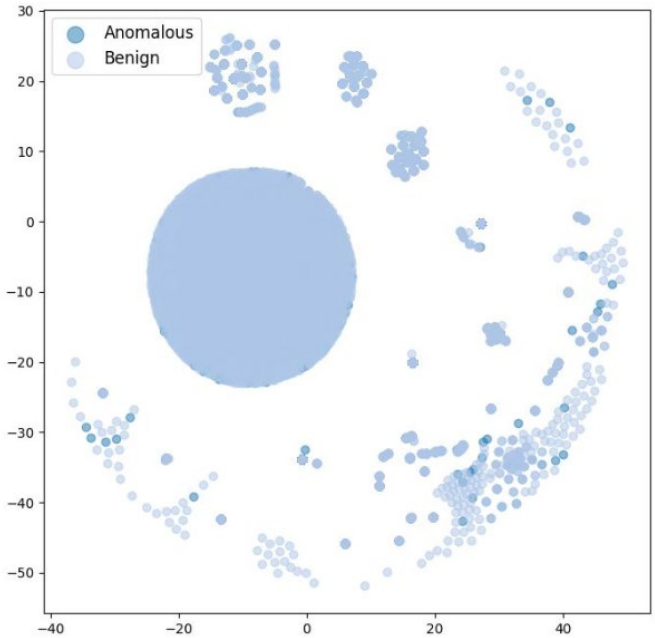

Figure 17. TSNE plot for 10% anomalous dataset. Large and small clusters of benign data but anomalous data shows no clustering.

2. 25% Anomalous Dataset

The learning curve in Figure 18 shows a typical validation loss vs. training loss trend during model training. After the first few epochs, training loss was consistently lower than the validation loss. The learning curve indicates that overfitting occurred. After epoch 8, the validation loss started to exhibit a slight upward trend while the training loss curve continued to decrease in value.

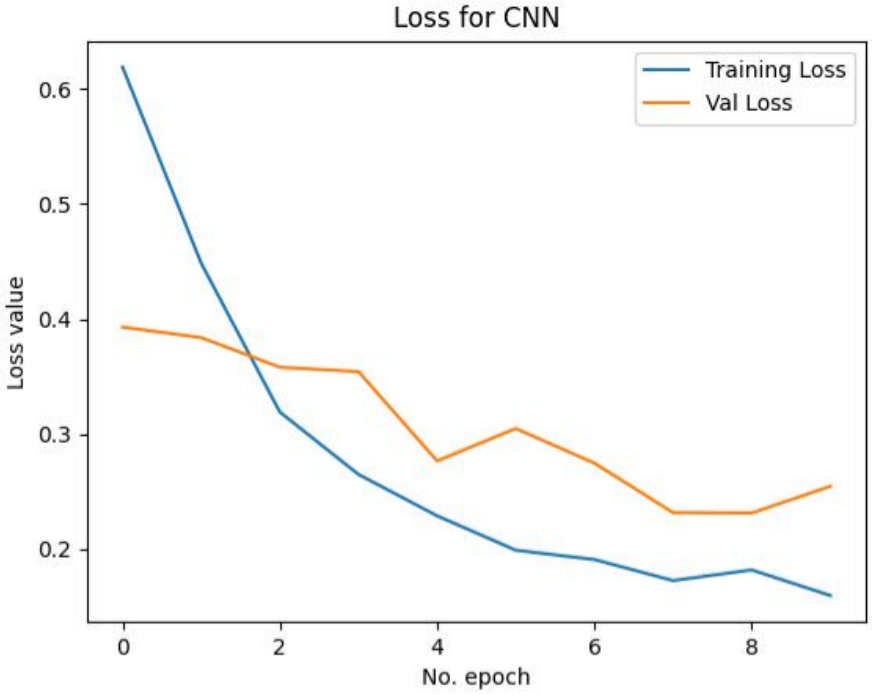

Figure 18. A typical learning curve of the 25% anomalous dataset. Overfitting is occurring due to the upward slope of the validation loss curve after epoch 8.

The confusion matrix in Figure 19 shows that 90% of anomalous packets were incorrectly classified as benign. For the benign packets, 90% of the packets were correctly classified as benign.

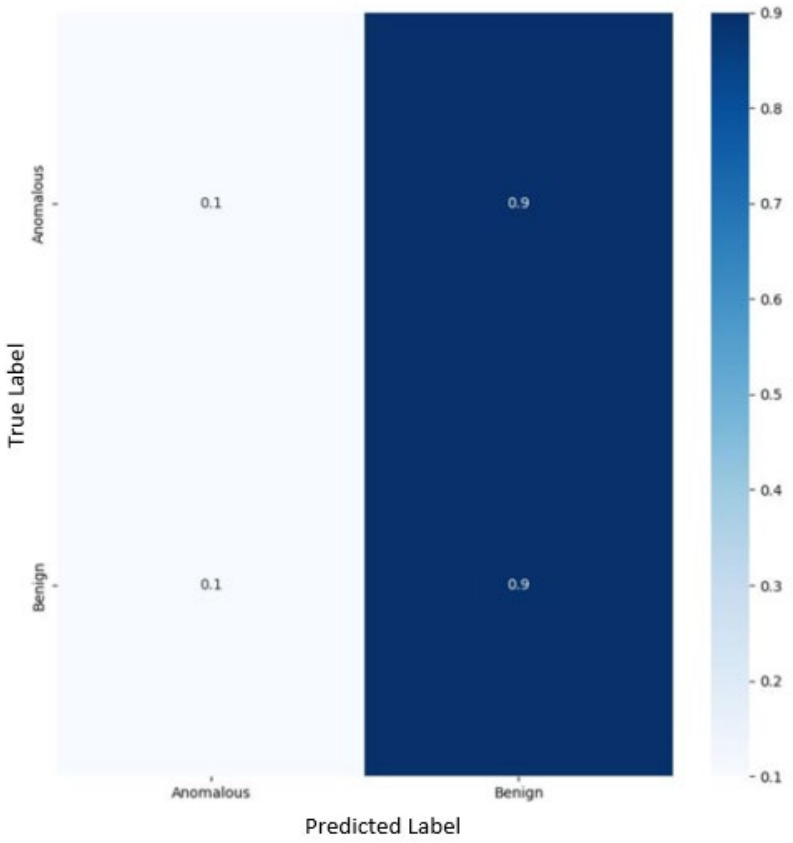

Figure 19. Confusion Matrix with Anomalous Data at 25%; 90% of anomalous data was incorrectly labeled.

The performance metrics in Table 2 indicate that the model is 68% accurate at identifying the correctly predicted labels. Despite the decent accuracy percentage (68%), and high TN rate (90%) achieved by the model, the model did not perform well. The rate of FNs was 90%, meaning that for all the anomalous packets processed by the model, only 10% of the anomalous packets were classified as anomalous. The high TN rate and decent accuracy score achieved by the model is similar to the case of the 10% anomalous dataset. The under representation of anomalous data causes the model to be skewed toward classifying anomalous packets as benign.

Table 2. Performance metrics on the 25% anomalous dataset: precision, recall, f1-score, and accuracy. CNN model is 68% accurate at identifying correctly predicted labels. Model did not perform well due to high FN rate of 90%.

	Precision	Recall	F1-score
Anomalous	.282	.101	.148
Benign	.726	.903	.805
Accuracy			.682

The TSNE plot for 25% anomalous data is shown in Figure 20. The figure still shows a large cluster and a few smaller clusters of benign data, but with a higher amount of anomalous data present throughout the smaller groupings of benign data. The anomalous data is more visibly clustered together within the benign data compared to the 10% anomalous data TSNE plot. This means the model has classified more anomalous data but does not take the FNs into account.

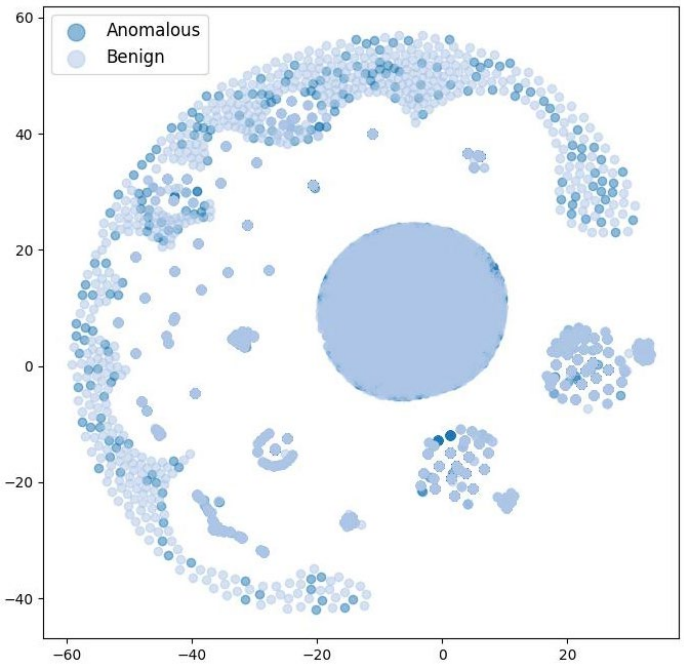

Figure 20. TSNE plot for 25% anomalous dataset. Benign data clustering is present and anomalous data is more visible.

