NETWORK INTRUSION DETECTION WITH A DEEP LEARNING APPROACH

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF INFORMATICS OF
MIDDLE EAST TECHNICAL UNIVERSITY
BY

EBRU KÜLTÜR

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE
IN
THE DEPARTMENT OF CYBER SECURITY

FEBRUARY 2022

Approval of the thesis:

NETWORK INTRUSION DETECTION WITH A DEEP LEARNING APPROACH

Submitted by EBRU KÜLTÜR in partial fulfillment of the requirements for the degree of **Master of Science in Cyber Security Department, Middle East Technical University** by,

Prof. Dr. Deniz Zeyrek Bozşahin
Dean, **Graduate School of Informatics** _____

Asst. Prof. Dr. Cihangir TEZCAN
Head of Department, **Cyber Security** _____

Asst. Prof. Dr. Aybar Can ACAR
Supervisor, **Health Informatics Dept., METU** _____

Examining Committee Members:

Asst. Prof. Dr. Cihangir TEZCAN
Cyber Security Dept., METU _____

Asst. Prof. Dr. Aybar Can ACAR
Medical Informatics Dept., METU _____

Assoc. Prof. Hacer KARACAN
Computer Engineering Dept., Gazi University _____

Date: *07.02.2022*

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last name: Ebru Kültür

Signature : _____

ABSTRACT

NETWORK INTRUSION DETECTION WITH A DEEP LEARNING APPROACH

Kültür, Ebru
MSc., Cyber Security
Supervisor: Asst. Prof. Dr. Aybar C. Acar

February 2022, 99 pages

With the rapid growth of the information technology in several areas, providing security of those systems has gained more importance. As a result of this development in information technology, the complexity of cyber-attacks has also significantly increased. Therefore, traditional security tools such as Signature-based Intrusion Detection Systems (SIDS) have become insufficient for detecting new attacks.

Intrusion Detection Systems (IDS) are used to monitor network traffic and capture malicious traffic. Traditional IDS are signature-based, meaning that they are capable of taking action against known threats using only predefined or custom rule sets. Traditional IDS fail to detect such attacks if unknown attacks or modified known attacks which does not correspond to the static signature occur.

New behavior-based anomalous activity detection approaches, such as deep learning, can offer a solution together with signature-based IDS in order to increase the performance of detecting new types of attacks and reduce the FP (false positive) and FN (false negative) rates. Since deep learning algorithms can learn from data and patterns, it will be possible to increase the detection rate of real malicious activities by estimating which traffic is normal or attack traffic. At the same time, they are capable of automating the detection process without the need for manual configuration in order to reduce false alarms.

In this thesis, we aim to investigate the efficiency of applying deep learning approaches by focusing on recurrent neural network architectures for network flow-based intrusion detection.

Keywords: Cybersecurity, network flow analysis, deep learning, intrusion detection

ÖZ

DERİN ÖĞRENME YAKLAŞIMI İLE AĞ SALDIRI TESPİTİ

Kültür, Ebru
Yüksek Lisans, Siber Güvenlik Departmanı
Tez Yöneticisi: Dr. Öğr. Üyesi Aybar C. Acar

Şubat 2022, 99 sayfa

Bilgi teknolojilerinin birçok alanda giderek yaygınlaşması ile birlikte, söz konusu sistemlerin güvenliğini sağlamak oldukça önem kazanmıştır. Bu büyümenin bir sonucu olarak, siber saldırıların karmaşıklığı da artmış durumdadır. Bundan dolayı, imza tabanlı Saldırı Tespit Sistemleri gibi geleneksel güvenlik araçlarının kullanımı da tatmin edici sonuçlar üretme konusunda artık yetersiz kalmış durumdadır.

Saldırı Tespit Sistemleri (STS) ağ trafiğini gözlemleme ve zararlı trafiği yakalamak için kullanılmaktadır. Geleneksel STS'ler imza tabanlıdırlar, yani sadece kural setleri kullanarak bilinen tehditlere karşı önlem alabilme kabiliyetine sahiptirler. Bilinmeyen saldırılar ya da bilinen saldırılarda statik imzaya karşılık gelmeyecek şekilde değişiklikler yapılması durumunda ise geleneksel STS'ler bu tür saldırıları tespitte başarısız olmaktadırlar.

Yeni tür saldırıları tespit edebilme performansını artırma, "false positive" ve "false negative" oranlarını azaltabilmek için, makine öğrenmesi gibi yeni davranış tabanlı anormal aktivite tespit yaklaşımları imza tabanlı STS'ler ile birlikte bir çözüm sunabilirler. Makine öğrenme algoritmaları veri ve örüntülerden öğrenebildiği için, hangi trafiğin normal ya da saldırı trafiği olduğu tahmin edilerek gerçek zararlı aktivitelerin tespit oranını artırmak mümkün olacaktır. Aynı zamanda yanlış alarmları azaltabilmek için ayrıca elle konfigürasyona ihtiyaç olmaksızın tespit etme işlemini bu şekilde otomatize edebilecektir.

Bu tezde, ağ akışı tabanlı saldırı tespiti için tekrarlayan sinir ağı mimarilerine odaklanarak derin öğrenme yaklaşımlarının uygulanmasının tespit performansına etkisini araştırmak amaçlanmaktadır.

Anahtar Kelimeler: Siber güvenlik, ağ akış analizi, derin öğrenme, saldırı tespiti

To My Family

ACKNOWLEDGMENTS

First of all, I would like to express my sincere gratitude to my super advisor Asst. Prof. Aybar Can Acar for his mentorship throughout my thesis. Even under very hard and unexpected conditions such as Pandemic, he was very supportive and understanding. He guided all the phases of my thesis by sharing his advices and wisdom.

I would like to thank to my parents for their support, patience and motivating me until the end.

I would also like to thank to my dear fiancé for being very supportive and gentle during this journey.

Without all the support and love of my beloved ones, this work would not have been possible.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

AE	Autoencoder
AIDS	Anomaly Based Intrusion Detection System
AUC	Area Under Curve
ANN	Artificial Neural Networks
CNN	Convolutional Neural Networks
DAE	Deep Autoencoder
DBM	Deep Boltzmann Machines
DBN	Deep Belief Networks
DDOS	Distributed Denial of Service
DNN	Deep Neural Networks
DOS	Denial of Service
FN	False Negative
FP	False Positive
FPR	False Positive Rate
FTP	File Transfer Protocol
HIDS	Host Based Intrusion Detection System
HOIC	High Orbit Ion Cannon
HTTP	Hyper Text Transfer Protocol
IDE	Integrated Development Environment
IDS	Intrusion Detection System
IPS	Intrusion Prevention System
KNN	k-Nearest Neighbors
LOIC	Low Orbit Ion Canon
LSTM	Long Short-Term Memory Recurrent Networks
MSE	Mean Squared Error
NIDS	Network Based Intrusion Detection System
PPS	Packets per Second
PR	Precision-Recall
RBM	Restricted Boltzmann Machines
RBM	
RF	Random Forest
RNN	Recurrent Neural Networks
ROC	Receiver Operating Characteristic
SIDS	Signature Based Intrusion Detection System

SMOTE	Synthetic Minority Over-sampling Technique
SQL	Structured Query Language
SSH	Secure Shell
TN	True Negative
TP	True Positive
TPR	True Positive Rate
XSS	Cross-Site Scripting

CHAPTER 1

INTRODUCTION

1.1. Problem Statement

With the evolution of information technology, number of used network devices and services have been increased. Since the attack surface area has become much larger with this increase, the need for taking precautions against possible attacks has gained a significant importance in terms of cyber security.

Moreover, due to recent COVID-19 Pandemic circumstances, we have been forced to change our existing habbits. Many compaines had to accelerate their digital transformation. This created a serious increase in demands such as remote working, online meetings, education, shopping, entertainment etc. This "new normal" also created much larger attack surface which attracted attackers' interest and increased number of conducted cyber attacks. (Cedric Nabe, 2021)

Cisco's "Future of Secure Remote Work Report" was commissioned as a global research survey across 21 markets in the Americas (AMER), Asia Pacific, Japan and China (APJC), and Europe, including 3000 IT decision makers from small businesses to large enterprises. According to this survey, remote working percentage has increased from 19% to 62% during pandemic.

% of respondents with more than half of their workforce working remotely				
	Global	APJC	AMER	Europe
Before COVID-19	19%	19%	24%	16%
Today (During COVID-19)	62%	56%	75%	67%
After COVID-19	37%	34%	46%	34%

Figure 1: Survey Results

Retrieved from (Cisco, 2020)

In addition, 85% of respondents globally have voted as cybersecurity is extremely important or more important than it was before the pandemic.

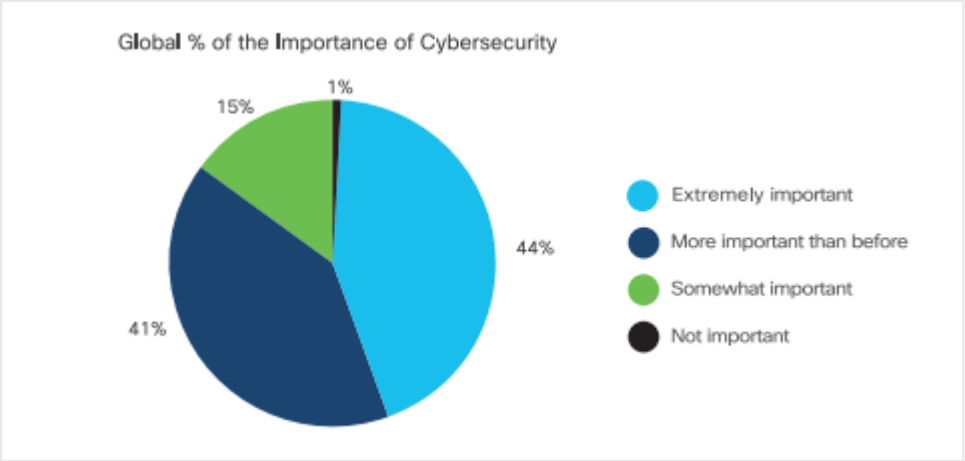

Figure 2: Global Importance of Cybersecurity

Retrieved from (Cisco, 2020)

As can be seen from Figure 3, by making use of remote working which is becoming widespread, malicious actors caused a significant growth in cybersecurity threats/alerts during pandemic.

Since the beginning of COVID-19, amongst the sixty-one percent of respondents whose organizations experienced a jump of 25% or more in cyber threats or alerts, Global, ADPJ, AMER and Europe countries share the results as shown in Figure 3. It is also very interesting that 8% of businesses globally do not have any idea about their current status of cyber threats/alerts which means either they have not been exposed to any cyber-attacks that does not make any sense regarding current cyber-attacks situation, or they are not capable of detecting cyber-attacks. Generally, for most of the organizations, second option is sadly in the case of point that should be considered very seriously.

Percentage of increase in cyber threats/alerts	Global	APJC	AMER	Europe
0% - 24%	31%	25%	31%	47%
25% - 50%	33%	35%	36%	23%
51% - 75%	23%	27%	24%	11%
76% - 100%	5%	6%	3%	2%
Don't know	8%	6%	5%	17%

25% or more threats

% of Increase in Cyber Threats or Alerts

Figure 3: Percentage of Increase in Cyber Threats

Retrieved from (Cisco, 2020)

In the light of the above findings, in order to increase visibility and detection of cyber threats, using the right technology and methods is very important. For the upcoming

parts in this thesis, we focus on detection methods and used technologies and how to improve them.

1.2. Research Questions

The research questions for this thesis are;

1. Due to the fact that traditional signature-based intrusion detection systems (IDS) are no longer sufficient to detect today's attack types, do Machine Learning based IDS perform better in detecting these attacks?
2. Is network flow information sufficient for detecting different kinds of attacks?
3. Which network flow extracted features are most effective for identifying each attack type?
4. Can RNNs give more efficient results in analyzing sequential information of network activity that occurs in timely manner?
5. What will be the difference in performance between binary and multi-class classification by using LSTM?

In this thesis, the main motivation is to find acceptable answers to these research questions and discuss the outcomes of the conducted study.

1.3. Proposed Approach

The proposed system applies neural network on a dataset composed of up-to-date attack scenarios and has a realistic distribution of benign and attack traffic, composed as a collaborative project between the Communications Security Establishment (CSE) & the Canadian Institute for Cybersecurity (CIC): CSE-CIC-IDS2018. We have implemented a solution to detect network anomaly traffic by using deep learning algorithms.

The selected dataset is composed of 80 features and 14 different types of intrusions extracted from the captured traffic. One of the most important features that lets the dataset be more realistic is that it includes timestamp values of each network flow. Among the research studies demonstrated by using various machine learning techniques, this information is mostly neglected. However, especially for cyber-attacks, association of network flows according to their timing is very important to detect anomalous behavior.

The strongest reason for selecting deep learning approach – Recurrent neural networks (RNNs) for this thesis work is that they are very efficient to solve sequence problems, time series prediction, speech recognition, generating texts, machine translation, generating image descriptions, handwriting recognition etc.

The Long Short-Term Memory (LSTM) network was invented with the aim of solving vanishing gradients problem of RNNs which is explained detailed in Chapter 2.4.3.1 of this thesis. We used LSTM RNN to learn time-ordered sequences of network flow

traffic and evaluate the performance of this model in terms of anomalous activity detection.

We constructed two different ML models; one of them containing only two labels that are “Benign” and “Attack” (all attack traffic combined under this label), the other one containing multi-class (“Benign”, and all 14 attack types as separate labels). We evaluated these two different models according to the performance of each model. In this way, the behavior of LSTM model was compared by evaluating the importance of features extracted from network flow for both binary and multi-class classification.

CHAPTER 2

BACKGROUND

2.1. Evolution of Cyber Attacks

The history of Cyber Attacks actually has started in 1970s, in the days when majority of people did not even have a computer.

In 1972, a computer program called “Creeper” was created by Bob Thomas for experimental purposes as a part of the research project on ARPANET (The Advanced Research Projects Agency Network). In order to detect and delete Creeper, Ray Tomlinson, who is the inventor of email, wrote the program Reaper. This program is known as the first antivirus software and also first computer worm because of its self-replicating feature.

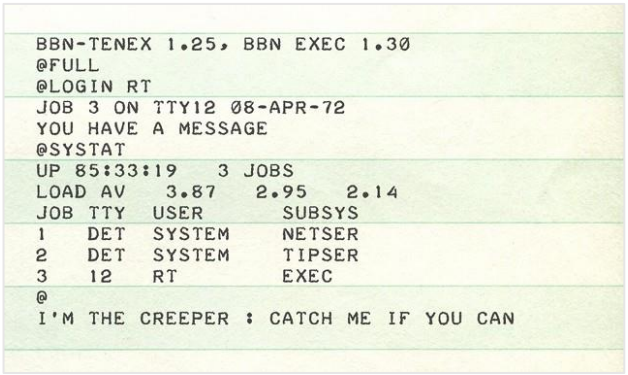

Figure 4: Creeper Program

In 1980s, with the transition from ARPANET to Internet, attacks became more complex. In 1988, one of the most important incidents occurred in the information security history; Morris Worm. Morris Worm was developed by Robert Morris and also known as the first DDoS (Distributed Denial-of-Service Attack). It successfully attacked about 6,000 of the 1988 internet's 60,000 servers by causing the targeted computers inoperable and slowing down the internet (Steven Vaughan-Nichols, 2018).

Between years 1990s-to-2000s, since the internet became more popular and available to the public, the amount of online stored information also increased. This situation attracted many attackers to steal information from organizations, governments and individual people by using this new strength of the internet. On the other hand, people started to give more importance to cyber security which led to development of first firewall program by a NASA researcher. Moreover, antivirus programs was also being mass produced in order to deal with these new security challenges. However, with new techniques and methods being used by attackers, these new precautions was only able to protect until some level.

In "The Sophos 2021 Threat Report" (Levy, 2020), which was prepared by SophosLabs on malware and spam analysis, and by the Sophos Rapid Response, Cloud Security, and Data Science teams; 20 year recent history of information security is discussed.

According to the corresponding report, 20 year history is divided into three major eras as follows;

- **2000-2004 - The Worm Era :** In this era, the main motivation of malware incidents were mostly curiosity or disruption.
  Some examples of incidents occurred in this era were:
  ILOVEYOU, CodeRed, CodeRed II, Nimda, SQL Slammer, Blaster, Welchia, Sobig, Sober, Bagle, MyDoom, Netsky, Sasser
- **2005-2012 - The Malware Monetization Era:** In this era, the main motivation of attacks was mostly stealth and profit – financial gain. Malvertising term also developed in this era in the result of exploitation against software vulnerabilities.

  Some examples of incidents occurred in this era were:

  Rx Spam, Storm, Zeus, Conficker, Stuxnet, Blackhole exploit kit, Malvertising

- **2013-Present – The Ransomware Era:** In this era, worms, security events such as banking Trojans, malvertising and spam has continued to occur. However, ransomwares have dominated all the security events. Moreover, ransomwares are the most destructive ones among the other attacks. The overall damage caused by exposing to ransomwares for past 7 years are estimated as trillions of dollars.
  Some examples of incidents occurred in this era were:
  Snowden leaks, Cryptolocker, Point-of-sale (POS) malware, Mirai, WannaCry, NotPetya, Magecart attacks, Extortion ransomware, APT tactics by threat actors (has played significant role in 2020)

According to McAfee Labs Threat Report published in April, 2021 (Cedric Nabe, 2021); Top 10 Attack Vectors belonging to each quarter of 2020 is given as in Figure 5. As can be seen, while new vulnerabilities vectors have increased 100% from Q3 to Q4 of 2020, new malware and targeted attacks each have increased 43%. Also, there has been approximately 50% of increase in unknown vectors in cyber attacks from Q3 to Q4 which can not be underestimated since mitigation is very hard without identifying the attack vector.

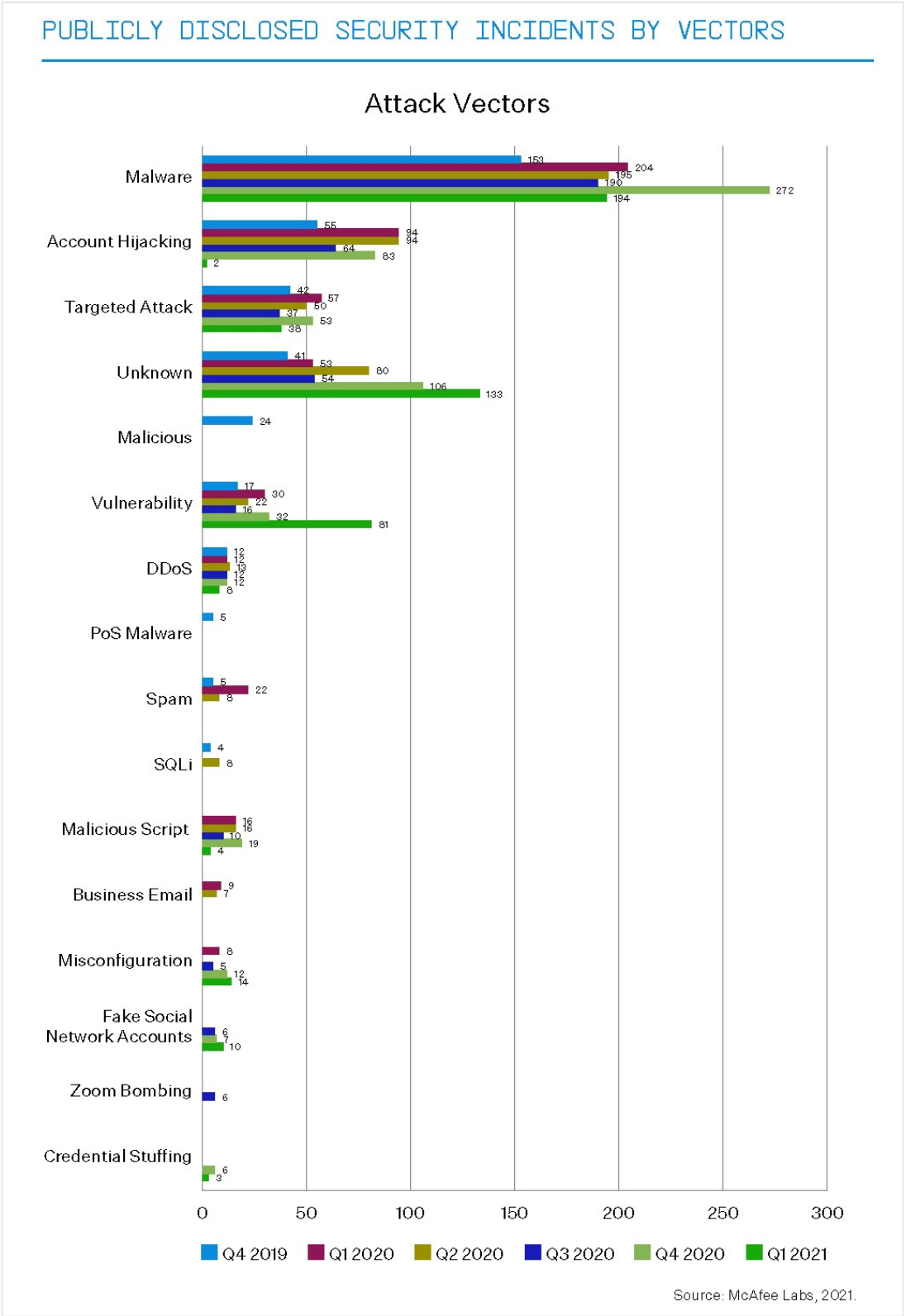

Figure 5: McAfee Labs Threat Report 06.21

Retrieved from (McAfee, 2021)

As can be concluded by the recent Cyber Attack Surveys, malware evolution throughout years has entailed the development of the design of intrusion detection systems (IDS). Detection of unknown and obfuscated malware has become important since attackers use several evasion techniques by hiding information to prevent detection. In addition, according to Sophos' 2021 Threat Report, APT tactics with the usage of nation-state tools and tactics in cyber attacks are more widely used in the past couple of years. Cyber attackers now mostly prefer to make use of sophisticated tools like Cobalt Strike to the increase damage.

In advanced persistent threat attacks (APT), attackers aim to gain access to the victim system and establish long-term presence on the network by using continuous, hidden and sophisticated hacking techniques. The main purpose in this attack type is to obtain highly sensitive data by mostly causing destructive activities. APT attacks are multi stage and on average it takes 240 days to detect an APT-related cyber attack.

One of the methods for detecting APT attacks is by analyzing network traffic. Network based Intrusion Detection Systems (NIDS) can be a significant part of the solution. Since APT attacks mostly present on the network in a prolonged period of time, traditional solutions including pre-defined rules will not be sufficient for detection.

2.2. Intrusion Detection Systems

According to RFC 4949, Internet Security Glossary (Shirey, 2007) the term intrusion is defined as;

"*A security event, or a combination of multiple security events, that constitutes a security incident in which an intruder gains, or attempts to gain, access to a system or system resource without having authorization to do so*."

Intrusion contains activities that damages confidentiality, integrity or availability of information obtained unauthorized.

Intrusion Detection Systems (IDS) are hardware or software used for identifying and mitigating intrusions. IDS are used for analyzing malicious activities, security violations and reporting abnormal activities to system administrators. These type of malicious activities cannot be identified by traditional firewalls since they are only capable of working up-to OSI Layer 4 (Transport Layer). That's why we need IDS for further investigation.

According to "2020 SANS Network Visibility and Threat Detection Survey", which was conducted among 213 respondents representing a broad cross-section of organizations with at least 1,000 employees, Network IDS/IPS technologies are being used by the majority of respondents for network visibility. (Ian Reynolds, 2020)

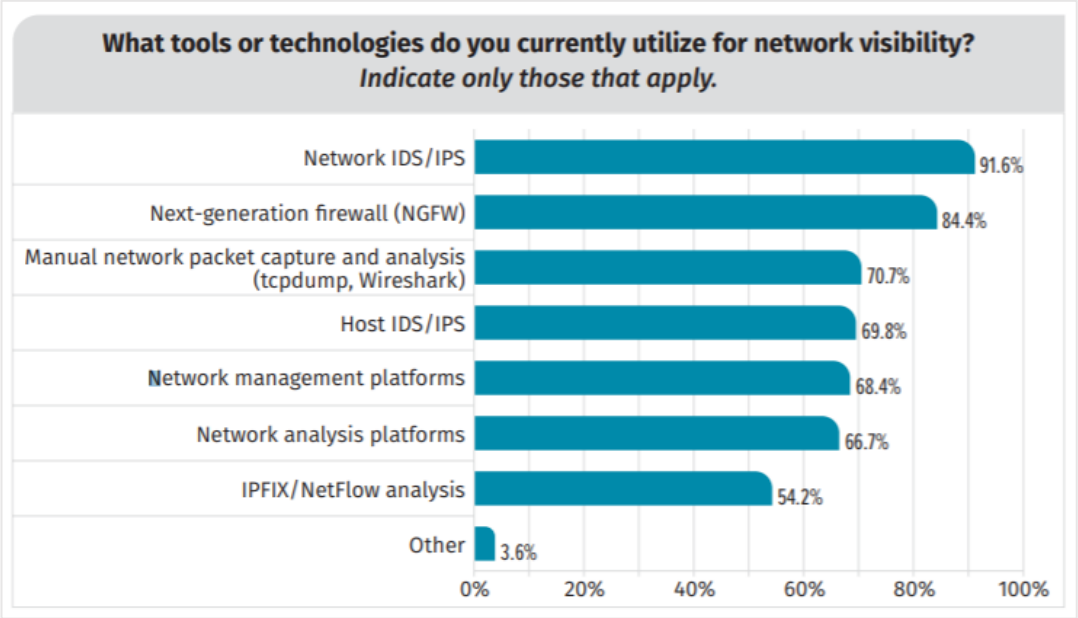

Figure 6: SANS 2020 Threat Detection Survey

As Guy Bruneau from SANS stated in "The History and Evolution of Intrusion Detection" paper (Guy Bruneau, 2001), the beginning of the history of IDS depends on a USAF (U.S. Air Force) paper published in October 1972. In this paper, the importance of growing number of computer security problems were emphasized and USAF operations and administration were affected by this problem.

