TRANSFER LEARNING FOR NETWORK TRAFFIC ANOMALY DETECTION

by

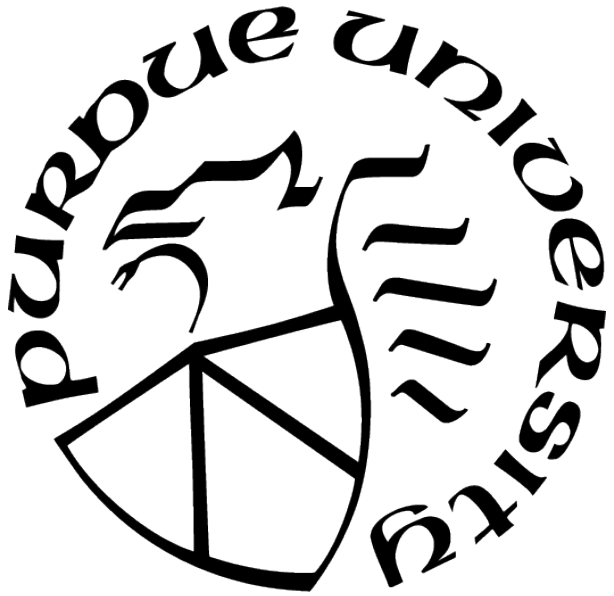
Shreya Ghosh

A Thesis

Submitted to the Faculty of Purdue University

In Partial Fulfillment of the Requirements for the degree of

Master of Science in Electrical and Computer Engineering



School of Electrical and Computer Engineering

West Lafayette, Indiana

May 2021

THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF COMMITTEE APPROVAL

Dr. Aly El Gamal, Chair

School of Electrical and Computer Engineering

Dr. Milind Kulkarni

School of Electrical and Computer Engineering

Dr. Arif Ghafoor

School of Electrical and Computer Engineering

Approved by:

Dr. Dimitrios Peroulis

ACKNOWLEDGMENTS

I would like to express my gratitude to my advisor Prof. Aly El Gamal for his continuous guidance and mentorship. I am extremely grateful to have received the opportunity to learn from someone as knowledgeable as him. I am grateful to him not only for his supervision of my research, but also for giving me the opportunity to learn from my mistakes and the advice and support he has given me throughout the course of my Master's degree. Next, I would like to express my gratitude to my committee members, Professor Kulkarni and Professor Ghafoor, for providing me with guidance and helpful advice throughout this time.

I take this opportunity to acknowledge my labmate, Shafin Jameel, for the informative research discussions we had. I would also like to thank him for having faith in my abilities and for giving me general academic advice. Lastly, I would like to thank my family and friends for their unconditional support and encouragement throughout my Master's journey. Thank You.

TABLE OF CONTENTS

LIST OF TABLES 8

LIST OF FIGURES 9

LIST OF SYMBOLS 10

ABBREVIATIONS 11

ABSTRACT 12

1 INTRODUCTION 14

1.1 Motivation 14

1.2 Contributions of This Work 15

2 BACKGROUND 18

2.1 Dataset 18

2.2 Attack Description 19

2.3 Class Imbalance Problem 21

2.4 Existing Machine Learning Approaches 21

3 HIDDEN MARKOV MODELS: A STATISTICAL MODELING BASED APPROACH 23

3.1 What are Hidden Markov Models? 23

3.2 Implementation 25

3.3 Results 27

4 NEURAL NETWORK ARCHITECTURES 28

4.1 Motivation 28

4.2 CLDNN 28
4.2.1 Architecture 28
4.2.2 Implementation 29
4.2.3 Results 30
Metrics 30
Confusion Matrices 31
4.3 CNN with BGRU layers and CNN with stacked BGRU layers 33
4.3.1 Architecture and Implementation 33
4.3.2 Results 36
4.4 One Class Neural Networks 38
4.4.1 What are One Class Neural Networks? 38
4.4.2 Metrics 39
4.4.3 Implementation and Results 40
4.5 CNN 41
4.5.1 Architecture and Implementation 41
4.5.2 Results 42
5 ATTACK CORRELATIONS 44
5.1 Base Architecture Trained on Original Dataset 44
5.1.1 Implementation 44
5.1.2 Results of Training with Original Dataset 44

5.2 Base Architecture Trained on Synthetic and Bootstrapped Dataset 47
5.2.1 Motivation 47
5.2.2 Synthetic Minority Oversampling Technique(SMOTE) 47
5.2.3 Bootstrapped Dataset 47
5.2.4 Results of Training with Synthetic and Bootstrapped Dataset 48
5.3 Attack Boosting Algorithm 51
5.3.1 Implementation 51
5.3.2 Metrics 53
5.3.3 Results 53
5.4 Discussion 54
6 HYPOTHESIS FOR ATTACK CORRELATIONS 56
6.1 Introduction 56
6.2 Feature Selection 56
6.2.1 Recursive Feature Elimination Algorithm 56
6.2.2 Implementation 56
6.2.3 Results 57
6.3 Hypothesis for Attack Correlations 59
6.3.1 Hypothesis 1 59
6.3.2 Hypothesis 2 60
6.3.3 Hypothesis 3 60

6.3.4 Discussion 61

7 CONCLUSION 62

7.1 Conclusion 62

7.2 Future Work 62

REFERENCES 64

LIST OF TABLES

2.1 Dataset Composition 18

3.1 HMM vs CLDNN 27

4.1 CLDNN Architecture 29

4.2 CNN with BGRU Architecture 34

4.3 CNN with Stacked BGRU Architecture 35

4.4 Comparison of performance of LSTM, BGRU and stacked BGRU architectures on all testing attacks when trained with Attack 4(DoS:HULK) 36

4.5 Comparison of performance of LSTM, BGRU and stacked BGRU architectures on all testing attacks when trained with Attack 2(DDoS) 36

4.6 Comparison of performance of LSTM, BGRU and stacked BGRU architectures on all testing attacks when trained with Attack 3(DoS:GoldenEye) 36

4.7 Comparison of performance of LSTM, BGRU and stacked BGRU architectures on all testing attacks when trained with Attack 5(DoS:Slowhttptest) 37

4.8 Comparison of performance of LSTM, BGRU and stacked BGRU architectures on all testing attacks when trained with Attack 6(DoS:Slowloris) 37

4.9 AUROC values for different seed values 41

4.10 CNN Architecture 42

5.1 Observed Correlations 46

5.2 Attack Boosting Algorithm on Selected Attacks 54

5.3 Attack Boosting Algorithm on Attack 6(DoS:Slowloris) when trained on SMOTE generated benign and attack data 54

6.1 Number of Features selected for select attack pairs by RFE algorithm 57

6.2 Features selected by RFE for select training-testing attack pairs 58

LIST OF FIGURES

3.1 Training stage: HMM 26

3.2 Testing stage: HMM 26

4.1 Attack Accuracy for CLDNN multi-class classification problem 31

4.2 Attack Accuracy for CLDNN for two-class classification problem 32

4.3 Example ROC curve 40

4.4 Results of CNN Architecture on multi-class classification problem 43

5.1 Overall Accuracy for the base architecture for the two-class classification problem 45

5.2 Attack Accuracy for the base architecture for the two-class classification problem 46

5.3 Confusion Matrix for Base Architecture trained on SMOTE generated attack data and tested on SMOTE generated benign data and real attack data . . . 49

5.4 Confusion Matrix for Base Architecture trained on SMOTE generated attack data and tested on real benign data and real attack data 50

5.5 Confusion Matrix for Base Architecture trained on bootstrapped attack data and tested on real benign data and real attack data 51

LIST OF SYMBOLS

$\sigma(.)$ sigmoid function

ABBREVIATIONS

HMM	Hidden Markov Model
DNN	Deep Neural network
LSTM	Long Short Term Memory
CNN	Convolutional Neural Network
CLDNN	Convolutional Long Short Term Memory Deep Neural Network
DDoS	Distributed Denial of Service
ReLU	Rectified Linear Unit
RFE	Recursive Feature Elimination
BGRU	Bidirectional Gated Recurrent Unit
BSRU	Bidirectional Simple Recurrent Unit
OCNN	One Class Neural Network

ABSTRACT

Statistics reveal a huge increase in cyberattacks making technology businesses more susceptible to data loss. With increasing application of machine learning in different domains, studies have been focused on building cognitive models for traffic anomaly detection in a communication network. These studies have led to generation of datasets containing network traffic data packets, usually captured using softwares like Wireshark. These datasets contain high dimensional data corresponding to benign data packets and attack data packets of known attacks. Recent research has mainly focused on developing machine learning architectures that are able to extract useful information from high dimensional datasets to detect attack data packets in a network. In addition, machine learning algorithms are currently trained to detect only documented attacks with available training data. However, with the proliferation of new cyberattacks and zero-day attacks with little to no training data available, current employed algorithms have little to no success in detecting new attacks. In this thesis, we focus on detecting rare attacks using transfer learning from a dataset containing information pertaining to known attacks.

In the literature, there is proof of concept for both classical machine learning and deep learning approaches for anomaly detection [1]. We show that a deep learning approach outperforms explicit statistical modeling based approaches by at least 21% for the used dataset. We perform a preliminary survey of candidate deep learning architectures before testing for transferability and propose a Convolutional Neural Network architecture that is 99.65% accurate in classifying attack data packets.

To test for transferability, we train this proposed CNN architecture with a known attack and test it's performance on attacks that are unknown to the network. For this model to extract adequate information for transferability, the model requires a higher representation of attack data in the training dataset with the current attack data comprising only 20% of the dataset. To overcome the problem of small training sets, several techniques to boost the number of attack data packets are employed like a novel synthetic dataset based training

and bootstrapped dataset training. For targeted training, a subset of training attacks are identified to maximize learning potential.

Our study results in identification of training-testing attack pairs that show high learning transferability. Most of the strong and consistent correlations are observed among Denial of Service(DoS) training-testing attack pairs. Furthermore, we propose hypotheses for model generalization. Our results are validated by a study of dataset features and attack characteristics using the Recursive Feature Elimination(RFE) algorithm.

1. INTRODUCTION

1.1 Motivation

In transfer learning, a neural network uses knowledge learned from a previous training to improve generalization on another similar task. Our goal is to leverage learning transferability to select attacks from the training dataset that causes the model to generalize for other attacks which can be extended to rare attacks.

In computer networks that employ machine learning algorithms for network security, it is difficult to provide guarantees about the kind of attacks that the network can expect to see and is susceptible to, especially with the rise in the number of novel attacks. Current machine learning algorithms are usually trained to detect a set of known attacks, or learn from attacks performed in the past. Therefore, it is hard to predict the range of attacks for which the employed machine learning algorithm is robust against.

Transferability studies can help us understand the range of attacks for which the system is actually robust against based on the attacks that the algorithm has been trained to detect and how those attacks enable the model to generalize for other attacks. This is especially useful when the trained model is able to generalize to unknown attacks without being trained on them explicitly because this provides a larger range of attacks that the network may be secure against. Studying these attack correlations indicates the attacks that the model can detect, and the accuracy with which the model can detect these unknown attacks. This helps us evaluate the risks of what the model can predict, if the observed correlations are consistent, and what kind of protection the algorithm provides at what computational cost. Transferability studies could discover that the model scales to other different attacks that it has learnt to detect without being trained on them explicitly and that provides assurance of detection for even broader range of attacks than the ones it has seen. For groups of similar attacks or correlated attacks, transfer learning enables us to find representative attacks from that group that help us train the model with smaller number of attacks yet achieve the same level of security as it would have if it was trained with all the attacks in the group.