3. 30% Anomalous Dataset

The learning curve in Figure 21 shows a typical validation loss vs. training loss trend during model training. The plot was similar to that of the 10% anomalous dataset model. After the first few epochs, training loss was consistently lower than the validation loss. The learning curve did not indicate that severe overfitting was occurring; validation loss was still trending downwards at a very gradual rate. Further epochs of training would have likely caused the plateau and/or increase in validation loss indicating overfitting.

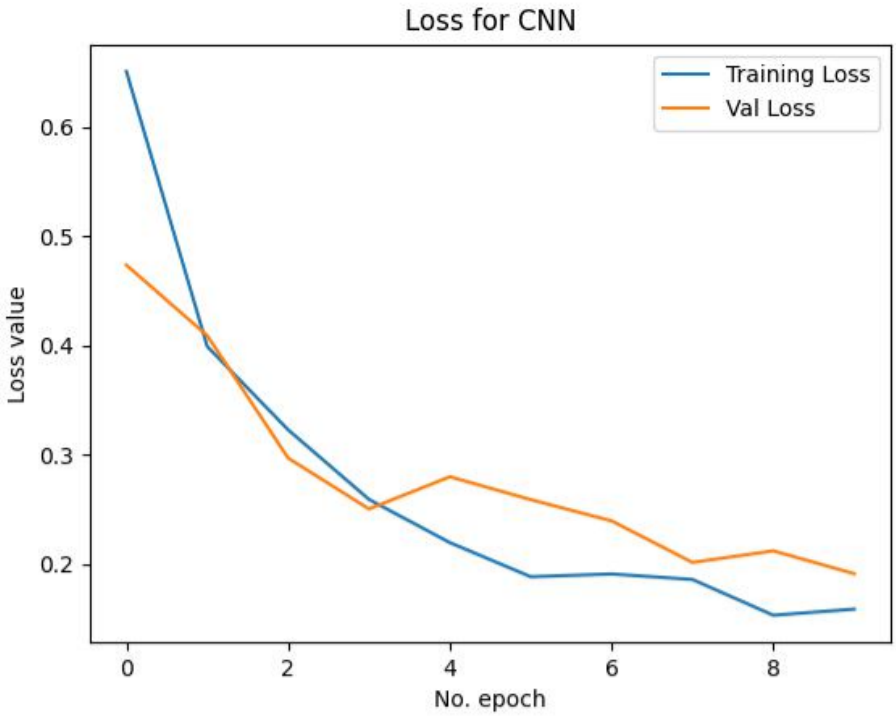

Figure 21. A typical learning curve of the 30% anomalous dataset. No severe overfitting occurred. Validation loss is gradually trending downward.

The confusion matrix in Figure 22 shows that 67% of anomalous packets were incorrectly classified as benign. For the benign packets, 67% of the packets were correctly classified as benign.

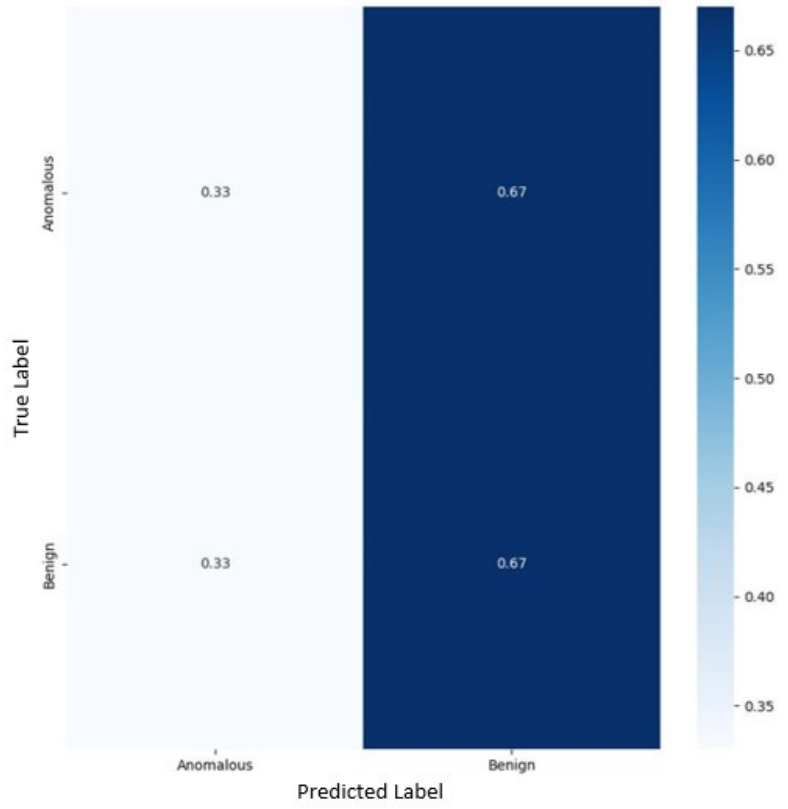

Figure 22. Confusion Matrix with Anomalous Data at 30%; 67% of anomalous packets were incorrectly classified as benign.

The performance metrics in Table 3 indicate that the model is 57% accurate at identifying the correctly predicted labels. The model was mediocre in its performance, with an accuracy that was only slightly better than a random binary guess. Additionally, the rate of FNs was 67%, meaning that for all the anomalous packets processed by the model, only 33% of the anomalous packets were classified as anomalous. While the metrics for the model are not great, the model did perform better than the 10% and 25% anomalous dataset models. This is likely due to the increased number of anomalous packets in the dataset; the model had more anomalous training samples and thus was able to better differentiate the anomalous packet patterns.

Table 3. Performance metrics on the 30% anomalous dataset: precision, recall, f1-score, and accuracy. CNN model is 57% accurate at identifying correctly predicted labels. Model did not perform well due to FN rate of 67%.

	Precision	Recall	F1-score
Anomalous	.299	.327	.313
Benign	.700	.672	.686
Accuracy			.569

The TSNE plot for 30% anomalous data, shown in Figure 23, shows a smaller large cluster of benign data and an increase in smaller groups from the 25% anomalous data TSNE plot. There are more smaller clusters of anomalous data spread throughout the groupings of the benign data. Based on the higher volume of anomalous clustering in the 30% anomalous data TSNE plot, this model is better at classifying benign and anomalous data, even though it visually does not look like 30% of the plot is anomalous data.

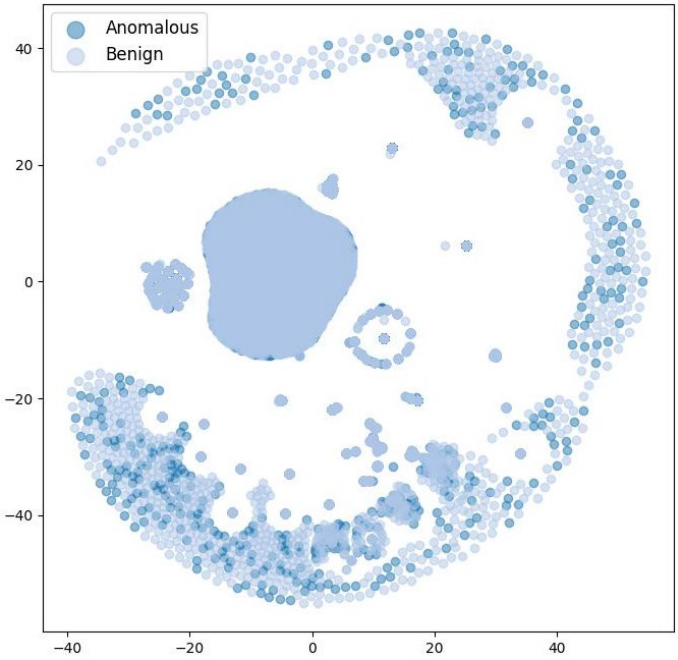

Figure 23. TSNE plot for 30% anomalous dataset. The larger cluster of benign data is smaller and very small groupings of anomalous data is spread throughout the benign data.

4. 40% Anomalous Dataset

The learning curve in Figure 24 shows a typical validation loss vs. training loss trend during model training and had similar trends to the plot for the 30% anomalous dataset model.