Following this paper, James P. Anderson has published a paper focusing on "How to use accounting audit files to detect unauthorized access" (Anderson, 1980). In this paper, a necessity of a risk assessment plan with the creation of a security policy were highlighted in order to identify and categorize threats, vulnerabilities, risks, possible penetrations.

The first model of a real-time IDS was first introduced as a result of the research and development study of Dorothy Denning and Peter Neumann. In 1987, Dorothy E. Denning has published "An Intrusion Detection Model" (Denning, 1987) which describes a real-time intrusion detection expert system (IDES). This system was designed as a rule-based expert system, which was capable of detecting known intrusions. This study has become a milestone for much of the research studies on IDS.

From then on, with increasing diversity of cyber attacks, IDS were also developed and categorized in many types in order to satisfy needs to detect different kinds of attack types. According to (Khraisat et al., 2019), IDS can be categorized according to different approaches.

First approach of categorization of IDS is the location of the intrusive activity. Therefore, IDS can be categorized as *host-based IDS (HIDS)* and *network-based IDS (NIDS)* according to their data sources.

NIDS are widely used for protecting networked systems by identifying unusual traffic. They are capable of detecting intrusions and security policy violations within a

network. Since NIDS are used only for detection, they are deployed so as to collect network traffic by receiving mirror traffic by network devices (switch, router, TAPs, network packet brokers etc.) in forms of Netflow and packet capture. Network-based IDS are focused on providing network security and perform detection while listening and analyzing network traffic.

HIDS are focused on the intrusive activities occurred on host computer while monitoring and analyzing corresponding system events. NIDS are able to monitor activities belonging to many machines connected to the network. NIDS are very advantageous to detect external malicious activities. On the other hand, HIDS are capable of detecting host initiated activities such as operating system, server logs, firewall logs, event logs, audit logs or database logs. Therefore, HIDS are more advantageous to detect insider attacks. But we should note that usage of both NIDS and HIDS will give the best detection rate.

The second approach of categorization of IDS is the data analytics methods used, such as signature-based (misuse-based), anomaly-based and hybrid which is a combination of the previous two methods. (Buczak et al., 2016)

In order to increase the performance of detecting new type of attacks and decrease false positive & false negatives, new behavioral based abnormal activity detection approaches such as machine learning can be combined with signature-based IDS. Since machine learning algorithms can learn from data and patterns, it is promising to increase the detection rate of actual malicious activity by predicting which traffic is expected or abnormal. This will also help automatizing detection process without having to make manual configurations to reduce false alarms.

In this thesis, we focus on methods to provide network security by using data analytics approaches for building a NIDS. Therefore, the categories are investigated more detailed in the following topics.

2.2.1. Signature-based intrusion detection systems (SIDS)

Traditional signature based intrusion detection systems (S-IDS) runs mostly according to pattern matching techniques to find a known attack. These techniques can be named as Knowledge-based Detection or Misuse Detection. In this approach, the attack or misuse patterns are learned according to previous activities. Then, with the knowledge obtained from the learned patterns, similar patterns can be detected. These learned attack and misuse patterns are called “signatures”. There are both pre-defined and user-defined signatures used to define attack patterns, and when a malicious network traffic matches the corresponding signatures, S-IDS generates alert.

A signature database consists of known signatures of known attack types is used to compare current activities and alert when there is a match. Moreover, to preserve detection efficiency, signature database should be updated regularly for the most recent attacks.

This method is very suitable and gives high detection accuracy when previously known attacks occur. However, this method lacks identifying modern attacks that span

multiple packets. IDS should keep the relevant past packets and make connection between ordered packets to identify the intrusion. There have been other approaches to solve this problem for S-IDS which propose that signatures should be created as state machines (Meiners et al., 2010) formal language string patterns or semantic conditions (Lin et al., 2011) .

Moreover, when it comes to zero day attacks or recent sophisticated cyber-attacks using irregular patterns to bypass security systems, S-IDS approach becomes insufficient for detection.

On the other hand, anomaly based intrusion detection systems (A-IDS) are capable of modelling normal behaviour so as to detect deviations from normal traffic. Compared with the S-IDS, this method is much more efficient for detecting unknown attacks.

2.2.2. Anomaly-based intrusion detection systems (A-IDS)

Machine learning, statistical-based or knowledge-based methods are used to create a normal model of the behavior of the corresponding system. When any deviations are detected between the observed and the model, it is treated as anomaly pointing to an intrusion. Main approach of this technique is that malicious user traffic should be different from normal user traffic which can be derived by modelling normal user traffic. In this way, attacks such as zero-day and insider can more easily be detected. The main dilemma of this technique is creating false positive alarms.

A-IDS methods can be applied both HIDS and NIDS to increase the detection performance of both systems. In this thesis, we focus on usage of A-IDS methods applied on NIDS by using machine learning techniques.

As mentioned earlier, A-IDS approaches can be discussed as statistics-based, knowledge-based and machine learning-based.

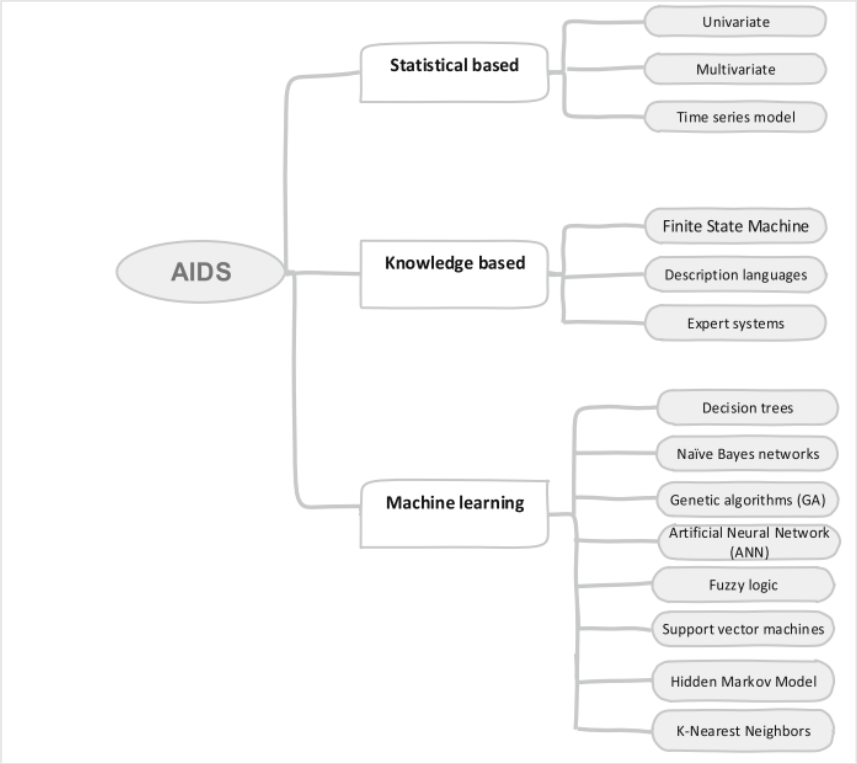

Figure 7: Types of Anomaly Based IDS

Retrieved from (Khraisat et al., 2019)

In the *statistics-based approach,* a distribution model is built to profile normal user behaviour. In this way, low probably occuring events are marked as potential intrusion activity. In this approach, statistical metrics such as the median, mean, mode and standard deviation of packets are considered to identify differences between normal and abnormal traffic. Univariate, multivariate or time series models are used in this approach.

In *knowledge-based approach*, normal user behaviour is reflected by creating a knowledge base. When an activity is identified as different from this profile marked as intrusion. This technique differs from other approaches since it depends on human knowledge to create the knowledge baseline. A set of rules are defined to define the normal user activity. In this way, false positive rates become smaller but since the rule definition phase is conducted manually, and needs to be updated regularly, this approach is insufficient in today's sophisticated nature.

Finite state machine (FSM), Description Language, Expert System and Signature Analysis are the methods used in this approach.

In this thesis, we focus on machine learning based A-IDS which is discussed in the next chapter.

Machine Learning Based A-IDS

Machine learning approaches are widely being used for anomaly based intrusion detection. AI-based IDS is still a hot research topic which gives very promising results comparing the mentioned limitations of SIDS. Several algorithms and their combinations are still being investigated by many researchers.

By using machine learning, statistics-based or knowledge-based methods, a baseline reflecting the normal behavior of the corresponding system is created. If any deviation from this model is detected, it is considered as anomaly (or intrusion). In order to build a model reflecting the normal behavior - baseline, the model is trained to learn the normal traffic profile. Then the trained model is tested with new data which is completely different from the one used in training phase. Finally, overall performance of the model is evaluated by comparing the predicted and actual results.

In this thesis, we discuss deep learning approaches - as a part of machine learning - used for anomaly based network intrusion detection.

In recent years, deep learning also gained a significant importance for its capability to learn complex data representations which is very promising to provide a satisfying solution to more detecting today's complex attacks by modelling high intensity network traffic.

Anomaly detection with deep learning focuses on learning feature representations or anomaly scores by using neural networks to detect anomalies. There are many deep learning algorithms such as Convolutional Neural Networks (CNN), Long Short Term Memory Networks (LSTMs), Recurrent Neural Networks (RNNs), Autoencoders that have been investigated to provide a solution for network anomaly detection.

Network traffic should be treated as sequential information since it occurs as time-correlated sequences. LSTM recurrent neural networks suits very well with this problem since they are able to learn ordered sequences by using their memory function which is be explained more detailed in Section 2.4.3.2. Amongst the recent studies conducted with many deep learning algorithms, LSTM has shown the most efficient results comparing others. For this reason, in this thesis , LSTM is decided to be the main deep learning algorithm to be focused on.

In order to train and test the A-IDS, labeled data sets are being used. Generally, as an evaluation criterion, when each data point is labeled as normal or attack class in corresponding dataset, detected attacks or false alarms can be taken into consideration for evaluating the algorithm. At this point, using a reliable dataset containing up to date network attacks and realistic network traffic simulation with optimum diversity of normal and attack traffic ratio. In this area, only reasonable number of datasets can be found public since it requires a serious amount of work to create a suitable topology with divergence of operating systems, services, clients, servers containing up to date attack scenarios with realistic network throughput rates.

2.3. Flow-based Network Anomaly Detection

As depicted in RFC 5101 (Claise, 2008), IP traffic flow is defined as;

"A set of IP packets passing an Observation Point in the network during a certain time interval. All packets belonging to a particular Flow have a set of common properties. Each property is defined as the result of applying a function to the values of:

1. *one or more packet header fields (e.g., destination IP address), transport header fields (e.g., destination port number), or application header fields (e.g., RTP header fields [RFC3550]).*
2. *one or more characteristics of the packet itself (e.g., number of MPLS labels, etc...)."*

Flow and connection (used in TCP) are two separate terms that should not be mixed. Flow does not depend on a connection, whether there is a TCP connection or not, flows can occur. There can be both TCP and UDP flows containing a set of packets with source/destination address, source/destination port and protocol no matter which size of the packet has.

According to article "(Singla et al., 2018)", an IDS is capable of working with the following types of data; (traffic aggregation at autonomous system (AS) is also included in this article but not considered in the scope of this thesis.)

- Packet data
- Flow records

Packet data is the most detailed data type amongst three. It is also used for deep packet inspection (DPI) purposes since it not only contains header information of the packet, but also payload part. However, there are many concerns about DPI since it violates user privacy and net neutrality which is still being discussed among authorities. Moreover, it brings complexity with the increasing throughputs and diversity in network topologies. Considering high percentage of internet traffic is encrypted in today's world, (for example, the total percentage of encrypted web traffic is now around 85%, (N. Shah, 2020)), it is impossible to inspect payload part of the traffic without decrypting for analyze. This process also demands extra performance and time which will be meaningless for especially real-time inspection under heavy traffic.

Flow records are inspected in flow-based intrusion detection. Flow records include only header information of the network packet such as source IP, destination IP, source port, destination port and protocol. Therefore, since no payload is included for the data to be investigated, there is not any violation for the user privacy and of course net neutrality. For this reason, encryption also becomes out of concern. Flow-based data can be described as the summary of the network traffic and its size is much smaller than packet data that makes them very convenient for high- speed networks.

As mentioned in (Ming et al., 2006) and (Lai et al., 2004) articles, payload-based NIDS processing capability is evaluated between 100 Mbps and 200 Mbps. Considering today's networks with high throughputs, many DPI-capable NIDS consumes high resources.

Table 1 belongs to (S. A. R. Shah & Issac, 2018) article containing several experiments conducted in different throughputs in order to compare two of the most well-known DPI-capable open-source NIDS, Snort and Suricata. Packet drop rates are shown under different network speed up to 10 Gbps. As can be seen, as network speed increases, packet drop rates are also increased.

Table 1: Comparison of Packet Drop Rates

Network Speed	Packet Drop Rates (average)					
	UDP		TCP		ICMP	
	Snort	Suricata	Snort	Suricata	Snort	Suricata
1 Gbps	3.2%	1.5%	1.5%	0%	7.0%	3.5%
2 Gbps	4.1%	2.0%	3.1%	1.0%	8.2%	4.1%
3 Gbps	4.8%	2.5%	4.9%	2.2%	9.8%	4.9%
4 Gbps	6.3%	2.9%	7.2%	3.6%	10.5%	5.5%
5 Gbps	7.5%	3.7%	9.5%	4.1%	11. %5	6.2%
6 Gbps	9.6%	5.0%	12.1%	5.3%	13.0%	6.9%
7 Gbps	10.8%	5.9%	14.8%	6.2%	14.2%	8.0%
8 Gbps	11.7%	6.6%	16.4%	7.4%	15. %6	8.6%
9 Gbps	12.8%	7.2%	18.2%	8.0%	16.1%	9.1%
10 Gbps	14.0%	8.0%	20.0%	9.0%	17.0%	10.0%

Due to these performance issues in high speed networks, flow-based intrusion detection approaches has raised attention throughout the years. In Figure 8, the evolution time line of intrusion detection and flow-based technologies is given.

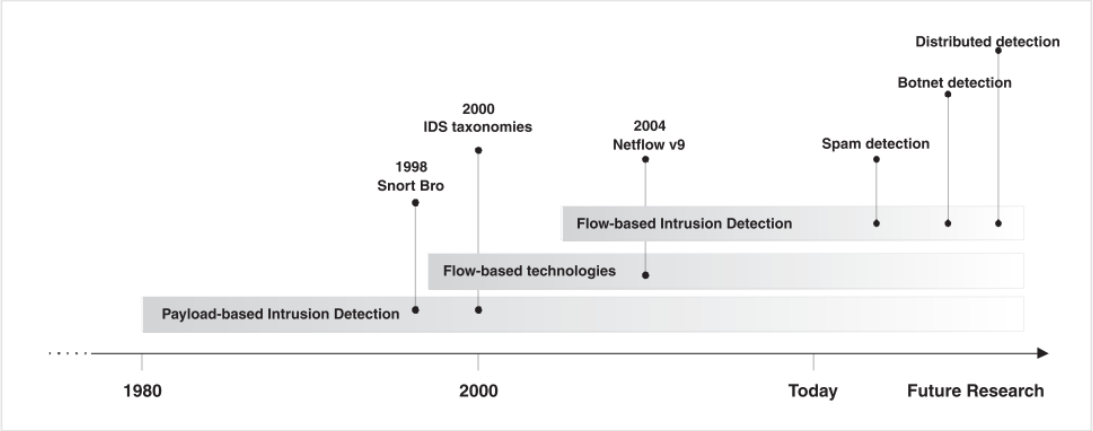

Figure 8: The Evolution of Intrusion Detection Techniques

Retrieved from (Sperotto et al., 2010)

Although, there are performance advantages to apply flow-based approaches in NIDS, DPI-based packet analysis aims to investigate network packets more detailed. Both approaches work best under different scenarios individually. Hence, each having its own pros and cons, should not replace each other. As mentioned, in this thesis, our main topic is to provide a solution for high-speed diverse networks by applying flow-based network anomaly detection with ML algorithms.

2.4. Machine Learning for Intrusion Detection

2.4.1. History of Machine Learning

With the evolution in machine learning in recent years, it has been started to be widely used in many areas. Cyber security has become one of the popular areas that has been benefiting machine learning solutions for detecting malicious activities. Since signature-based approaches are now evaluated as insufficient to detect current cyber threats, machine learning approaches have gained significant importance to fill this gap.

In 1943, neurophysiologist Warren McCulloch and mathematician Walter Pitts has published a paper "(Mcculloch & Pitts, 1990)" where they explained the working principle of human nervous system. They have proposed a model imitating the communication between neurons using an electrical circuit which is now called the born of neural networks.

In 1950, "The Imitation Game (also called as Turing Test)" was invented by Alan Turing. The aim of this test was to determine if the machine is capable of behaving/thinking as an intelligent minded like humans.

In 1952, Arthur Samuel has written a checkers-playing program and completed this first learning program in 1955. Since it is the first ever learning program as it ran, it is accepted as a milestone in artificial intelligence.

In 1957, the psychologist Frank Rosenblatt has designed the first artificial neural network based on ideas about the work of the human nervous system. He has created a neural network called "Rosenblatt Perceptron" which consists of a single artificial neuron. This was a binary single neuron model used for pattern and shape recognition. However, its single neuron approach was not sufficient for solving non-linear separable problems. This artificial model is acknowledged as an ancestor of today's complex neural networks.

Following this progress, in 1959, ADALINE (Adaptive Linear Neuron or later Adaptive Linear Element) model was created by Bernard Widrow and Marcian Hoff at Stanford University. This model also consisted of single-layer artificial neural network and differs in terms of weight adjustment from the previous model. Moreover, another model called MADALINE (many ADALINE) was proposed which was a multilayer network version of ADALINE. MADALINE consisted of three layers (input, hidden, output) and used ADALINE in its hidden and output layers. This model is still being used today.

In 1967, the nearest neighbor algorithm was developed by Marcello Pelillo to map routes in order to find the most efficient route as a solution for travelling salesperson. This then constituted the basis of basic pattern recognition.

The next important milestone of neural networks can be recorded as in 1986. The term backpropagation which is used to train deep neural networks rediscovered to be used in neural networks and was first announced by researchers Rumelhart, Hinton and Williams from Stanford psychology department, in addition to the previous work of Widrow and Hoff.

In 1990s, studies conducted in machine learning has changed their approach from knowledge-driven to data-driven. Support-vector machines (SVM) and Recurrent Neural Networks (RNN) have gained significance during this era.

Following these developments, in 1997, IBM computer Deep Blue beat the world chess champion Garry Kasparov. Then, in 1998, as a result of research at AT&T Bell Laboratories on digit recognition by using back-propagation, an important success was achieved in terms of accuracy in detecting handwritten postcodes from the US Postal Service.

In 2000s, unsupervised machine learning methods gained importance with Support-Vector Clustering and kernel methods.

In 2006, the term deep learning was first introduced by Geoffrey Hinton. Following his previous article on backpropagation in 1986, Hinton has published (Hinton et al., 2006) where a new approach is introduced to train multiple layered network of restricted Boltzmann machines (RBM).

In this article this new "deep" approach is defined as;

"Using complementary priors, we derive a fast, greedy algorithm that can learn deep, directed belief networks one layer at a time, provided the top two layers form an undirected associative memory."

Since the beginning of 2010s, deep learning approaches has become widespread for solving much complex problems. Recent studies mostly conducted by using deep learning algorithms. The following lists several examples of these studies;

- Google Brain (2012)
- AlexNet (2012)
- DeepFace (2014)
- OpenAI (2015)
- Amazon's Machine Learning Platform (2015)
- ResNet (2015)
- U-net (2015)
- DeepMind (2016)
  Created AlphaGo program that managed to beat Lee Sedol who is known as the greatest Go player of the past decade in 2016.

2.4.2. Machine Learning Approaches for Intrusion Detection Systems

In survey (Farah et al., 2015), the machine learning techniques used for intrusion detection systems are classified as three major categories; supervised learning, unsupervised learning and reinforcement learning.

- **Supervised Learning**
  Also known as classification, in this learning technique, the model is trained with a dataset containing labelled instances. Hence, supervised learning algorithms aim to predict the output values for new data by modelling dependencies and connection between input features and labels of the dataset which will be predictions after the training phase. Commonly used supervised algorithms are;
  Artificial Neural Network, Bayesian Statistics, Gaussian Process Regression, Lazy learning, Nearest Neighbor algorithm, Support Vector Machines (SVM), Hidden Markov Model, Bayesian Networks, Decision Trees (C4.5, ID3, CART, Random Forrest), K-nearest neighbor, Boosting, Ensembles classifiers (Bagging, Boosting), Linear Classifiers (Logistic regression, Fisher Linear discriminant, Naive Bayes classifier, Perceptron, SVM), Quadratic classifiers etc.

- **Unsupervised Learning**
  In this unsupervised learning, the model is trained with unlabeled dataset. Clustering is one of the mostly used methods in this learning technique. The algorithms are mostly used for pattern detection and descriptive modeling. Since there are not any labels to learn, algorithms used in this technique mine the unlabeled input data by summarizing and grouping related data points, detecting patterns in order to obtain significant information and make predictions accordingly.
  Commonly used unsupervised algorithms are;
  Cluster analysis (K-means clustering, Fuzzy clustering), Hierarchical clustering, Self-organizing map, Apriori algorithm, Eclat algorithm and Outlier detection (Local outlier factor)

- **Reinforcement Learning**
  Besides being a part of Machine Learning techniques, reinforcement learning is also a branch of artificial intelligence. In this method, the algorithm continues to learn iteratively from the environment it is running according to a reward mechanism. The main aim is to maximize the cumulative reward until the full range of possible states is achieved. Commonly used reinforcement learning algorithms are;
  Q-Learning, Temporal Difference (TD), Deep Adversarial Networks

(Mishra et al., 2019) provided a survey on the analysis of machine learning techniques for intrusion detection. In this survey, they highlighted several machine learning approaches with examples of related works.

According to this survey, generic framework for network anomaly detection is used for machine learning. At first, the labeled or unlabeled dataset is decided to be the input to the chosen technique. After that, in order to ease the operation, dataset is reorganized and prepared in the data processing phase. According to the chosen machine learning technique, anomaly detection is made as supervised or unsupervised and finally generates output as prediction of the model. Then as the final phase, the model is evaluated according to the scores or labels produced.

2.4.3. Deep Learning

Deep learning is a subset of machine learning and artificial intelligence. As mentioned in the previous chapter, it has become widespread since the beginning of 2010s.

With the enhancements in the artificial neural networks, deeper neural networks were started to be trained. This new approach is called Deep Learning and achieved very promising results in many areas. Anomaly detection is one of the popular fields where deep learning techniques applied.

As mentioned in (Vinayakumar et al., 2019), by the help of hidden layers, deep learning is able to learn hierarchical feature representations and sequential relationships of more complex datasets which is very efficient for network intrusion detection. Deep learning is also applied in several fields such as image processing, machine translation, speech recognition, natural language processing etc.

Deep learning strongly depends on neural network architectures which are a combination of multiple layers of neurons. In, (Amini et al., 2006), neural networks are categorized by their *architecture, learning algorithm* and *activation functions*.

Figure 9 represents a three-layered fully connected neural network. The layer on the left side is called “input layer” and the neurons in this layer are called input neurons. The layer on the right side is called “output layer” containing a single output neuron. The middle layer is called “hidden layer” that constructs a bridge between input and output layers. This architecture may be sufficient for relatively more simple tasks, however, in case of solving more complex problems, we need to divide the main problem into sub-problems to ease the process of design.

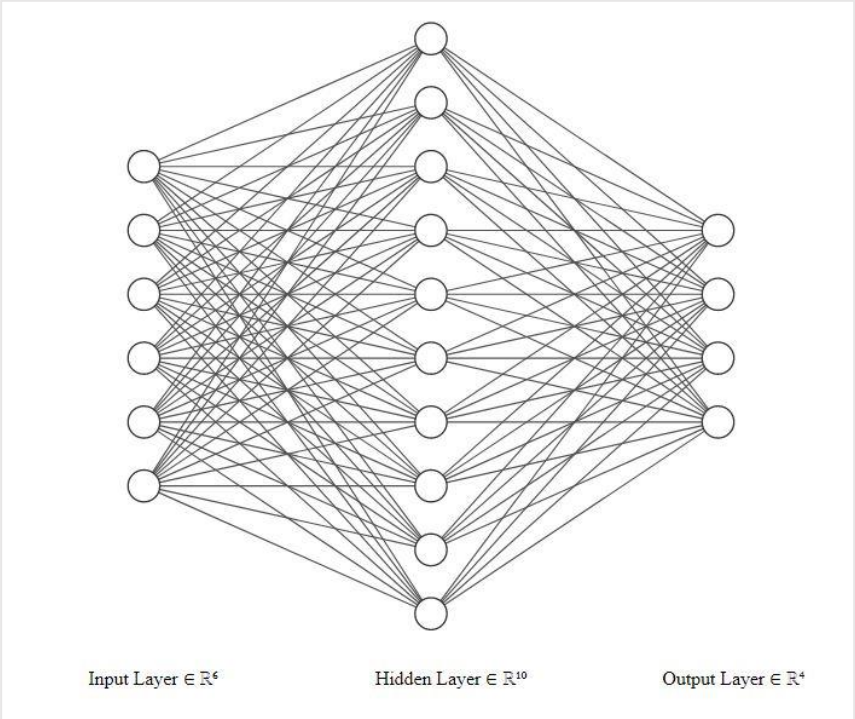

Figure 9: 3- Layered Neural Network

In order to obtain a deeper network for a multi-layered approach, intermediate/hidden layers are used. When we add more hidden layers, the architecture gets deeper. This approach helps to solve complex pattern recognition problems.