The increasing applications of machine learning has also seen deep learning models going into hardware chips. The advantage of transferability is that we could identify smaller

training sets yet making the model generalize well for a broader range of attacks. This decreases training times, making the process of training computationally efficient and having lower memory requirements. This is important especially if this algorithm is deployed on a resource constrained device while performing at par with a model trained on a full training set.

Increasing research in this domain has led to an increase in network traffic datasets. Some of the common datasets used are CAIDA 2007, DARPA 98, KDD 99, CSE-CIC-IDS 2018 to name a few. We use the traffic dataset CICIDS 2017 for training our model. Traffic monitoring softwares like Wireshark are used to log and monitor network traffic packets for creating these datasets. While these datasets provide a complete description of network packets, they have a high dimensional feature space. In our preliminary study of a potentially suitable neural network, we develop an architecture that is able to extract meaningful data from high dimensional training data and limited number of attack data packets. After identifying a suitable network, we focus on attack correlations and comparisons for why training our model with one attack scales for another and if these correlations are symmetric or asymmetric. We describe the contributions of this work in three sections : 1) Developing a DNN architecture 2) Testing our proposed architecture for transferability 3) Hypotheses for attack correlations.

1.2 Contributions of This Work

Developing a DNN architecture

We carry out a comparative study of a Hidden Markov Model, as a statistical modeling based approach, and a candidate Convolutional LSTM Deep Neural Network(CLDNN) architecture and show that the deep learning approach scales better for our dataset. Further, we explore deep learning models for the classification task. Our candidate DNN architectures include a Convolutional LSTM Deep Neural Network(CLDNN), Convolutional Neural Network with BGRU recurrent layers, One Class Neural Networks (OCNN), and a Convolutional Neural Network(CNN). We evaluate the performance of these models when trained and tested on all attacks in the dataset as well as learning transferability. It is worth noting

that all the above architectures had overall accuracies of above 90% however, this dataset is highly unbalanced with a majority of the dataset consisting of benign data packets. The models are able to detect benign attack packets with a high accuracy thereby boosting the overall accuracy of the models. Therefore, a major criteria for architecture selection is not the overall accuracy but the percentage of attack data packets it is able to correctly classify. We study attack transferability with some of these architectures however none of the candidate models other than the proposed architecture showed very strong correlations between training-testing attack pairs. Our proposed architecture is the model that shows maximum attack data classification accuracy for the multi-class classification problem. This architecture is able to perform deep feature extraction using flow based features not only for adequately sampled attacks, but also for severely under represented attacks.

Testing our proposed architecture for transferability

We use our proposed architecture to test for attack transferability. In this section, we train the model with one kind of attack from the training dataset and test it on another attack. The entire dataset consists of 20% of attack data that are further divided into fourteen categories of attacks. Therefore, representation of each attack class is low. For learning transferability, we require a larger number of attack data packets. To address this problem, we use two techniques to increase the number of attack data packets: 1) Using SMOTE generated data 2) Using a bootstrapped dataset. For the first method, we use a Synthetic Minority Oversampling Technique(SMOTE) to synthetically generate more attack data packets in each attack class. For the second method, the original attack data is resampled, shuffled and added to the training dataset to match the number of benign data packets. Furthermore, we explore the possibility that the model may exhibit higher testing accuracies for a particular attack if it is trained with a subset of other attacks in the dataset. An *attack boosting algorithm* is employed to select a subset of attacks from the training dataset to maximize the classification accuracy when tested on a particular attack.

Hypotheses for correlated attacks

The study of attack correlations reveals training testing attack pairs that exhibit strong correlation with each other. We propose hypotheses for these observed attack correlations.

We study attack characteristics and important features to validate our hypotheses. The Recursive Feature Elimination(RFE) algorithm is used to identify the dominant features for learning. This not only reduces dimensionality of the training data but also reduces training times. We make inferences about the properties of the features selected and the number of features selected for a pair of correlated attacks. This results in identification of usable attacks in our dataset for transfer learning and also explaining why they scale for certain attacks.

2. BACKGROUND

2.1 Dataset

The dataset used is the Canadian Institute of Cybersecurity Intrusion Detection Systems dataset: CICIDS 2017. Overall, the dataset consists of 8 separate CSV files, with data corresponding to attacks simulated in 8 sessions included. The packet data that is included is for the period of - 9a.m Monday, July 3, 2017 to 5 p.m on Friday July 7, 2017. The entire dataset consists of data corresponding to 14 types of attacks and benign traffic. The dataset describes 78 flow features. 80% of the data entries are benign data and 20% of the dataset consists of attack data. The dataset was created by simulating 14 different types of attacks that are: Botnet, DDoS, DoS-GoldenEye, DoS-Hulk, DoS-Slowhttptest, DoS-Slowloris, Brute Force-FTP Patator, Heartbleed, Infiltration, PortScan, Brute Force-SSH Patator, Web Attack-Brute Force, Web Attack-SQL Injection, Web Attack-XSS.

Table2.1. Dataset Composition

Classes	**Attack**	**Percentage of data**
0	BENIGN	80.3
1	Botnet	0.069
2	DDoS	4.52
3	DoS GoldenEye	0.36
4	DoS Hulk	8.16
5	DoS Slowhttptest	0.19
6	DoS Slowloris	0.2
7	FTP-Patator	0.28
8	Heartbleed	0.00038
9	Infiltration	0.0012
10	PortScan	5.61
11	SSH-Patator	0.2
12	Web Attack Brute Force	0.053
13	Web Attack SQL Injection	0.00074
14	Web Attack XSS	0.023

2.2 Attack Description

Botnets

An attacker gains control of several machines and servers connected to the internet and uses these servers to carry out cyberattacks against a target. Due to multiple synchronized attacking machines, the attack data volumes are huge thereby facilitating a strong attack against the target machine. Botnets are often used to carry out DDoS attacks.

Denial of Service(DoS)

Denial of Service(DoS) attacks are attacks aimed at making a server unresponsive to the requests of legitimate users. They usually work by sending enormous traffic thereby flooding the server and hitting it's resource pool by requesting resources or by sending obfuscated data that makes the server unavailable to real requests.

DDoS Attacks

Distributed Denial of Service(DDoS) attacks are DoS attacks conducted by different servers at once on a victim server. These servers have different IP addresses thus making it difficult to track down one single IP address for attacker identification that makes it hard to mitigate the attack. In the case of flooding attacks, the volume of data generated to flood the victim server is larger making the DoS attack strong and harder to terminate.

DoS:GoldenEye

DoS:GoldenEye is an attack tool that identifies vulnerabilities in a target server. It explores the capability of victim servers to form multiple HTTP connections thereby using up all possible connections on that server. This attack can be operated as a distributed attack as well. DoS GoldenEye uses Keep Alive headers and Caching Control options to keep the connection alive, preventing the target server from shutting down the connections and making the server unresponsive to non-attack requests.

DoS: HULK

DoS:HULK is HTTP Unbearable Load King attack. It is a flooding attack that floods the target servers with large volume of HTTP requests, requesting data or resources or sending unclear HTTP packets. This floods the target machine with HTTP data packets and the load is unbearable for the server that makes the server unreachable.

DoS: Slow HTTP Attacks

DoS: Slowhttptest and DoS:Slowloris are attack tools for carrying out Slow HTTP attacks. Slow HTTP attacks form multiple connections with the target server and try to keep the connections open. Some slow HTTP attacks keep the connection open by declaring a large amount of data to be sent and send the data at very large intervals of time, almost equal to the timeout period, however the server cannot close the connection and the connection does not time out. This keeps all available connections alive and keeps the server from responding to legitimate requests. Slow HTTP attacks can be done using Slow Header attacks where the packet header arrives at large intervals or Slow Body attacks where the body of the data arrives very slowly.

Brute Force

Brute Force attacks are targeted towards gaining access of authentication keys by trying all possible combinations of the key. File Transfer Protocol is a network protocol that can be used to transfer files. Users connect to the server using a FTP client using username and password authentication. Brute Force FTP attacks are an attack on the username and password. Brute Force SSH gains access to valid login credentials to authenticate a SSH access to a server by trying all possible combinations.

HeartBleed

Heartbleed could be classified as a protocol based attack where the attack utilizes a packet header field required for the transport layer security protocol that causes a machine to dump out it's entire memory, including confidential and protected data. It does so without leaving traces in the target machine, making it undetectable.

Infiltration

In an infiltration attack, the attacker gains access to a protected network or system and finds vulnerabilities in the machines or devices connected to the network. After identifying vulnerabilities, the attacker attacks the machine or device to steal private information.

Port Scan

In this kind of attack, the attacker scans the ports of the target server and sends requests to a range of ports on the target server. The attacker finds an active port on the server and exploits a known vulnerability of that service to attack the target server.

Web Attacks

Web Attacks identify weaknesses in web applications to gain access to private or protected data. These weaknesses could be exploited using data entry or using injection attacks where malicious scripts are injected into data entries of otherwise harmless websites, or by using brute force. For example, for a web application that uses an underlying SQL database and has an option for user inputs, malicious inputs could be given that cause the database to dump out confidential information. DDoS attacks are also a kind of web attacks.

2.3 Class Imbalance Problem

As we can observe from Table 2.1, the CICIDS 2017 dataset is highly unbalanced with 80% of the dataset consisting of benign data and 20% of the dataset consisting of attack data packets. The attack data is further divided unequally into fourteen different types of attacks. Because of the unequal division, certain attack classes have a very sparse representation for example, just 11 or 36 data packets throughout the entire dataset. This makes it difficult to generate realistic synthetic data to augment the number of data packets and limits model learning even when using a bootstrapped dataset for these classes of attacks. Therefore, when we are testing our proposed architecture for attack transferability, we eliminate those classes of attacks that cannot be used for transferability of learning.

2.4 Existing Machine Learning Approaches

In the literature, there are different machine learning techniques that deal with intrusion detection using flow characteristics. Alkasassbeh et al. explore MLP, Naive-Bayes, Random Forest classification algorithms for Distributed Denial of Service attack detection and show that MLP achieves the highest accuracy [2]. Lopez et al. presents a study of machine learning techniques for traffic anomaly detection and proves that Random Forest based decision classifier is the best model for anomaly detection and a Dense Neural Network is a good classifier for some types of DDoS attacks with methods to boost number of attack samples of under represented attack types [1]. Vinayakumar et al. carry out a comprehensive study of DNNs and Machine Learning classifiers that learn abstract and high dimensional

data representation using the KDDCup99 dataset and test their model performance on other datasets, such as NSL-KDD, UNSW-NB15, Kyoto, WSN-DS, and CICIDS 2017. They also propose a hybrid DNN framework which can be deployed in real time to monitor network traffic and events to detect possible network attacks [3]. Sharafaldin et al. generate a dataset consisting of benign and seven types of attack data. They also evaluate the performance of machine learning algorithms to identify the best subset of features for certain types of attacks [4]. Ferrag et al. analyzes RNNs, DNNs, restricted Boltzmann machines, Deep Belief networks, CNNs, Deep Boltzmann machines and deep autoencoders for traffic data classification using CSE-CIC-IDS2018 dataset and the Bot-IoT dataset [5].

3. HIDDEN MARKOV MODELS: A STATISTICAL MODELING BASED APPROACH

3.1 What are Hidden Markov Models?

HMM is a Markov Model in which the process being modeled is a Markov Process. Markov models are used to model processes that change stochastically. We consider that the phenomenon being modeled is a first order Markov Process where the next state of the phenomenon is only dependent on the current state of the phenomenon. This is called the Markovian Property. If q_t represents the state of a model at time instant t, then the conditional probability of it's next state will be as given:

$$P(q_t = S_{\mathrm{i}}|q_{t-1} = S_{\mathrm{j}}, q_{t-2} = S_k, ...) = P(q_t = S_{\mathrm{i}}|q_{t-1} = S_{\mathrm{j}})$$

where S_{i} is the current state and S_{j}, S_k represents the past states.