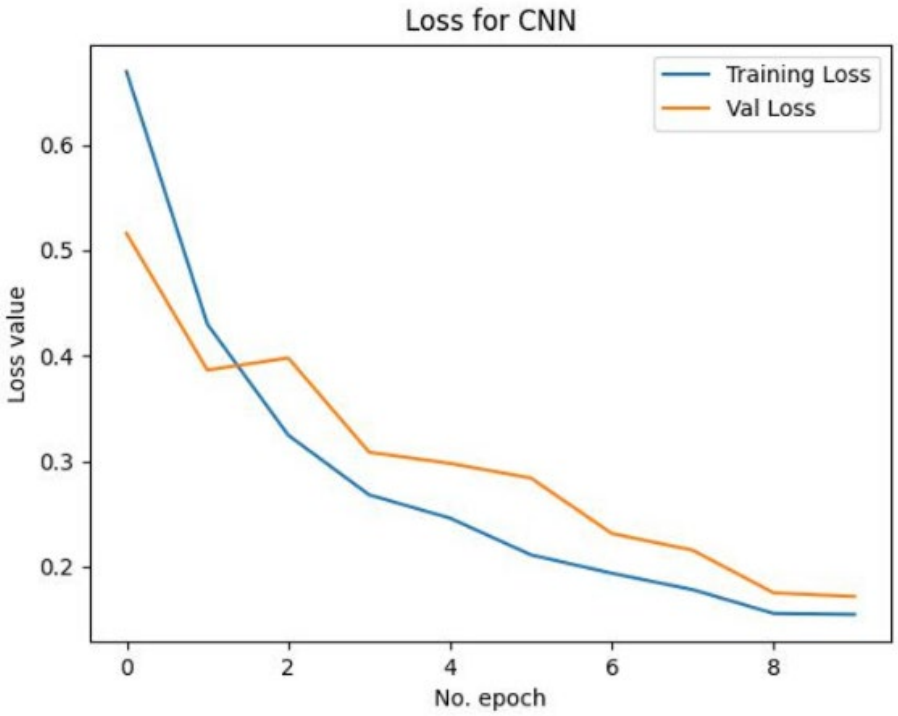

Figure 24. A typical learning curve of the 40% anomalous dataset. No severe overfitting occurred. Validation loss is gradually trending downward.

The confusion matrix in Figure 25 shows that 60% of anomalous packets were incorrectly classified as benign. For the benign packets, 61% of the packets were correctly classified as benign.

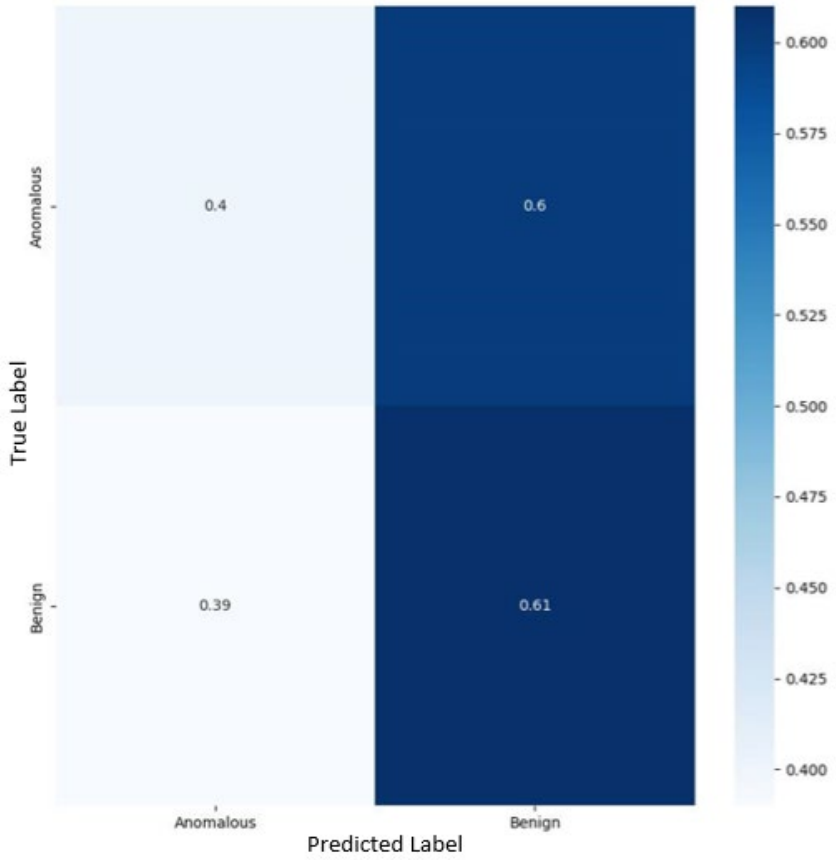

Figure 25. Confusion Matrix with Anomalous Data at 40%; 60% of anomalous packets were incorrectly classified as benign.

The performance metrics in Table 4 indicate that the model is 52% accurate at identifying the correctly predicted labels. Similar to the 30% anomalous data model, the 40% anomalous data model had mediocre performance. The accuracy was slightly better than a random binary guess. While the accuracy of the 40% model was lower than that of the 30% model, the rate of FNs was 60%, meaning that the 40% data set model was able to identify 7% more anomalous packets than the 30% model. The increase in anomalous packets in the training set, while hurting overall accuracy, is better in classifying anomalous data.

Table 4. Performance metrics on the 40% anomalous dataset: precision, recall, f1-score, and accuracy. CNN model is 52% accurate at identifying correctly predicted labels. Best model performance due to the FN rate being 60%.

	Precision	Recall	F1-score
Anomalous	.401	.397	.399
Benign	.601	.605	.603
Accuracy			.522

The 40% anomalous data TSNE plot in Figure 26 shows two smaller large clusters of benign data and about the same anomalous groupings from the 30% anomalous data TSNE plot. Based on the similar volume of anomalous clustering in the 30% anomalous data TSNE plot, this model visually has the same results in classifying benign and anomalous data.

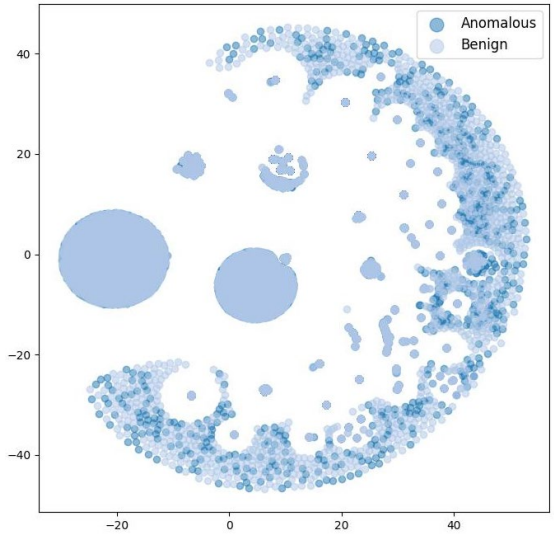

Figure 26. TSNE plot for 40% anomalous dataset. Similar volume of anomalous and benign clustering compared to 30% anomalous data.

5. 50% Anomalous Dataset

The learning curve in Figure 27 shows a typical validation loss vs. training loss trend during model training and had similar trends to the plot for the 40% anomalous

dataset model. However, after epoch 8, the validation loss steeply declined while the training loss had plateaued, suggesting the model training was near optimal for the given dataset.

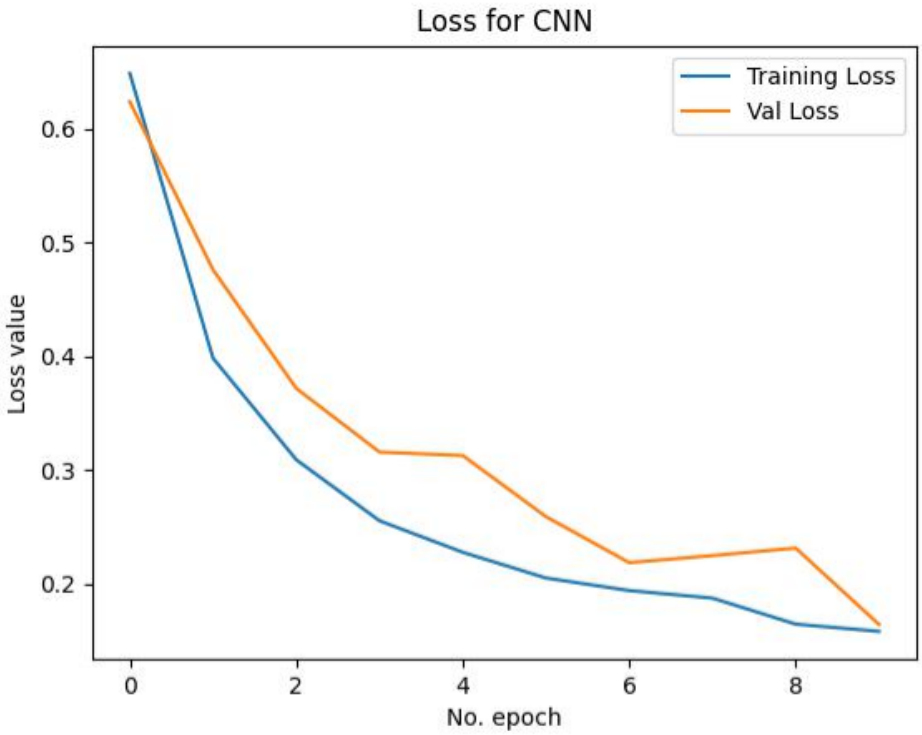

Figure 27. A typical learning curve of the 50% anomalous dataset. Validation loss steeply declines after epoch 8 while training loss is flattening. Model training is near optimal for the given dataset.