Figure 10 shows three hidden layered fully connected deep neural network architecture.

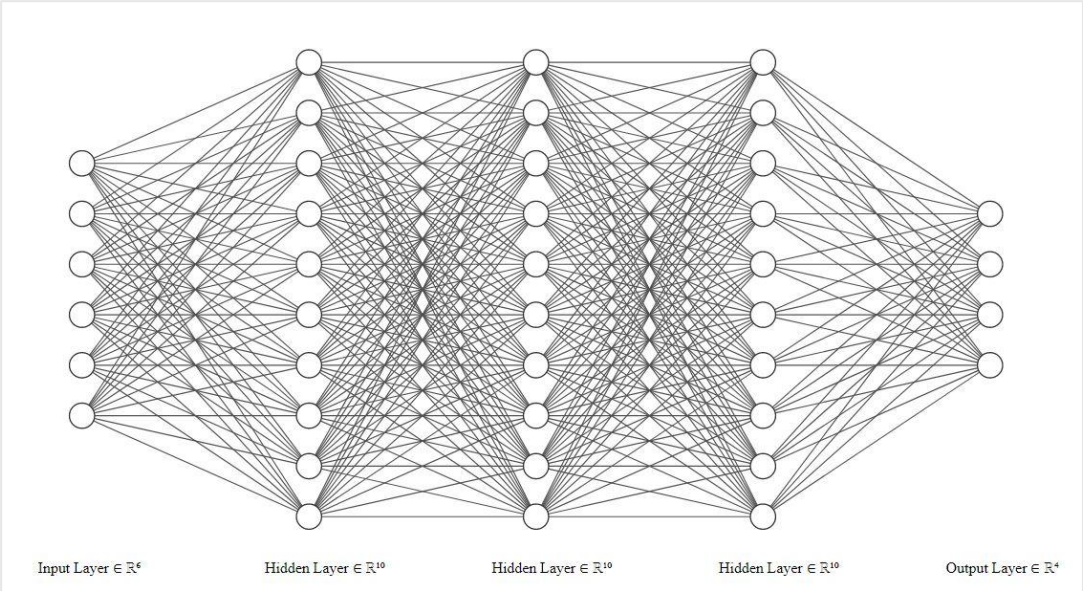

Figure 10: Deep Neural Network with 3-Hidden Layers

According to (Kwon et al., 2019), deep learning can be categorized into three classes;

- Deep networks for unsupervised or generative learning: Restricted Boltzmann Machine (RBM), deep Boltzmann Machine (DBM), Deep Belief Network (DBN), Autoencoders, Deep Autoencoders
- Deep networks for supervised learning: Convolutional Neural Networks (CNN), Recurrent Neural Networks (RNN)
- Hybrid deep networks: Composed of two or more deep learning algorithms

In this thesis, we focus on deep networks for supervised learning algorithms to build our solution.

2.4.3.1. Recurrent Neural Networks (RNN)

As stated in the previous topic, RNN is a supervised deep learning algorithm. RNNs make use of sequential data or time-series data to build a model. RNNs are currently being used in many fields such as language translation, natural language processing (nlp), speech recognition, and image captioning. Popular applications such as Siri, voice search, and Google Translate also uses RNN.

In Chapter 10, "Sequence Modeling: Recurrent and Recursive Nets" of (Goodfellow et al., 2016), it is depicted that, RNNs operate on a sequence which contains vectors (t) with the time step index between 1 and τ. (in this case, sequence of values will be in the form of: x (1), . . ., x(τ))

In practice, this time step index value does not have to depend on time, it can also refer to the position in the sequence.

There are different representations of RNNs. As shown on the left side of Figure 11, "rolled RNN" is the squashed version of the whole network. It summarizes different layers of RNN into one input, hidden and output layer with a temporal loop. As shown on the right side of Figure 11, "unrolled RNN" is a more detailed representation where each circle of rolled RNN corresponds to whole layer of neurons.

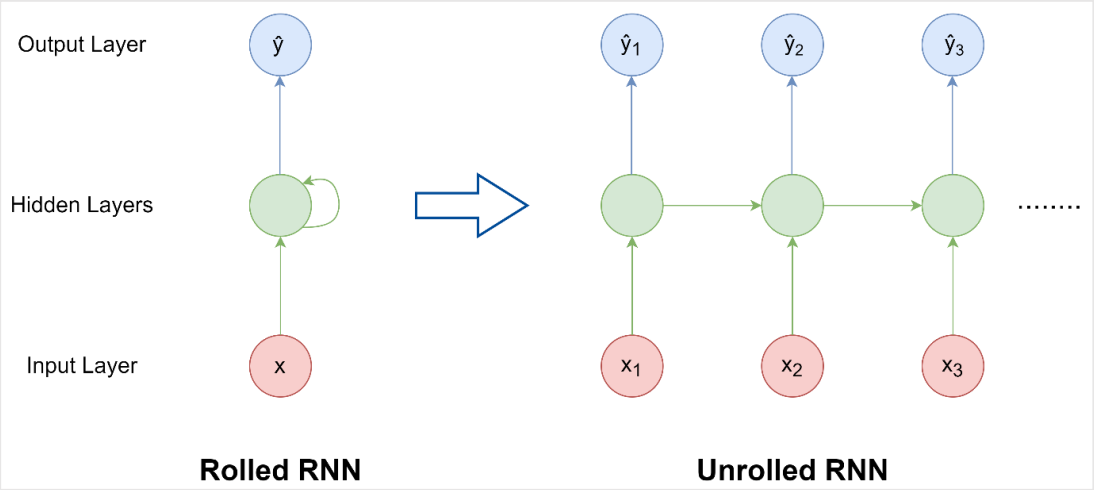

Figure 11: Rolled and Unrolled RNN

Most used RNN design types are listed in the following:

- Recurrent networks that produce an output at each time step and have recurrent connections between hidden units. (The variable "s" represents the state of the system, "x" represents the input value and "f" is the function that maps the state at t-1 to the state at t.) Equation (1) belongs to 3 timesteps where t is equal to 3 in this case.

$$s(3) = f(f(s(1);\theta);\theta) \quad (1)$$

- Recurrent networks that produce an output at each time step and have recurrent connections only from the output at one time step to the hidden units at the next time step where the state contains information about the whole past sequence. (The variable "s" represents the state of the system, "x" represents the input value and "f" is the function that maps the state at t-1 to the state at t.)

$$s(t) = f(s(t-1), x(t);\theta) \quad (2)$$

- Recurrent networks with recurrent connections between hidden units, that read an entire sequence and then produce a single output. Equation (3) is used by many RNN to define the values of hidden units. (The variable "h" represents the state, "x" represents the input value and "f" is the function that maps the state at t-1 to the state at t.) In addition, output layers are added to the RNN architecture that benefits state h to make predictions.

$$h(t) = f(h(t-1), x(t);\theta) \quad (3)$$

There are different types of applications of RNNs which are;

(The illustrations shown in Figure 12, Figure 13, Figure 14, Figure 15 and Figure 16 are adapted from Stanford University's knowledgebase (Afshine Amidi, n.d.))

Here x is to the input, $\hat{y}$ is the output, a is the activation, Tx and Ty are the time steps.

- **One-to-one:** This is the traditional neural network where there is only one input and one output. (Tx=Ty=1) Shown in Figure 12.
- **One-to-many:** This type of RNNs mostly used in music generation where there are one input and multiple outputs. (Tx=1, Ty>1) Shown in Figure 13
- **Many-to-one:** This type of RNNs consist of multiple inputs and one output. This model is mostly used for sentiment analysis. (Tx>1, Ty=1) Shown in Figure 14.
- **Many-to-many:** In this model, multiple inputs and outputs are included. This model can be divided onto two sub models where number of inputs and outputs are equal and not equal to each other.
  As shown in Figure 15, inputs and outputs are equal to each other (Tx=Ty).
  This sub model is widely used for named-entity recognition (NER).
  As shown in Figure 16, there are multiple inputs and outputs that are not equal to each other. (Tx≠Ty) This sub model is most commonly used for machine translation, in other words language translation. Since word count can differ while translating a sentence in one language into another, this sub model fits well as a solution.

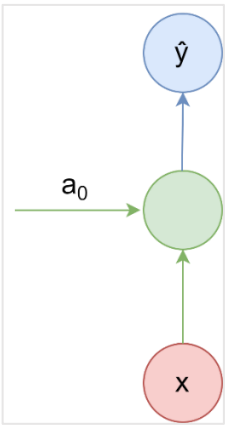

Figure 12: One to One RNN

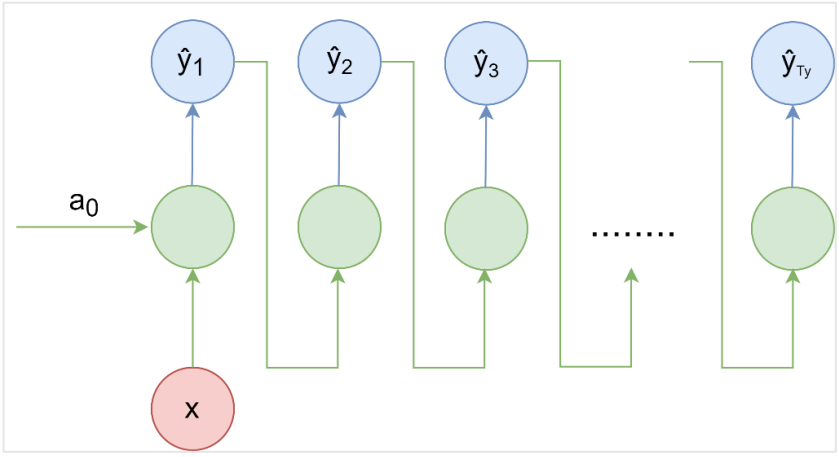

Figure 13: One to Many RNN

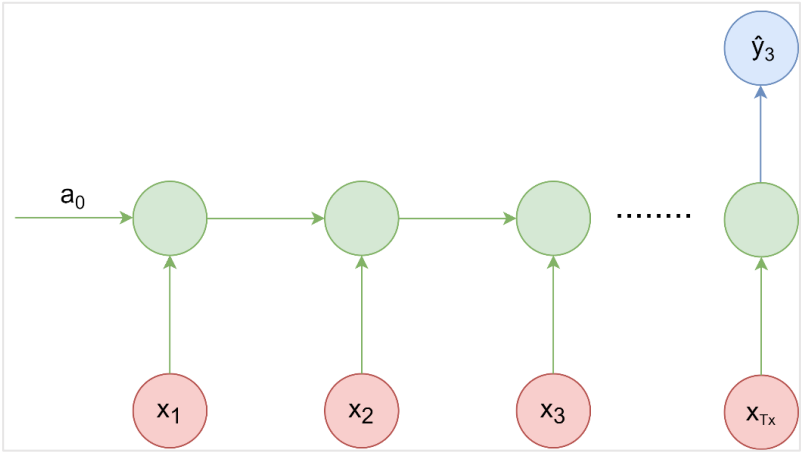

Figure 14: Many to One RNN

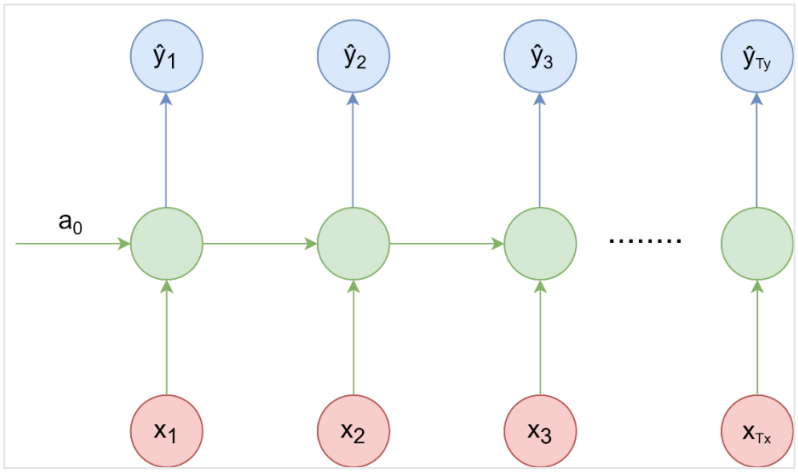

Figure 15: Many to Many RNN (Tx=Ty)

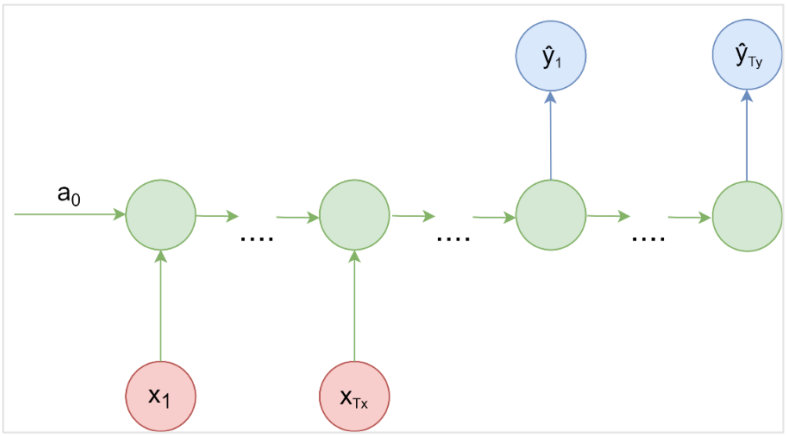

Figure 16: Many to Many RNN (Tx≠Ty)

Vanishing/Exploding Gradient Problem

Especially when training large networks, one of the most important problems when training recurrent neural networks with gradient-based learning methods and backpropagation is the vanishing or exploding gradients. This problem was first introduced by Sepp (Joseph) Hochreiter back in 1991 as a diploma thesis (Hochreiter, 1991).

When training RNNs, cost function compares the predicted output and real output to adjust the weights accordingly with back propagation. Gradient descent algorithm is used to minimize the cost function by calculating the global minimum of it for optimization of the network.

In RNNs, not only the information of the given inputs at the specified time (t), but also the information belonging to the previous time points (t-1) is used as an input to the next time points (t+1) and so on. For each time point, in order to optimize the network, cost function is calculated.

Figure 17 is originated from (Pascanu et al., 2013), when backpropagation is applied to the unrolling RNN.

Supposing that;

- x_t is the state of the network at time t
- u_t is the input of the network at time t
- W_{rec} is the recurrent weight matrix
- b is the bias
- W_{in} is the input weight matrix
- ε_t is the cost function obtained from the output at time t
- σ is an element-wise function

Also note that all the parameters are collected in θ.

$$\mathrm{x}_t = F(\mathrm{x}_{t-1}, \mathrm{u}_t, \theta) \tag{4}$$

$$\mathrm{x}_t = W_{rec}\sigma(\mathrm{x}_{t-1}) + W_{in}u_t + b \tag{5}$$

The unrolling recurrent neural network schema in Figure 17 shows the relationship between these parameters in case of forward and back propagation.

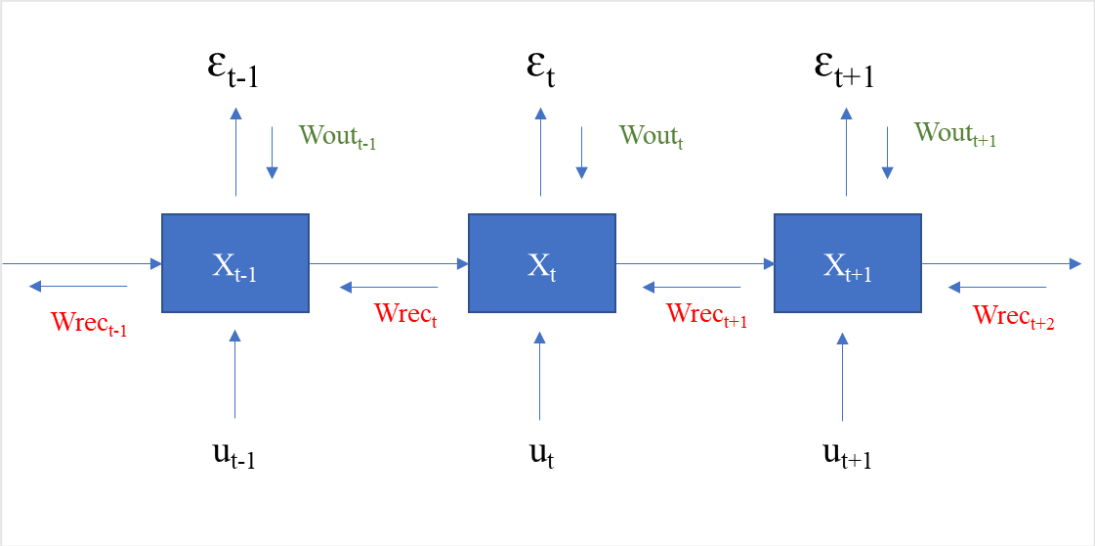

Figure 17: Unrolled Network Schema of RNN

The parameters are listed in (6),(7),(8),(889),(10),(11) & (12)8 where Wrec is the recurring weight.

$$\text{Wout}_{t-1} = \frac{\partial \varepsilon_{t-1}}{\partial x_{t-1}} \tag{6}$$

$$\text{Wout}_{t} = \frac{\partial \varepsilon_{t}}{\partial x_{t}} \tag{7}$$

$$\text{Wout}_{t+1} = \frac{\partial \varepsilon_{t+1}}{\partial x_{t+1}} \tag{8}$$

$$\text{Wrec}_{t-1} = \frac{\partial x_{t-1}}{\partial x_{t-2}} \tag{9}$$

$$\text{Wrec}_{t} = \frac{\partial x_{t}}{\partial x_{t-1}} \tag{10}$$

$$\text{Wrec}_{t+1} = \frac{\partial x_{t+1}}{\partial x_{t}} \tag{11}$$

$$\text{Wrec}_{t+2} = \frac{\partial x_{t+2}}{\partial x_{t+1}} \tag{12}$$

After calculation of the cost function, in order to update the weights accordingly and minimize the error, backpropagation should be done. Considering above unrolled structure of RNNs, each weight belonging to all the neurons of previous timesteps is needed to be updated.

Let's have a look at formula (13) below;

$$\frac{\partial \varepsilon}{\partial \theta} = \sum_{1 \le t \le T} \frac{\partial \varepsilon_t}{\partial \theta} \tag{13}$$

Applying the chain rule, the equation becomes as Formula (14);

$$\frac{\partial \varepsilon_t}{\partial \theta} = \sum_{1 \le k \le t} \left(\frac{\partial \varepsilon_t}{\partial x_t} \frac{\partial x_t}{\partial x_k} \frac{\partial^+ x_k}{\partial \theta} \right) \tag{14}$$

Combining x_i with the previous equation;

$$\frac{\partial x_t}{\partial x_k} = \prod_{t \ge i > k} \frac{\partial x_i}{\partial x_{i-1}} = \prod_{t \ge i > k} Wrec^T diag(\sigma'(x_{i-1})) \tag{15}$$

As can be seen from the resulting equation, to get the state of x_t , x_{t-1}should be multiplicated with $Wrec$. Similarly, when we want to get x_{t-2} , we need to multiply this value again with $Wrec$. Therefore, it has a huge impact on the final result. If $Wrec$ is too small, the result will decrease exponentially with each multiplication while propagating back to the initial layers. In this case, while training the neural network, the weights and biases of the initial layers will not be able to be updated properly with small gradient. Since in RNN, all layers are connected each other which means that updating a weight in any part of the network will also affect the rest of the weights, whole network will be exposed to the same problem. This situation is called "**Vanishing Gradient Problem**" which should be fixed in order to train deep neural networks accordingly. (SuperDataScience Team, 2018)

Likewise, if $Wrec$ is too large, the gradient will increase exponentially which causes an unstable ML model that weights are updated with large changes in cost. This situation is called "**Exploding Gradient Problem**" that should also be fixed for a stable model. (Sherstinsky, 2020) (Goodfellow et al., 2016)

2.4.3.2. Long Short-Term Memory Recurrent Neural Networks (LSTM)

LSTM is a special kind of recurrent neural networks which was firstly introduced by (Hochreiter & Schmidhuber, 1997) as a new gradient-based method in order to provide a solution for the problems introduced in (Hochreiter, 1991). In the light of the previous studies, it shows that RNNs are not efficient for learning long-term dependencies. On the other hand, LSTMs are designed to solve this problem and proved to be capable of learning long-time dependencies without losing short-time lag

capabilities. This is achieved by using constant error flow instead of vanishing or exploding gradients while “Back-Propagation Through Time" (BPTT). As Hochreiter and Schmidhuber highlighted in their study, LSTM has a special type of linear unit with a self-connection of value 1. By making use of input and output gates, the information flow is controlled in order not to deal with vanishing gradient problem.

(Figure 18, Figure 19, Figure 20, Figure 21, Figure 22 are retrieved from (Olah, 2015) with the permission of the writer.)

In Figure 18, there is an LSTM diagram showing the four different gates which are called;

- Forget gate
- Input gate
- Candidate gate
- Output gate

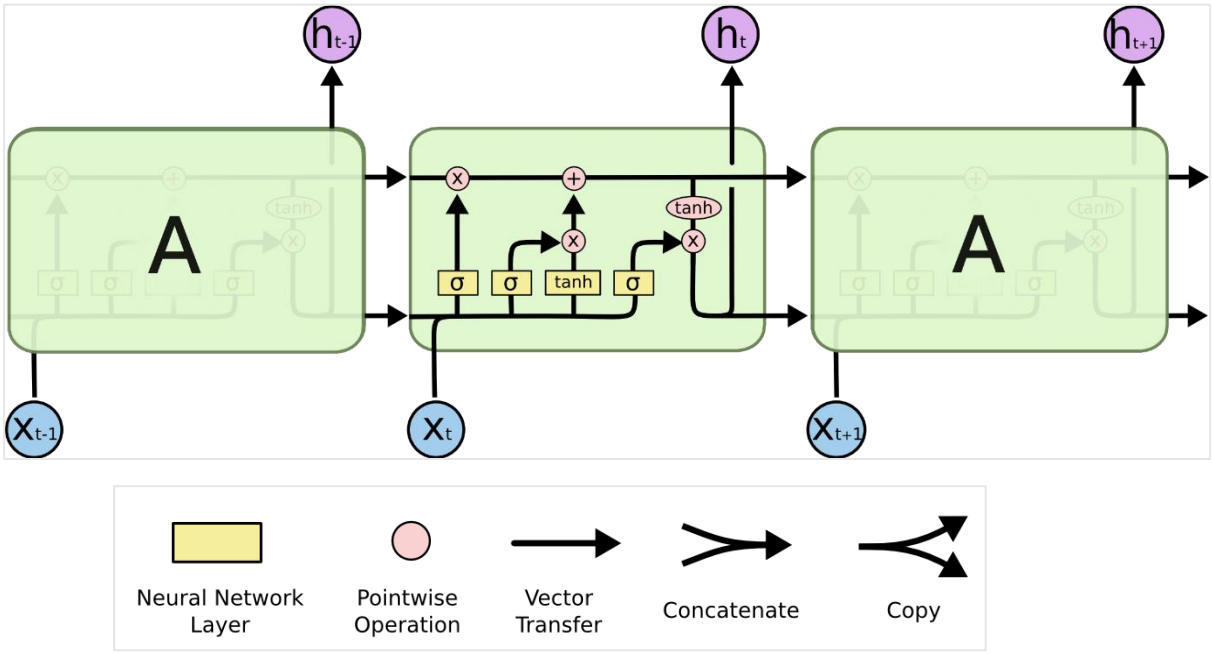

Figure 18: General Architecture of LSTM

According to (Rian Dolphin, 2020), the output of LSTM depends on the following;

- Cell state (long-term memory of the network at time t which is shown as the horizontal line at the top of the Figure 18)
- Hidden state (output of the previous time step)
- Input data (at time t)

The term “gate” is very important in LSTM since it controls all the information flow through the network. Forget gate, input gate and output gate are the three significant gates used in a standard LSTM.

Figure 19 shows the ***forget gate*** (or sigmoid) where it is decided that which information of the cell state should be kept or not. Since sigmoid function is used, the output of this gate will be in the range of [0,1]. If the output is close to 0, the information will not be kept, otherwise, if the output is close to 1, the information will be kept. In this way, irrelevant information is removed from the network that may negatively affect the training process.

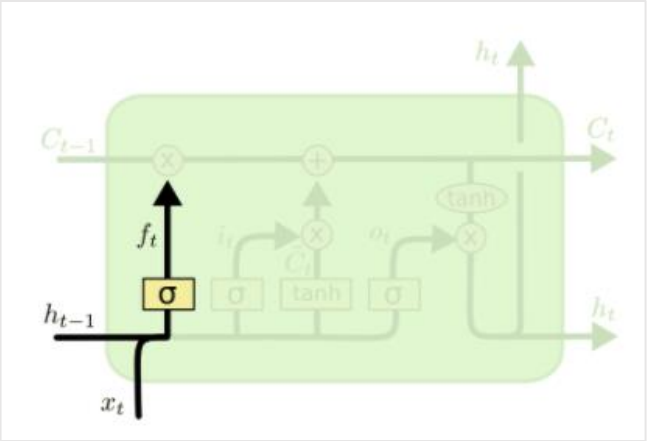

Figure 19: LSTM Forget Gate

In the formula (16), f_t is the forget gate, h_{t-1} is the previous hidden state of the network, x_t is the input data, W_f is the input weight and b_f is the bias. σ is the pointwise nonlinear activation function of this gate. $(\sigma(x) = \frac{1}{1+e^{-x}})$

$$f_t = \sigma(W_f[h_{t-1}, x_t] + b_f) \tag{16}$$

Figure 20 shows the process of selecting which information to be stored in the cell state. Input gate and candidate gate are both used in combination for this process.