A stochastic process is a process that describes it's succession through time. HMM models a doubly stochastic process. It assumes that there are two random processes at play: one that can be observed and one that drives the observations. Our goal is to learn more about the random process that drives the observations by modeling the random process that presents as observations. We model these observations in states which are called hidden states. This makes HMMs a good predictive model for multi-stage attacks. In multistage attacks, attackers compromise a system and explore vulnerabilities in the system, and the attack progresses in stages before attacking a machine or device on the network. HMMs are particularly useful in modeling these attacks as the hidden states of the HMM are in correspondence with the stages of progression of multi-stage attacks. HMM can model this process by tuning and updating it's parameters to predict which kind of attack is in progress. We assume that the underlying random process can be characterized using parameters and that the HMM parameters can be learned precisely. The parameters that the model will learn from the training data is:

$$\lambda = (A, B, \boldsymbol{\pi})$$

where A is the Transition Probability Matrix, B is the Observation Probability Matrix and $\boldsymbol{\pi}$ is the Initial State Matrix. In our implementation of the HMM models, we use models with three hidden states. Therefore, following explanation of the parameters are with reference to three hidden states.

Initial State Matrix($\boldsymbol{\pi}$)

This matrix contains the probabilities of the HMM being in a certain hidden state when the process of observation of the stochastic process begins. Given below is the initial state matrix for a HMM with three hidden states. Initial State Matrix($\boldsymbol{\pi}$):

$$\boldsymbol{\pi}= \begin{bmatrix} \boldsymbol{\pi}_1 & \boldsymbol{\pi}_2 & \boldsymbol{\pi}_3 \end{bmatrix}$$

$\boldsymbol{\pi}_1, \boldsymbol{\pi}_2, \boldsymbol{\pi}_3$ represent the probabilities of the HMM being in State 1, State 2 and State 3 when the observations start presenting.

Transition Probability Matrix(A)

This probability matrix indicates the probability of transition from one hidden state to another. For a set of T observations, the state of the model may change when the next observation is being observed. This matrix indicates the probability of this happening. If there are three states: a_{ij} indicates the probability of transition from state i to state j. Given below is the transition probability matrix:

$$a_{\mathrm{ij}} = P(q_t = S_{\mathrm{j}} | q_{t-1} = S_{\mathrm{i}})$$

where q_t represents the present state, q_{t-1} represents the past state and S_{i} and S_{j} are the states.

$$\mathrm{A}= \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

Observation Probability Matrix(B)

The observation probability matrix represents the probability of observing a certain observation given that the model is already in a hidden state. b_{ij} represents the probability of observing the j-th observation given that the model is in the i-th hidden state. Let the observations be

$$O = \{O_1, O_2, O_3, ..., O_T\}$$

,

The observation probability matrix for T observations and three hidden states is as follows:

$$\text{B}=\begin{bmatrix} b_{11} & b_{12} & .... & b_{1T} \\ b_{21} & b_{22} & .... & b_{2T} \\ b_{31} & b_{32} & .... & b_{3T} \end{bmatrix}$$

During the training stage, each HMM tunes it's parameters to maximize the expectation of $P(O|\lambda)$ for a set of observations belonging to one class of data. For a multi-class classification problem, there are as many number of trained HMMs as there are classes and each HMM is trained with one class of data. We assume that the model is able to tune it's parameter to detect that class of data during testing. During the testing phase, a set of T observations is fed to all the trained models and we identify the model that yields the maximum $\text{P(O}|\lambda)$. The class of data that this model is trained with is the predicted label of the testing observations.

3.2 Implementation

In this problem, the observations are the features corresponding to a data sample. 40 flow features from the dataset are taken for each data sample and is fed to each HMM as 40 observations. Each CSV file in the dataset is treated as a separate dataset. We evaluate the performance of HMMs on a 2 class and 4 class classification problem. In those cases, we use 2 HMMs and 4 HMMs respectively for training. T observations are taken at a time from the testing dataset and each model is tested with the T observations. The model that generates the highest probability $\text{P(O}|\lambda)$ implies that the set of observations O belong to that HMM and class as seen in Figure 3.2.

Computation of $\text{P(O}|\lambda)$ using hidden states. Let Q be the set of hidden states:

$$O = \{O_1, O_2, O_3, ..., O_T\} \text{ where T=40.}$$

$$P(O|\lambda) = P(\{O_1, O_2, O_3, ..., O_T\}|\lambda) = \sum_Q P(O|Q, \lambda)P(Q|\lambda)$$

where Q is the set of all possible hidden states. To compute $P(O|\lambda)$ using the method above, the complexity is of the order $O(2TN^T)$ for a general system with N states and T observations. To compute the same quantity, a forward-backward algorithm is used. This reduces the computational complexity to $O(N^2T)$.

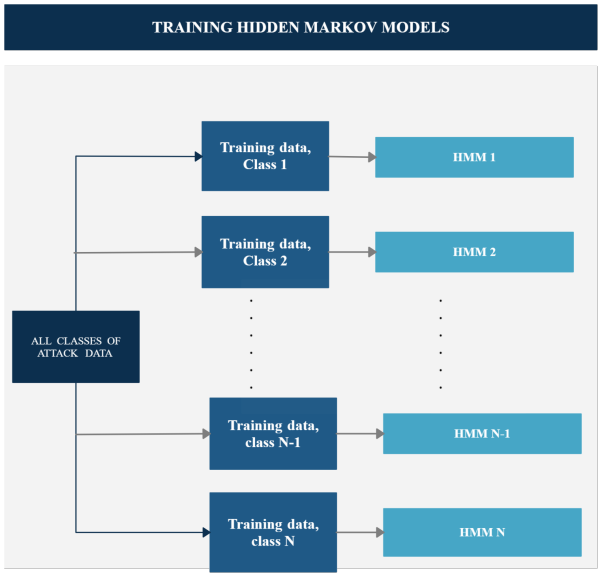


Figure3.1. Training stage: HMM

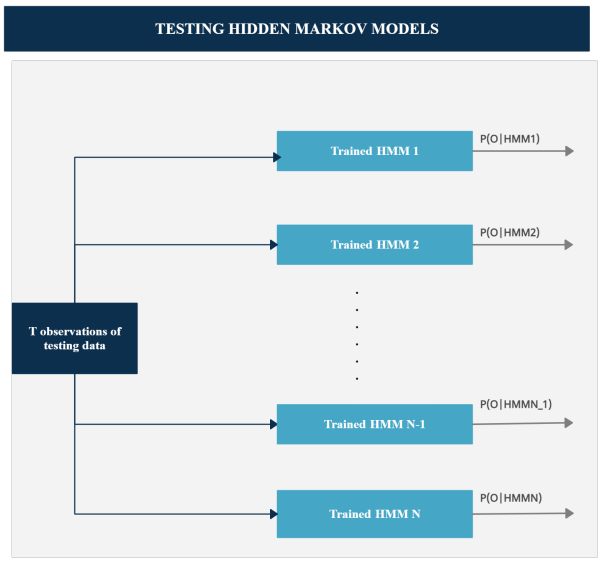


Figure3.2. Testing stage: HMM

3.3 Results

To observe if the statistical modeling based approach scales better than the deep learning approach, we compare the performance of the HMM with a candidate CLDNN model (Table 3.1). We observe that the CLDNN model is able to achieve an overall accuracy of 99.69% for 2 classes of data and 99.57% for 4 classes of data whereas the HMM model is only 78.7% accurate for 2 classes of data and 50.8% accurate for 4 classes of data. This could be because not all the attacks in the dataset are multi-stage attacks that correspond to the hidden states of the HMMs. The performance of the HMM model deteriorates as the number of classes in the dataset is increased.

Comparison between HMM performance and CLDNN performance

Table3.1. HMM vs CLDNN

Number of Classes of data in dataset	**HMM accuracy**	**CLDNN accuracy**
2 Classes	78.7%	99.69%
4 Classes	50.8%	99.57%

We conclude from Table 3.1 that the deep learning approach shows more promise for the multi-class classification problem. Therefore, we explore more deep learning architectures in the sections to follow to identify the best architecture to use to study attack transferability patterns. Architecture details of the used candidate CLDNN model is given in the next section.

4. NEURAL NETWORK ARCHITECTURES

4.1 Motivation

We test Deep Neural Network architectures to identify the best architecture to use to study attack correlations. The first two architectures that we test are a CLDNN and CNN with BGRU layers. These neural networks are part of a class of neural networks called CRNN which is CNN with Recurrent Layers. We use convolutional layers because they capture spatial features from the dataset and gated RNNs to capture temporal patterns in the sequence. Bidirectional GRUs capture temporal patterns not only forward in time but also backward in time. In addition, we also evaluate the performance of a One Class Neural Network(OCNN) that is good for minority data detection in large datasets. Finally, we evaluate the performance of a Convolutional Neural Network without Recurrent layers.

4.2 CLDNN

4.2.1 Architecture

The Convolutional LSTM Deep Neural Network architecture has 6 hidden layers. The first layer is a convolutional layer with 256 feature maps, and a $(1, 3)$ convolution kernel and 20% dropout. The second hidden layer is a convolutional layer with 256 feature maps, and a $(2, 3)$ convolution kernel. The third layer is a convolutional layer with 80 feature maps and a $(1, 3)$ kernel with 20% dropout. The fourth hidden layer is a convolutional layer with 80 feature maps and a $(1, 3)$ kernel. The fifth hidden layer is an LSTM layer with 50 cells. The sixth hidden layer is a fully connected layer with 128 neurons. All hidden layers have Rectified Linear Unit(ReLU) activation as:

$$g(x) = max(0, x)$$

We use this architecture to evaluate it's performance on: i) The multi-class classification problem when it is trained with all classes of data and tested on all classes of data and ii) The two-class classification problem when it is trained with one class of data and tested on all other classes of data. For the multi-class classification problem, the output layer has

Table4.1. CLDNN Architecture

CLDNN Architecture
Conv2D 1x3x256, ReLU, Dropout(rate=0.2)
Conv2D 2x3x256, ReLU
Conv2D 1x3x80, ReLU, Dropout(rate=0.2)
Conv2D 1x3x64, ReLU
LSTM 50
Dense, 128, ReLU
Dense, 2, Softmax

15 output classes. For the two class classification problem, the output layer has 2 output classes. In both implementations, the output layer has softmax activation.

Softmax activation can be given as:

$$\sigma_{\mathrm{i}}(z) = \frac{\mathrm{e}^{z_{\mathrm{i}}}}{\sum_{\mathrm{j=i}}^{J} \mathrm{e}^{z_{\mathrm{j}}}} \text{ , where } J \text{ is the total number of classes.}$$

4.2.2 Implementation

All 78 features are used for training. For the two-class classification problem, we train our model on one kind of attack and evaluate it's performance when tested on all the other attacks. In this case, all benign data is labeled zero and all attack data is labeled one. For the multi-class classification problem, the model is trained with all 15 classes of data. The total dataset is divided into a fifty-fifty split, 50% of the data is used for training and 50% of the data is used for testing. The loss function used is Categorical CrossEntropy and the optimizer is Adam. The batch size is 1024. This model is trained for 50 epochs. Categorical CrossEntropy loss function is given by:

$$L_{CE} = -\frac{1}{N} \sum_{\mathrm{i}} t_{\mathrm{i}} log(p_{\mathrm{i}})$$

where t_{i} is the true label of the i^{th} data point, p_{i} is the predicted probability of the data point belonging to class t_{i} and N is the batch size.