The confusion matrix in Figure 28 shows that 50% of anomalous packets were incorrectly classified as benign. For the benign packets, 51% of the packets were correctly classified as benign.

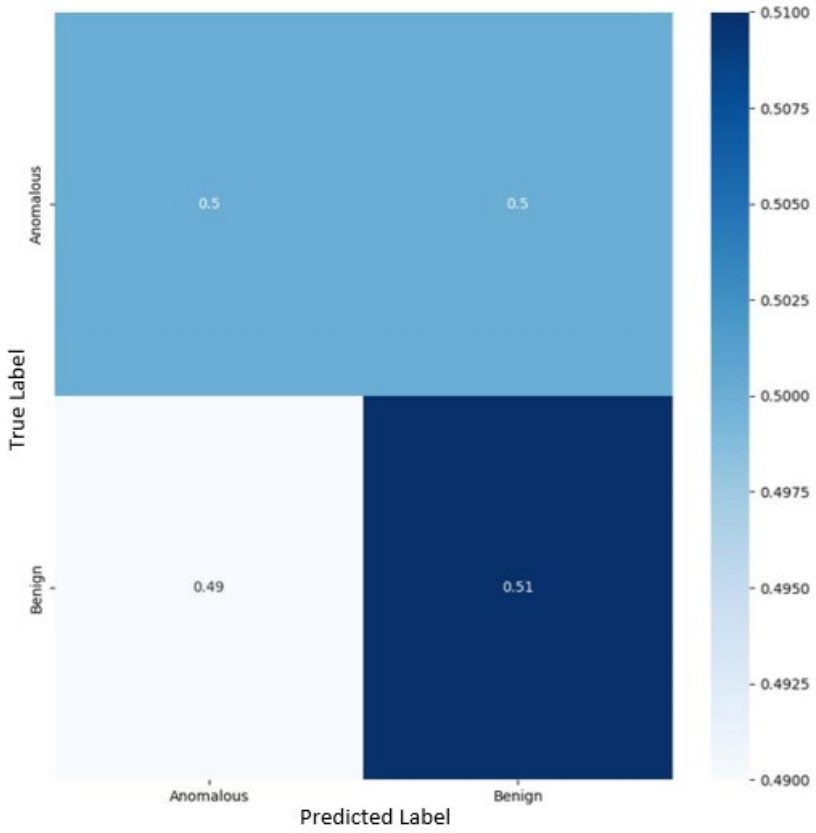

Figure 28. Confusion Matrix with Anomalous Data at 50%; 50% of anomalous packets were incorrectly classified as benign while 51% were correctly classified as benign.

The performance metrics in Table 5 indicate that the model is 50% accurate at identifying the correctly predicted labels. Overall, the model performs like random binary guessing. With regards to achieving a lower FN rate, this model is the best performance we were able to achieve. Unfortunately, the FN rate performance came at a tradeoff with the lack of performance of benign packet classification.

Table 5. Performance metrics on the 50% anomalous dataset: precision, recall, f1-score, and accuracy. CNN model is 50% accurate at identifying correctly predicted labels, which is the worst accuracy in the group. Best FN rate performance of 50%.

	Precision	Recall	F1-score
Anomalous	.502	.498	.500
Benign	.502	.507	.505
Accuracy			.502

The 50% anomalous data TSNE plot in Figure 29 shows consistent clusters of benign data, but less cluster of anomalous data from the 30% and 40% anomalous data TSNE plots. Visually it can be determined that even though we had anomalous data at 50%, there is an abundance of benign clustering compared to anomalous. Based solely on the 50% anomalous data TSNE plot, the visual classifications are by far the worst at classification.

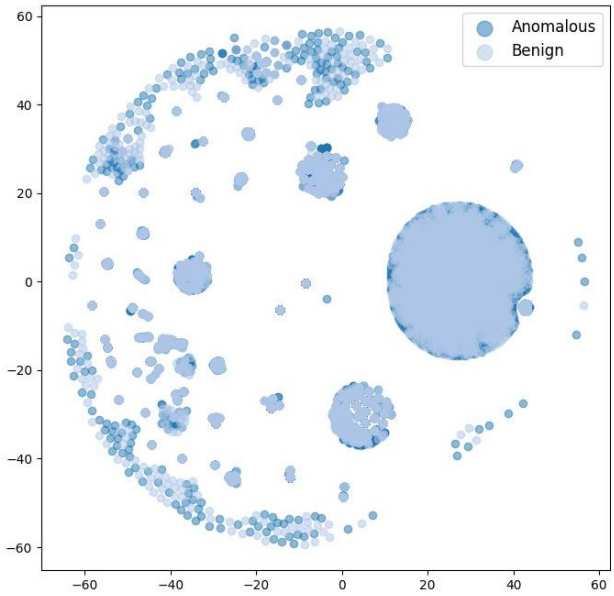

Figure 29. TSNE plot for 50% anomalous dataset. Benign data appears to be more than 50% and anomalous data appears to be less than the TSNE plot for the 30% anomalous dataset.

6. Summary of Findings

As more anomalous packets were added to the training dataset, the models were able to achieve reduced FN rates. However, the reduction in FN rates came at the cost of decreasing the correct classification of benign packets. Overall, the best performing model was the 40% anomalous data model. Although the model exhibited a higher FN rate (60%) than the 50% anomalous data model (50% FN rate), the performance of benign classification (60%) was still significantly greater than random binary guessing. The 40% anomalous data model outperformed the lower percentage models due to the reduction in FN rates. The TSNE plots did have a huge impact at deducing whether the CNN model is correctly classifying the datasets.

C. RECOMMENDATIONS FROM FINDINGS

By not obtaining a true dataset from the MCAS Miramar network, we attempted to create a synthetic 5G network dataset by incorporating generic classification labels of ones and zeros as anomalous and benign data. The datasets were not effective given the overall poor performance of the CNN results. Through our findings the following are recommendations and lessons learned. The poor performance of the model can be attributed to how we labeled the anomalies in the dataset. The random assignment of zeros and ones may not be representative of actual anomalies, which with a true dataset would have more features to enable classification. This is proven by comparing the FPs and FNs for each anomaly dataset. In Table 6 we compare the number of FNs and FPs for the five datasets used. Figure 30 shows the plot of these values. From both Table 6 and Figure 30, we can see the data converge, meaning the generic labeling method used in [25] did not work well in our process. The CNN did not have enough anomalous features to properly classify between anomalous and benign packets. If the p1, p2, and p3 PCAPs were to be replayed (rather than transferred) over the NPS 5G network, there would likely be more anomalous features to increase the CNN model performance.

Table 6. Comparison of the FPs and FNs for each anomalous data set. For the 50% anomalies dataset, the FP and FN values are similar because there are few features to rely on when labeling anomalous data.

% of Anomalies	FP	FN
10	.96	.03
20	.90	.10
30	.67	.33
40	.60	.39
50	.50	.49

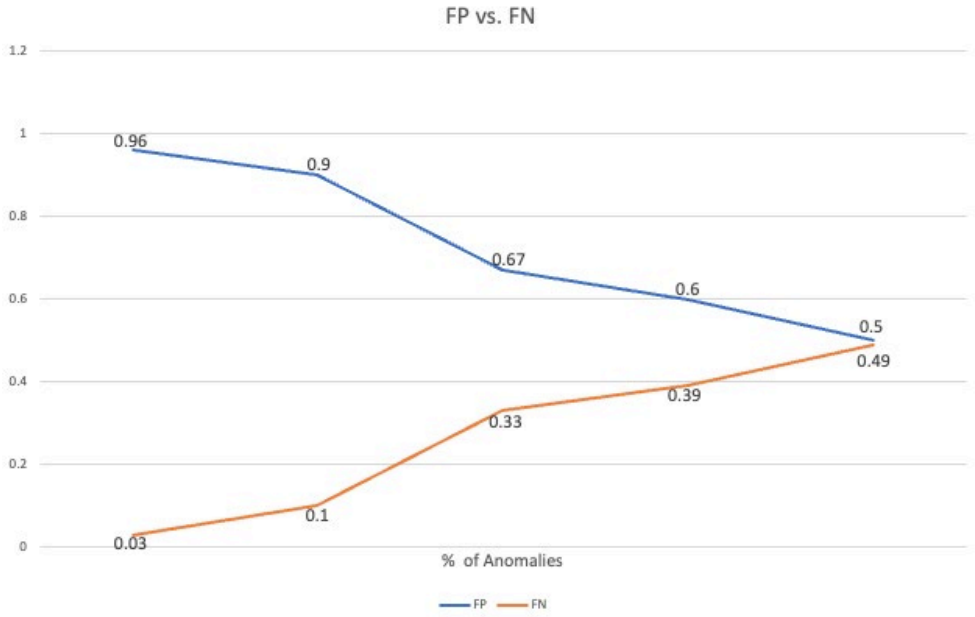

Figure 30. A plot of the FPs compared to FNs for each data set. As the percentages of anomalies increases, the FP and FN converge to 50%.