Input gate is used to select the values that will be updated. Therefore, memory contents that are already been stored are protected from any disruption by unrelated inputs. (Hochreiter & Schmidhuber, 1997)

After the decision of the input values, new candidate values $\tilde{C}_t$ are created by the ***candidate gate (or tanh gate)***.

tanh is the hyperbolic tangent function that outputs values in the range of [-1,1], having the following formula (17);

$$\tanh(x) = \frac{\sinh(x)}{\cosh(x)} = \frac{e^{2x} - 1}{e^{2x} + 1} \tag{17}$$

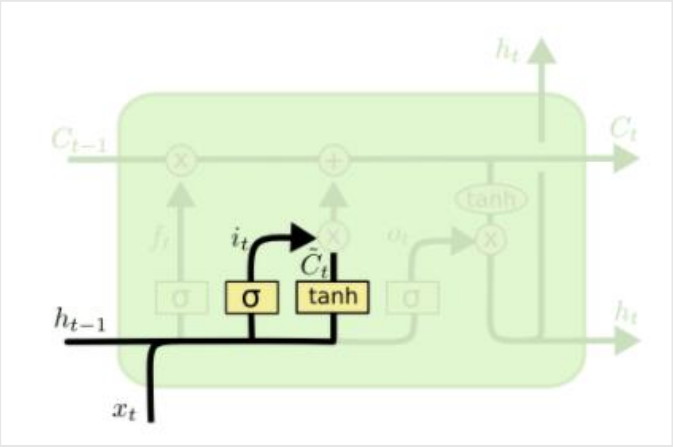

Figure 20: LSTM Input and Candidate Gates

In the following formulas (18-26);

i_t denotes the selection function of the input gate.

$\tilde{C}_t$ denotes the candidate values created as a result of the hyperbolic tangent ($tanh$) function.

$tanh$ is used as the block input and output activation function, σ is used as the gate activation function. (Greff et al., 2017)

W_i and W_C are the input weights belonging to input and candidate gates.

b_i and b_c are the biases of input and candidate gates.

h_{t-1} is the previous hidden state of the network and x_t is the input data.

$$i_t = \sigma(W_i[h_{t-1}, x_t] + b_i) \quad (18)$$

$$\tilde{C}_t = tanh(W_C[h_{t-1}, x_t] + b_C) \quad (19)$$

Now that the candidate values are determined, the ***cell state*** should be updated accordingly. As in Figure 21, at first, the previous state of the network should be multiplicated by f_t to reset the network state by forgetting irrelevant data as determined in the forget gate. After that, new candidate values $\tilde{C}_t$ and selected input values i_t are multiplicated and added to the last result.

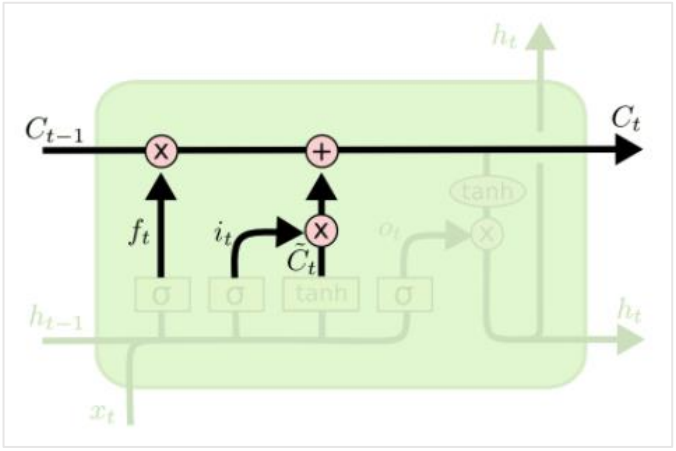

Figure 21: LSTM Cell State Update Process

The following formula (20) shows the final memory cell.

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t \quad (20)$$

After updating the cell state by determining which information should be forgotten and which candidate values should be chosen, there is still so much information to keep in the hidden state. In the ***output gate,*** new hidden state is aimed to be decided. Since each gate of LSTM consists of hidden state, this gate becomes critical where there is an evaluation of which part of the memory C_t should be kept or not in the new hidden state. Similar to forget gate and input gate, logistic sigmoid is used as the gate activation function. The process is illustrated in Figure 22.

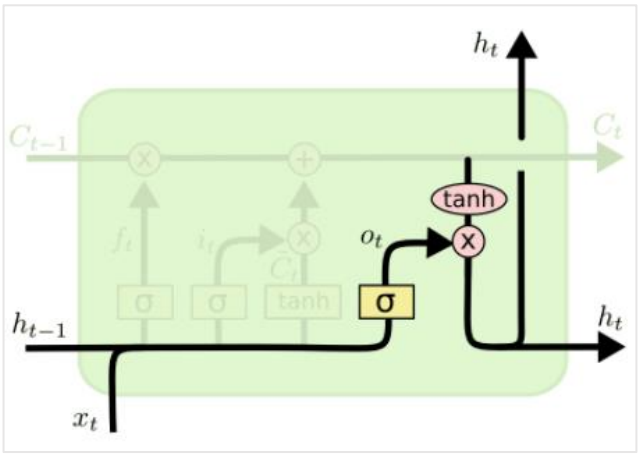

Figure 22: LSTM Output Gate

As shown in the equation (21), a sigmoid function is used to select the output of cell state. Here, previous hidden state h_{t-1}, new input data x_t with the input weight W_o and bias b_o are used as inputs of the sigmoid function which will result in [0,1].

$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o) \quad (21)$$

As in formula (22), newly update cell state C_t is given as an input of tanh to obtain the hidden vector h_t values between (-1,1). Moreover, hidden state is calculated by the

multiplication of the output of the input gate. Hence, only the selected parts will be evaluated as output.

$$\mathrm{h}_t = \mathrm{o}_t * tanh(C_t) \tag{22}$$

The output will be;

$$\mathrm{y}_t = W_o * h_t \tag{23}$$

(The following formulas (24), (25) and (26) are adapted from (Weber, 2017).)

Previously in the "Vanishing/Exploding Gradient Problem" topic, the gradient in RNN while BPTT was calculated as follows; (x_t is changed by h_t as it denotes the hidden state of the network in LSTM)

$$\frac{\partial h_t}{\partial h_k} = \prod_{t \geq i > k} \frac{\partial h_i}{\partial h_{i-1}} = \prod_{t \geq i > k} Wrec^T diag(\sigma'(h_{i-1})) \tag{24}$$

Considering that C_t is a function of f_t , i_t and $\tilde{\mathrm{C}}_t$ and they are all functions of h_{t-1} and therefore C_{t-1}, the previously used chain rule can be modified as follows;

$$\begin{aligned} \frac{\partial C_t}{\partial C_{t-1}} = {} & \frac{\partial C_t}{\partial f_t}\frac{\partial f_t}{\partial h_{t-1}}\frac{\partial h_{t-1}}{\partial C_{t-1}} + \frac{\partial C_t}{\partial i_t}\frac{\partial i_t}{\partial h_{t-1}}\frac{\partial h_{t-1}}{\partial C_{t-1}} \\ & + \frac{\partial C_t}{\partial \tilde{\mathrm{C}}_t}\frac{\partial \tilde{\mathrm{C}}_t}{\partial h_{t-1}}\frac{\partial h_{t-1}}{\partial C_{t-1}} + \frac{\partial C_t}{\partial \mathrm{C}_{t-1}} \end{aligned} \tag{25}$$

$$\begin{aligned} \frac{\partial C_t}{\partial C_{t-1}} = {} & C_{t-1}\sigma' W_f * o_{t-1} tanh'(C_{t-1}) + \tilde{\mathrm{C}}_t \sigma' W_i \\ & * o_{t-1} tanh'(C_{t-1}) + i_t tanh' W_C \\ & * o_{t-1} tanh'(C_{t-1}) + f_t \end{aligned} \tag{26}$$

In this case, unlike $\frac{\partial h_t}{\partial h_{t-1}}$ ***always*** having values in range of [0,1] or greater than 1 that was causing the vanishing/exploding gradient problem, $\frac{\partial C_t}{\partial C_{t-1}}$ will take values between (0,1) or greater than 1 ***at any time step***. The main difference is that although it may occur, this situation is not expected to cause a vanishing/exploding gradient problem since there is a new parameter called forget gate f_t which is added at the end of the new equation. This value can be adjusted accordingly by the decision of the network as a result of the learning process to overcome potential gradient problems as the amount of time steps increases.

To sum up, as explained briefly in this chapter why LSTMs are preferred for long time dependency problems, in case of network traffic analysis and anomaly detection which is highly time dependent, this algorithm is chosen as the best fit as a solution in this thesis.

2.4.4. State of the Art

Throughout years, there have been many researches and studies conducted to build an intrusion detection system by using machine learning techniques. Whilst machine learning methods such as J48, ANN, RF, SVM, k-NN, Naïve Bayes were mostly popular in previous works, more recent works include deep learning methods such as RNN, LSTM and CNN which give very promising results comparing to the previous machine learning methods.

According to a recent survey on neural networks usage on intrusion detection systems, (Drewek-Ossowicka et al., 2021), many significant works were compared with each other by which dataset(s), methods were used and which attack types were analyzed.

In (Tang et al., 2016), NSL-KDD dataset was used with a three hidden layered DNN. As a result of this study, comparing the outcome with Naïve Bayes, SVM and Decision Tree algorithms, it is concluded that DNN gives the highest accuracy which was calculated as 75.75% for binary classification.

In (Yin et al., 2017), NSL-KDD dataset was analyzed by using RNN. In this study, RNN with different numbers of hidden nodes and learning rates have been tested by tuning parameters and overall accuracy and performance were compared among different topologies. The highes accuracy was calculated as 99.81% for binary classification and 99.53% for multi-class classification.

In (Kim et al., 2016), NSL-KDD dataset was used with LSTM-RNN deep learning method. As a result, they obtained detection rate as 98.88% and accuracy as 96.93%. Both values are evaluated as the highest comparing with the GRNN, PNN, RBNN, k-NN, SVM and Bayesian machine learning methods.

As can be seen, there are many studies conducted by using KDD Cup 1999 or NSL-KDD datasets. However, in order to build an intrusion detection system that is able to detect today's attacks, these datasets become insufficient since they include out-of-date attacks. For this reason, CSE-CIC-IDS2018 dataset, which is one of the most recent network-based public datasets is examined in this thesis.

While the usage of this dataset becomes more popular, a survey (Leevy & Khoshgoftaar, 2020) was made by just including the studies on CSE-CIC-IDS2018 dataset. In this survey, several research papers were investigated and compared by their performance scores (accuracy, precision, recall and AUC values), proposed models and computing environment.

In (Atefinia & Ahmadi, 2021), the main aim was to gain a higher precision rate than the previous works. For this reason, a multi-architectural modular deep neural network model as a combination of four different architectures was proposed as a solution. Deep Feed-Forward Neural Network (DFFNN), Stacked Restricted Boltzmann Machine (SRBM), Gated Recurrent Unit (GRU) and LSTM models were integrated with an aggregator module developed as a weighted averaging technique that produces output for the overall network. Multi-class classification was applied and the highest accuracy, precision and recall rates were obtained as 100% for DoS, DDoS and brute force attack types. These results are highly indicating an overfitting problem. Moreover, all experiments were conducted using the subsets of dataset. Since only

results for each 6 attack class were included, this paper did not include overall results. Also, performance metrics did not cover specific metrics for imbalanced datasets such as G-mean score, balanced accuracy, ROC AUC & PR AUC. Although there were results shown with N/A values, the reason of this absence was not explained. Another limitation of this work is that there was not sufficient information for the used hyperparameters for each model which highly affects the reproducability. In Table 2, only results with the highest and lowest scores are included.

In (Basnet et al., 2019), Multilayer Perceptron (MLP) model was trained for multi-class and binary classification. The main focus of this paper was to compare various deep learning frameworks such as Keras, TensorFlow, Theano, fast.ai and PyTorch by their performance. Accuracy and confusion matrix were used as performance metrics. The results were compared according to accuracy scores and processing time. However, this work also had its limitations. In this work, there was not sufficient information of the hyperparameters used to obtain these results. The only explanation was that at least 3 hidden layers were used. This situation highly affects the reproducability of this work. Moreover, there was no mention of the performance metrics used specifically for class imbalance problem. In Table 2, only results with the highest and lowest scores are included.

As mentioned in (Ferrag et al., 2020), deep learning approaches used for cyber security intrusion detection are summarized as ten. (DNN: **D**eep **N**eural **N**etwork, FFDNN: **F**eed **F**orward **D**eep **N**eural **N**etwork, RNN: **R**ecurrent **N**eural **N**etwork, CNN: **C**onvolutional **N**eural **N**etwork, RBM: **R**estricted **B**oltzmann **M**achine, DBN: **D**eep **B**elief **N**etwork, DA: **D**eep **A**utoencoder, DML: **D**eep **M**igration **L**earning, STL: **S**elf **T**aught **L**earning, ReNN: **R**eplicator **N**eural **N**etwork)

In this study, seven deep learning models (recursive neural networks (RNN), deep neural networks (DNN), restricted Boltzmann machines (RBM), deep belief neural networks (DBNN), convolutional neural networks (CNNs), deep Boltzmann machines (DBM), deep autoencoders) were applied to two real traffic datasets which are CSE-CIC-IDS2018 and Bot-IoT dataset. The applied method contained four stages such as datasets, pre-processing, training and testing stage. The training and testing stages were performed on only a 5% of the entire dataset. As a result, the highest accuracy was obtained for CSE-CIC-IDS2018 as 97.376% with CNN. In addition, highest recall on this dataset was obtained as 98.18% by using deep autoencoder.

The major limitation of this work is that only 5 % of the dataset was used for training the model. In addition, used methods for the pre-processing phase were not explained in detail. For evaluating the model, only Detection Rate and Accuracy were used as performance metric. There was no mention of the other performance metrics used for class imbalance problem. Moreover, the used hyperparameters were not shared for the multi-class classification that affects the reproducability of this work.

In (Gamage & Samarabandu, 2020), a benchmark including four deep learning models for intrusion detection is presented. Besides comparing the outcomes of the studies conducted with several datasets and models, three different approaches are introduced as a contribution to the previous works. These approaches consist of; feed-forward neural networks for flow instance classification, LSTM for classifiying sequential flows, autoencoder and deep belief networks for semi-supervised training of the

model. Also, CSE-CIC-IDS2018 dataset is among the modern datasets tested in this research. According to the performance results for this dataset, the highest accuracy is obtained as 98.88% by using LSTM.

While whole dataset samples and wide variety of performance metrics were included while evaluating the model, only performing binary classification is one of the limitations of this study. Another limitation was that there was not neough information for the pre-processing phase which is necessary to reproduce this work.

To conclude, due to the structure of the dataset, LSTM gives one of the highest results for learning the sequential data. The reason of obtaining different performance results with the same dataset and deep learning approach is using different parameters with different topology and also data preprocessing part is handled in many different ways. Although most of the papers did not include details about how they perform pre-processing, it is one of the most important phases of data preparation for learning. Also, used hyperparameters for obtaining each results was not mentioned in detailed in most of these studies which is also necessary for reproducability.

Table 2 shows the results obtained with different models and hyperparameters for each related studies conducted by using CSE-CIC-IDS2018 dataset.

In this thesis, we aim to analyze the dataset by including all the details about how we prepare the data and how we build our topology and select parameters in order to obtain the best performance by comparing our results and approach with the previous studies.

Table 2: Related works with CSE-CIC-IDS2018

Reference Study	Classification Type	Model	Hyperparameters	Results
(Atefinia & Ahmadi, 2021)	Binary classification	SRBM	Hidden layers: 6	Brute-Force:
			Learning Rate Decay Value: 0.9	Accuracy: 100 %
				Precision: 1.0
				Recall: 1.0
				F1 Score: 1.0
		DFNNN	Hidden Layers: 12	DoS & DDoS:
			Batch Size: 10000	Accuracy: 100 %
			Learning Rate: 0.01	Precision: 1.0
			Optimizer: Adam	Recall: 1.0
			Activation: Categorical cross-entropy (multi-class)	F1 Score: 1.1
		GRU	Hidden layers: 6	Web Attacks:
				Accuracy: N/A
				Precision: N/A
				Recall: N/A
				F1 Score: N/A
		LSTM		Infiltration:
				Accuracy: N/A
				Precision: N/A
				Recall: N/A
				F1 Score: N/A
(Basnet et al., 2019)	Multi-class classification	Multilayer Perceptron (MLP)	Hidden Layers: 3 (at least)	fast.ai:
				Accuracy: 98.31 %
				Keras-TensorFlow:
				Accuracy: 94.73 %
	Binary classification			fast.ai:
				Accuracy: 98.68 %
				Keras-TensorFlow:
				Accuracy: 94.40 %

Table 2 (cont.)

(Ferrag et al., 2020)	Binary classification	RNN	Hidden Nodes: 100 Learning Rate: 0.5	Accuracy: 97.31 %
		CNN	Hidden Nodes: 100 Learning Rate: 0.5	Accuracy: 97.38 %
	Multi-class classification	RNN	N/A	Detection Rate: SSH-Bruteforce: 100 % FTP-Bruteforce: 100 % SQL Injection: 100 % DoS attacks-GoldenEye: 98.330 %
		CNN	N/A	Detection Rate: SSH-Bruteforce: 100 % FTP-Bruteforce: 100 % SQL Injection: 100 % DDOS attack-LOIC-HTTP: 98.991 %
(Gamage & Samarabandu, 2020)	Binary classification	ANN	Hidden Layers: 4 Hidden Nodes: 128,64,32,16 Dropout Rate: 0.2 Batch Size: 256	Accuracy: 98.38 % Detection Rate: 0.9836 Precision: 0.9754 FAR: 0.0492 FNR: 0.0162 F1 Score: 0.9785
		AE + ANN	Layers: 3 Hidden Nodes: 128,64,32,16 Dropout Rate: 0.2 Batch Size: 256	Accuracy: 98.22 % Detection Rate: 0.9822 Precision: 0.9750 FAR: 0.0514 FNR: 0.0178 F1 Score: 0.9768
		DBN + ANN	Layers: 3 Hidden Nodes: 128,64,32,16 Dropout Rate: 0.2 Batch Size: 256	Accuracy: 98.31 % Detection Rate: 0.9831 Precision: 0.9761 FAR: 0.0496 FNR: 0.0169 F1 Score: 0.9778
		LSTM(shallow)	Layers: 1 Hidden Nodes: 32 Dropout Rate: 0.2 Batch Size: 256 Sequence length: 32	Accuracy: 98.88 % Detection Rate: 0.9888 Precision: 0.9840 FAR: 0.0509 FNR: 0.0112 F1 Score: 0.9839

CHAPTER 3

PROPOSED APPROACH

3.1. Dataset

Considering emerging cyber-attacks, building an intrusion detection system by training a model with the most up-to-date realistic dataset plays a major role for an efficient solution. Therefore, within the scope of this thesis, CSE-CIC-IDS2018 public dataset is decided to be used. This dataset was created by a collaborative project between the Communications Security Establishment (CSE) & the Canadian Institute for Cybersecurity (CIC) in 2018. This dataset was mainly focused on network-based anomaly detection by providing several real-world alike scenarios within a diverse environment. (*CSE-CIC-IDS2018 on AWS*, n.d.)

The diverse environment was built on the AWS computing platform. The overall infrastructure was divided into two as attacking and victim organization. The attacking organization was composed of 50 machines. The victim organization consisted of 420 machines and 30 servers. This dataset was created by collecting all the network traffic and system logs from those machines.

The following six different attack scenarios were conducted during the experiments;

- DDoS
- DoS
- Brute-force
- Botnet
- Infiltration
- Web Attack

As shown in Figure 23, so as to simulate all attack scenarios, different operating systems (Windows, Ubuntu, Mac) with different versions (Ubuntu 12.04, 16.04, Windows Vista,7,8.1,10 etc.) running various services were included in the environment.

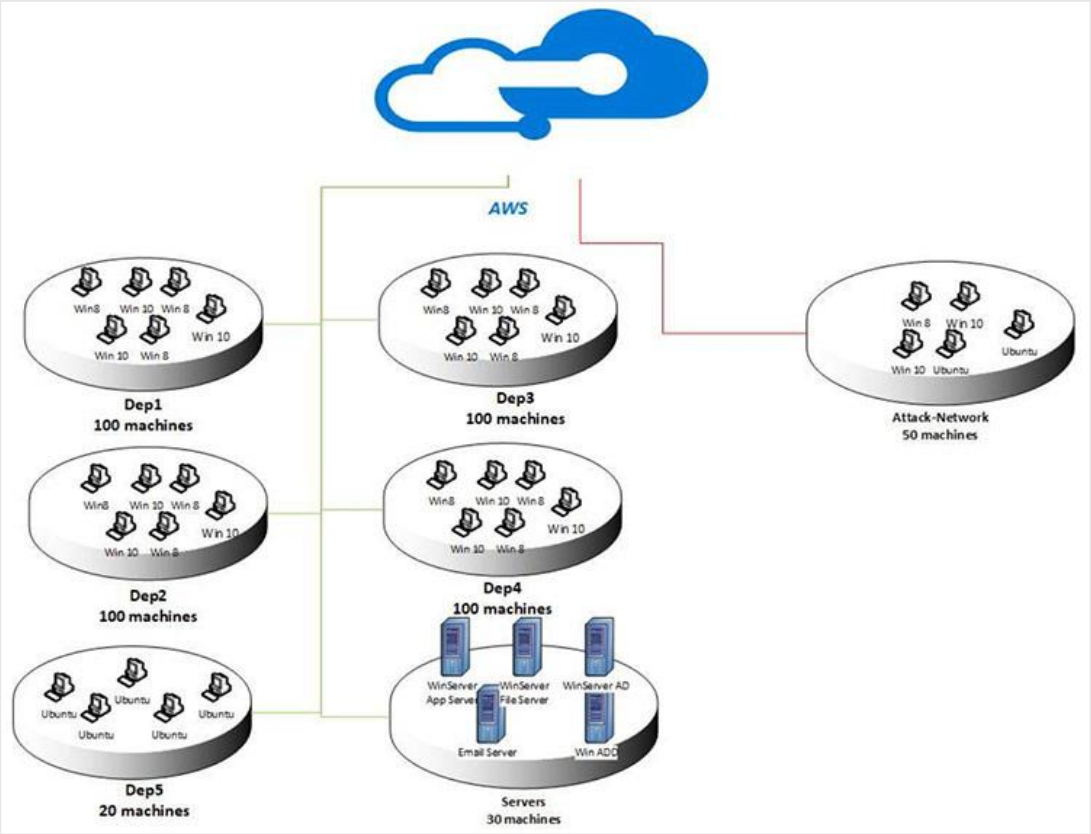

Figure 23: CSE-CIC-IDS2018 Attack Topology

One of the realistic specialties this dataset offers is 83% benign traffic percentage. Since most of the traffic is also expected to be normal in a real-world topology, training the anomaly detector model is very critical by making use of this produced benign traffic. This will make an advancement in our model's capability to differ benign traffic from attack or suspicious traffic which will decrease false positive rate and increase precision.

Each attack was executed from one or more machines outside the target network. Similar to real-world network topologies, 5 different subnets simulating different departments and server rooms of the company were deployed.

The network traffic distribution of this dataset is given in Table 3. All 14 attack types are matched with the attack scenarios.

(Values are taken from (Leevy & Khoshgoftaar, 2020))

Table 3: Attack Distribution of the Dataset

Attack Scenario	**Attack Name**	**Distribution (%)**
Benign	None	83.07
DDoS	DDoS attacks-LOIC-HTTP	7.786
	DDOS-LOIC-UDP	
	DDOS-HOIC	
DoS	DoS-GoldenEye	4.031
	DoS-Slowloris	
	DoS-SlowHTTPTest	
	DoS-Hulk	
Brute force	FTP-BruteForce	2.347
	SSH-BruteForce	
Botnet	Bot	1.763
Infiltration	Infiltration	0.997
Web Attack	Brute Force -Web	0.006
	Brute Force -XSS	
	SQL Injection	

This dataset also provides collected raw network traffic data distributed day by day among 10 CSV files which can be downloaded from AWS cloud. By using CICFlowMeter-V3 as a network traffic flow generator and analyzer, more than 80 features are extracted from network traffic data. (Lashkari, 2020) (Canadian Institute of Cyber Security, n.d.)

CICFlowMeter is capable of generating bidirectional network flows. This is very advantageous for detecting cyber-attacks since it gives results such as Duration, number of packets, number of bytes, packet length in both forward (source to destination) and backward (destination to source) directions.

Table 4 shows the attack tools, victim and attacker environments used to execute each kind of attacks. (*CSE-CIC-IDS2018 on AWS*, n.d.)