4.2.3 Results

Metrics

Attack Accuracy

Other than the overall accuracy metric, we use an attack accuracy metric to evaluate the performance of a model. We use this metric because the number of benign packets is much larger than the attack data. A gross misclassification of the attack data packets does not diminish the overall accuracy considerably. A more accurate metric of performance evaluation would be the percentage of correctly classified attack data. Therefore, we define the attack accuracy as the ratio of the number of correctly classified attack packets in a class to the total number of attack packets in that class. This is also called true positive rate more generally.

$$\text{Attack Accuracy} = \frac{\text{Number of correctly classified attack packets in class i}}{\text{Total number of attack packet packets in class i}}$$

Confusion Matrices

1) Result of the CLDNN Architecture on the Multi-class Classification Problem

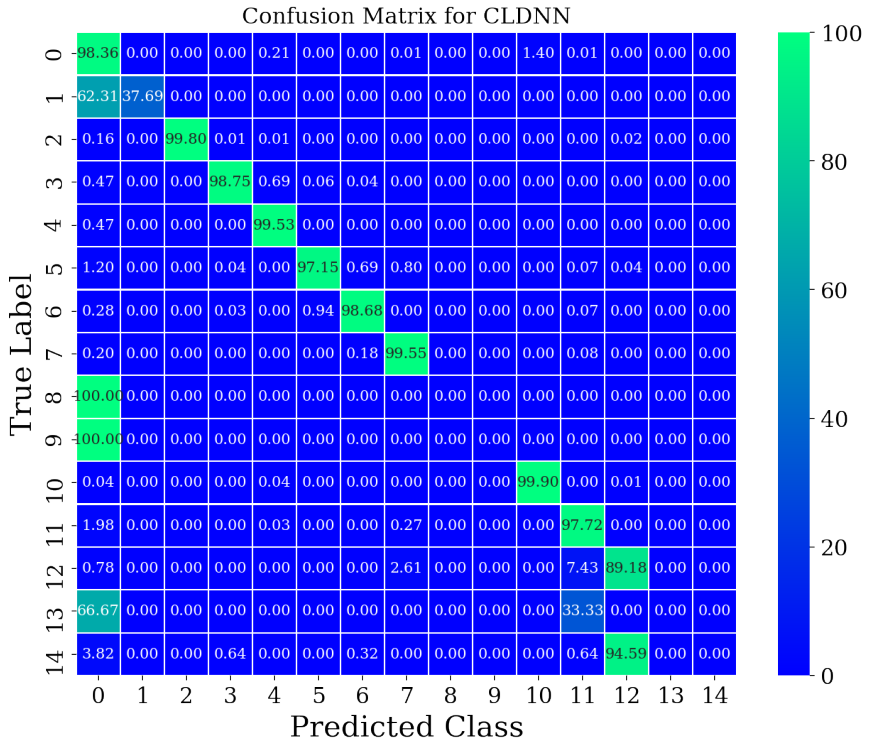


Figure4.1. Attack Accuracy for CLDNN multi-class classification problem

Figure 4.1 is the confusion matrix of the performance of this CLDNN architecture. The labels on the Y-axis of the matrix indicate the true labels of the testing data and the labels on the X-axis of the matrix indicate the predicted labels of the testing data. For a model with a good performance, we would observe high accuracies along the diagonal of the confusion matrix which indicate that the model correctly classifies the attack that it is trained with, with a high accuracy. We observe that our CLDNN architecture achieves an overall accuracy of 98.53%. For most classes, the model is able to correctly classify the attacks. However, it can be observed that although the model is being trained on data from Attacks 8 (HeartBleed) , 9 (Infiltration), and 13 (Web Attacks: SQL Injection), it is not able to correctly classify them with a good attack accuracy. This is largely because the Attacks

8, 9 and 13 are very under represented in the dataset. It could also be that there are not a lot of dominant features that influence the learning of the model. In addition, it is also a possibility that the important features do not have a large variance in its distribution. This makes it hard for the model to learn patterns to differentiate between the classes of data.

2) Result of the CLDNN Architecture on the Two-class Classification Problem

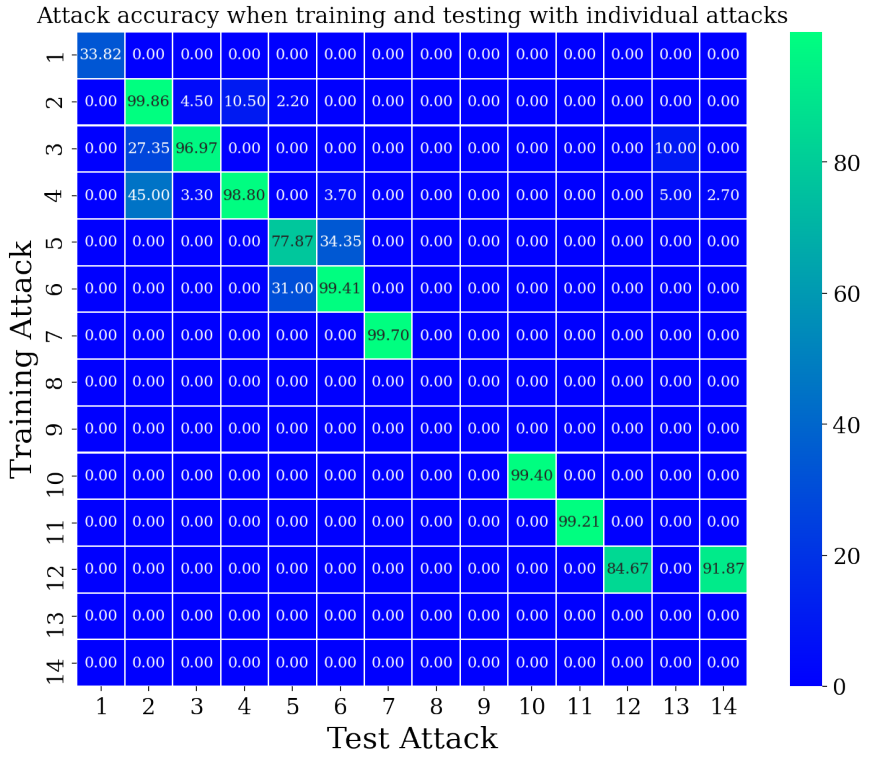


Figure4.2. Attack Accuracy for CLDNN for two-class classification problem

Figure 4.2 is the confusion matrix for the two-class classification problem. In this confusion matrix, the labels on the Y-axis indicate the attack with which the model has been trained with. The labels on the X-axis represent the class of attack that the model is tested on. For a model trained on one attack, it is tested on all other attacks for transferability. For a model that exhibits attack transferability, more off the diagonal correlations are preferred. High classification accuracies on the diagonal represent the performance of the model when it is trained and tested on the same attack. However, to observe transferability in learning, we would want to observe high accuracies when the model is tested on a attack class that it

has not been trained on. We can observe that training this CLDNN model with one attack does not correlate to any other attack. Most of the correlations observed are on the diagonal of the confusion matrix, which means the model only performs well when tested on the training attack. It exhibits low transferability like for example, when the model is trained with Attack 12, it can identify attacks in class 14 with 91.87% accuracy. It does not exhibit other strong off the diagonal correlations. Therefore, we conclude from these results that transferability of learning cannot be tested using this architecture.

4.3 CNN with BGRU layers and CNN with stacked BGRU layers

4.3.1 Architecture and Implementation

Gated Recurrent Units are different from traditional RNNs without gates as they are able to control the amount of previous data to carry over to the next iteration using hidden states. GRUs usually have a reset gate and update gate as opposed to a forget gate, input gate and output gate as in LSTMs. This makes training the model with GRUs faster than using LSTMs. BGRUs have connections that go backward in time, enabling them to capture temporal patterns backward and forward in time. The CNN with Bidirectional Gated Recurrent Units and Stacked Bidirectional Gated Recurrent Units architecture is the same as the architecture of the CLDNN used, except in these architectures there are BGRU and stacked-BGRU layers in place of the LSTM layer. Each BGRU layer has 50 cells. For a stacked-BGRU approach, there are two BGRU layers consecutively with 50 cells in each layer. Each model is trained for 150 epochs, with a batch size of 1024. The loss function used is mean squared error and the optimizer used is RMSProp. Table 4.2 and Table 4.3 are the architectures of these two DNNs.

Table4.2. CNN with BGRU Architecture

CNN with BGRU Architecture
Conv2D 1x3x256, ReLU, Dropout(rate=0.2)
Conv2D 2x3x256, ReLU
Conv2D 1x3x80, ReLU, Dropout(rate=0.2)
Conv2D 1x3x64, ReLU
BGRU 50
Dense, 128, ReLU
Dense, 2, Softmax

Table4.3. CNN with Stacked BGRU Architecture

CNN with Stacked BGRU Architecture
Conv2D 1x3x256, ReLU, Dropout(rate=0.2)
Conv2D 2x3x256, ReLU
Conv2D 1x3x80, ReLU, Dropout(rate=0.2)
Conv2D 1x3x64, ReLU
BGRU 50
BGRU 50
Dense, 128, ReLU
Dense, 2, Softmax

4.3.2 Results

These results present a comparative study of the performance of the CLDNN, CNN with BGRU layer and CNN with stacked BGRU layers for select attacks for the two-class classification problem. We also evaluate the performance of a CNN with a BGRU layer with 10 cells. We evaluate the attack accuracy when tested on each attack. All the entries are attack accuracies in terms of percentages.

Table4.4. Comparison of performance of LSTM, BGRU and stacked BGRU architectures on all testing attacks when trained with Attack 4(DoS:HULK)

Testing Attack	1	2	3	4	5	6	7	8	9	10	11	12	13	14
LSTM	0	45	3.3	98.8	0	3.7	0	0	0	0	0	0	0	2.7
BGRU(10)	0	46	0	98.33	0	0	0	0	0	0	0	0	0	2.6
BGRU(50)	0	43.7	47	98.79	9	2.1	0	0	0	0	0	0	0	3
StackedBGRUs	0	47.7	31.3	99	3.2	0	0	0	0	0	0	0	0	3

Table4.5. Comparison of performance of LSTM, BGRU and stacked BGRU architectures on all testing attacks when trained with Attack 2(DDoS)

Testing Attack	1	2	3	4	5	6	7	8	9	10	11	12	13	14
LSTM	0	99.86	4.5	10.5	2.2	0	0	0	0	0	0	0	0	0
BGRU(10)	0	99.81	0	3.3	4	0	0	0	0	0	0	0	0	0
BGRU(50)	0	99.79	0	0	4.4	0	0	0	0	0	0	0	0	0
Stacked BGRU	0	99.84	0	0	4.1	0	0	0	0	0	0	0	0	0

Table4.6. Comparison of performance of LSTM, BGRU and stacked BGRU architectures on all testing attacks when trained with Attack 3(DoS:Golden-Eye)

Testing Attack	1	2	3	4	5	6	7	8	9	10	11	12	13	14
LSTM	0	27.35	96.97	0	0	0	0	0	0	0	0	0	0	0
BGRU(10)	0	0	95.66	0	0	0	0	0	0	0	0	0	0	0
BGRU(50)	0	11	97	0	0	0	0	0	0	0	0	0	0	0
Stacked BGRU	0	30.7	96.5	0	0	0	0	0	0	0	0	0	0	0

Table4.7. Comparison of performance of LSTM, BGRU and stacked BGRU architectures on all testing attacks when trained with Attack 5(DoS:Slowhttptest)