This work solidified that the building of the CNN model can be useful and potentially effective if properly trained on a true dataset. The model can perform classification tasks based off the benign classifications being higher than binary guessing for the 40% anomalous dataset; however, the model is not learning to classify actual anomalies present in the packets due to the non-representative random assignment of labels. The 18 features currently chosen to create the images through the preprocessing

technique can be thoroughly selected and defined based on a true dataset. This will highlight the differences even further between anomalous and benign packets. By narrowing down the features and making them more relevant to the MCAS Miramar network, model performance will likely increase. The design of the model is potentially effective based on its ability to learn the patterns present with the anomalous and benign images, meaning the model can classify better if given a true dataset.

Based on the findings it would be beneficial to train the model on a larger dataset with more epochs to decrease the potential of overfitting in the models that had slight indication. By increasing the dataset size, the model will have access to more anomalous images and thus will have a better chance of learning the patterns present in the anomalous image. The increased pattern recognition capacity of the model will result in better classification performance. The anomalous data created for the model was synthetically generated and only used two classifiers (1 or 0) to indicate benign or anomalous. A true dataset from the energy communications network at Miramar would allow us to create a more balanced dataset which may result in higher model performance. The advantage of using the developed model is that it can autonomously classify anomalous data flowing through the 5G network without the need for human intervention. This is an effective way of limiting user oversight in anomaly detection of the network. This work is the first step in studying ML models for the MCAS Miramar network and lessons can and have been learned.

THIS PAGE INTENTIONALLY LEFT BLANK

V. CONCLUSION

The purpose of this thesis is to develop an anomaly detection model using supervised ML for the energy communications infrastructure at MCAS Miramar. This research provides a preliminary cyber anomaly detection platform that could be potentially used on the Miramar network. We developed a CNN and trained the model using an experimental collection of network data transferred over a 5G connection.

A. SUMMARY

MCAS Miramar is the most energy forward defense installation in the nation mainly due to the operations of the EWOC and the future capability of wireless communications between the systems [3], [4]. Currently, the Verizon 5G NSA network is deployed at Miramar, which means the network architecture does not have cyber anomaly detection. This thesis has explored that a supervised ML detection platform for cyber anomaly detection although it may not be the best option for the task at hand.

This thesis begins by obtaining the 5G benign dataset collected from the AT&T 5G cellular tower at the NPS SLAMR facility. We created our own anomalous dataset from the benign dataset by pulling features from packets and establishing an additional feature that categorized whether a packet was anomalous or benign. Five different anomalous datasets were created to test the model: 10%, 25%, 30%, 40%, and 50%. Next, we implemented a preprocessing technique that allowed an image input from the datasets to be fed into our CNN ResNet 50 design.

To demonstrate the ability of anomaly detection using the CNN, we trained, validated, and tested each anomalous dataset. To measure CNN performance results, we used graphical representations and metrics from TensorFlow. Experiments showed that as more anomalous packets were added to the training dataset, the models achieved a reduced FN rate. The reduction of FN rates costed the ability of the model to properly classify benign packets. The best performance occurred with the 40% anomalous data model because it shows the most balance between a high TP rate and a low FN rate.

Overall, the CNN model did not have a good performance due to the synthetically generated data, which did not allow the model to accurately classify anomalous data from benign data. A true balanced 5G NSA dataset from the energy communications network at Miramar will also help model performance

B. FUTURE WORK

The work presented in this thesis the first step of incorporating ML solutions for the MCAS Miramar network. We tested our model on a synthetic dataset generated from the NPS network. Thus, further experimentations can be conducted to advance this research. To that end, the following is recommended: First, create a larger dataset from the 5G data collected from the NPS network. Currently the data in the archive can be imitated to create a bigger dataset than the one used during the initial tests. By increasing the dataset size, the model has the potential of creating a better learning curve therefore getting better performance. Second, collect data from the MCAS Miramar network. Once the MCAS Miramar 5G NSA network is flowing, a dataset can be put together using real data from the source. This potentially could lead to testing where Verizon can simulate cyber anomalies or attacks on the network to create a realistic dataset to help test the model. Lastly, test an alternative machine learning method. An unsupervised machine learning method such as an autoencoder would be beneficial. The autoencoder requires no classification labels and may be better suited for detecting anomalous data in the 5G network as it requires no need to transform the data into an image as input to the model. The MCAS Miramar base would benefit from exploring an unsupervised approach.

APPENDIX A. IMAGE GENERATOR

The following script is the process of turning packet data into images using a preprocessing technique.

```
import tensorflow as tf
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import sys
import IPython

NUM_OF_FEATURES = 11
FINAL_DIR = "C:\\Users\\hatha\\OneDrive\\Documents\\AshThesisFolder"

def get_max_corr_features(df_corr):
    # Intent here is to: for each feature in the correlation matrix, index
      that into that feature so you can figure out
    # the 7 most correlated features to it.

    corr_list = []
    for i in range(0,NUM_OF_FEATURES):
        df_corr_feature = df_corr.iloc[i]
        df_corr_feature_maxes = df_corr_feature.nlargest(n=9)
        corr_list.append(df_corr_feature_maxes)

    return corr_list

def generate_surrounding_matrix(df, df_corr_list):
    # Todo: for each record in the df, generate the surrounding
      correlation matrix for each feature
    # Will look like: [[0, 1, 2], [7, FEATURE, 3], [6, 5, 4]] where each
      number is the N'th
    # highest correlation feature to the anchor feature

    # Resulting matrices will be calculated for every numeric feature in
      the record, then all of
    # the matrices will be concatenated. This represents 1 image.

    # This process is repeated for entire passed df. Once complete, the
      next step will be to use
    # matplotlib to color the matrices, resulting in our actual image.

    df_length = len(df)
    sm_list = []
    df_corr_list = df_corr_list[:-1]

    # outer for loop will iterate through every row(record) in the df.
```

```
    for i in range(0, df_length):
    #for i in range(0, 50000):
        record = df.iloc[i]
        sm_record_matrix = []
        record_start_flag = True

        # inner for loop will iterate through every feature in the record
        for each in record.index:
            # check to make sure we only consider numeric features
            if (pd.api.types.is_numeric_dtype(record.loc[each])) ==
False:
                continue

            sm_constructor = []
            record_values = []

            # inner nested loop matches feature to what is in the
             df_corr_list and grabs
            # those column names

            for items in df_corr_list:
                if each == items.index[0]:
                    column_names = items.index[1:]
                    break
            # second inner nested loop now grabs the record values for the
             column names
            # listed in the column_names
            for name in column_names:
                record_values.append(record.loc[name])

            center = record.loc[each]
            f0 = record_values[0]
            f1 = record_values[1]
            f2 = record_values[2]
            f3 = record_values[3]
            f4 = record_values[4]
            f5 = record_values[5]
            f6 = record_values[6]
            f7 = record_values[7]

            # Will look like: [[0, 1, 2], [7, FEATURE, 3], [6, 5, 4]]
             where each number is the N'th
            sm_constructor.append([f0, f1, f2])
            sm_constructor.append([f7, center, f3])
            sm_constructor.append([f6, f5, f4])

            if record_start_flag:
                sm_record_matrix = sm_constructor
            else:
                sm_record_matrix[0] += sm_constructor[0]
                sm_record_matrix[1] += sm_constructor[1]
```

```
                sm_record_matrix[2] += sm_constructor[2]

            record_start_flag = False

        sm_list.append(sm_record_matrix)
    return sm_list

def convert_to_image(df_SM, label_flag):
    # Grab each image matrix and convert to an actual image

    if label_flag == 0:
        IMAGES_DIR =
"C:\\Users\\hatha\\OneDrive\\Documents\\AshThesisFolder\\images\\10_percen
t\\images_benign"


    if label_flag == 1:
        IMAGES_DIR =
"C:\\Users\\hatha\\OneDrive\\Documents\\AshThesisFolder\\images\\10_percen
t\\images_anomalous"

    cnt = 0
    for each in df_SM:
        plt.imsave(IMAGES_DIR + f"\\image_{cnt}.jpg", each,
cmap='viridis')
        cnt += 1


def main():

    DATA_DIR =
"C:\\Users\\hatha\\OneDrive\\Documents\\AshThesisFolder\\csv\\pw1.csv"
    percent = int(sys.argv[1])/100

    # Read in our data
    df = pd.read_csv(DATA_DIR)
    # Clean up df by removing NaN rows and filling NaN values with 0
    df = df.dropna(how='all')
    df = df.fillna(0)

    # Seperate out traffic by the benign=0 or anomalous=1 label based on
      our cmd line percentage
    df1 = df.sample(frac=percent, replace=False, random_state=42)
    df0 = df.drop(df1.index)

    # Generate our Correlation Matrices
    df0_corr = df0.corr()
    df1_corr = df1.corr()

    df0_corr_list = get_max_corr_features(df0_corr)
    df1_corr_list = get_max_corr_features(df1_corr)
```

```
    df0_SM = generate_surrounding_matrix(df0, df0_corr_list)
    df1_SM = generate_surrounding_matrix(df1, df1_corr_list)

    convert_to_image(df0_SM, 0)
    convert_to_image(df1_SM, 1)

    print(f"Your {len(df0_SM)}/{len(df1_SM)} images have been saved to
{FINAL_DIR}...")
```