Table 4: Attacks Environment of the Dataset

Attack	Tools	Attacker	Victim
Bruteforce attack	FTP – Patator SSH – Patator	Kali linux	Ubuntu 16.4 (Web Server)
DoS attack	Hulk, GoldenEye, Slowloris, Slowhttptest	Kali linux	Ubuntu 16.4 (Apache)
DoS attack	Heartleech	Kali linux	Ubuntu 12.04 (Open SSL)
Web attack	Damn Vulnerable Web App (DVWA) In-house selenium framework (XSS and Brute-force)	Kali linux	Ubuntu 16.4 (Web Server)
Infiltration attack	First level: Dropbox download in a windows machine Second Level: Nmap and portscan	Kali linux	Windows Vista and Macintosh
Botnet attack	Ares (developed by Python): remote shell, file upload/download, capturing screenshots and key logging	Kali linux	Windows Vista, 7, 8.1, 10 (32-bit) and 10 (64-bit)
DDoS+PortScan	Low Orbit Ion Canon (LOIC) for UDP, TCP, or HTTP requests	Kali linux	Windows Vista, 7, 8.1, 10 (32-bit) and 10 (64-bit)

Table 5 shows the list of extracted network traffic features and their description. In addition to the 75 features for network flow traffic, 5 additional features were also added to categorize the traffic: *FlowID, SourceIP*, *DestinationIP*, *SourcePort, DestinationPort*, and *Protocol.*

Table 5: Description of Dataset Features

Feature Name	Description
Flow Duration	Flow duration
Tot Fwd Pkts	Total packets in the forward direction
Tot Bwd Pkts	Total packets in the backward direction
TotLen Fwd Pkts	Total size of packet in forward direction
Fwd Pkt Len Max	Maximum size of packet in forward direction
Fwd Pkt Len Min	Minimum size of packet in forward direction
Fwd Pkt Len Mean	Average size of packet in forward direction
Fwd Pkt Len Std	Standard deviation size of packet in forward direction
Bwd Pkt Len Max	Maximum size of packet in backward direction

Bwd Pkt Len Min	Minimum size of packet in backward direction
Bwd Pkt Len Mean	Mean size of packet in backward direction
Bwd Pkt Len Std	Standard deviation size of packet in backward direction
Flow Byts/s	flow byte rate that is number of packets transferred per second
Flow Pkts/s	flow packets rate that is number of packets transferred per second
Flow IAT Mean	Average time between two flows
Flow IAT Std	Standard deviation time two flows
Flow IAT Max	Maximum time between two flows
Flow IAT Min	Minimum time between two flows
Fwd IAT Tot	Total time between two packets sent in the forward direction
Fwd IAT Mean	Mean time between two packets sent in the forward direction
Fwd IAT Std	Standard deviation time between two packets sent in the forward direction
Fwd IAT Max	Maximum time between two packets sent in the forward direction
Fwd IAT Min	Minimum time between two packets sent in the forward direction
Bwd IAT Tot	Total time between two packets sent in the backward direction
Bwd IAT Mean	Mean time between two packets sent in the backward direction
Bwd IAT Std	Standard deviation time between two packets sent in the backward direction
Bwd IAT Max	Maximum time between two packets sent in the backward direction
Bwd IAT Min	Minimum time between two packets sent in the backward direction
Fwd PSH Flags	Number of times the PSH flag was set in packets travelling in the forward direction (0 for UDP)
Bwd PSH Flags	Number of times the PSH flag was set in packets travelling in the backward direction (0 for UDP)
Fwd URG Flags	Number of times the URG flag was set in packets travelling in the forward direction (0 for UDP)
Bwd URG Flags	Number of times the URG flag was set in packets travelling in the backward direction (0 for UDP)
Fwd Header Len	Total bytes used for headers in the forward direction
Bwd Header Len	Total bytes used for headers in the forward direction
Fwd Pkts/s	Number of forward packets per second
Bwd Pkts/s	Number of backward packets per second
Pkt Len Min	Minimum length of a flow

Pkt Len Max	Maximum length of a flow
Pkt Len Mean	Mean length of a flow
Pkt Len Std	Standard deviation length of a flow
Pkt Len Var	Minimum inter-arrival time of packet
FIN Flag Cnt	Number of packets with FIN
SYN Flag Cnt	Number of packets with SYN
RST Flag Cnt	Number of packets with RST
PSH Flag Cnt	Number of packets with PUSH
ACK Flag Cnt	Number of packets with ACK
URG Flag Cnt	Number of packets with URG
CWE Flag Count	Number of packets with CWE
ECE Flag Cnt	Number of packets with ECE
Down/Up Ratio	Download and upload ratio
Pkt Size Avg	Average size of packet
Fwd Seg Size Avg	Average size observed in the forward direction
Bwd Seg Size Avg	Average size observed in the backward direction
Fwd Byts/b Avg	Average number of bytes bulk rate in the forward direction
Fwd Pkts/b Avg	Average number of packets bulk rate in the forward direction
Fwd Blk Rate Avg	Average number of bulk rate in the forward direction
Bwd Byts/b Avg	Average number of bytes bulk rate in the backward direction
Bwd Pkts/b Avg	Average number of packets bulk rate in the backward direction
Bwd Blk Rate Avg	Average number of bulk rate in the backward direction
Subflow Fwd Pkts	The average number of packets in a sub flow in the forward direction
Subflow Fwd Byts	The average number of bytes in a sub flow in the forward direction
Subflow Bwd Pkts	The average number of packets in a sub flow in the backward direction
Subflow Bwd Byts	The average number of bytes in a sub flow in the backward direction
Init Fwd Win Byts	Number of bytes sent in initial window in the forward direction
Init Bwd Win Byts	# of bytes sent in initial window in the backward direction
Fwd Act Data Pkts	# of packets with at least 1 byte of TCP data payload in the forward direction
Fwd Seg Size Min	Minimum segment size observed in the forward direction

Active Mean	Mean time a flow was active before becoming idle
Active Std	Standard deviation time a flow was active before becoming idle
Active Max	Maximum time a flow was active before becoming idle
Active Min	Minimum time a flow was active before becoming idle
Idle Mean	Mean time a flow was idle before becoming active
Idle Std	Standard deviation time a flow was idle before becoming active
Idle Max	Maximum time a flow was idle before becoming active
Idle Min	Minimum time a flow was idle before becoming active

Having a closer look into the attack profiles and scenarios applied for this dataset;

FTP-Bruteforce and **SSH-Bruteforce** labeled attacks were executed by using Patator which is a multi-purpose brute-forcer tool. While executing brute force attacks, the main motivation of the attacker is to gain unauthorized access to personal/organizational/system/network accounts. In order to achieve this, attackers mostly use automatized tools that work with trial-and-error method by making use of huge amount of user names, passwords, encryption keys, login credentials etc. and their possible combinations. Patator is one of those automated tools. It is multi-threaded tool and written in Python. "ssh_login" and "ftp_login" modules of this attack tool were used to conduct this experiment.
According to the paper from the creators of CICIDS2017 dataset (includes very similar attack types with CSE-CIC-IDS2018), the top most effective features to determine brute force attack type are; (Sharafaldin et al., 2018) Also note that since both SSH and FTP based on TCP, the extracted network flow features are all relative to this protocol.
(The analysis is done with the Wednesday 14-02-2018 dataset which includes only Brute force attack and Benign traffic.)

- ***Init Fwd Win Byts:*** The total number of bytes sent in the TCP initial window in the forward direction. If we examine the day when Patator attacks were executed, the mean value of brute force labelled TCP initial window size is calculated as 20327.14. and benign labelled TCP initial window size is calculated as 4895.22. This gives us a clue that attacks handled by Patator tool set their window size bigger so that they can receive more data from the victim in a smaller amount of time.
- ***Subflow Fwd Byts:*** The average number of bytes in a subflow in the forward direction. When FTP-Bruteforce and SSH-Bruteforce attacks are filtered out, despite having approximately equal average values, it is seen that the maximum subflow forward bytes of Patator attacks is 2024 and benign traffic is 8591554. Also, while Patator attack traffic has the same subflow bytes for multiple flows, Benign traffic has varying subflow bytes. This can indicate an automated attack including several attempts including dictionary selected values having similar lengths.
- ***TotLen Fwd Pkts:*** Total length of the packets sent in the forward direction. This feature works similar to "Subflow Fwd Byts". When FTP-Bruteforce and SSH-Bruteforce attacks are filtered out, it is seen that the maximum subflow

forward bytes of Patator attacks is 2024 and benign traffic is 8591554. This can also indicate an automated attack including several attempts including dictionary selected values having similar lengths.

- ***ACK Flag Cnt:*** TCP packet count with ACK Flag set. This is one of the most critical features to detect brute force attacks. Since with every trial, there will be ACK packet sent from the target which will be proportional to the total trial count. This can be also deterministic whether the victim has any authentication policy that prevents trials after some threshold or not.
- ***Fwd PSH Flags:*** Total count of PSH flag set packets count in the forward direction. In brute force attack, every time a new combination of the credentials is tried, TCP "PSH" and "ACK" flags are set while sending the new data to the victim.
- ***SYN Flag Cnt:*** Number of packets with TCP SYN Flag set. In order to initiate the connection with the victim, during the TCP three-way handshake phase, TCP packets with SYN Flag set are sent. When FTP-Bruteforce and SSH-Bruteforce attacks are filtered out, it is seen that the average SYN Flag Count of brute force attacks are 0 while benign is larger than zero. This may indicate that since TCP SYN Flags are set during the TCP three-way handshake, the brute force attacks are detected and labelled after the handshake. Therefore, this can be a measure for detection.
- ***Fwd Pkts/s:*** Number of packets sent in the forward direction. This is also very related to brute force attacks since it is expected to have more TCP packets in the forward direction (towards the victim) than the backward direction under huge amount of trial and errors. This is also proved when we investigate the dataset. While benign flows have average forward packets/sec 27585.37, Patator flows have a quite larger average value as 340832.66. Most of the time, network security products block or the server does not respond to the incoming packets that causes TCP timeouts.

(*Patator Usage Example*, 2021)
(Lanjelot, 2021)

DoS Attack labeled attacks were executed by using several different attack tools. The purpose of DoS (Denial of Service) attack is to make a resource (website, application or server) incapable of performing its functions. By exhausting the victim service while sending too many requests in a specific time, it may become unavailable to legitimate users. (Nsrav, n.d.)

DoS Attacks-GoldenEye: GoldenEye is a HTTP DoS test tool that uses attack vectors 'HTTP Keep Alive + NoCache'. It is mostly used for testing whether a website is susceptible to DoS attacks or not. This tool at first makes a full TCP connection with the target website. Then it sends multiple requests at long-term and regular intervals by attempting to keep the connection alive. When it is combined by several attacking machines, in other words, botnets, the server's limit of concurrent connections is easily reached so that it cannot respond to the requests of legitimate users. (Jan Seidl, 2020)

DoS Attacks-Hulk: Hulk flood is a DDoS attack tool developed in Python. It aims to exhaust the target resources by generating unique and obfuscated requests to single or multiple URLs from several attack machines. The main difference of this tool from other HTTP Flood tools is that it is able to create unique pattern for each request. By this way, bypassing intrusion detection and prevention systems becomes much easier. (Abdellah Af, 2021)

DoS Attacks- SlowHTTPTest: SlowHTTPTest is a tool developed for simulating several Layer 7 DoS attacks used for exploiting slow HTTP DoS vulnerabilities. Slow HTTP attacks are generally executed by sending HTTP requests slowly to a Web server. If the request is not complete or the transfer rate is too slow, server normally waits for the data transfer is complete. During this time, the server's concurrent connection pool reaches the maximum limit which means that incoming requests from legitimate users cannot be responded anymore. Unlike other DoS attacks, this type of attack is much easier to accomplish since it does not require so much resources.
Attacks that can be implemented using this tool; (*Slowhttptest Usage Example*, n.d.)

- Slowloris
- Slow HTTP POST
- Slow Read attack (based on TCP persist timer exploit) by draining concurrent connections pool
- Apache Range Header attack by causing very significant memory and CPU usage on the server.

In this attack, the attacker exploits the HTTP GET request to exceed the number of HTTP connections limit on the server. In this way, legitimate clients' requests to the server become unresponsive by letting the attacker open multiple HTTP connections to the server.
DoS Attacks- Slowloris: Slowloris Perl-based tool is used to execute this denial-of-service attack. Firstly, a full TCP connection is made to the victim server. In order to keep the connection alive, valid and incomplete crafted HTTP requests are sent to the server at specific time intervals so as to make the sockets stay open. Therefore, incoming new requests requiring new open sockets will be denied and no new connection can be established.
Below features are selected to have the most impact for detecting DoS attacks.
(The analysis is done by combining Thursday 15-02-2018 and Friday 16-02-2018 datasets which include only DoS attack and Benign traffic.)

- ***Bwd Pkt Len Std:*** During Hulk Flood attacks, server keeps sending replies to the attacking machine(s) until its resources are exhausted. When we examine the dataset, the mean standard deviation of the packets in the backward direction of the
  Benign traffic is 138.97 and the DoS traffic is 50.99. The reason of this difference is that during HTTP DoS attack, either the victim waits for the reply from the attacker by keeping the connection open or sends specific HTTP replies (For example, HTTP 200 OK in case of DoS Hulk). This creates a relatively low standard deviation in the packets flowing from the victim to the attacker with respect to the benign traffic.
- ***Flow IAT Min/Mean/Std:*** Flow IAT statistics indicating the time between flows is very important for determining DoS attacks. If the DoS attack type is Slowloris or GoldenEye, the requests are sent slowly in regular periods. When the dataset is analyzed, it can be deducted that the average of benign traffic Flow IAT values are relatively small compared to DoS Slowloris and GoldenEye. Moreover, DoS Hulk has smaller Flow IAT values than Benign since it executes a type of HTTP Flood. When it comes to SlowHTTPTest, the average of Flow IAT values are smaller than the benign traffic in the dataset.
  Although there is not any clear explanation of what kind of attack is handled by using SlowHTTPTest tool, this result shows that "Apache Range Header

attack (or Apache Killer)" is executed. The reason is, during Apache Killer attack, the attacker sends HTTP requests with Range Header set. With this header, byte ranges to be sent to the client are determined in a way that forces the victim to send only the specified parts at once. This creates a bottleneck on the Apache server as it tries to create separate copies of the requested resource. Hence, it becomes eventually unresponsive. Nevertheless, having the victim running an Apache web server is an indicator of this attack. Due to the nature of this attack, smaller Flow IAT values are expected since the request and response are divided into several parts.

- ***Flow Duration:*** This is another feature that needs to be paid attention to while especially determining which type of DoS attacks has been executed. For example, DoS Slowloris and DoS SlowHttp sends requests slower than the legitimate traffic. This makes the mean value of Flow Duration of these attacks higher than the benign traffic. On the other hand, while Hulk Flood, many requests are sent in a limited time that the flow duration regarding to each request/response will be shorter than the benign traffic. Moreover, SlowHTTPTest, namely, Apache Killer is a different kind of DoS attack since it aims to take advantage of the vulnerability of the way that Apache handles HTTP requests. Hence, it has the smallest Flow Duration among the other DoS attacks and benign traffic**.**
- ***Active Min/Mean:*** The minimum and mean active time of the flow. Both features play an important role while determining the type of HTTP DoS attacks. Slow HTTP attacks handled by Slowloris and GoldenEye have greater average value of Active Min and Active Mean time and with respect to Hulk Flood and Apache Killer which is an expected result since their aim is to keep the connection alive for a longer time. On the other hand, DoS Hulk has smaller average value of Active Min and Active Mean than the benign and Slow HTTP attacks which is a common behavior of Flood attacks. Moreover, SlowHTTPTest has zero Active Min and Active Mean value. This may be an indicator of an unsuccessful attack where there is not any active connection started.
- ***Bwd IAT Mean:*** This feature depends on the reply of the victim server. It is deterministic for the detection of specific type of DoS attacks. For example, slow DoS attacks such as Slowloris and GoldenEye have significantly higher average value of Backward IAT Mean than the other DoS attacks and benign traffic. Also, DoS Hulk attack has lower average value of Backward IAT Mean than slow DoS attacks and benign traffic which is an expected behavior of flood attacks. SlowHTTPTest has zero Backward IAT Mean value which is proportional to its Active Mean value. If the attack is unsuccessful, it is expected that no reply is sent from the victim which explains this result.

Brute Force-Web: This attack is executed by using a custom developed code with in-house Selenium framework. The victim machine is Damn Vulnerable Web App (DVWA) which has several vulnerabilities. Brute force attacks are usually executed by using automated tools and predefined or self-prepared dictionaries consisting of username/password pairs, word list of known pages etc. They are very effective in breaking into networks that are vulnerable for having accounts with weak username/password combinations and weak password policy. Web brute force attacks

are executed to obtain hidden content/pages in a web application. By sending HTTP GET or POST requests to the server with a word list of known pages, attacker can easily obtain the available pages on the victim server. (Gsami, Rezos, Thiagoalz, KristenS, D0ubl3 h3lix, Andrew Smith, Jenjava1762, Mtesauro, n.d.)

Brute Force -XSS: Similar to brute force web, this attack is also executed with an automated code developed in Selenium framework on the DVWA victim machine. This attack is executed by injecting malicious code into victim website, generally consisting of client-side scripts. In this way, attackers can bypass access controls and sensitive information kept by users' browsers to be used by the victim website. These attacks can be successful when user input is not validated or encoded correctly.

SQL Injection: This attack is also executed on DVWA including vulnerable PHP/MySQL web application. In this attack, the attacker aims to obtain sensitive information from the database by manipulating it with SQL command injection. These attacks can be successful when the website lacks the input validation or violates the "least privilege" principle while accessing the database.

(The analysis is done by combining Thursday 22-02-2018 and Friday 23-02-2018 datasets which include only Web attacks and Benign traffic.)

Below features are selected to have the most impact for detecting DoS attacks.

- ***Init Fwd Win Byts:*** For detection of Web attacks, TCP initial window size is one of the significant features to focus on. When we look at the standard deviation of "Brute Force XSS", "Brute Force Web" and "SQL Injection", they are all similar and have an average value of 3479.87. However, benign traffic has larger standard deviation as 17256.30 which is an expected result of a randomized normal traffic. This indicates an automated tool usage while conducting attacks.
- ***Subflow Fwd Byts:*** One of the common specialties of Brute Force Web, Brute Force XSS and SQL Injection attacks is taking advantage of improper input validation on the victim side. To do this, several malicious statements are sent by the attacker to trigger the related vulnerability. Therefore, it is expected to have flow length in the forward direction larger than the benign traffic. The dataset statistics also shows the similar result, the mean length of subflow bytes in the forward direction is 400.06 for the benign traffic and 14754.35 for the web attack traffic.
- ***Init Bwd Win Byts:*** This feature is deterministic to specify in which direction the data is flowing. In our case, the average value of the initial TCP window size in backwards direction (from server to client) for web attacks is 655.78. The value for benign traffic is 10338.79. This decrease of the window size in the backwards direction indicates that the data is sent from server to client during the web attack. This is an expected result since the main aim of these attacks is to retrieve information from the victim.
- ***TotLen Fwd Pkts:*** When the dataset is examined, it is seen that the average value of total length of the packets in the forward direction belonging to the web attack traffic (14754.35) is greater than the benign traffic (400.06). Depending on the attack type, Brute Force XSS attacks have the highest value (26515.36) than the others (Brute Force Web is 12352.07 and SQL Injection is

533.19). Since generally a client-side script is injected to the victim website in this attack type, it is meaningful to have higher packet sizes than brute force web which is using pre-defined dictionaries of credentials or SQL injection using SQL statements.

Infiltration: This attack is executed by exploiting a vulnerable application. (In our case, Adobe Acrobat Reader 9 is exploited.) The attacker sends the victim an email containing a malicious file which is prepared to be opened by the vulnerable application. If the exploitation step succeeds, a backdoor is expected to be placed on the victim machine, making it a part of the botnet, communicating Command and Control (C2C) servers to be able to get commands from the attacker and also send sensitive information (network reconnaissance results, credentials, sensitive files etc.) to the attacker. As mentioned in (*CSE-CIC-IDS2018 on AWS*, n.d.), our dataset scenario includes victim downloading a malicious file from Dropbox. After that, the file is opened with a vulnerable program letting it to scan the network and collect sensitive information.

(The analysis is done by combining Wednesday 28-02-2018 and Thursday 01-03-2018 datasets which include only Infiltration and Benign traffic.)

- ***Subflow Fwd Byts:*** When we investigate the dataset, the standard deviation of the subflow bytes in the forward direction for Infiltration traffic (35334.91) is relatively smaller than the benign traffic (172535.72). This can indicate an automated job which is a part of the infiltration process such as malicious code running on the victim computer sending cached credentials on the browser to the attacker periodically.
- ***TotLen Fwd Pkts:*** The standard deviation of the total length of packets sent in the forward direction of infiltration traffic is smaller than the benign traffic. Similar to the subflow bytes, this can also be an indicator of an automated job running for infiltration.
- ***Flow Duration:*** It is investigated from the dataset that the mean, standard deviation, maximum and minimum values of the flow duration for infiltration and benign traffic is very similar to each other. However, the unique flow duration values of the benign traffic are approximately 4 times of the infiltration traffic. This indicates that more similar actions are handled in the infiltration traffic than normal traffic which is expected since foreplaned actions are applied during infiltrations compared to more random benign traffic.
- ***Active Mean:*** Similar to flow duration; the mean, standard deviation, maximum and minimum values of the active mean for infiltration and benign traffic are close to each other. When we look at the unique values, it is seen that the benign traffic is approximately 4 times of the infiltration traffic. Therefore, unique values are evaluated as more deterministic. Having the same reason with the flow duration, this situation is also expected for more automated nature of the infiltration traffic.

DDOS-LOIC-UDP & DDoS attacks-LOIC-HTTP: In order to conduct distributed denial or service attacks, Low Orbit Ion Cannon (LOIC) open-source tool is used. LOIC tool is mostly used for network stress testing and executing DoS and DDoS attacks. This tool works by flooding the victim services with illegitimate TCP, UDP

or HTTP packets and generating huge amount of network traffic. Like all DDoS attacks, the aim is to exhaust network or application resources of the victim. To have such an impact, this type of attacks is executed by using multiple computers.

DDOS-HOIC: High Orbit Ion Cannon (HOIC) is developed to replace LOIC. It is also an open-source tool commonly used for network stress testing and executing denial of service attacks. For creating this dataset with DDoS attacks, HOIC tool is used by 4 different attack computers. Below features are driven as the top features that affects the detection ratio.

(The analysis is done by combining Tuesday 20-02-2018 and Wednesday 21-02-2018 datasets which include only DDoS attacks and Benign traffic.)

- ***Bwd Pkt Len Std:*** When we examine the dataset, it is seen that the standard deviation of the packet length in the backward direction for the DDoS attacks take only 3 values which are 0, 467.5 and 482 while benign traffic takes several different values varying between 0 and 22448.41. This situation is compatible for DDoS attacks since during the attack, there is not any connection established with the victim, either the victim does not reply back or the attacker does not reply back to the victim after certain reply is received. Hence, it is expected to have minimal changes in the reply sizes of the flows.
- ***Pkt Size Avg:*** For the DDoS attacks in the dataset, the maximum value of the average packet size is depicted as 185.71. However, this value is 16801.14 for benign traffic. This is similar with the packet length in the backward direction since there is not any connection established with the victim, there is not any condition to increase the packet size.
- ***Flow Duration:*** This feature is one of the most deterministic features for DDoS attacks. In contrast with the features related to the packet sizes, the flow duration of the flood attacks is expected to be lower than the benign traffic. When we look at the average value of Flow Duration for DDoS attacks, it is seen that it is half of the benign traffic which is an expected result.
- ***Flow IAT Std:*** This is another feature that is used to detect DDoS attacks. The average value of the standard deviation time between flows for the benign traffic is approximately ten times of the DDoS attack traffic. (1148293.31/121302.69). This is also an expected result and compatible with the nature of the flood attacks. Especially for the distributed denial of service attacks, while using multiple machines sending continuous packets to the victim machine, this difference is greater.

Botnet: A robot network (botnet) is a remotely controllable network by attackers. It consists of several hijacked machines that are infected by bot malware which is controlled by the attacker. Through the botnet, attacker is able to send commands to each bot machine and also bot machines can send sensitive information to the attacker. In our case, for botnet connectivity Zeus is used as a Trojan horse malware package. Zeus is capable of man-in-the-browser keystroke logging and form grabbing to obtain sensitive information from the victim. Along with Zeus, Ares botnet is also used to infect the victim machines. Ares is a Python Remote Access Tool and made of two programs including a command-and-control server and an agent program. It is capable

of opening remote shell, file upload/downloading, taking screenshots, key logging and keeping its persistence on the victim machine.

(The analysis is done with Friday-02-03-2018 dataset which includes only Botnet and Benign traffic.)

- ***Subflow Fwd Byts:*** Botnet traffic includes the communication between the attacker and the infected victim. During this communication, attacker sends several commands requesting information from the victim. Hence, it is expected to have smaller subflow bytes from attacker to victim than the normal traffic. When we examine the dataset, it gives us similar results that the benign traffic has approximately three times average value of subflow forward bytes than the botnet traffic.
- ***TotLen Fwd Pkts:*** This value is the same as the subflow bytes in the forward direction. The average value of the botnet traffic has smaller total packet length than the benign traffic that gives us an expected result.
- ***Fwd Pkt Len Mean:*** When we investigate the dataset, the maximum value of the mean value of packet length in the forward direction for the botnet traffic is smaller than the benign traffic. This is proportional to the average value of the subflow bytes in the forward direction which results in for the similar reason.
- ***Bwd Pkts/s:*** Since automated botnet tools are used during the experiment, the commands from the attacker is simulated in specific time periods. Therefore, it is expected to have standard deviation of the pps (packets per second) from the victim to attacker is smaller than the benign traffic. When we examine the dataset, there is a significant difference between the standard deviation of the pps in the backwards direction for the botnet traffic and benign traffic. The benign traffic is approximately 219 times of the botnet traffic which is deterministic for detection.
-

3.2. Experiment Setup

The following were included in the experiment environment;

- Anaconda Individual Edition Python distribution platform with the latest stable version by the time the experiments were handled;
  - Conda 4.10.3
  - Python 3.9.7
- Tensorflow Platform for Machine Learning
- Keras as a deep learning API of Tensorflow
- Scikit-learn for preprocessing the dataset and evaluating the model

Anaconda Individual Edition Python distribution platform was installed. The latest stable version (according to the time this thesis was written) was Conda 4.10.3 with python 3.9.7.