Testing Attack	1	2	3	4	5	6	7	8	9	10	11	12	13	14
LSTM	0	0	0	0	77.87	34.35	0	0	0	0	0	0	0	0
BGRU(10)	0	3.2	0	0	98.83	35.9	0	0	0	0	0	0	0	0
BGRU(50)	0	0	0	0	76	34.16	0	0	0	0	0	0	0	0
Stacked BGRU	0	0	0	0	90.56	34.7	0	0	0	0	0	0	0	0

Table4.8. Comparison of performance of LSTM, BGRU and stacked BGRU architectures on all testing attacks when trained with Attack 6(DoS:Slowloris)

Testing Attacks	1	2	3	4	5	6	7	8	9	10	11	12	13	14
LSTM	0	0	0	0	31	99.41	0	0	0	0	0	0	0	0
BGRU(10)	0	0	0	0	33	99.44	0	0	0	0	0	0	0	0
BGRU(50)	0	0	0	0	31.13	98.24	0	0	0	0	0	0	0	0
Stacked BGRU	0	0	0	0	28.91	98.24	31	0	0	0	0	0	0	0

From Table 4.4, 4.5, 4.6, 4.7, 4.8, the models yield high accuracies only when tested on the training attack. Small correlations can be studied for example when the models are trained with Attack 5(DoS:Slowhttptest) and tested on Attack 6(DoS:Slowloris) and vice versa. However, the prediction accuracies are still low(about 28%-35%) which are not sufficient to conclude that those attacks have a strong correlation. We can also tell from Table 4.4 that training the model with Attack 4(DoS:HULK) has the potential to scale for Attack 2(DDoS) and Attack 3(DoS:Hulk) but this model only detects those attacks with 31%-47% accuracy. We conclude from this comparative study that the architectures explored so far do not exhibit very strong attack transferabilities. We explore another deep learning architecture, a One Class Neural Network in the next section and evaluate it's performance for anomaly detection.

4.4 One Class Neural Networks

4.4.1 What are One Class Neural Networks?

One Class Neural Networks(OCNN) are used for anomaly detection in complex datasets that require highly non linear decision boundaries. The loss function of OCNNs is derived from the loss function of OC-SVMs. In an OC-SVM, for input data X with N input samples $\{X_{\mathrm{i}}\}_{\mathrm{i}=1}^{N}$. A $\phi(X_n)$ which is a reproducing kernel Hilbert space that is a mapping from the input space to feature space. All the input data points are labeled one and the only negative point is the origin. A hyperplane separates the origin from the mapped $\phi(X_n)$s. The hyperplane in the feature space is given by $f(X_n) = w^T\phi(X_n) - r$. v is the parameter to control the trade off between maximizing the distance and number of data points falsely classified as positive. r is the hyperplane bias. A OCNN has a feedforward neural network with one hidden layer and one output node. Here, w is the scalar output from the hidden to output layer and V is the weight matrix from the input to hidden layer. The hidden layer has a linear or sigmoidal activation given by $g(.)$

The optimization problem for OCNNs is:

$$\min_{w,V,r} \tfrac{1}{2}\|w\|_2^2 + \tfrac{1}{2}\|V\|_F^2 + \tfrac{1}{v} \cdot \tfrac{1}{N} \textstyle\sum_{n=1}^{N} max(0, r - \langle w, g(VX_n)\rangle) - r$$

(w, V) are updated using normal backpropagation. The model is trained using important features extracted from an autoencoder instead of using the raw data. The OCNN algorithm is implementation from Chalapathy et al [6]:

Algorithm 1 OC-NN algorithm

Input:A set of points X_n, n=1,...,N

Output:A set of decision scores $S_n = \hat{y_n}$, n=1,....,N for X

Initialise $r^{(0)}$

$t \leftarrow 0$

while *(no convergence achieved)* **do**

    Find $(w^{(t+1)}, V^{(t+1)})$

    $r^{t+1} \leftarrow v^{th}$ quantile of $\{\hat{y_n}^{t+1}\}_{n=1}^{N}$

    $t \leftarrow t + 1$

Compute decision score $S_n = \hat{y_n} - r$ for each X_n

if $S_n \geq 0$ **then**

    X_n is a normal point

else

    X_n is an anomalous point

return $\{S_n\}$

4.4.2 Metrics

1. **True Positive Rate(TPR)/Sensitivity**

   Also called sensitivity, True Positive Rate is the proportion of true positive classifications among all the positive data points. The best True Positive Rate is 1 which indicates that all positive data points were classified correctly and the worst is 0 which indicates that all positive data points were incorrectly classified.

$$\text{TPR}=\frac{\textbf{Number of True Positives}}{\textbf{Number of True Positives+Number of False Negative}}$$

2. **False Positive Rate(FPR)**

   The False Positive Rate is the proportion of negative data points classified falsely as positives. The best FPR is 0 which indicates that none of the truly negative data points were misclassified as positive and the worst FPR is 1 which indicates that all the negative data points were misclassified as positives.

$$\text{FPR}=\frac{\textbf{Number of False Positives}}{\textbf{Number of True Negatives + Number of False Positives}}$$

3. **ROC curve**

Receiver Operating Characteristics(ROC) is a plot of TPR vs FPR for different decision thresholds. Area Under ROC (AUROC) is a measure of model performance. Higher area under the curve implies better classification accuracy for the model. Ideally the ROC curve would look like a step function with a true positive rate of 1 and a false positve rate of 0 which achieves a maximum AUROC of 1. However, in realistic systems, achieving an AUROC of above 0.7-0.8 is considered an acceptable model performance. AUROC of 0.5 is as good as random guessing. The black dotted line in Figure 4.3 indicates an area of 0.5 under the curve. Higher AUROC values indicates better model performance.

Example ROC curve generated using the classes of the iris dataset

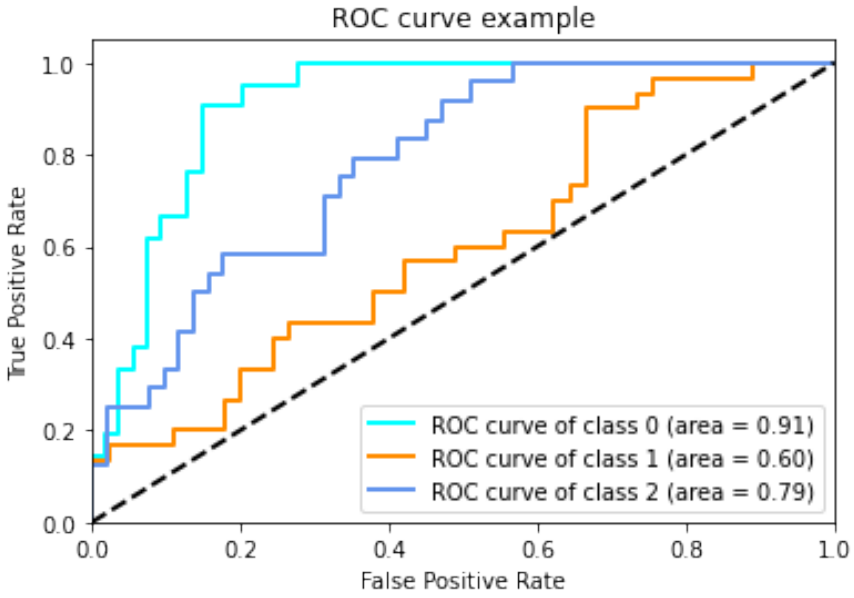


Figure4.3. Example ROC curve

4.4.3 Implementation and Results

The implementations is the implementation by Chalapathy et al [6]. The raw data is passed through an autoencoder to find important features. It is then passed through a one

layer NN and the function it optimizes is given above. For CICIDS dataset, the model AUROCs were computed for different seed values.

Table4.9. AUROC values for different seed values

AUROC values for different seed values
0.6047022068674568
0.6040465887906412
0.5290125251557807
0.48844116825440265
0.549943844886464
0.6398034285974904
0.5154925402064912
0.561410034947294
0.6227060493594897
0.5054273774706824

From Table 4.9, we can observe that the model performs as good as random guessing in 30% of the runs, marginally better than random guessing in 60% of the runs and worse than random guessing in 10% of the runs. On an average, it achieves an AUROC of 0.55. Therefore, we conclude from the AUROC values that this model does not cannot detect anomalies with a high accuracy and would not be suitable for the two-class classification task at hand. We evaluate the performance of a Convolutional Deep Neural Network in the next section.

4.5 CNN

4.5.1 Architecture and Implementation

The Convolutional Deep Neural Network architecture consists 5 hidden layers. The first hidden layer is a Convolutional layer with 256 feature maps, and a $(1, 3)$ convolution kernel. The second hidden layer is a convolutional layer with 256 feature maps, and a $(2, 3)$ convolution kernel. The third hidden layer is a convolutional layer with 256 feature maps and a $(1, 3)$ convolution kernel and dropout of 20%. The fourth hidden layer is a convolutional layer with 80 feature maps and a $(1, 3)$ convolution kernel. The fifth hidden layer is a fully connected layer with 128 neurons. The output layer has 2 output nodes and

Table4.10. CNN Architecture

CNN Architecture
Conv2D 1x3x256, ReLU
Conv2D 2x3x256, LeakyReLU(0.3)
Conv2D 1x3x256, LeakyReLU(0.3), Dropout(rate=0.2)
Conv2D 1x3x80, ReLU
Dense, 128, ReLU
Dense, 2, Sigmoid

sigmoid activation. Layers 1,4 and 5 have ReLU activation and layers 2 and 3 have Leaky ReLU activation with $\alpha = 0.3$. All 78 features from the dataset were used for training. This model's performance is evaluated on the mutli-class classification problem.

Leaky ReLU can be given as:

$$g(x) = max(0, x) + \alpha min(0, x),\ \alpha \text{ is a small value}$$

Sigmoid activation can be given as:

$$\sigma_{\mathrm{i}}(z) = \frac{1}{1+\mathrm{e}^{-z}}$$

The loss function used is Binary Cross Entropy(BCE).

$$L_{BCE} = -\frac{1}{N}\sum_{\mathrm{i}=1}^{N}(t_{\mathrm{i}} * log(p_{\mathrm{i}}) + (1 - t_{\mathrm{i}}) * log(1 - p_{\mathrm{i}}))$$

where t_{i} is the true label, p_{i} is the probability of the i^{th} data point having a predicted label of t_{i} and N is the batch size.

The model is trained on 70% of data and tested on 30% of the data. 10% of the training data is used as validation data. The model is trained for 100 epochs.

4.5.2 Results

The overall accuracy of the CNN is 99.65% whereas the CLDNN achieved an overall accuracy of 98.53%.