APPENDIX B. DATASETS

The following script highlights training, validation, and test datasets.

```
import numpy as np
import os
import tensorflow as tf

def fetch_data(train_dir, val_dir, test_dir, model_type):
    # build in unit tests for obtaining shape of image and input data for
troubleshooting
    import pathlib
    from tensorflow.keras.preprocessing.image import ImageDataGenerator

    train_data_dir = pathlib.Path(train_dir)
    val_data_dir = pathlib.Path(val_dir)
    test_data_dir = pathlib.Path(test_dir)

    # may need to adjust image size
    IMAGE_SIZE = (224,224)
    model_type = model_type

    if model_type == "mobilenetv2":
        from tensorflow.keras.applications.mobilenet_v2 import
preprocess_input

    if model_type == "resnet50":
        from tensorflow.keras.applications.resnet50 import
preprocess_input

    if model_type == "vgg16":
        from tensorflow.keras.applications.vgg16 import preprocess_input

    if model_type == "vgg19":
        from tensorflow.keras.applications.vgg19 import preprocess_input

    if model_type == "densenet121":
        from tensorflow.keras.applications.densenet import
preprocess_input


    batch_size = 32
    train_datagen = ImageDataGenerator(
            preprocessing_function=preprocess_input,
            shear_range=0.2,
            zoom_range=0.2,
            horizontal_flip=True,
            vertical_flip=True,
            rotation_range=45)
```

```
train_set = train_datagen.flow_from_directory(
        train_data_dir,
        target_size=IMAGE_SIZE,
        color_mode = 'rgb',
        batch_size=batch_size,
        shuffle=True,
        seed=42,
        class_mode='categorical')

val_datagen = ImageDataGenerator(
    preprocessing_function=preprocess_input)

val_set = val_datagen.flow_from_directory(
        val_data_dir,
        target_size=IMAGE_SIZE,
        color_mode = 'rgb',
        batch_size=batch_size,
        shuffle=True,
        seed=42,
        class_mode='categorical')

test_datagen = ImageDataGenerator(
    preprocessing_function=preprocess_input)

test_set = test_datagen.flow_from_directory(
        test_data_dir,
        target_size=IMAGE_SIZE,
        color_mode = 'rgb',
        batch_size=batch_size,
        shuffle=True,
        seed=42,
        class_mode='categorical')

return train_set, val_set, test_set
```

APPENDIX C. TASKS

The following script creates the data class models and callback models to drive model training.

```
import os
import sys
#import yaml

import tensorflow as tf
import math
import pathlib
import models
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
import pickle
import callbacks
import data_class
import PIL
import PIL.Image

from sklearn.manifold import TSNE
from sklearn.decomposition import PCA
from matplotlib import cm

from sklearn.metrics import classification_report, confusion_matrix
import IPython

def pca_tsne(checkpoint_path, test_set, pca_filename, save_directory):

    model = tf.keras.models.load_model(checkpoint_path, compile=False)
    model.summary()

    feature_extractor = tf.keras.Model(model.input, model.layers[-
2].output)
    feature_extractor.summary()

    embeddings = feature_extractor.predict(test_set)
    print(embeddings.shape)

    true_labels = test_set.labels

    tsne = TSNE(2)
    pca = PCA(n_components=2)

    pca_result = pca.fit_transform(embeddings)
```

```
    tsne_result = tsne.fit_transform(embeddings)

    cmap = cm.get_cmap('tab20')
    fig, ax = plt.subplots(figsize=(8,8))
    number_categories = 2
    classes=['Anomalous', 'Benign']
    for lab in range(number_categories):
        indices = (true_labels == lab)
        ax.scatter(tsne_result[indices,0], tsne_result[indices,1],
c=np.array(cmap(lab)).reshape(1,4), label = classes[lab], alpha=0.5)
    ax.legend(fontsize='large', markerscale=2)
    #plt.show()
    final_filename = os.path.join(save_directory + "\\Plots\\" +
pca_filename + ".jpg")
    plt.savefig(final_filename)

    print("Explained variation per principal component:
{}".format(pca.explained_variance_ratio_))
    return

def metrics(checkpoint_path, test_set, metrics_filename, save_directory):

    # Load the Model
    model = tf.keras.models.load_model(checkpoint_path, compile=True)

    # Create a Confusion Matrix
    y_pred= model.predict(test_set)
    predicted_labels = np.argmax((y_pred), axis=1)
    print('Confusion Matrix')
    #quitIPython.embed()
    con_mat = confusion_matrix(test_set.labels, predicted_labels)

    #Normalization Confusion Matrix
    con_mat_norm = np.around(con_mat.astype('float') /
    con_mat.sum(axis=1)[:, np.newaxis], decimals=2)
    print(con_mat_norm)

    # Print and Save Classification Report
    classes=['Anomalous', 'Benign']
    class_report= classification_report(test_set.labels, predicted_labels,
target_names = classes, digits=3)
    print(class_report)
    final_text = os.path.join(save_directory + "\\Metrics\\" +
metrics_filename + ".txt")
    out_text = open(final_text, "w")
    out_text.write(class_report)
    out_text.close()

    # Build Confusion Matrix DF
    con_mat_df = pd.DataFrame(con_mat_norm,
                      index = classes,
```

```
                    columns = classes)

    # Plot Confusion Matrix and Heat map
    figure = plt.figure(figsize=(8, 8))
    sns.heatmap(con_mat_df, annot=True,cmap=plt.cm.Blues)
    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    #plt.show()
    final_filename = os.path.join(save_directory + "\\Metrics\\" +
metrics_filename + ".jpg")
    plt.savefig(final_filename)

    return

def main(train_dir, val_dir, test_dir, checkpoint_path, save_directory,
model_type, model_dir, pca_tsne_flag, metrics_flag, fine_tune_flag):

    train_set, val_set, test_set = data_class.fetch_data(train_dir,
val_dir, test_dir, model_type)
    #IPython.embed()
    filename = str(input("Please enter desired filename for model: "))

    if pca_tsne_flag == True:
        #pca_tsne:
        #pca_filename = str(input("Please enter desired filename for pca:
      "))
        pca_filename = filename
        pca_tsne(checkpoint_path, test_set, pca_filename, save_directory)
        return


    if metrics_flag == True:
        # Get Metrics Report Including Confusion Matrix
        #metrics_filename = str(input("Please enter desired filename for
      metrics: "))
        metrics_filename = filename
        metrics(checkpoint_path, test_set, metrics_filename,
save_directory)
        return

    model = models.create_model(model_type, checkpoint_path,
fine_tune_flag)

    if not fine_tune_flag:

model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
                                #loss='binary_crossentropy',
                                loss='categorical_crossentropy',
                                metrics=['accuracy'])
```

```
    model.summary()

    history = model.fit(train_set,
                                epochs=10,
                                validation_data = val_set,
                                steps_per_epoch = (len(train_set)//32),
                                #validation_steps = 50,

callbacks=callbacks.make_callbacks(model_dir))

    # Plot Learning Curve
    plt.plot(history.history['loss'], label='Training Loss')
    plt.plot(history.history['val_loss'], label='Val Loss')
    plt.title('Loss for CNN')
    plt.ylabel('Loss value')
    plt.xlabel('No. epoch')
    plt.legend(loc="upper right")

    # Save the Plot
    #img_save_path = os.path.join(hparams.model_dir ,
      "learning_curve.jpg")
    img_save_path = os.path.join(save_directory + "\\Plots\\" + filename +
"_learning_curve.jpg")
    #img_save_path = os.path.join(save_directory + "\\Plots\\" +
      "learning_curve.jpg")
    plt.savefig(img_save_path)

    return


if __name__ == "__main__":
    train_dir =
"C:\\Users\\hatha\\OneDrive\\Documents\\AshThesisFolder\\images\\10_percen
t\\images_train" #directory path
    val_dir =
"C:\\Users\\hatha\\OneDrive\\Documents\\AshThesisFolder\\images\\10_percen
t\\images_val" #directory path
    test_dir =
"C:\\Users\\hatha\\OneDrive\\Documents\\AshThesisFolder\\images\\10_percen
t\\images_test" #directory path
    checkpoint_path =
"C:\\Users\\hatha\\OneDrive\\Documents\\AshThesisFolder\\checkpoints\\Save
_Directory\\Models\\10_percent\\checkpoint10-0.14.pb\\"
    save_directory =
"C:\\Users\\hatha\\OneDrive\\Documents\\AshThesisFolder\\checkpoints\\Save
_Directory" #directory path
    model_type = "resnet50" # (resnet50, vgg16, vgg19, mobilenetv2,
densenet121, fully_connected)
    #model_type = "simpleCNN" # (resnet50, vgg16, vgg19, mobilenetv2,
densenet121, fully_connected)
```

```
    model_dir =
"C:\\Users\\hatha\\OneDrive\\Documents\\AshThesisFolder\\checkpoints\\Save
_Directory\\Models"
    pca_tsne_flag = True
    metrics_flag = False
    fine_tune_flag = False
    main(train_dir, val_dir, test_dir, checkpoint_path, save_directory,
model_type, model_dir, pca_tsne_flag, metrics_flag, fine_tune_flag)
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX D. CNN MODEL

The following script is Tensorflow's ResNet 50 CNN model.