Spyder was used as IDE which was pre-installed with Anaconda.

Tensorflow and Keras were used for creating and training the LSTM model.

System specifications of the experiment environment;

- ***Processors:*** Intel® Core™ i5-8400 @2.80 GHz 6 Cores
- ***RAM:*** 64 GB
- ***GPU:*** NVIDIA GeForce GTX 1060 3 GB
- ***Operating System:*** Windows 10

So as to make use of GPU while training, Nvidia Cuda and CuDNN were installed and configured. At the time of the experiments, Cuda Version was 11.2.1 and CuDNN was 8.1 which were compatible with tensorflow and tensorflow-gpu version 2.5.0.

3.3. Preprocessing

Since the aim of this thesis is to build a model that predicts benign traffic with 14 different attack types by outperforming traditional intrusion detection systems, multi-class classification approach was applied.
The methodology followed during the experiments is shown in . The Preprocessing was divided into three main Phases such as “Data Initialization”, “Data Preparation” and “Split and Normalize”. During this phase, the methodology shown in Figure 24 is used.

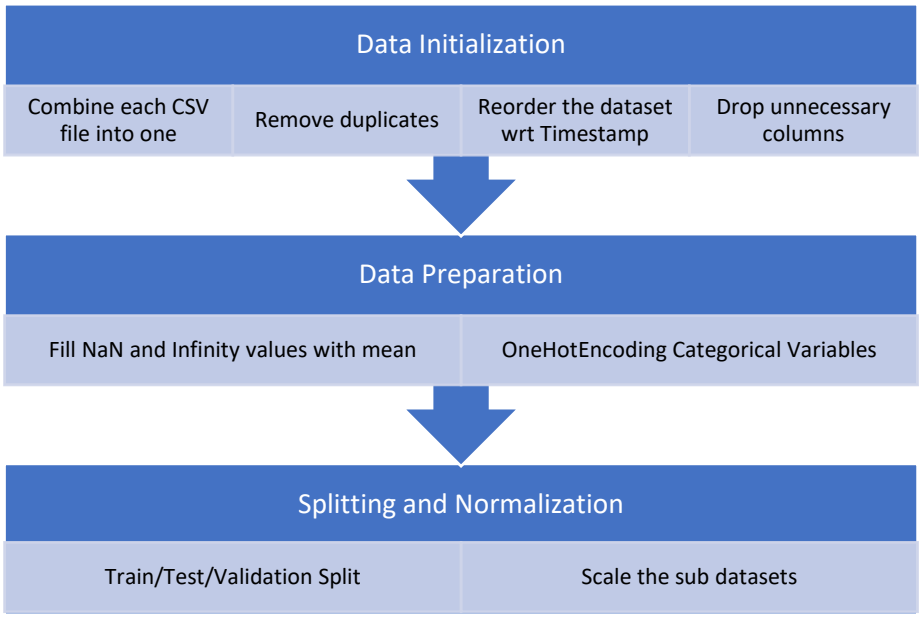

Figure 24: Data Preprocessing Methodology

The dataset consists of 10 separate CSV files belonging to different days, all separate CSV files were combined into one. Most of the related works were focused on specific attack types included in individual csv files. However, since the chosen deep learning algorithm is LSTM, the historical data is also considered while training. This is also

compatible with the nature of the real-world traffic. It makes more sense especially categorizing benign traffic of all days and detecting abnormal behavior by learning the normal behavior of the traffic and including as many samples as possible. Also, there are another attack types dependent on the historical traffic such as DoS/DDoS, brute force, bot and infiltration. Therefore, the model is trained by using the complete preprocessed dataset.

After concatenating the dataset into one csv file, the duplicate rows were removed. Then, the dataset is reordered with respect to the “Timestamp” feature. As mentioned before, LSTM model is highly dependent of the historical data. While training our model, we feed the model as batches and iterate over the batches for each epoch. In order to detect the attacker behavior in a more realistic way, we should analyze the traffic in a timely manner.

In order to make predictions independent from specific socket information; the following columns were dropped: "Src IP", "Src Port", "Dst IP". (“Dst Port” is kept since it is effective for detecting specific attacks such as FTP and SSH brute force, web attacks, certain types of application layer DoS attacks etc.)

In addition to the previous point, there were some columns detected as having not enough impact on the classification. For example, “Bwd PSH Flags” and “Bwd URG Flags” columns are detected as having the same value (=0) throughout the whole dataset. In addition, “Flow ID” is a unique value generated for each flow and it is not meaningful to use it as a feature for attack detection. “Timestamp” feature is used to reorder the dataset in the previous step. However, since we do not want our model to learn attack and benign traffic according to their timing, this column is also dropped before training. As a result, final feature size is dropped from 83 to 76.

After “Data Initialization Phase” is completed, the data is supposed to be prepared for the “Splitting and Normalizing Phase”. Infinity values were detected at Flow Byts/s and Flow Pkts/s columns. These values were replaced with Nan and then the dataset is separated as two NumPy arrays as “X” and “y” according to the dependent and independent variables. In our case, only “Label” column depicting the attack types is chosen as dependent. The resulting “X” shape is obtained as (15677150, 76) and “y” as (15677150,). Then, by using Simple Imputer from scikit-learn, Nan values of X were filled with the mean value of the related column. Simple Imputer basically transforms the dataset by completing the missing values according to the given strategy. The strategy can be “mean, “median”, “most frequent” or “constant”. We used the default strategy “mean” for our case.

When we analyze the dataset, it is seen that all independent variables were in the numerical form. However, dependent variables were in categorical and string form. Since the ML model expects numerical values, before the training Phase, all values should be converted to numerical form. Therefore, “Label” column, y, is transformed into numeric array by using Categorical Encoder.

For binary classification (see the model structure in 3.4.1), in y column, all benign traffic is converted into “0.0” and all attack traffic is converted into “1.0”.

For multi-class classification (see the model structure in 0), OneHotEncoding approach is decided to be applied. OneHotEncoder encodes categorical features as a

numeric array. In this approach, data is fit into one-hot numeric array consisting of 0's and 1's unlike the other popular method, LabelEncoder. Machine learning models, including LSTM, tend to give more weight and importance to higher values while training. This behavior misleads the model to make wrong estimations while capturing the relationships. This situation gains more importance in our case since while making multi-class classification among different attack types, all categories have equal importance in terms of intrusion detection.

As a result of OneHotEncoding, new shape of y is obtained as (15677150,15) having the following unique rows for each Label;

[[0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1.],

[0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0.],

[0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0.],

[0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0.],

[0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0.],

[0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0.],

[0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0.],

[0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0.],

[0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0.],

[0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0.],

[0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0.],

[0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],

[0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],

[0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],

[1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.]]

In the "Splitting and Normalization Phase", as a starter, scikit-learn train-test-split is used. Test size is determined as 20% of the whole dataset. The rest 80% is again split into train and validation with the same ratio. The resultant sub dataset ratios are obtained as:

- Train: 64 %
- Test: 20 %
- Validation: 16 %

Table 6 shows the class distribution among train, test and validation datasets for the multi-class classification.

Table 6: Train, Test, Validation Dataset Distribution – Binary Classification

Classes	**train**	**test**	**val**
Benign	8584473	2682916	2146601
Attack	1448903	452514	361743

Table 7 shows the class distribution among train, test and validation datasets for the multi-class classification.

Table 7: Train, Test, Validation Dataset Distribution – Multi-class Classification

Classes	**train**	**test**	**val**
Benign	8584473	2682916	2146601
Bot	178520	55866	44198
Brute Force -Web	398	121	92
Brute Force -XSS	148	39	43
DDOS attack-HOIC	416771	130521	104454
DDoS attacks-LOIC-UDP	1107	345	278
DDoS attacks-LOIC-HTTP	368509	115302	92348
DoS attacks-GoldenEye	26565	8211	6628
DoS attacks-Hulk	264264	82378	65519
DoS attacks-SlowHTTPTest	3881	1202	967
DoS attacks-Slowloris	6422	1976	1612
FTP-BruteForce	12589	3979	3172
Infiltration	103656	32189	26021
SQL Injection	59	18	10
SSH-Bruteforce	66014	20367	16401

After splitting is completed, the sub datasets are scaled between (0,1) for normalization. In order to achieve this, scikit-learn's "MinMaxScaler" are used. The reason of this selection is that this scaler preserves the shape of the original distribution without causing any distortion to the original data. MinMaxScaler is simply responsible for scaling and translating each feature according to the given range ((0,1) as default) by using the translation formulas (27) and (28).

$$X_{std} = \frac{X - X.\min(axis = 0)}{X.\max(axis = 0) - X.min(axis = 0)} \quad (27)$$

$$X_{scaled} = X_{std} * (max - min) + min \quad (28)$$

*min, max = feature_range

Finally, our data has become ready to be fit into our ML model for Training Phase.

3.4. Model

While creating the model, Tensorflow Platform with Keras API is used.

Figure 27 illustrates the model created for multi-class classification. The model consists of four hidden LSTM layers. After several experiments with different sizes of layers, four is chosen as the most optimum size.

The unit size for all layers is chosen as 76 which is equal to the total count of features in our dataset.

There are also "kernel" and "recurrent_kernel" sizes shown on each LSTM cell. Both parameters correspond to the weights that are used in LSTM cells. As mentioned in 2.4.3.2, there are four different types of weights used to transform the input state for each LSTM gate.

- W_i is used for the input gate,
- W_C is used for the candidate gate,
- W_f is used for the forget gate,
- W_O is used for the output gate.

While each value is being sent in a neuron, its corresponding four weights should be also included for each neuron. This makes the kernel size of each LSTM cell equal to $1 * (4 * \text{units}) = 1 * (4 * 76) = 1 * 304$.

In addition, to transform the hidden state, LSTM uses four more weights.

When we extend the formula explained in 2.4.3.2, the following formulas (29),(30),(31),(32) & (33) are driven from (Graves, 2013) ;

$$i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci}c_{t-1} + b_i) \quad (29)$$

$$f_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf}c_{t-1} + b_f) \quad (30)$$

$$c_t = f_t \odot c_{t-1} + i_t \odot tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c) \quad (31)$$

$$o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + W_{co}c_t + b_o) \quad (32)$$

$$h_t = o_t \odot \tanh(c_t) \quad (33)$$

Here;

- σ is the sigmoid function
- i_t is the input gate activation vector
- f_t is the forget gate activation vector
- c_t is the cell/candidate input activation vector
- o_t is the output gate activation vector
- h_t is the hidden state
- $\odot$ operator is the Hadamard product.

W_{xi} , W_{ci}, W_{xf}, W_{cf}, W_{xc}, W_{xo}, W_{co} are the weights used to transform the input state for the input, forget, candidate and output gates.

There are also additional weights used to transform the hidden state of the four LSTM gates such as; W_{hi} as the hidden-input gate weight, W_{hf} as the hidden-forget gate weight, W_{cf} as the candidate-forget gate weight, W_{ho} as the hidden-output gate weight.

Considering these additional four weights for hidden state, recurrent kernel size for each LSTM cell is equal to $\text{units} * (4 * \text{units}) = 76 * (4 * 76) = 76 * 304$

Similarly, there are four biases to be considered for each neuron, therefore it gives us $4 * \text{units} = 4 * 76 = 304$ per LSTM cell.

While training with Deep Neural Networks, in order to reduce the ratio of possible overfitting, adopting a regularization method helps to increase the performance of the model. For this reason, "Dropout" method is decided to be used.

As mentioned in (Srivastava et al., 2014), there occurs two main problems correspond to two different methods while regularizing a model. First method is to calculate the average of the predictions applying different settings of the parameters and then determine the weights of each setting consistent with the posterior probability. However, when working with huge datasets and relatively complex models with limited computation this approach is expected to be not suitable.

Second method is the model combination. In this method, taking the average of the output of many separately trained models is proposed. This requires many different model architectures, training on different data, parameter tuning for each model etc. Therefore, a similar drawback occurs for demanding a lot of computation for each of the tasks.

As a solution for both of the problems, "Dropout" method is proposed. In this method, hidden and visible units are dropped out randomly and temporarily with their connections from the network. Hence, while propagating forward, the randomly dropped neurons' effect on the activation of downstream neurons is ignored. Similarly, during backward propagation, there is no update on weights of these neurons. Since this step repeats randomly, it helps the neural network become less sensitive to the specific weights of neurons so that the possibility of overfitting decreases and model is more capable of generalization.

In our model, we add Dropout Layer between hidden layers. The Dropout rate is set as 20% so that 1 out of every 5 inputs is randomly dropped out for each iteration. The most optimum results obtained with this Dropout rate.

In all four of the LSTM hidden layers, default tanh activation function is used. As explained detailed in 2.4.3.2, this function applies for the candidate hidden state and output hidden state. This is used to determine the how the transformation of the weighted sum of the input is handled. As can be seen from Figure 25, tanh function takes values between (-1,1). Moreover, since its derivative takes values between (0,1), it is an optimal choice for vanishing gradient problem (2.4.3.2). It also helps gradient computation less intensive and converges faster.

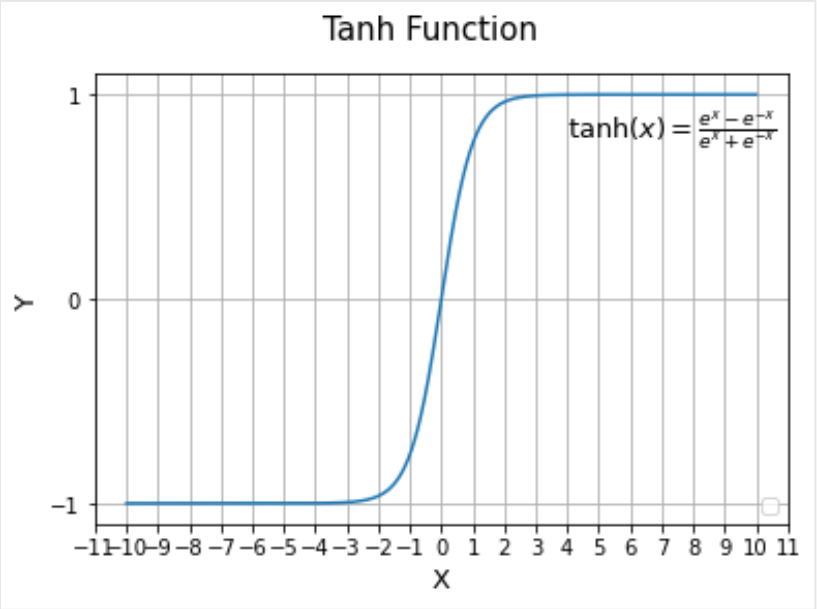

Figure 25: Tanh Activation Function

3.4.1. Binary Classification

Since we have two different outputs, the Dense output layer units are selected as 1. As an activation function in the output Dense layer, “Sigmoid” is used.
As shown in Figure 26, Sigmoid is a logistic function that takes output values varying between (0,1). The output of the function becomes saturated when the input x value gets too negative or too positive. In that case, if there is a small change in input values, the function becomes so saturated that it is insensitive to detect those changes. (Goodfellow et al., 2016) This behavior is rated as an advantage for classification problems.

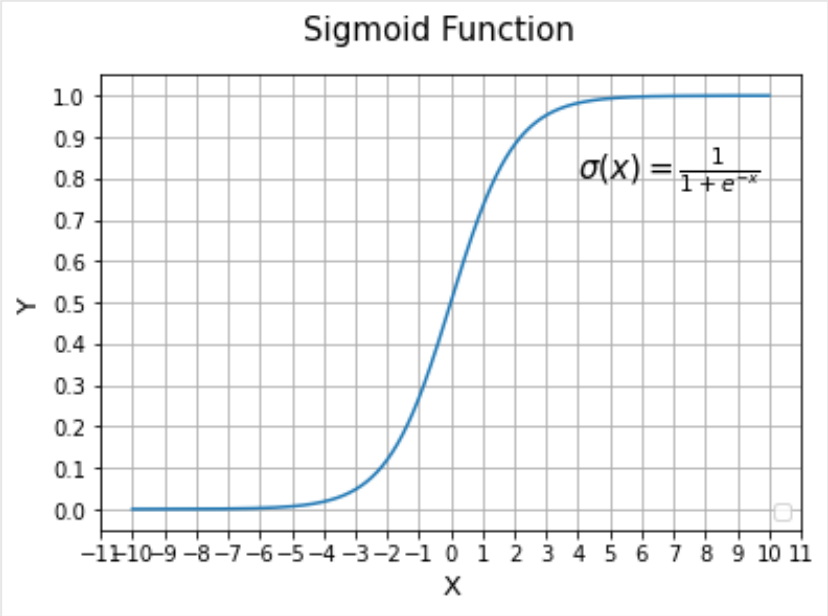

Figure 26: Sigmoid Activation Function

As optimizer, Adam is decided to be used with Learning Rate 0.0002. As introduced in (Kingma & Ba, 2015), Adam provides a new approach for classical stochastic gradient descent. It aims to optimize the stochastic objective functions by first order gradients. In this way, the model becomes less memory consumptive. Unlike classical stochastic gradient descent, for different network weights individual adaptive learning rates are computed. They are separately adapted during training.

Learning rate simply determines the update ratio of the weights of the network. Therefore, it is a very critical parameter for the performance of the model. As depicted in (Kingma & Ba, 2015), if learning rate is chosen too large, the update ratio of the weights will also increase with the gradient descent leading higher training loss. This may affect the robustness of the network and create oscillation of the metrics during training. Likewise, when the learning rate is chosen too small, the weights will be updated so slowly that may not be sufficient for the network to learn or it may get stuck with a high training loss.

In the light of the above drawbacks, different learning rates combined with different batch sizes are applied to our model. The resultant optimum learning rate is determined as 0.0002 with batch size 1024.

Finally, the loss function is chosen as “binary crossentropy” which is one the most optimal functions for binary classification. It gives outputs as probabilities in range (0,1). It has the same formula (38) with the categorical cross entropy.

3.4.2. Multi-class Classification

The output activation function is different than others since it strongly depends on the type of the prediction. In our case, multi-class classification is needed for 15 different classes. Hence, "Softmax" is chosen for the output activation function. Softmax function gives outputs as probabilities so that the sum of the outcomes is equal to 1. When applied in a multi-class model, it gives outputs between (0,1), having the target class with the highest probability.

As mentioned in Chapter 6 "Deep Feedforward Networks" of (Goodfellow et al., 2016),

Softmax can be calculated as;

$$softmax(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \text{ for i} = 1, \dots, \text{K} \tag{34}$$

Considering z as the input of K real numbers, when predicting the probability of the first variable of the input array having length of 3 the equation is expected to be as in the following;

$$softmax(z)_1 = \frac{e^{z_1}}{\sum_{j=1}^{K} e^{z_j}} = \frac{e^{z_1}}{e^{z_1} + e^{z_2} + e^{z_3}} \tag{35}$$

As optimizer, Adam is decided to be used with Learning Rate 0.0002. Batch size is also selected as 1024 which is the same as binary classification.

Last but not least, the loss function for the multi-class classification is chosen as "categorical cross entropy" which is a type of cross-entropy which is derived from Information Theory. Since SoftMax activation function is used for multinomial probability distribution, cross entropy is mostly suggested as loss function for the error calculation. This loss function is created to measure the difference between different probability distributions.

As described in (Murphy, 2022), the cross entropy aims to calculate the average number of bits required to transmit data from a distribution to another or create a representation for the specific data.

Categorical cross entropy between two probability distributions (p, q) are given in the driven from (Murphy, 2022);

$$H(p, q) = -\sum_{n=1} p_k log q_k \tag{36}$$

The categorical loss function is referred as negative log likelihood in this paper. The formula (37) stands for the loss function;

$$NLL(w) = -\sum_{i=1}^{N} [y_i log \mu_i + (1 - y_i) \log(1 - \mu_i)] \quad (37)$$

Here; y_i stands for the target value, μ_i is the offset term in the output of the model, w is the weight vector, N is the output size that corresponds to number of scalar values in the model output.

If we adapt the formula (37) to our solution, we can obtain formula (38);

$$Loss = -\sum_{i=1}^{N} [y_i \, log \hat{y}_i] \quad (38)$$

Here $\hat{y}_i$ is the predicted label of the i^{th} sample and y_i is the corresponding actual label in our dataset.

Finally, we use one output Dense Layer with unit size 15 which is equal to the class count that is to be predicted.

Figure 27 and Figure 28 show the final model for each classification.

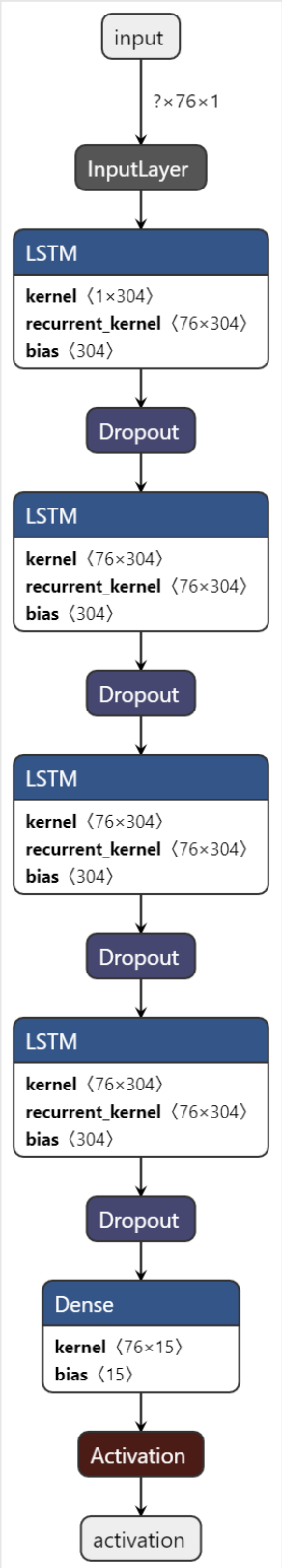

Figure 27: Multi-class Classification LSTM Model

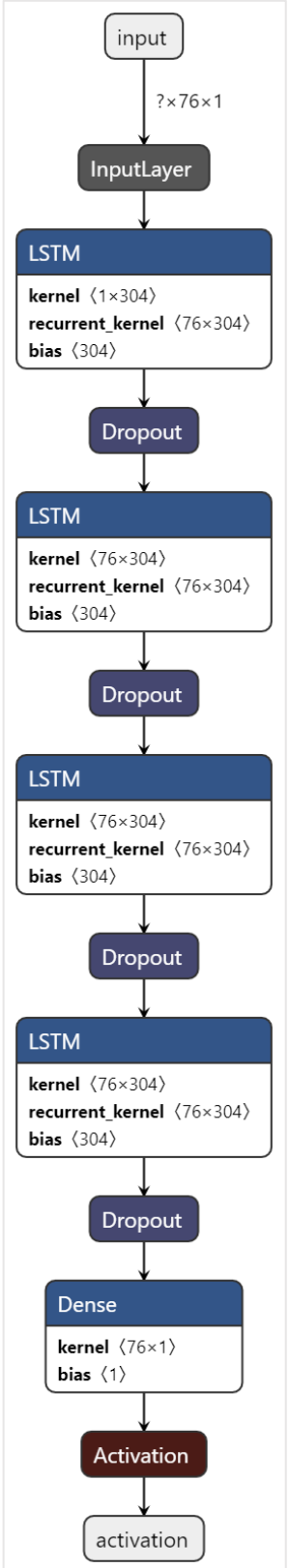

Figure 28: Binary Classification LSTM Model

3.5. Training

After creating the model, in the training phase, the model is fit with the train and validation datasets. Several batch sizes are included in the experiments. As a result, most optimum batch size is found as 1024 with learning rate 0.0002. In addition, LSTM is used with the default settings which makes it stateless. Stateless LSTM means that after each batch is fit into the model, all states of the gates are reset to their initial state. Although it is beneficial in terms of the resource consumption, it has come to conclusion that while doing time series prediction, selecting relatively smaller batch sizes may cause the model to fail to learn hierarchical feature representations and sequential relationships adequately. Therefore, 1024 batch size is seen to show better performance compared to smaller sizes. (64,128,256 and 512)

The other important parameter is the epoch count selection during training. During training, it is observed that the model usually reaches its peak training and validation accuracy until the 40^{th} epoch. However, this ratio may change with different experiments and scenarios. At this point, callbacks from Tensorflow Keras are decided to be used. This feature helps to set the strategy and patience while training and store and restore the best weights obtained from the epoch having the minimum loss. The obtained results are given more detailed in Chapter 4.

Moreover, with the synchronization of Wandb weights and biases platform, several graphs, resource consumption, logging weights and biases through epochs is able to be implemented.

(*Wandb Main Page*, n.d.)

CHAPTER 4

EVALUATION AND DISCUSSION

4.1. Performance Evaluation Metrics

The performance evaluation metrics for our model is directly dependent on the dataset characteristics and our performance criteria.