Result of CNN Architecture on multi-class classification problem

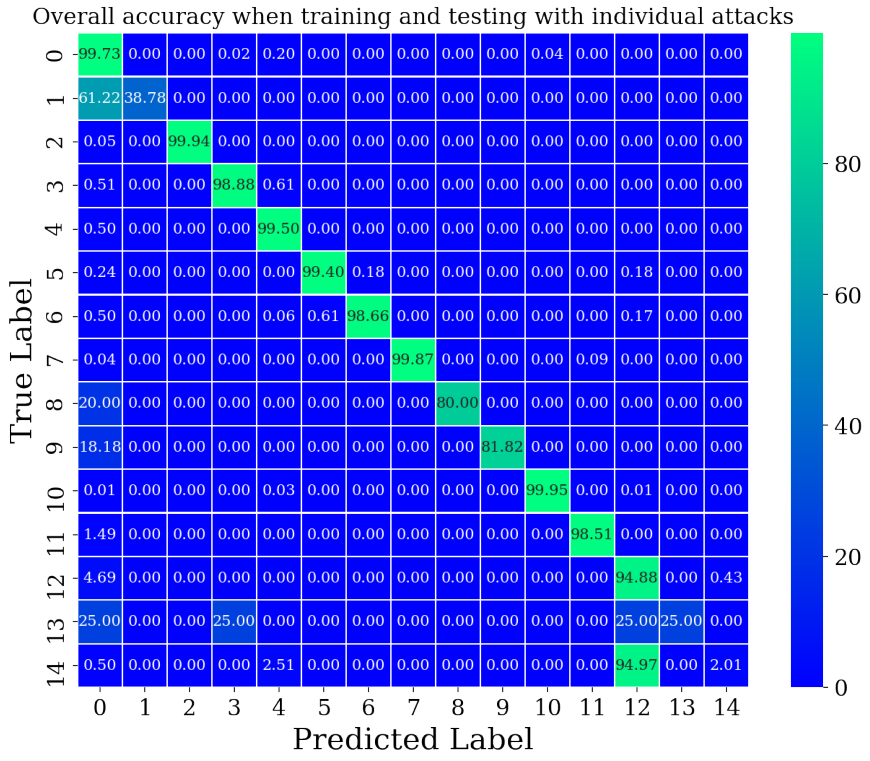


Figure4.4. Results of CNN Architecture on multi-class classification problem

In the confusion matrix, the diagonal elements represent the percentage of data in each class correctly classified. We observe from the confusion matrix that the model is able to correctly classify Attack 8(HeartBleed) and Attack 9(Infiltration) that have 11 and 36 data packets in the dataset upto 80% and 81.82% accuracy. Thus this model is able to learn the important features from a high dimensional feature set. We conclude that this is this architecture has performed the best and is the best architecture for studying attack transferability. This architecture does not have any recurrent layers thereby speeding up training time while increasing classification accuracy by 1.12%. This is our proposed base architecture.

5. ATTACK CORRELATIONS

5.1 Base Architecture Trained on Original Dataset

5.1.1 Implementation

In this section, we study the transferability of learning using our base architecture. We train the CNN architecture with one attack and test on all the other attacks. To implement this, we consider this as a two-class classification problem. We label all attack data as one class and all benign data as another class. We train the model on benign data and attack data corresponding to the attack whose transferability we want to study. As seen in Table 2.1, attacks like Attack 8 (HeartBleed), Attack 9 (Infiltration) and Attack 13 (Web Attack: SQL Injection) are severely under represented in the dataset. We address this using a synthetic and bootstrapped dataset in the next section. In this section, the models are trained with only the original dataset. In the confusion matrices in results section, Attack 8, 9, and 13 are not included as there are not enough original samples in the dataset for transferability in learning. The model is trained for 20 epochs for each attack, the loss function used is Binary CrossEntropy and the optimizer used is Adam.

5.1.2 Results of Training with Original Dataset

Figure 5.1 represents the overall accuracy of the base architecture. Figure 5.2 is the confusion matrix for when the base architecture is tested for the two-class classification task. The diagonal elements are placeholders as they indicate the performance of the model when tested on the training attack. From Figure 4.4, we have observed that the model performs with high accuracies when trained and tested on the same attack. We observe transferability in the off diagonal elements as:

1. Training with DoS:Golden Eye(Attack 3) scales for DDoS attacks(Attack 2), DoS:HULK(Attack 4), DoS:Slowhttptest(Attack 5) and DoS:Slowloris(Attack 6).

2. Training with DDoS Attack(Attack 2) scales for DoS:HULK(Attack 4) and vice versa.

3. Training with DoS:Slowhttptest(Attack 5) scales for DoS:GoldenEye(Attack 3) and DoS:Slowloris(Attack 6).

4. Training with DoS:Slowloris(Attack 6) scales for DoS:GoldenEye(Attack 3), DoS:HULK(Attack 4) and DoS:Slowhttptest(Attack5).

Result of the CNN Architecture when trained with one attack and benign data, tested on all attacks(Average Overall Accuracy)

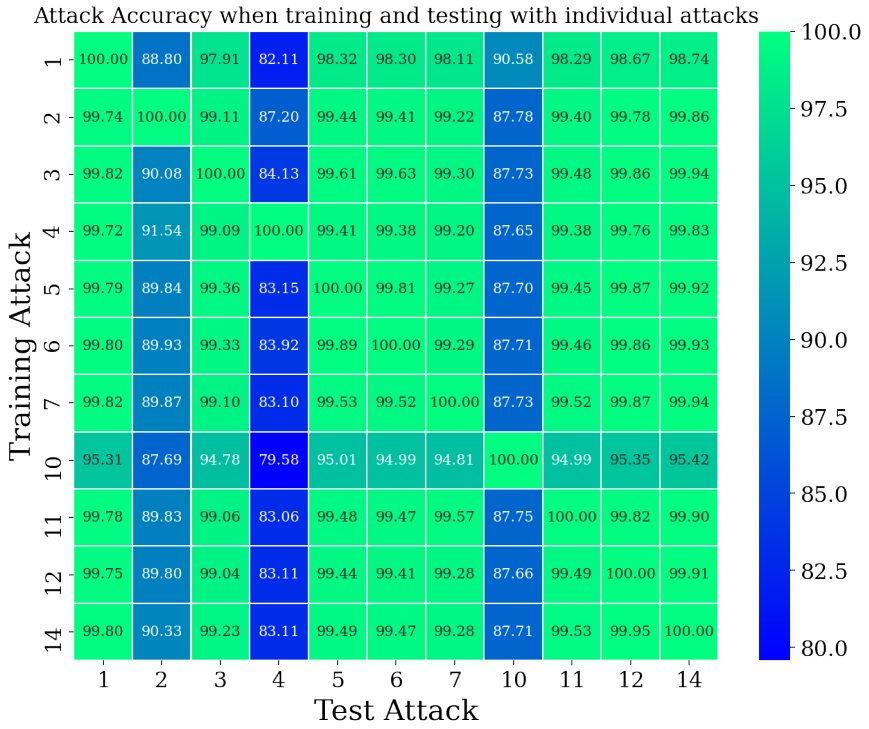


Figure5.1. Overall Accuracy for the base architecture for the two-class classification problem

Result of the CNN Architecture when trained with one attack and benign data, tested on all attacks(Average Attack Accuracy)

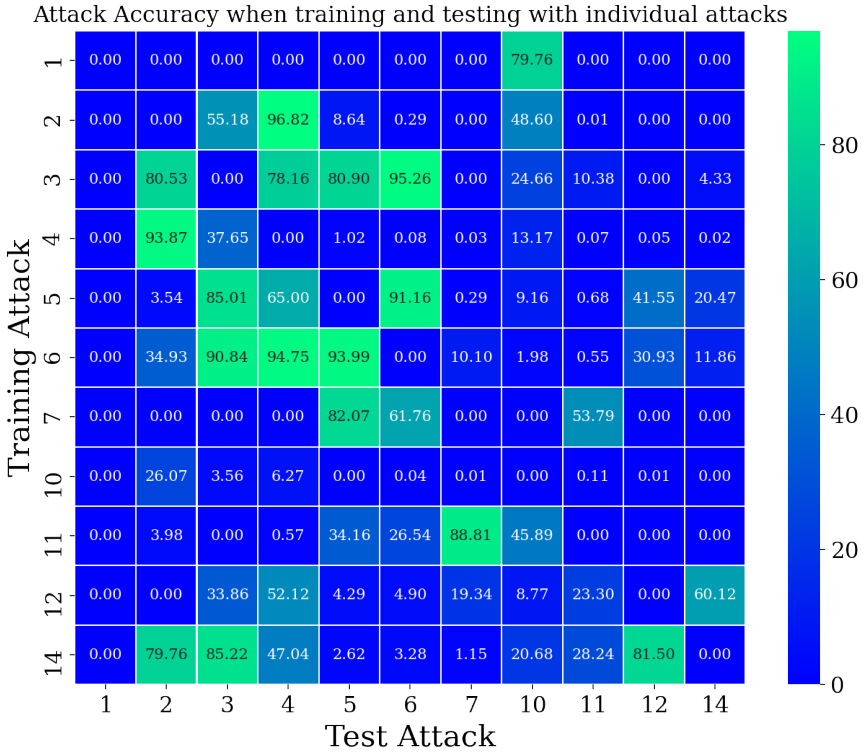


Figure5.2. Attack Accuracy for the base architecture for the two-class classification problem

Table5.1. Observed Correlations

Training Attack	**Correlated Attacks**
DDoS(2)	DoS:HULK(4)
DoS:GoldenEye (3)	DDoS(2),DoS:Hulk(4),DoS:Slowhttptest(5),DoS:Slowloris(6)
DoS:Hulk(4)	DDoS(2)
DoS:Slowhttptest(5)	DoS:GoldenEye(3),DoS:Slowloris(6)
DoS:Slowloris(6)	DoS:GoldenEye(3),DoS:HULK(4),DoS:Slowhttptest(5)

5.2 Base Architecture Trained on Synthetic and Bootstrapped Dataset

5.2.1 Motivation

The CICIDS 2017 dataset is unbalanced, with 80% of the data samples being benign data packets and 20% of the data samples belonging to attack data. The total attack data is further divided into fourteen classes of attacks thus each class is severely under represented in the dataset which is not ideal for learning. Our goal in this section is to test the model for strong and consistent attack correlations when it is trained with more number of samples from the minority attack classes.

5.2.2 Synthetic Minority Oversampling Technique(SMOTE)

We generate more number of attack data per class using Synthetic Minority Oversampling Technique(SMOTE). SMOTE selects examples that are close to each other in the feature space. This technique selects one point from the minority attack class and uses K nearest neighbours algorithm to find the K nearest data points around the randomly chosen point. The number K is chosen based on how much we want to oversample the minority classes. In our implementation, we take 5 nearest neighbours. It then randomly selects one neighbour point out of the K neighbours and creates a synthetic data point on the line joining the two selected points. Using this technique, we identify specific regions in the feature space that can be used to generate more samples belonging to a certain class of data. The number of attack packets generated using this technique is used to match the number of benign data packets in the dataset. This process is repeated for each attack. The model trained with this dataset is tested on: 1) SMOTE generated benign data and real attack data for evaluation of accuracy when tested on a synthetically generated benign dataset and 2) Real benign data and real attack data as this scenario best emulates a real life scenario.

5.2.3 Bootstrapped Dataset

Another technique used to create a balanced training dataset is for every attack that the model is trained with, the attack data is resampled to match the number of benign data

packets. This training dataset is bootstrapped dataset. This process is used to replicate attack data corresponding to all attacks. The base architecture is trained with this enhanced dataset and tested on real attack data. During training, it is validated on 20% of the attack data. During training with the SMOTE dataset as well as bootstrapped dataset, the model is trained for 20 epochs per training attack, the loss function used is Binary CrossEntropy, the optimizer is Adam.

5.2.4 Results of Training with Synthetic and Bootstrapped Dataset

Figure 5.3, 5.4 and 5.5 are the confusion matrices of the base architecture when trained on the SMOTE dataset and bootstrapped dataset and tested on SMOTE and real attack data. From Figure 5.2, we can observe that most of the attack pairs that showed a correlation when the model was trained with the original training data also show correlations when the model is being trained with synthetic and bootstrapped data. Like the previous section, the diagonal elements in the confusion matrices are placeholders as they represent the accuracies when training and testing on the same attack that we have already observed scales well with accuracies above 90% from Figure 4.4.