```
import tensorflow as tf
import os
from tensorflow import keras


def fine_tuning_option(checkpoint_path):
    #load checkpoint weights and put into model
    fine_tune_at = 150
    model = tf.keras.models.load_model(checkpoint_path, compile=True)
    model.trainable = True

    # freeze all the layers before the fine_tune_at (based on ResNets (our
      highest performing model) base) 175 layers
    base_model_layer = model.layers[1]
    print("Number of layers in the base model: ",
len(base_model_layer.layers))
    for layer in base_model_layer.layers[:fine_tune_at]:
        layer.trainable = False

    return model

def create_fully_connected_model(input_shape):
    model = tf.keras.models.Sequential([
          tf.keras.layers.Flatten(input_shape),
          tf.keras.layers.Dense(200, activation='sigmoid'),
          tf.keras.layers.Dense(60, activation='sigmoid'),
          tf.keras.layers.Dense(10, activation='softmax')
        ])
    return model

def create_simple_CNN_model(input_shape):
    model = tf.keras.models.Sequential()
    model.add(tf.keras.layers.Conv2D(32, (3, 3), activation='relu',
input_shape=(32, 32, 3)))
    model.add(tf.keras.layers.MaxPooling2D((2, 2)))
    model.add(tf.keras.layers.Conv2D(64, (3, 3), activation='relu'))
    model.add(tf.keras.layers.MaxPooling2D((2, 2)))
    model.add(tf.keras.layers.Conv2D(64, (3, 3), activation='relu'))
    model.add(tf.keras.layers.Flatten())
    model.add(tf.keras.layers.Dense(64, activation='relu'))
    model.add(tf.keras.layers.Dense(2, activation='softmax'))
    return model

def create_vgg16_model(input_shape):
```

```
    base_model = tf.keras.applications.VGG16(input_shape=input_shape,
                                             include_top=False,
                                             weights='imagenet')

    base_model.trainable = False
    print("Number of layers in the base model: ", len(base_model.layers))
    global_average_layer = tf.keras.layers.GlobalAveragePooling2D()
    prediction_layer = tf.keras.layers.Dense(8, activation='softmax')
    inputs = tf.keras.Input(shape=(224, 224, 3))

    x = base_model(inputs, training=False)
    x = global_average_layer(x)
    x = tf.keras.layers.Dropout(0.2)(x)
    outputs = prediction_layer(x)
    model = tf.keras.Model(inputs, outputs)

    return model

def create_vgg19_model(input_shape):
    base_model = tf.keras.applications.VGG19(input_shape=input_shape,
                                             include_top=False,
                                             weights='imagenet')

    base_model.trainable = False
    print("Number of layers in the base model: ", len(base_model.layers))
    global_average_layer = tf.keras.layers.GlobalAveragePooling2D()
    prediction_layer = tf.keras.layers.Dense(8, activation='softmax')
    inputs = tf.keras.Input(shape=(224, 224, 3))

    x = base_model(inputs, training=False)
    x = global_average_layer(x)
    x = tf.keras.layers.Dropout(0.2)(x)
    #x = tf.keras.layers.Dense(100, activation='relu')(x)
    #x = tf.keras.layers.Dense(50, activation='relu')(x)
    outputs = prediction_layer(x)
    model = tf.keras.Model(inputs, outputs)

    return model

def create_mobilenetv2_model(input_shape):
    base_model =
tf.keras.applications.MobileNetV2(input_shape=input_shape,
                                             include_top=False,
                                             weights='imagenet')

    base_model.trainable = False
    print("Number of layers in the base model: ", len(base_model.layers))
    global_average_layer = tf.keras.layers.GlobalAveragePooling2D()
    prediction_layer = tf.keras.layers.Dense(8, activation='softmax')
    inputs = tf.keras.Input(shape=(224, 224, 3))
```

```
    x = base_model(inputs, training=False)
    x = global_average_layer(x)
    x = tf.keras.layers.Dropout(0.2)(x)
    outputs = prediction_layer(x)
    model = tf.keras.Model(inputs, outputs)

    return model

def create_resnet50_model(input_shape):
    base_model = tf.keras.applications.ResNet50(input_shape=input_shape,
                                                include_top=False,
                                                weights='imagenet')

    base_model.trainable = False
    print("Number of layers in the base model: ", len(base_model.layers))
    global_average_layer = tf.keras.layers.GlobalAveragePooling2D()
    #prediction_layer = tf.keras.layers.Dense(2, activation='softmax')
    prediction_layer = tf.keras.layers.Dense(2, activation='softmax')
    inputs = tf.keras.Input(shape=(224, 224, 3))

    x = base_model(inputs, training=False)
    x = global_average_layer(x)
    x = tf.keras.layers.Dropout(0.2)(x)
    # x = tf.keras.layers.Dense(100, activation='relu')(x)
    # x = tf.keras.layers.Dense(50, activation='relu')(x)
    outputs = prediction_layer(x)
    model = tf.keras.Model(inputs, outputs)

    return model

def create_densenet121_model(input_shape):
    base_model =
tf.keras.applications.DenseNet121(input_shape=input_shape,
                                                include_top=False,
                                                weights='imagenet')

    base_model.trainable = False
    print("Number of layers in the base model: ", len(base_model.layers))
    global_average_layer = tf.keras.layers.GlobalAveragePooling2D()
    prediction_layer = tf.keras.layers.Dense(4, activation='softmax')
    inputs = tf.keras.Input(shape=(224, 224, 3))

    x = base_model(inputs, training=False)
    x = global_average_layer(x)
    x = tf.keras.layers.Dropout(0.2)(x)
    outputs = prediction_layer(x)
    model = tf.keras.Model(inputs, outputs)

    return model

def create_model(model_type, checkpoint_path, fine_tune_flag):
```

```
if fine_tune_flag:
    return fine_tuning_option(checkpoint_path)

model_type = model_type.lower()
input_shape = (224, 224, 3)
if model_type == 'fully_connected':
    return create_fully_connected_model(input_shape)

if model_type == 'vgg16':
    return create_vgg16_model(input_shape)

if model_type == 'vgg19':
    return create_vgg19_model(input_shape)

if model_type == 'mobilenetv2':
    return create_mobilenetv2_model(input_shape)

if model_type == 'resnet50':
    return create_resnet50_model(input_shape)

if model_type == 'densenet121':
    return create_densenet121_model(input_shape)

if model_type == 'simpleCNN':
    return create_simple_CNN_model(input_shape)


else:
    print('unsupported model type %s' % (model_type))
    return None
```

APPENDIX E. CALLBACKS

The following script allows for the saving of parameters after every epoch.