First of all, when we look at the dataset distribution as shown in terms of percentages in Figure 29 , we conclude that there is a significant class imbalance issue within our dataset.

$$Imbalance\ Ratio = \frac{Majority\ Class\ Instances}{Minority\ Class\ Instances} \quad (39)$$

The imbalance ratio of the majority vs rest (Benign vs Attack in our case) is calculated as;

$$Imbalance\ Ratio = \frac{13413990}{2263160} = 5.927$$

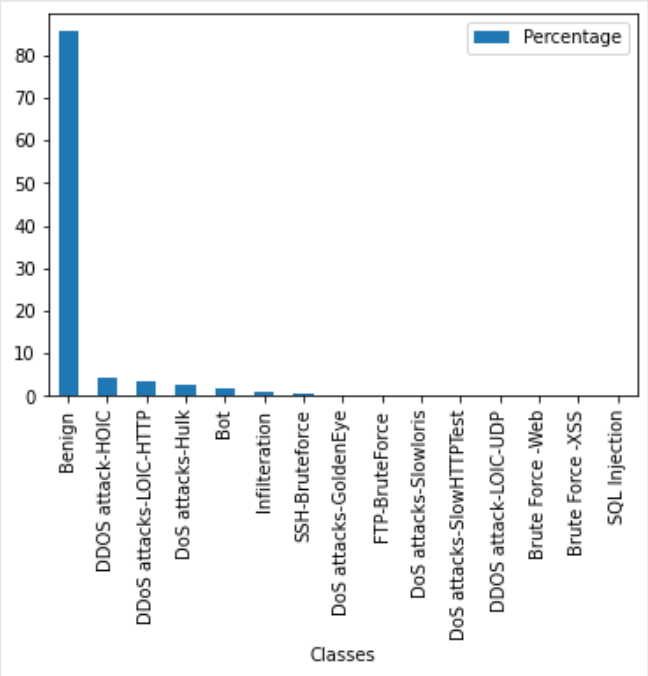

Figure 29: Dataset Class Distribution Percentage

Considering above findings, flowchart from the (Jason Brownlee, 2020)is adapted as a reference point which can be seen in Figure 30.

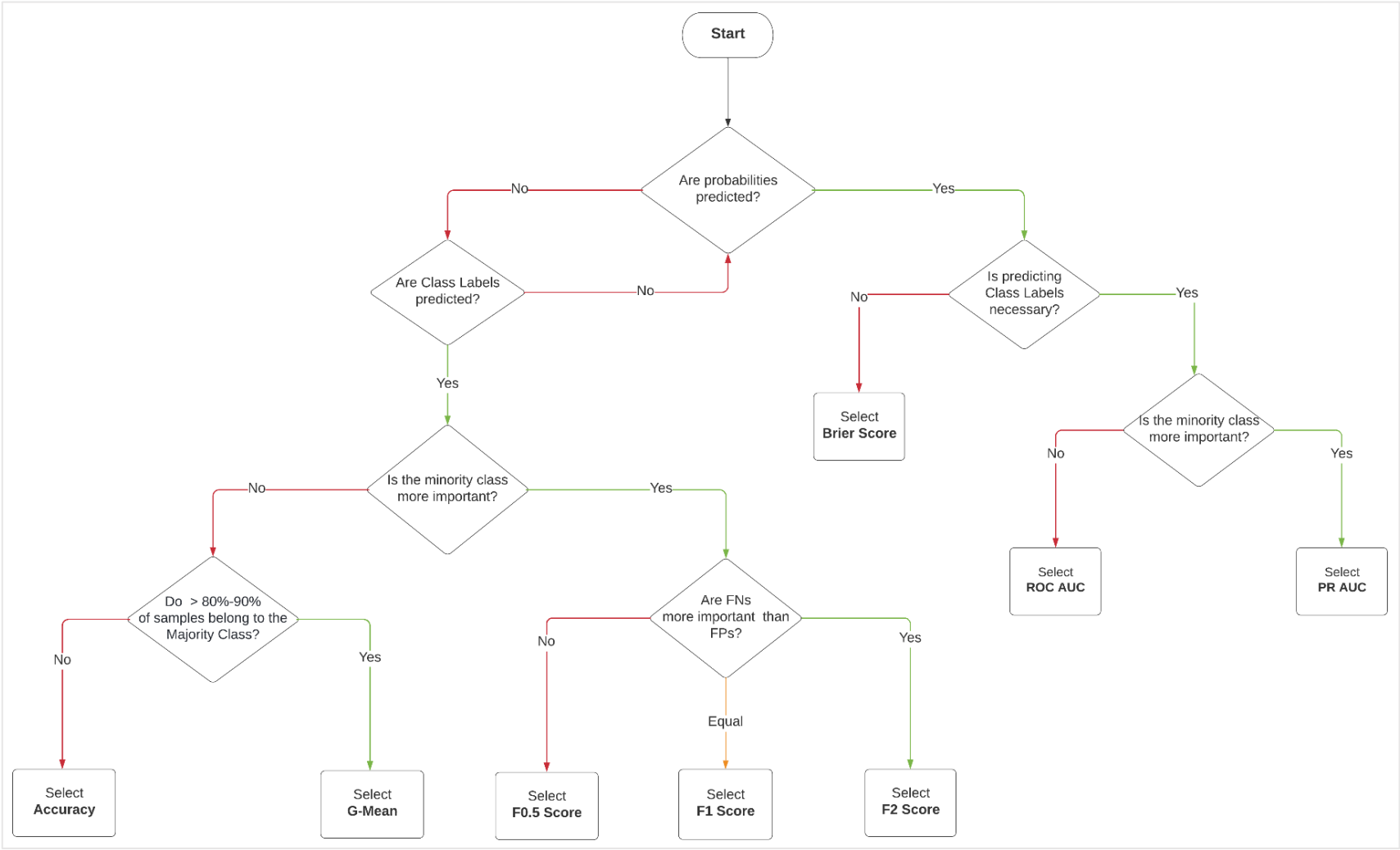

Figure 30: Performance Metrics Selection for Imbalanced Datasets

As explained in Chapter 3.4, since we are using Softmax as an activation in the output layer, when we predict the model, we obtain the probabilities. In terms of intrusion detection, we need to predict the Class Labels from the probabilities. In addition, whilst eliminating the False Positives is also essential for the model performance, the sensitivity ratio and False Negatives are evaluated as more vital for detection. Therefore, PR AUC is selected as the most appropriate metric for our case.

Besides PR AUC, other metrics explained in the Flowchart are also included. Since we are predicting Class Labels having the majority class weight with 83.07 % (Chapter 3.1), G-Mean with weighted Accuracy and F1 Score for weighting false positives and false negatives equally are also selected as a performance metric. Therefore, we both calculate the predictions and with 0.5 threshold value, we predict class labels consisting of one hot arrays with 0 and 1.

Below is an example of the prediction showing the most probability for the Benign Class;

[9.9667859e-01, 1.9083198e-10, 2.8668619e-08, 8.9975778e-09, 8.4159159e-11, 6.9023204e-11, 5.9489613e-10, 1.2886682e-09, 5.6413567e-12, 2.3829304e-12, 7.2299183e-10, 5.0482535e-12, 3.3214393e-03, 9.9054132e-10, 6.5570696e-12]

Same prediction converted into one-hot array;

[1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.]

To have a closer look into the formulas of these performance metrics;

(Note that the metrics formulas are given as a function of True Positives (TP), True Negatives (TN), False Positives (FP), False Negatives (FN), True Positive Rate (TPR) and False Positive Rate (FPR))

Accuracy

The accuracy function is given as in formula (40). It is beneficial to evaluate the correct prediction rate of the classifier. The result varies between (0.0, 1.0). Having the accuracy score closer to 1.0 is accepted as ideal for a good classifier.

$$Accuracy = \frac{\mathrm{TP} + \mathrm{TN}}{\mathrm{TP} + \mathrm{TN} + \mathrm{FP} + \mathrm{FN}} \tag{40}$$

Although this metric is widely used for evaluating most classification models, this gives very misleading results when it comes to imbalanced classification. The reason is, if we have a majority class that overweighs the others, just learning and predicting this class correctly is enough to obtain high accuracy results. Therefore, it will not give any sufficient result for the performance of predicting minority classes.

Balanced Accuracy

Balanced accuracy is suggested to be used for binary and multi-class classifications while dealing with imbalanced datasets.

The result varies between (0.0, 1.0). Having the balanced accuracy score closer to 1.0 is accepted as ideal for a good classifier.

The balanced accuracy gives the arithmetic mean of sensitivity (TPR/Recall) and specificity (TNR) for binary classification.

$$Balanced\ Accuracy = \frac{1}{2}\left(\frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FN}} + \frac{\mathrm{TN}}{\mathrm{TN}+\mathrm{FP}}\right) \tag{41}$$

For multi-class evaluation, it is suggested to compute balanced accuracy for each class and take the average of the results of all classes. (Urbanowicz & Moore, 2015)

Precision

Precision summarizes the fraction of examples assigned the positive class that belong to the positive class. The result varies between (0.0, 1.0).

The formula is given as;

$$Precision = \frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FP}} \tag{42}$$

Since having minimum FPs are ideal for a good classifier, having precision value closer to 1.0 gives the most promising results.

Recall

Recall summarizes how well the positive class is predicted and is the same calculation as sensitivity. The result varies between (0.0, 1.0).

$$Recall = \frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FN}} \tag{43}$$

Since having minimum FNs are ideal for a good classifier, having recall value closer to 1.0 gives the most promising results.

Geometric Mean Score (G-mean)

The geometric mean is calculated by taking the root of the multiplication all the sensitivity values for each class. The root degree is equal to the number of classes. The result varies between (0.0, 1.0). It is more acceptable to have score close to 1.0.

For multi-class classifications, G-mean can be not only calculated overall but also can be calculated for each class separately as if it is a binary classification problem with one-vs-rest strategy.

The formula is given in (44) where n is the number of classes.

$$\mathrm{G-mean} = \left(\prod_{i=1}^{n} \mathrm{Sensitivity}_i \right)^{-1/n} \tag{44}$$

F-Score (F-Measure)

Precision and recall can be combined into a single score that seeks to balance both concerns, called the F-score or the F-measure.

$$F - Score = \frac{2 * \mathrm{Precision} * \mathrm{Recall}}{\mathrm{Precision} + \mathrm{Recall}} \tag{45}$$

The F-Measure is a popular metric for imbalanced classification.

ROC Curve and AUC

ROC Curve is basically a plot of FPR vs. TPR that is used to evaluate the performance of the binary classification model according to the minority class.

TPR is also called “Sensitivity” or “Recall” having the same formula;

$$TPR = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}} \tag{46}$$

The formula of FPR is given in Formula (47);

$$FPR = \frac{\mathrm{FP}}{\mathrm{FP} + \mathrm{TN}} \tag{47}$$

While plotting these two functions, x axis becomes FPR and y axis becomes TPR. Since having minimum FNs and FPs whilst having maximum TPs and TNs is ideal for a model, the resulting curve is expected to be closer to y axis.

AUC score is the are under ROC curve. ROC-AUC value is expected to be between (0.0,1.0). Having ROCAUC value closer to 1.0 is accepted as ideal for a good classifier.

Precision Recall Curve and AUC (PR-AUC)

Precision Recall curve is the plot of Recall vs Precision on x and y axis. The ideal model is expected to have the plot close to (1,1) coordinate.

AUC score is the area under PR ROC curve. It gives a summarized result ranging between (0.0,1.0). The values closer to 1.0 is accepted as ideal for a good classifier.

4.2. Results

In this section, the results obtained by using LSTM for binary and multi-class classifications are shown in detail. Depending on the type of the classification, different models with different unit size, activation and loss functions are used. Moreover, in order to increase the performance of the multi-class classification, several methods to overcome the imbalance problem of the dataset are applied. These methods are SMOTE oversampling and LSTM training with custom class weights. Their effect on the results is also discussed in this section.

For the reproducibility of this study, the source code used during these experiments can be reached via the following GitHub link;

https://github.com/ebrukultur/ms-thesis

4.2.1. The Development of the Model

For both binary and multi-class classification, LSTM model was built by using different hyperparameters until reaching the most optimum results.

For this purpose, different values for layers, learning rate, loss function, activation function, dropout and batch size were experimented. *Table 8* shows the experimented hyperparameters and the related results obtained.

Table 8: Hyperparameters of the Previous Experiments

Classification	Output Layers	Hidden Layers	Learning Rate	Epoch Count	Loss Function	Batch Size	Dropout	Output Activation Function	Optimizer	Accuracy	Precision	F1	Recall
Binary	2	2	0.00005	10	mse	64	0.2	relu	adam	0.949	0.924	0.934	0.949
Binary	1	2	0.00005	10	mse	64	0.2	relu	adam	0.979	0.995	0.958	0.924
Binary	1	4	0.001	22	binary crossentropy	128	0.2	sigmoid	adam	0.979	0.988	0.947	0.911
Multiclass	1	2	0.00005	10	mse	65	0.2	relu	adam	0.894	0.807	0.848	0.894
Multiclass	1	4	0.00005	10	mse	512	0.2	relu	adam	0.977	0.974	0.972	0.977
Multiclass	1	4	0.00005	10	mse	128	0.2	relu	adam	0.967	0.942	0.954	0.967
Multiclass	1	4	0.00005	10	mse	128	0.2	softmax	adam	0.975	0.966	0.965	0.975
Multiclass	1	4	0.0002	40	categorical crossentropy	1024	0.1	softmax	adam	0.983	0.978	0.978	0.983
Multiclass	1	4	0.0002	40	categorical crossentropy	1024	0.5	softmax	adam	0.978	0.968	0.973	0.978
Multiclass	1	4	0.0001	24	categorical crossentropy	1024	0.2	softmax	adam	0.983	0.973	0.978	0.983
Multiclass	2	4	0.00002	33	categorical crossentropy	1024	0.2	softmax	adam	0.982	0.973	0.977	0.982
Multiclass	2	4	0.00002	40	categorical crossentropy	256	0.2	softmax	adam	0.969	0.965	0.969	0.969
Multiclass	1	2	0.0002	12	categorical crossentropy	1024	0.2	softmax	adam	0.979	0.969	0.974	0.979

For binary and multi-class classification, we tried different values for output layers, however no such difference occurred in the model performance when we increased the total count of the output layers. Also, the model showed higher performance when we decreased the output layers which indicates a mismatching model complexity with the dataset. Therefore, “1” output layer was chosen.

Moreover, when we increased the hidden layers from “2” to “4”, it was seen that the model performance has increased proportionally. Therefore, “4” hidden layers were chosen as the best fit.

We tried mse and cross-entropy loss functions for both binary and multi-class. Binary cross-entropy and categorical cross-entropy were chosen as the best fit according to the obtained results.

In order to make our model more capable of generalization, we used dropout between each hidden layer. We have chosen values lower than “0.5” since higher rates is evaluated to cause our network to under-learn. For this purpose, we tried different values for dropout such as “0.1”, “0.2” and “0.5”. Among these values, “0.2” was chosen as the best fit.

As output activation function, relu, sigmoid and softmax were experimented. Since our problem is classification rather than regression, we preferred sigmoid and softmax since they are capable of predicting probabilities of the outputs. Also, we obtained higher results with these functions.

We tried different learning rate and batch size values. Considering the prediction time and using stateless LSTM, keeping batch size larger and adjusting learning rate according to the batch size, “0.0002” learning rate with “1024” batch size showed the best performance. With these values, the prediction has lasted for approximately 934 seconds.

4.2.2. Binary Classification

Similar to many other works conducted by using machine learning techniques for the same dataset, as a starter, binary classification is tested for this dataset. As the name implies, binary classification is used for predicting two outcomes. In our case, it is expected to predict whether the flow belongs to “Attack” or “Benign” class. The preprocessing phase and model creation is performed as in Sections 3.3 and 3.4.1.

Binary training is performed with 10 epochs set which is evaluated as the most optimum count for the training loss and accuracy.

Figure 31 and Figure 32 show the training accuracy and training loss throughout epochs.

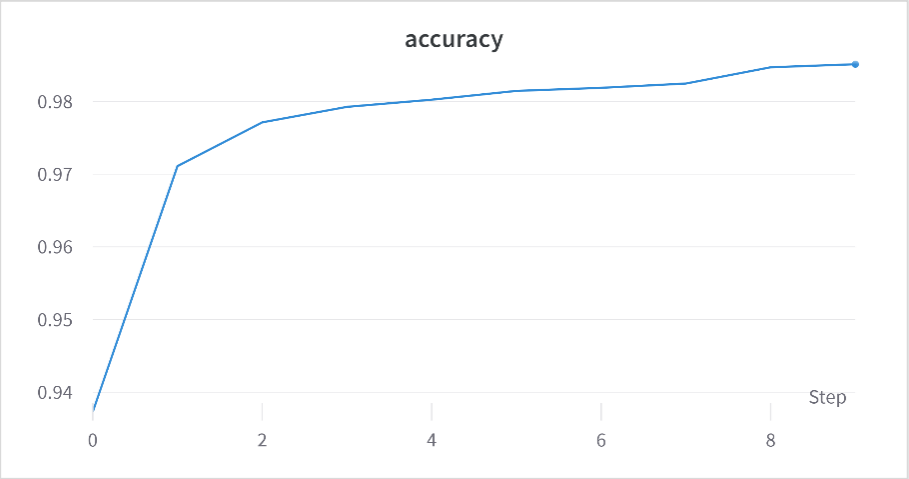

Figure 31: Training Accuracy - Binary Classification

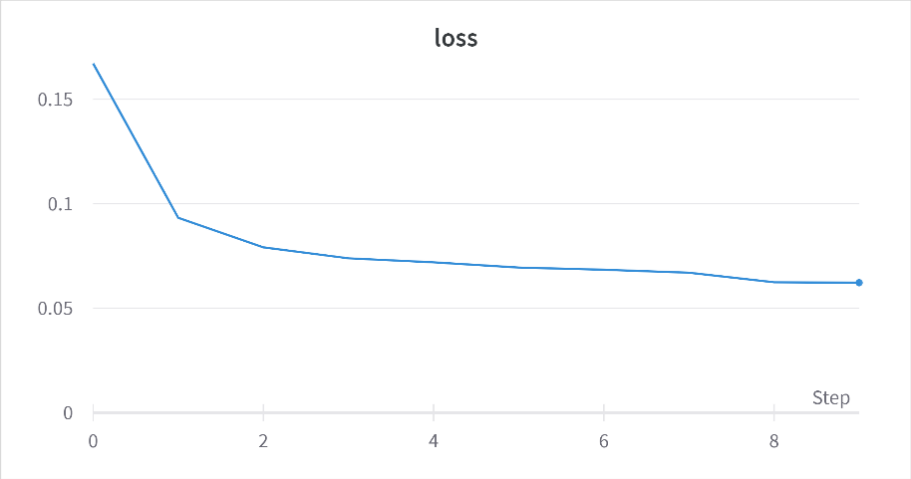

Figure 32: Training Loss - Binary Classification

Overall performance results are shown in Table 9 below;

Table 9: Overall Performance Results – Binary Classification

Metric	**Result**
Accuracy	0.985
Recall	0.985
Precision	0.985
F1 Score	0.985
G-Mean	0.958
Balanced Accuracy	0.959

The confusion matrix is given in Figure 33. Since Benign traffic dominates the dataset, the TN ratio is the largest among the other metrics. Having only 0.40% of FP ratio is very promising because of the fact that traditional signature-based IDS having high FP rates. As depicted in (S. A. R. Shah & Issac, 2018), under 10 Gbps throughput, the average FPR value for both Snort and Suricata is calculated above 55%. In our case FPR is 0.4 % which is relatively too small to compare.

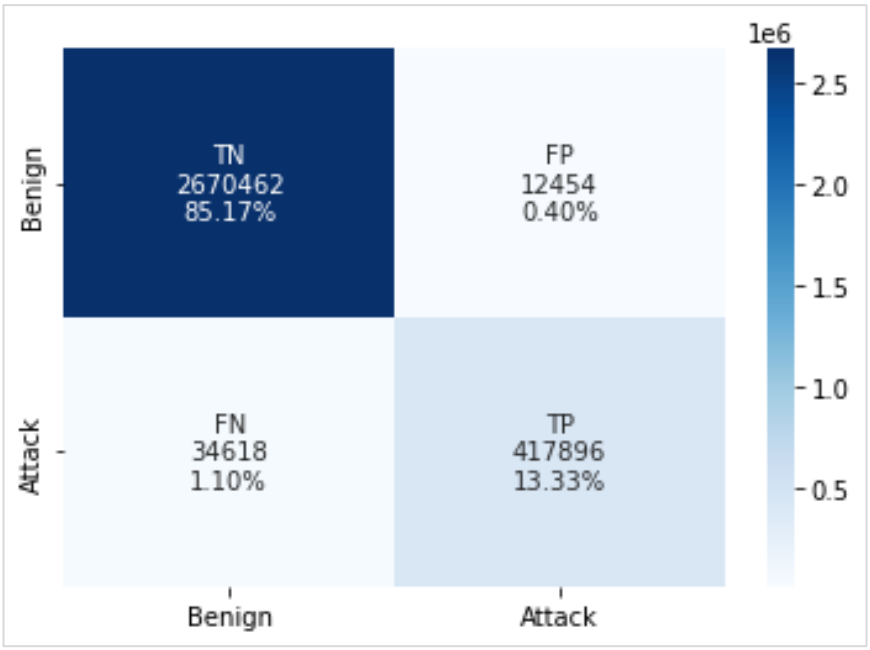

Figure 33: Confusion Matrix - Binary Classification

The classification report is given in Figure 34. Here, “macro avg” value denotes the average of the mean value of precision, recall and F1-Score for each label without giving any weight per label. “Weighted avg” value shows the average of the support weighted precision, recall and F1 score for each label.

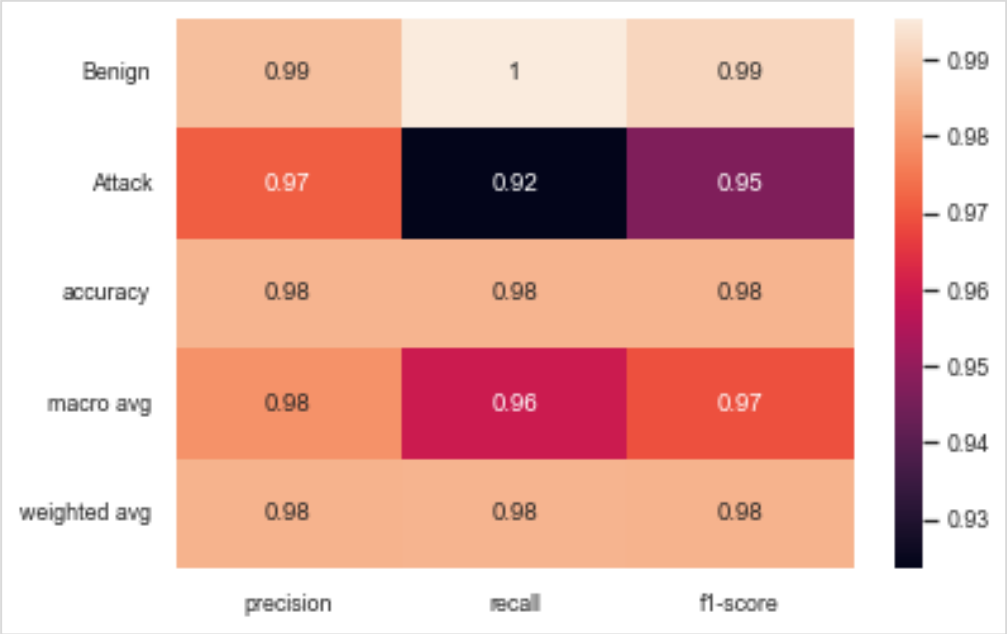

Figure 34: Classification Report - Binary Classification

The overall accuracy is higher than the similar conducted experiments. For example, according to (Ferrag et al., 2020), by using LSTM with different parameters for binary classification of the same dataset *CSE-CIC-IDS2018,* 97.310% is the highest accuracy they obtained.

When we examine the ROC Curve and AUC values shown in Figure 35, FPR vs TPR ratio for both benign and attack traffics are above the threshold line (FPR=TPR). It is an expected result for a good classifier.

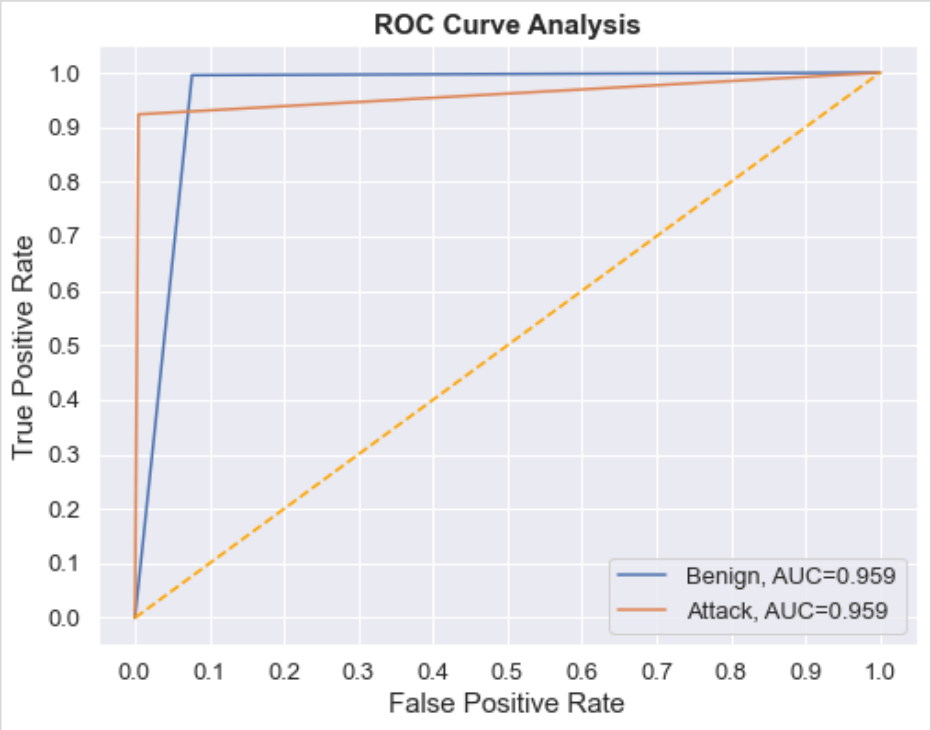

Figure 35: ROC AUC Curve – Binary Classification

When we investigate the PR Curve and the AUC values shown in Figure 36, it is seen that both precision and recall curve of benign and attack traffic is above the threshold line (unskilled classifier with precision=0.5). It is also an expected result for a good classifier.