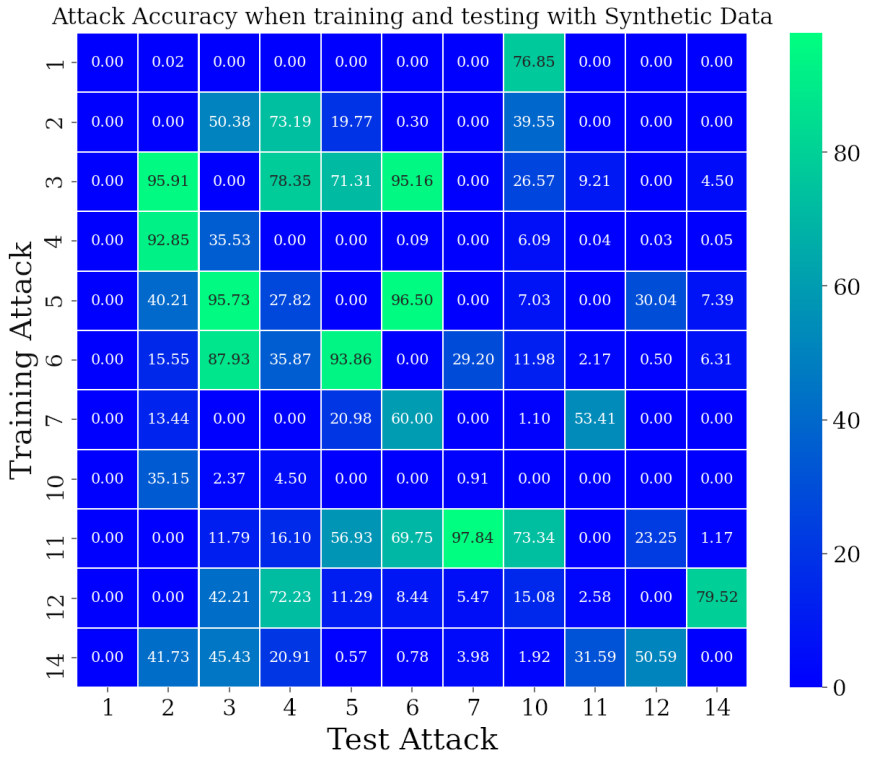


Figure5.3. Confusion Matrix for Base Architecture trained on SMOTE generated attack data and tested on SMOTE generated benign data and real attack data

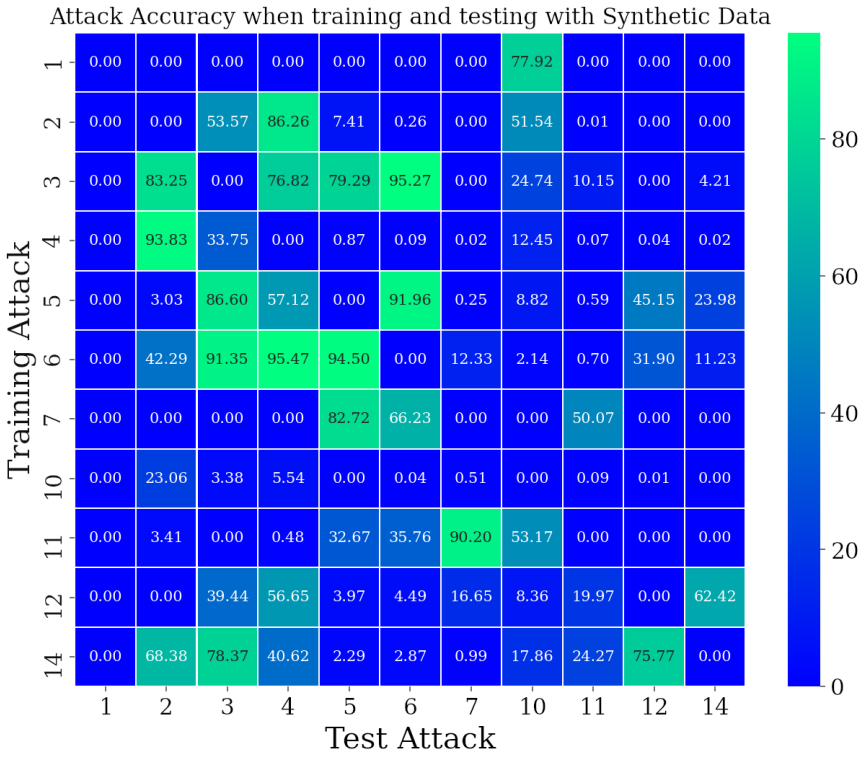


Figure5.4. Confusion Matrix for Base Architecture trained on SMOTE generated attack data and tested on real benign data and real attack data

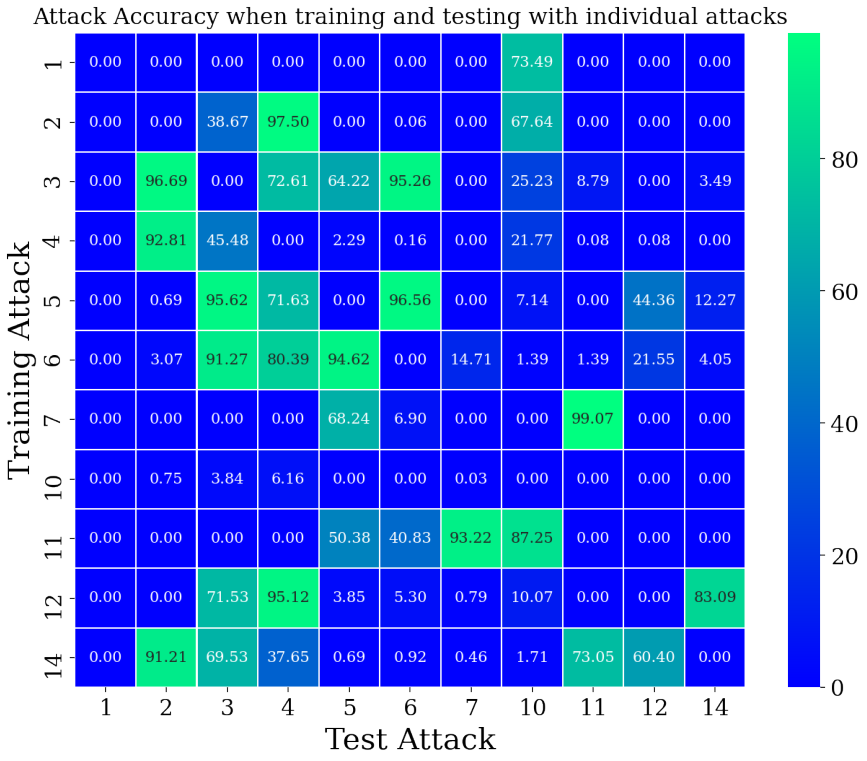


Figure5.5. Confusion Matrix for Base Architecture trained on bootstrapped attack data and tested on real benign data and real attack data

5.3 Attack Boosting Algorithm

5.3.1 Implementation

In the previous sections, we have explored the transferability in learning when the model is trained with a single attack and tested on other attacks. In this section, we aim to identify a subset of training attacks that can be used to train the model to cause it to scale for other attacks. We explore the possibility of the model scaling for a certain test attack when it has been trained with a combination of training attacks (that do not include the testing attack). We aim to observe consistent and strong off the diagonal correlations and identify training attacks. To do this, we use the attack boosting algorithm as outlined in Algorithm 2. 50% of the set is used only as training data for selecting the training attacks and the model performance is evaluated on the remaining 50% of the set that serves as the validation data. The testing data consists of the attack for which we want to select a subset of training attacks

along with benign data. The loss function used is Binary CrossEntropy and optimizer used is Adam.

Algorithm 2 Attack Boosting Algorithm

Result: Selected Training Attack List

while *Accuracy increase=1 and remaining attack list is not empty* **do**

  for *single attack in remaining attack list* **do**

    append single attack data to the current training set (without updating current training set), train model and evaluate on validation set

    if *accuracy > appending accuracy* **then**

      appending attack=single attack

    end

  end

  if *appending accuracy > current accuracy* **then**

    current accuracy=appending accuracy

    update selected attacks list

    remove appending attack from *remaining attack list*

    update *Accuracy increase*

  end

end

5.3.2 Metrics

We define some metrics for performance measurement used for evaluating the performance of the attack boosting algorithm.

1. **Positive Rate(PR)**

   Positive Rate is the percentage of true positive classifications among all positively classified data packets.

$$\text{Positive Rate}=\frac{\textbf{Number of True Positives}}{\textbf{Number of True Positives+Number of False Positives}}$$

2. **Negative Rate(NR)**

   Negative Rate is the percentage of true negative classifications among all the negatively classified data packets.

$$\text{Negative Rate}=\frac{\textbf{Number of True Negatives}}{\textbf{Number of True Negatives+Number of False Negatives}}$$

3. **Weighted Accuracy**

   The weighted accuracy is the average of the Positive Rate and Negative Rate expressed as a percentage.

$$\text{Weighted Accuracy}=\frac{\textbf{PR+NR}}{2} * 100\%$$

5.3.3 Results

As observed from Table 5.2, the algorithm chooses Attack 11 (Brute Force:SSH Patator) as the selected training attack when tested on Attack 6 (Slowloris). This produces a very low PR. In Figure 5.2, we can observe that training the model with Attack 3 (DoS:GoldenEye) alone yields a true positive accuracy of 95.26% and Attack 5 (DoS:Slowhttptest) alone yields a true positive accuracy of 91.16% when tested on Attack 6 (DoS:Slowloris). In Table 5.3, we observe a Positive Rate of 0.877 when the model was trained on Attack 11 data and tested on Attack 6. However, this does not outperform the performance of the model which was

trained with a single attack. From Table 5.2 we can also observe that the selected attacks for Attack 1 cannot correctly classify Attack 1. The selected attack for testing attack Attack 2 also achieves a low PR. From Figure 5.2, we can observe that training the model with attack data corresponding to only Attack 3 or Attack 4 yields true positive accuracies of 80.53% and 93.87% when tested on Attack 2. The selected attacks for Attack 3 cannot detect attack data packets however training the model with Attack 5 or Attack 6 yields a true positive accuracy of 85.01% and 90.84% when tested on attack 3 (as seen in Figure 5.2).

Table5.2. Attack Boosting Algorithm on Selected Attacks

Test Attack	Selected Attacks	Accuracy	PR	NR	Weighted Acc.
1	10,7	92.75%	0.0	0.99	49.88%
2	10	85.3%	0.392	0.906	64.91%
3	10,7	91.94%	0.0	0.988	49.66%
6	11	99.38%	0.002	0.995	49.82%

Table5.3. Attack Boosting Algorithm on Attack 6(DoS:Slowloris) when trained on SMOTE generated benign and attack data

Selected Training Attacks	Accuracy	PR	NR	Weighted Acc.
11	99.54%	0.877	0.995	93.6%

Comparing the positive rates from Table 5.2 and 5.3 with that in Figure 5.2, we conclude that training with one kind of attack scales better than training with a subset of attacks. Therefore, we do not employ this algorithm in our further study of attack correlations.

5.4 Discussion

We conclude from our study of attack correlations that the correlations observed in Figures 5.2 are the only strong and consistent correlations that exist. The model exhibits the same correlations even when it is trained with larger number of samples from the minority attack data classes using synthetic and bootstrapped datasets. Figure 5.3, 5.4 and 5.5 indicate the same correlations as Figure 5.2 when the model is trained with the augmented

datasets. We further propose hypotheses in the next section to explain these observed correlations.

6. HYPOTHESIS FOR ATTACK CORRELATIONS

6.1 Introduction

In this section, we propose hypotheses for observing the correlations as in Figure 5.2. In doing so, we use the Recursive Feature Elimination Algorithm to identify the important and dominant features for model learning. We propose hypotheses based on the number of important features selected by the algorithm as well as the properties of the attacks and features selected.