```
import os
import tensorflow as tf
import math

def make_callbacks(model_dir):
    checkpoints = []
    #if 'checkpoint' in callback_list:
    checkpoints.append(_make_model_checkpoint_cb(model_dir))
    #if 'csv_log' in callback_list:
    checkpoints.append(_make_csvlog_cb(model_dir))
    #if 'reducelr' in callback_list:
    checkpoints.append(_make_model_reducelr_cb())
    #print("reduce lr was called")
    return checkpoints

def _make_model_checkpoint_cb(model_dir):

    checkpoint = tf.keras.callbacks.ModelCheckpoint(
        os.path.join(model_dir, "checkpoint{epoch:02d}-
{val_loss:.2f}.pb"),
        monitor='val_loss',
        verbose=1,
        save_best_only=True,
        save_weights_only=False,
        mode='min',
        save_freq='epoch')
    return checkpoint

def _make_model_reducelr_cb():
    reduce_lr = tf.keras.callbacks.ReduceLROnPlateau(
        monitor = 'val_loss',
        factor = 0.1,
        patience = 7,
        verbose = 1,
        min_lr = 1e-7)
    return reduce_lr

def _make_csvlog_cb(model_dir):
    print("entered_csv_log")
    csv_log = tf.keras.callbacks.CSVLogger(os.path.join(model_dir,
"log.csv"), append=True, separator=';')
    return csv_log
```

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

[1] E. W. Prehoda, C. Schelly, and J. M. Pearce, "U.S. strategic solar photovoltaic-powered microgrid deployment for enhanced national security," *Renewable and Sustainable Energy Reviews*, vol. 78, pp. 167–175, 2017

[2] Energy.gov, "How microgrids work," Jun. 2014 [Online]. Available: https://www.energy.gov/articles/how-microgrids-work

[3] Black & Veatch, "Marine Corps Air Station Miramar microgrid: From design and construction to operations and commissioning," Jun. 2022 [Online]. Available: https://www.bv.com/projects/marine-corps-air-station-miramar-microgrid-design-and-construction-operations-and

[4] US Marines, "Microgrid at Marine Corps Air Station Miramar," Jun. 2021 [Online]. Available: https://www.marines.mil/News/News Display/Article/ 2677033/microgrid-at-marine-corps-air-station-miramar/

[5] USIgnite, "Request for Proposal (RFP) restoration of existing Energy Management System (EMS) & support of EMS integration with Photovoltaic (PV) array inverters & backup generators at MCAS Miramar," Miramar, CA, U.S., Aug. 2021

[6] USIgnite, "Request for Proposal (RFP) integration of existing Energy Management System (EMS) with Distributed Energy Resources (DER) at MCAS Miramar," Miramar, CA, USA, Jan. 2021.

[7] P. Kim, *Matlab Deep Learning with Machine Learning, Neural Networks and Artificial Intelligence*, New York, NY, USA: Apress, 2017

[8] Ujjwalkarn, "An intuitive explanation of convolutional neural networks," Aug. 2016 [Online]. Available: https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/

[9] B. Ding, H. Qian, and J. Zhou, "Activation functions and their characteristics in deep neural networks," in 2018 Chinese control and decision conference (CCDC). *IEEE,* 2018, pp. 1836–1841.

[10] X. Ying, "An overview of overfitting and its solutions," *Journal of Physics: Conference Series*, vol. 1168, no. 2, pp. 022022, 2019.

[11] Aurélien Géron, *Hands-On Machine Learning with Scikit-Learn and TensorFlow Concepts, Tools, and Techniques to Build Intelligent Systems*, O'Reilly Media, Inc., 2019.

[12] G. Liu, Y. Huang, Z.Chen, L.Liu, Q. Wand and N. Li, "5G deployment: Standalone vs. non-standalone from the operator perspective," *IEEE Communications Magazine*, vol. 58, no. 11, pp. 83–89, 2020.

[13] Verizon, "What is 5G NR," Dec. 2019 [Online] Available: https://www.verizon.com/about/our-company/5g/what-is-5g-nr

[14] G. Holtrup *et al.*, "5G system security analysis," Aug. 2021 [Online]. Available: https://arxiv.org/pdf/2108.08700.pdf

[15] S. Teral, "5G best choice architecture IHS Markit Technology | White Paper," Jan. 2019 [Online]. Available: https://www.redestelecom.es/siteresources/files/894/48.pdf

[16] P. Frenger and R. Tano, "More capacity and less power: How 5G NR can reduce network energy consumption," *2019 IEEE 89th Vehicular Technology Conference (VTC2019-Spring)*, pp. 1–5, 2019.

[17] Positive Technologies, "5G security issues," Nov 2019. [Online]. Available: https://www.gsma.com/membership/wp-content/uploads/2019/11/5G-Research_A4.pdf

[18] CISA and USD R&E, "5G security evaluation process investigation version 1," May 2022 [Online]. Available: https://www.cisa.gov/sites/default/files/publications/5G_Security_Evaluation_Process_In vestigation_508c.pdf

[19] S. Sridharan, "Machine Learning (ML) in a 5G standalone (SA) self organizing network (SON)," *International Journal of Computer Trends and Technology*, vol. 68, no. 11, pp. 43–48, 2020.

[20] D. Darah, "5G NSA vs. SA: How does each deployment mode differ," May 2022 [Online]. Available: https://www.techtarget.com/searchnetworking/feature/5G-NSA-vs-SA-How-does-each-deployment-mode-differ

[21] J. Lam, R. Abbas, "Machine learning based anomaly detection for 5G networks," Mar. 2020 [Online]. Available: https://arxiv.org/abs/2003.03474

[22] M. Alabadi and Y. Celik, "Anomaly detection for cyber-security based on convolution neural network: A survey," *2020 International Congress on Human-Computer Interaction, Optimization and Robotic Applications (HORA)*, pp. 1–14, 2020.

[23] M. N. I. Farooqui, J. Arshad, and M. M. Khan, "A layered approach to threat modeling for 5G-based systems," *Electronics*, vol. 11, no. 12, p. 1819–1836, Jun. 2022 [Online]. Available: https://www.mdpi.com/2079-9292/11/12/1819

[24] A. L. Perales Gomez *et al.*, "On the generation of anomaly detection datasets in industrial control systems," *IEEE Access*, vol. 7, pp. 177460–177473, 2019.

[25] S. Sevgican, M. Turan, K. Gokarslan, H. B. Yilmaz, and T. Tugcu, "Intelligent network data analytics function in 5G cellular networks using machine learning," *Journal of Communications and Networks*, vol. 22, no. 3, pp. 269–280, 2020.

[26] Ookla, "Speedtest by Ookla – The global broadband speed test," 2019 [Online]. Available:https://www.speedtest.net/

[27] L. Weston, "New LP-CRADA between NPS, TMGcore focused on high-density computing," Oct. 2021 [Online] Available: https://nps.edu/-/new-lp-crada-between-nps-tmgcore-focused-on-high-density-computing

[28] Netresec, "SCADA / ICS PCAP files from 4SICS," Jan. 2018 [Online] Available: https://www.netresec.com/?page=PCAP4SICS

[29] tjcruz-dei, "Release modbus TCP SCADA #1 tjcruz-dei/ICS_PCAPS," *GitHub*, Jan.2019 [Online] Available: https://github.com/tjcruzdei/ICS_PCAPS/releases/tag/MODBUSTCP%231

[30] J. Smith, "ICS-pcap," *GitHub*, Jul. 2022 [Online] Available: https://github.com/automayt/ICS-pcap

[31] X. Liu, Z. Tang, and B. Yang, "Predicting network attacks with CNN by constructing images from netFlow data," *2019 IEEE 5th Intl Conference on Big Data Security on Cloud (BigDataSecurity), IEEE Intl Conference on High Performance and Smart Computing, (HPSC) and IEEE Intl Conference on Intelligent Data and Security (IDS)*, pp. 61–66, 2019.

[32] TensorFlow, "An end-to-end open-source machine learning platform," 2022 [Online]. Available: https://www.tensorflow.org/

[33] TensorFlow, "TensorFlow Hub," May 2022 [Online]. Available: https://tfhub.dev/

[34] M. Talo, "Convolutional neural networks for multi-class histopathology image classification," Mar. 2019 [Online] Available: https://arxiv.org/pdf/1903.10035.pdf

[35] L. Ali, F. Alnajjar, H. Jassmi, M. Gochoo, W. Khan, and M. Serhani, "Performance evaluation of deep CNN-based crack detection and localization techniques for concrete structures," *Sensors*, vol. 21, pp.1688-1710, 2019.

[36] S. Verma, “Understanding input and output shapes in convolution neural network| Keras,” Apr. 2020 [Online]. Available: https://towardsdatascience.com/understanding-input-and-output-shapes-in-convolution-network-keras-f143923d56ca

[37] Devopedia, “Confusion matrix,” Aug. 2020 [Online] Available: https://devopedia.org/confusion-matrix

[38] P. Schneider and K. Böttinger, “High-performance unsupervised anomaly detection for cyber-physical system networks,” in *Proceedings of the 2018 workshop on cyber-physical systems security and privacy*, 2018, pp. 1–12.

[39] Google Developers, “Classification: Machine learning crash course,” Jul. 2022 [Online] Available: https://developers.google.com/machine-learning/crash-course/ml-intro

[40] J. T. Vanderplas, *Python Data Science Handbook: Essential Tools for Working With Data*, O’Reilly Media, Inc., 2017.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California