6.2 Feature Selection

6.2.1 Recursive Feature Elimination Algorithm

Recursive Feature Elimination is a feature selection algorithm. A different ML algorithm is used to identify important features in a given feature set and wrapped by the RFE algorithm to do so. This ML alogrithm at the core of the wrapper does not have to be the same as the classifier used for the classification task at hand, it can be other algorithms like Decision Trees, Random Forests to name a few. RFE provides us with a subset of the entire feature set that are important for learning. This is especially important for datasets with large number of features to identify the important features for learning and to reduce training times significantly. RFE works by recursively training the model with reduced features and removing features without which the model performs with a higher accuracy. RFE also ranks the features in order of importance. The top n features can be selected based on this algorithm.

6.2.2 Implementation

The CICIDS 2017 dataset has 78 features which is a relatively large number of features. Our goal is to identify a smaller subset of features that are important for learning and scalability of our model, as well as speed up training times. We have two implementations of the RFE algorithm. For the first implementation, we select a single attack from all the attacks and use the RFE algorithm to identify features that the classifier uses to separate from the

benign data. We repeat this process for all attacks in the dataset and identify the common features between attacks to validate our hypotheses. The second implementation involves taking two correlated attacks(as observed in Figure 5.2) and using the RFE algorithm to identify the number of features that the classifier uses to differentiate both attacks from benign data. We use the Decision Tree Classifier as the core model for RFE. The training data for both these implementations contains attack and benign data.

6.2.3 Results

Table 6.1 shows the number of features selected for select attack pairs. For attack pairs like (3,2), (3,6) and (3,4) that are correlated as seen in Figures 5.2, 5.3, 5.4 and 5.5, the algorithm selects only 4,6 and 6 features out of a total of 78 features in the dataset. For uncorrelated attacks like (3,7), the number of features selected by the algorithm is 24 which is considerably higher than correlated attacks. Similarly for (3,1) which are also uncorrelated, 21 features are selected. This forms the base for our hypotheses. We also observe some selected features from correlated attacks in Table 6.2. We use some of the features listed in Table 6.2 to strengthen our hypotheses in the next section.

Table6.1. Number of Features selected for select attack pairs by RFE algorithm

Attack(s)	**N Features Selected**
(3,2)	6
(3,6)	9
(3,4)	6
(4,6)	11
(3,7)	24
(3,1)	21

Table6.2. Features selected by RFE for select training-testing attack pairs

Select Attack Pairs	**Selected Features**
(3,2)	Destination Port, Total Length of Fwd Packets, Subflow Fwd Packets, Subflow Fwd Bytes, Subflow Bwd Packets, Init Win bytes forward
(3,4)	Destination Port, Bwd Packet Length Std, Flow IAT Mean, Packet Length Mean, Init Win bytes forward, Idle Min
(6,4)	Destination Port, Total Length of Bwd Packets, Bwd Packet Length Std, Flow IAT Mean, Flow IAT Std, Bwd IAT Mean, Bwd IAT Std, Packet Length Mean, Bwd Avg Bulk Rate, Subflow Fwd Packets, Init Win bytes forward, Init Win bytes backward, Active Mean, Active Std

6.3 Hypothesis for Attack Correlations

It is important to notice that in Figure 5.2, the model exhibits some symmetric as well as asymmetric correlations. For example, for attacks (3,5,6), training the model with any one of these attacks causes the model to scale for the other two attacks. Similarly, training the model with Attack 4 scales for Attack 2 and vice versa. However, training the model with Attack 3 scales for Attack 2, but training the model with Attack 2 does not scale for Attack 3. Similarly, training the model with Attack 6 scales for Attack 4 but not vice versa. We address these observations as well in this section.

6.3.1 Hypothesis 1

We observe from Figure 5.2 that training the base architecture with Dos:GoldenEye(Attack 3) attack causes it to perform well on Slow HTTP Attacks like Attack 5 and Attack 6 that are Slow HTTP Attacks performed with Slowloris and Slowhttptest attack tools and vice versa. We hypothesize that this is because Attacks 3, 5, and 6 are very similar in the way they are performed.

Training with DoS: slowHTTPattack simulated with Slowhttptest scales well for Slowloris and vice versa as they are essentially the same attack but simulated using different tools. In Slow HTTP attacks, the attacker opens multiple HTTP connections with the target machine and keeps them open by sending packet headers or body slowly thereby keeping all available connections occupied, making the target unavailable to legitimate requests. Since Attack 5 and Attack 6 are both the same kind of attack, it is possible for the model to learn attacking trends.

DoS:GoldenEye is an attack tool that explores vulnerabilities in the target, it tests how susceptible an a target is to a DoS attack. DoS:GoldenEye opens multiple HTTP connections with the target server and keeps those connections open with keep alive headers and caching control options because of which the connections cannot be closed, making the server unavailable to other HTTP requests. Slow HTTP attacks also open multiple HTTP connections with the target and keep open connections by sending large volumes of data at a slow rate, thereby making the target unavailable to other users. Whether it's for the keep alive

header or data being received slowly, the target is unable to close these connections. The same applies to Slowloris/Slowhttptest testing tools. This implies that although DoS:GoldenEye is not a Slow HTTP attack, it operates similar to Slow HTTP attacks. This could explain why training with DoS:GoldenEye Scales for both the Slow HTTP attacks.

6.3.2 Hypothesis 2

We observe from Figure 5.2 that training the model with attack data from DoS:HULK (Attack 4) causes it to scale for DDoS attack (Attack 2) and vice versa. Training with Distributed Denial of Service attacks scales for DoS:HULK attack because the DDoS training attack was generated with Low Orbit Ion Cannon that simulated UDP,TCP or HTTP flooding attacks by sending enormous traffic volume to the target machine. DoS: HTTP Unbearable Load King is a flooding attack where the target is flooded with HTTP data packets, sending GET requests and hitting it's resource pool. Both these attacks are flooding attacks that may cause them to scale for each other.

6.3.3 Hypothesis 3

We observe from Figure 5.2 that training the model with DoS:GoldenEye (Attack 3) causes the model to scale for DDoS (Attack 2) and DoS:HULK (Attack 4), however this observed correlation does not apply the other way around. Similarly, training the model with DoS:Slowloris (Attack 6) causes it to scale for DoS:HULK (Attack4) but not vice versa. We focus on these asymmetric comparisons in this hypothesis. To explain this, we hypothesize that correlated attacks will have few dominant features selected by the algorithm which may explain the correlations between the attacks whereas the uncorrelated attacks will have many features selected by the RFE algorithm indicating that there are not many dominant features that are important for learning which could explain the low correlations. This can be strengthened by the results in Table 6.1 for Attack pairs (3,2), (3,4), and (3,6) that have 6, 6 and 9 features selected by the algorithm. Each of these attack pairs are correlated. However for uncorrelated attacks like (3,7),(3,1), the RFE algorithm selects 24 and 21 features which are considerably more.

6.3.4 Discussion

In addition to these hypotheses, some features observed from the selected feature list from Table 6.2 provide validation for our hypotheses. For example, in addition to DoS:GoldenEye and DDoS attacks having only a few features selected in their reduced feature set, the features selected as indicated in Table 6.2 are Subflow Fwd packets, Subflow Fwd bytes, Subflow Bwd packets. Subflow Fwd packets is the average number of packets in a sub flow in the forward direction, Subflow Fwd bytes is the average number of bytes in a subflow in the forward direction, Subflow Bwd packet is the average number of bytes in a subflow in the backward direction. Subflows in the forward and backward direction are usually associated with distributed systems which points to these two attacks being distributed attacks. DoS: GoldenEye attack tool can be used as a distributed attack as well by specifying the number of workers while simulating the attack. This provides further explanation for generalization.

Similarly, to study the correlation between DoS:GoldenEye, DoS:Slowhttptest, DoS:Slowloris and DoS:HULK, the RFE algorithm chooses features Inter-Arrival Time, Bytes Sent in the Forward Direction, Bytes sent in the Backward Directions in the Initial Window which is coherent with the hypotheses for symmetric comparisons.

7. CONCLUSION

7.1 Conclusion

We show that the deep learning approach generalizes well for this dataset in comparison to explicit statistical modeling based methods. After evaluating the performance of several candidate DNN architectures like CLDNN, CNN with BGRU, OCNN and CNN, we propose a CNN architecture that has the highest accuracy in the multi-class classification problem (Figure 4.4). Further, we use this architecture to study attack transferability and observe it's performance as in Figure 5.2. We identify training attacks for which the model generalizes well on certain testing attacks. We tackle the problem of under represented attacks in the dataset using synthetic and bootstrapped dataset based training methods. Figure 5.3, 5.4 and 5.5 show that we can observe the same attack correlations as we observed in Figure 5.2 for the performance of the model trained on SMOTE dataset and bootstrapped datasets. This confirms our identification of training-testing pairs of attacks that exhibit attack transferability. Furthermore, we propose hypotheses for the observed correlations. We use Reduced Feature Elimination Algorithm to strengthen our hypotheses. We can observe the relationship between the number of features chosen and the training-testing attack pairs from Table 6.1. Table 6.2 lists features selected by the algorithm for selected attacks that are coherent with our hypotheses.

7.2 Future Work

In this work, we validate our hypotheses with features from the RFE algorithm. Future work could focus on validating these hypotheses using transferability in learning for different datasets for example, DARPA 2009 IDD, KDDCup99, NSL-KDD, UNSW-NB15, WSN-DS. Future work could also be focused on analyzing trends in specific or selected features in studying attack correlations. It is interesting to observe from Figure 5.2 that we only observed correlations in attacks that are different types of DoS attacks. We did not observe correlations from other kinds of attacks for example, Attack 1 (Botnets), Attack 7 (Brute Force: FTP-Patator), Attack 8 (Heartbleed), Attack 9 (Infiltration) and Attack 13 (Web

Attack: SQL Injection). Future research can focus on identifying attributes that could help the model scale for these attacks and testing for transferability of learning for these attacks. Future work could also focus on attack detection when there is more than one attack being performed at the same time.

REFERENCES

[1] A. Lopez, A. Mohan, and S. Nair, "Network traffic behavioral analytics for detection of ddos attacks", *SMU Data Science Review*, vol. 2, no. 1, 2019.

[2] M. Alkasassbeh, G. Al-Naymat, A. Hassanat, and M. Almseidin, "Detecting distributed denial of service attacks using data mining techniques", *International Journal of Advanced Computer Science and Applications*, vol. 7, Jan. 2016. DOI: 10.14569/IJACSA.2016.070159.

[3] R. Vinayakumar, M. Alazab, K. P. Soman, P. Poornachandran, A. Al-Nemrat, and S. Venkatraman, "Deep learning approach for intelligent intrusion detection system", *IEEE Access*, vol. 7, pp. 41 525–41 550, 2019. DOI: 10.1109/ACCESS.2019.2895334.

[4] I. Sharafaldin, A. H. Lashkari, Ghorbani, and A. A, "Toward generating a new intrusion detection dataset and intrusion traffic characterization.", in *ICISSp*, 2018, pp. 108–116.

[5] M. A. Ferrag, L. Maglaras, S. Moschoyiannis, and H. Janicke, "Deep learning for cyber security intrusion detection: Approaches, datasets, and comparative study", *Journal of Information Security and Applications*, vol. 50, Dec. 2019. DOI: 10.1016/j.jisa.2019.102419.

[6] R. Chalapathy, A. K. Menon, and S. Chawla, *Anomaly detection using one-class neural networks*, 2019. arXiv: 1802.06360 [cs.LG].