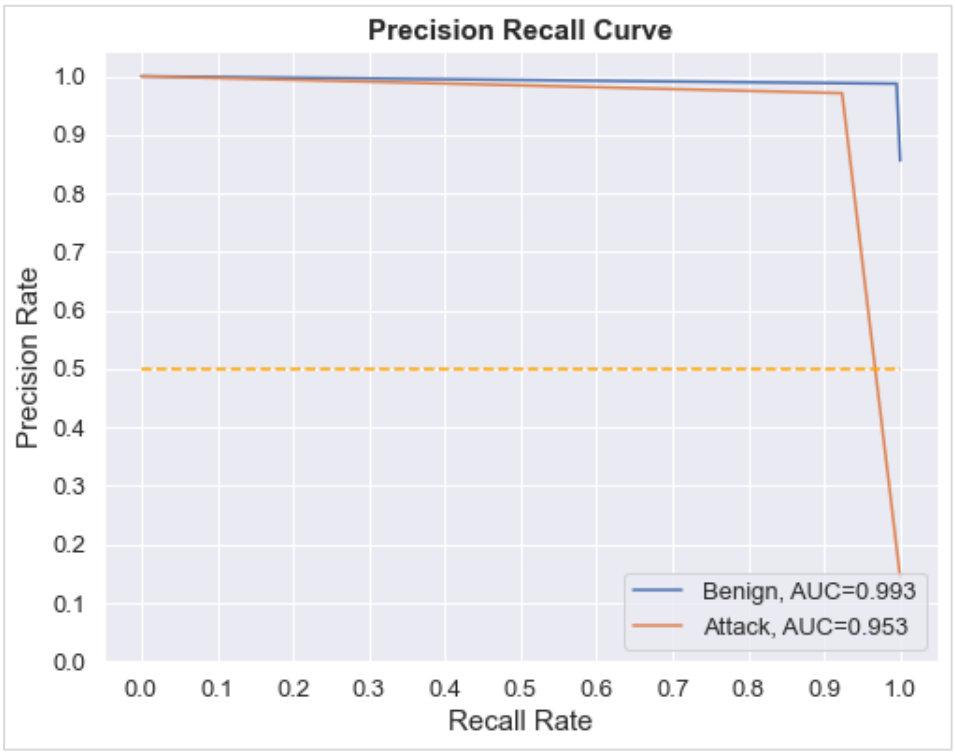

Figure 36: Precision Recall Curve – Binary Classification

4.2.3. Multi-class Classification

With the aim of to make the prediction based on the attack types, multi-class classification with LSTM is applied.

Multi-class training is performed with 40 epochs set and 5 epoch training loss value tolerated callback. The training has stopped at the 30th epoch where the training loss has started to increase.

Figure 37 and Figure 38 show the training accuracy and training loss throughout epochs.

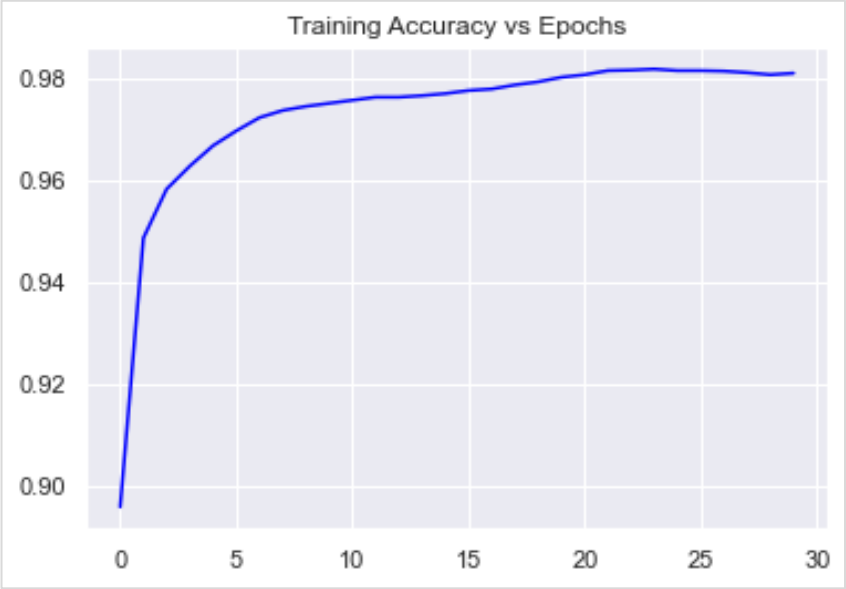

Figure 37: Training Accuracy - Multi-class Classification

Figure 38: Training Loss - Binary Classification

Overall performance results are shown in Table 10. The overall accuracy, recall, precision and F1 Score is very similar to binary classification. However, for class imbalance deterministic metrics such as G-Mean and balanced accuracy, they are way smaller than binary classification. The main reason for G-Mean to be zero is that some minority classes have zero Recall value. The balanced accuracy is also smaller than binary classification. The reason is, it gives more importance for FNs than normal accuracy. Also, in case of multi-class classification, FN values of each class vs rest FNs is calculated. Having some minority classes with small recall values has led this result.

Table 10: Overall Performance Results – Multi-class Classification

Metric	Result
Accuracy	0.986
Recall	0.986
Precision	0.981
F1 Score	0.981
G-Mean	0.000
Balanced Accuracy	0.630

The confusion matrix is given in Figure 39 for each class based on one-vs-rest. On the figure, the performance metric having the highest value is marked with dark blue color. In accordance with an imbalanced dataset having Benign class as majority, the TN values of the attack traffic (minority class) have dominance over other metrics

The TP value of Brute Force Web, Brute Force XSS and SQL Injection attacks is all equal to zero which means that our model is incapable of learning these attacks. The common feature of these attacks are they have the least percentage in the dataset. (See Figure 29)

For Brute Force Web, Brute Force XSS and SQL Injection type of attacks, the model classifies them as either DDoS Attacks LOIC HTTP or Benign.

In addition, Infiltration has high FN value that makes the precision and recall value lower. Although it has the 6^{th} order of the percentage through all dataset, this type of attack cannot be detected properly with our model. The model predicts Infiltration traffic as either Benign, Bot, HTTP DoS/DDoS, FTP/SSH Brute Force.

Table 11: Confusion Matrix Reference Table

Actual/Predicted	Predicted: NO	Predicted: YES
Actual: NO	TN	FP
Actual: YES	FN	TP

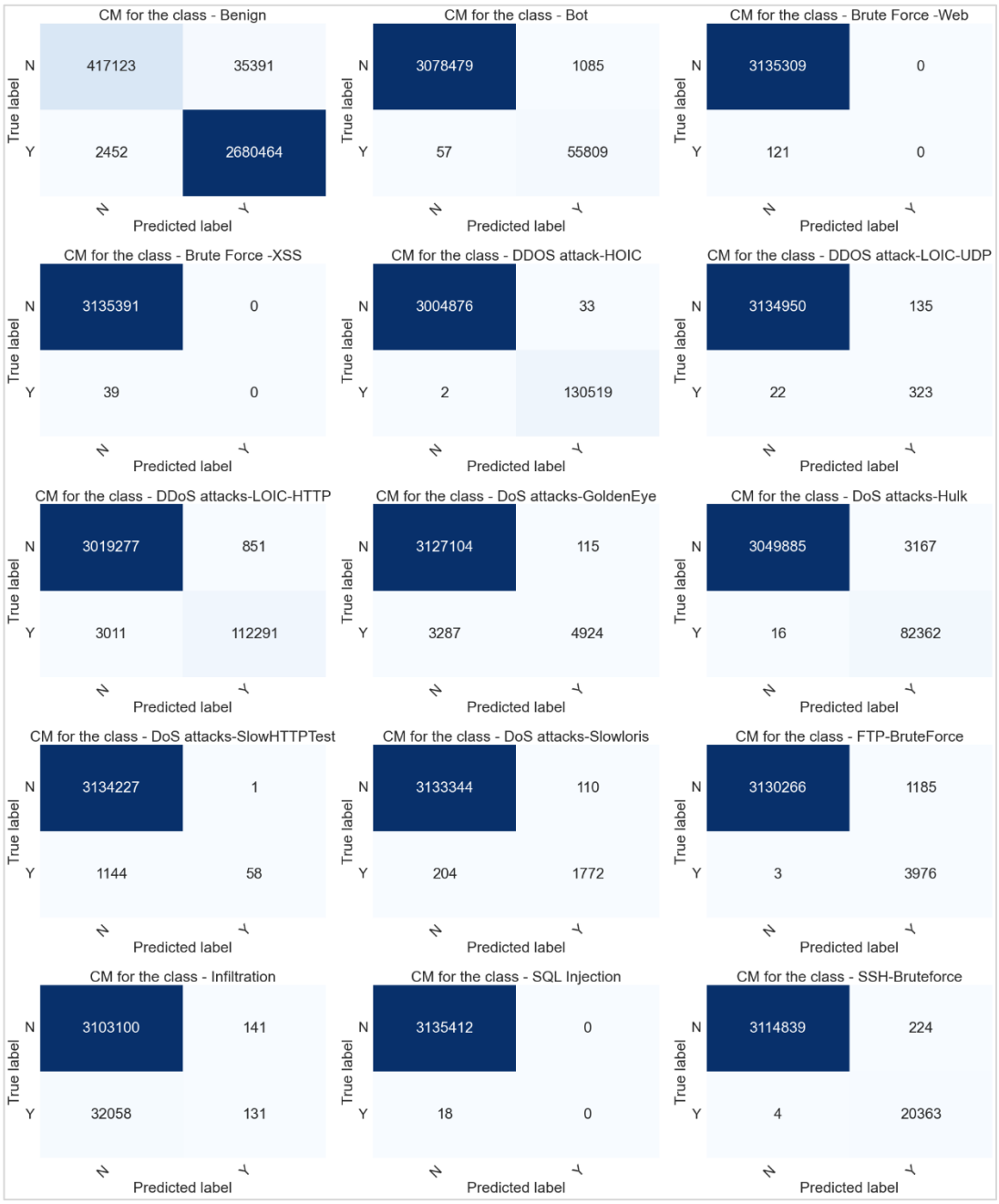

Figure 39: Confusion Matrix - Multi-class Classification

The classification report is given in Figure 40. The micro average stands for multi-class classification problems where it calculates the overall result by taking the average of TP, TN, FN and FP of each label. It is misleading for imbalanced datasets. Macro average value is more deterministic to evaluate overall performance of our model. It includes the unweighted average of all labels while calculating.

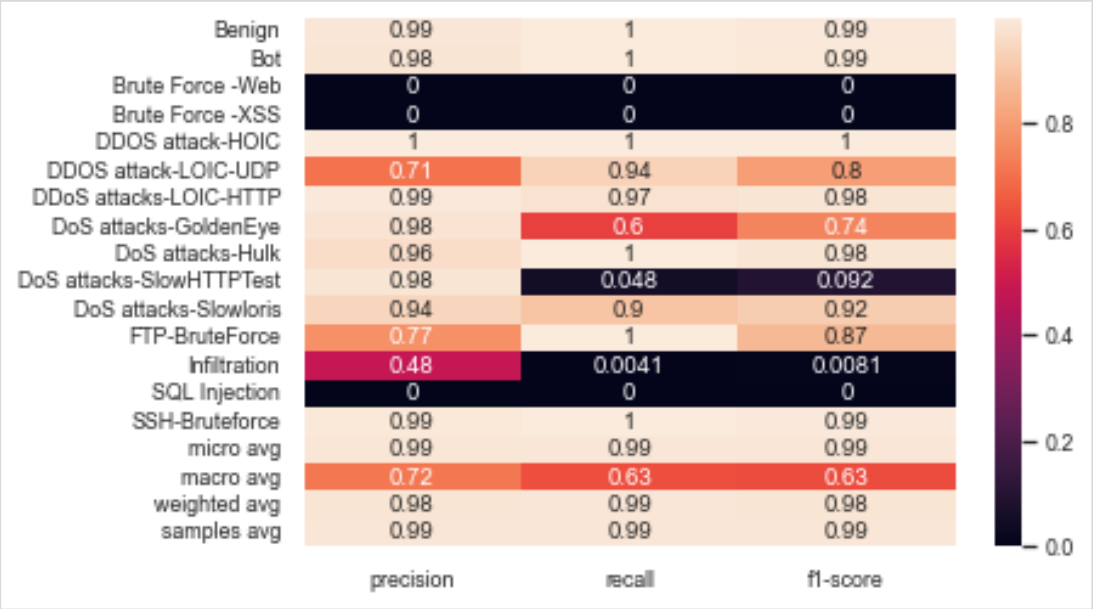

Figure 40: Classification Report – Multi-class Classification

The ROC-AUC Curve for each class is given in Figure 41. While having low precision and recall values, all classes are shown as above the threshold. This situation is misleading for imbalanced multi-class classification since TPR and FPR mostly focus on the majority class.

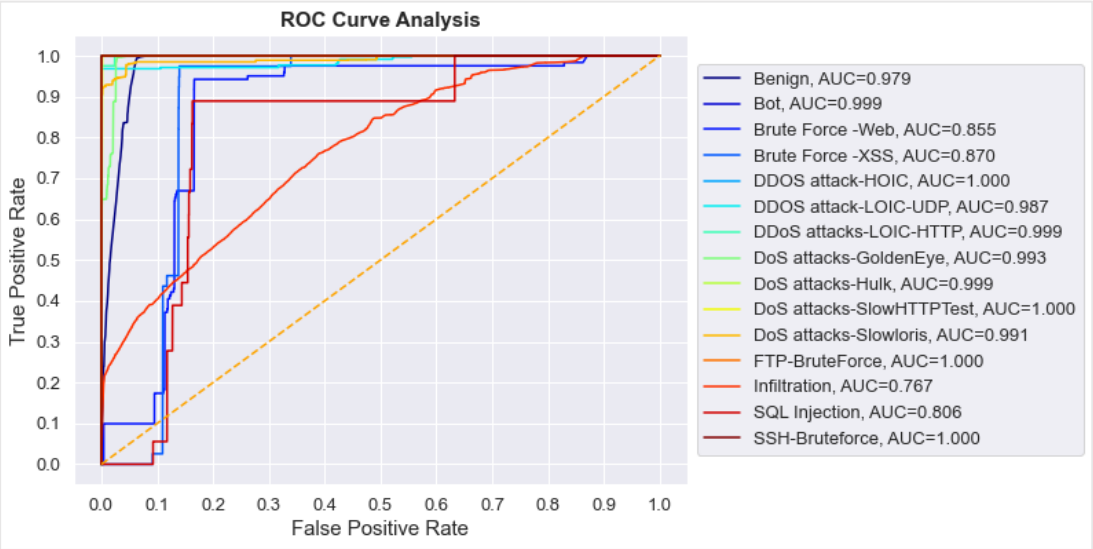

Figure 41: ROC AUC Curve – Multi-class Classification

The Precision-Recall Curve for each class is given in This gives more realistic results for our case since it focuses on Precision and Recall values.

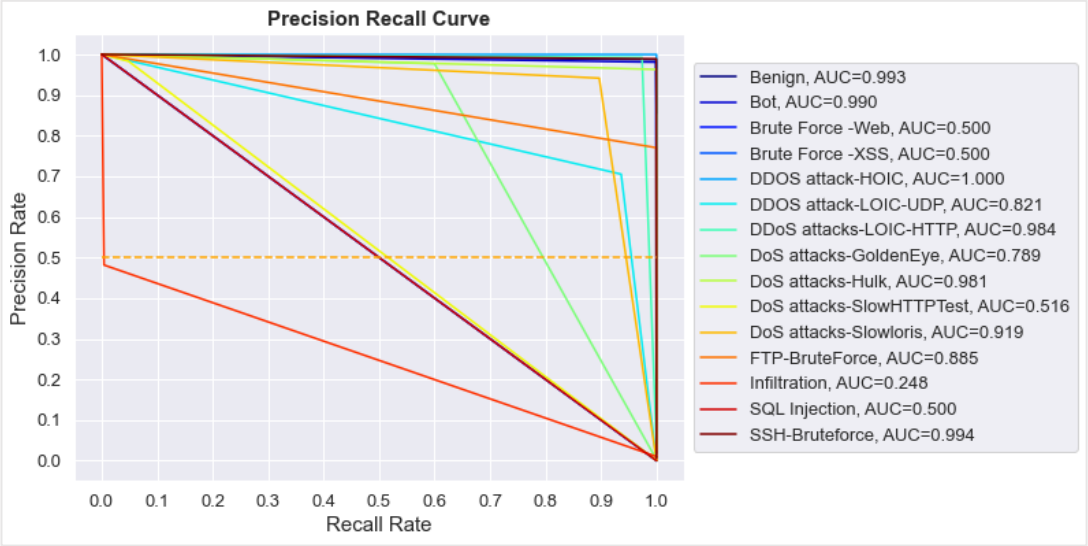

Figure 42: Precision Recall Curve – Multi-class Classification

4.2.4. Evaluation of Imbalanced Classification Problem

As explained in Section 4.1, and in the light of the results found in Section 4.2.3, there is a class imbalance problem in the dataset that prevents our model to predict minority classes. To overcome this problem, several methods are investigated. One of these methods is Sampling. It aims to change the majority and minority class ratio by changing the sample counts of majority and minority classes by using different strategies. The other method to be discussed is modifying the class weights while training. Normally, machine learning algorithms tend to give equal weight to classes while learning by default. Tweaking this default behavior by giving more weight to minority classes can be interpreted as a solution.

4.2.4.1. Sampling Strategies for Imbalanced Classification

According to (He, 2011), “Random over-sampling” and “Random under-sampling” were the first sampling methods developed for imbalanced classification.

Random over-sampling aims to increase the rate of the minority class samples for a more balanced dataset. During over-sampling, the minority class samples are randomly selected and duplicated to reach a desired rate.

Random under-sampling aims to decrease the rate of the majority class samples so as to obtain a balanced distribution in the dataset. While under-sampling the majority class samples, the samples are randomly selected and removed from the dataset until reaching the desired state.

There are two major concerns for applying these methods. One of them is for random over-sampling. Since minority class samples are duplicated, the dataset will be filled with replications of the same samples. This may cause the model memorize these values so that it can become insensitive to detect samples with small changes that leads serious overfitting problems. Other concern is for random under-sampling. Depending on the imbalance ratio of the dataset, the majority class samples are removed. Considering a dataset with very high imbalance ratio (in our case, Benign traffic dominates 83% of the dataset), high value of the majority samples has to be removed. This may lead a significant loss of information so that our model can become incapable of properly learning and the classifier performance will decrease.

As a result of these bottlenecks, different sampling strategies were introduced. All these strategies aim to overcome the problem of overfitting for over-sampling and losing useful information problem for under-sampling. Although there are new methods developed for under-sampling, we also share the similar concern for losing meaningful data. Therefore, we decided to more focus on over-sampling techniques.

At that point, SMOTE (Synthetic Minority Over-sampling Technique) is evaluated as useful for our problem. This approach was first introduced by (Chawla et al., 2002). The main difference of this approach is creating synthetic instances rather than duplicating them. These synthetic instances are created based on k-NN (k nearest

neighbors) algorithm. Referencing the original samples, k-nearest neighbors are randomly chosen and introduced as new synthetic samples (k value is 5 as default) In that way, it is aimed to create a more convincing dataset distribution for the model.

For implementing the SMOTE algorithm in our solution, "imbalanced-learn" open source, MIT-licensed library is used (Lemaître et al., 2017). As shown in Figure 24, after the preprocessing phase, SMOTE over-sampling method is applied only on the training datasets. Then, our model is trained with the new generated training datasets and tested and evaluated with the original split test datasets.
Another important point is to select the most convenient strategy to determine the sampling ratio for over-sampling. For this reason, we adopt three different strategies. Throughout all experiments, the over-sampling ratio is determined by disregarding the dominating Benign samples and focusing on the attack samples per label.

Experiment No#1 – Majority Class Referenced

The main purpose of this experiment is to make the four classes for which our model has shown the worst performance the majority class among all attack classes.
The Table 12 shows class distribution after applying SMOTE oversampling to training dataset. When we exclude Benign traffic, "DDoS attacks-LOIC-HTTP" attack type becomes the majority class. The problematic 4 class types are oversampled to have the same train samples as this majority class.

Table 12: Dataset Distribution After SMOTE #1

Classes	train
Benign	8584473
Bot	178520
Brute Force -Web	416771
Brute Force -XSS	416771
DDOS attack-HOIC	**416771**
DDoS attacks-LOIC-UDP	1107
DDoS attacks-LOIC-HTTP	368509
DoS attacks-GoldenEye	26565
DoS attacks-Hulk	264264
DoS attacks-SlowHTTPTest	3881
DoS attacks-Slowloris	6422
FTP-BruteForce	12589
Infiltration	416771
SQL Injection	416771
SSH-Bruteforce	66014

In that way, the resulting class imbalance ratio of the train dataset based on benign vs attack traffic has decreased from 5.92 to 2.85 by applying the formula (39).

The training has lasted for 40 Epochs. Learning Rate is 0.0002 and batch size is 1024. Also, the same LSTM model with the multi-class classification is used.
The Training Accuracy and Loss of the model is shown in Figure 43 and Figure 44.

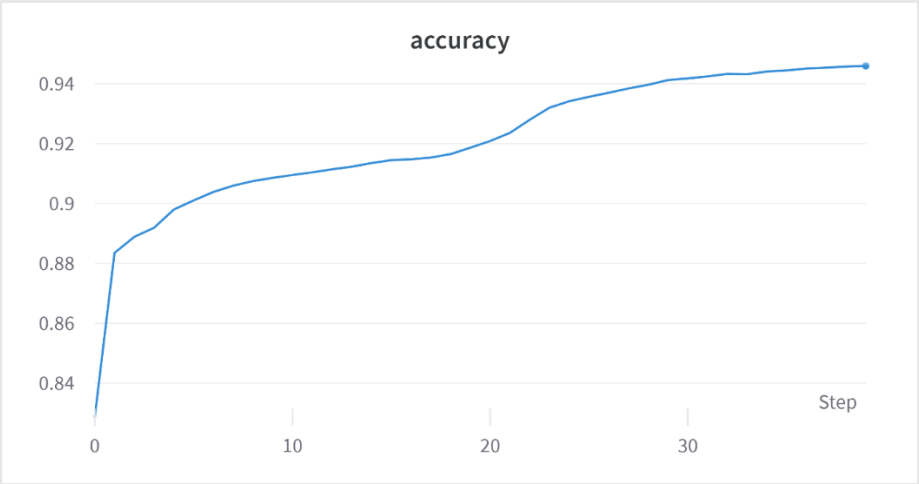

Figure 43: Training Accuracy – SMOTE #1

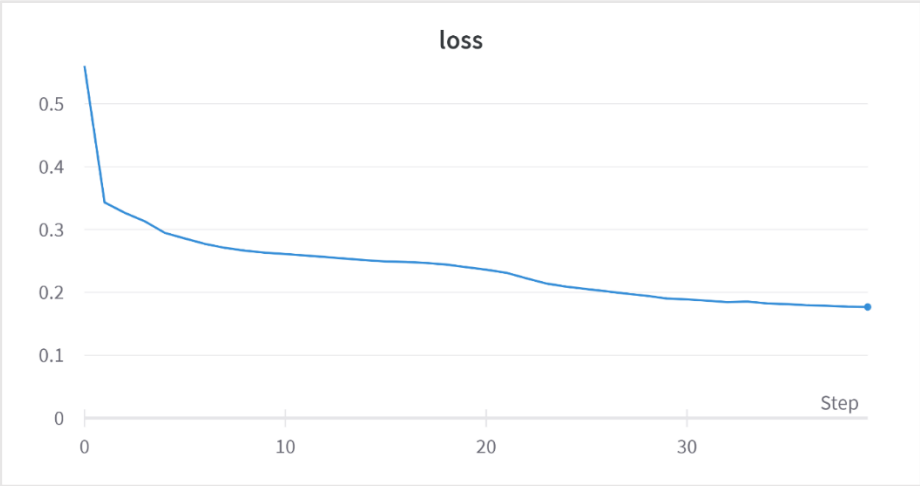

Figure 44: Training Loss – SMOTE #1

Overall performance results are shown in Table 13. Comparing with the multi-class classification results, there is a small decrease in overall accuracy and increase in overall recall, precision and F1 Score. This is an expected result since we increase the minority class samples, the domination rate of the benign traffic has decreased which caused this decrease in the accuracy. Moreover, the balanced accuracy and g-mean vales have increased which indicates more balanced classification.

Table 13: Overall Performance Results – SMOTE #1

Metric	Result
Accuracy	0.985
Recall	0.985
Precision	0.983
F1 Score	0.983
G-Mean	0.689
Balanced Accuracy	0.823

The confusion matrix is given in Figure 45 for each class based on one-vs-rest. When we investigate Brute Force Web, Brute Force XSS, Infiltration and SQL Injection classes, we conclude that there has been an improvement for detecting these attacks. Especially, the true positive rates have significantly increased for each of these attacks.

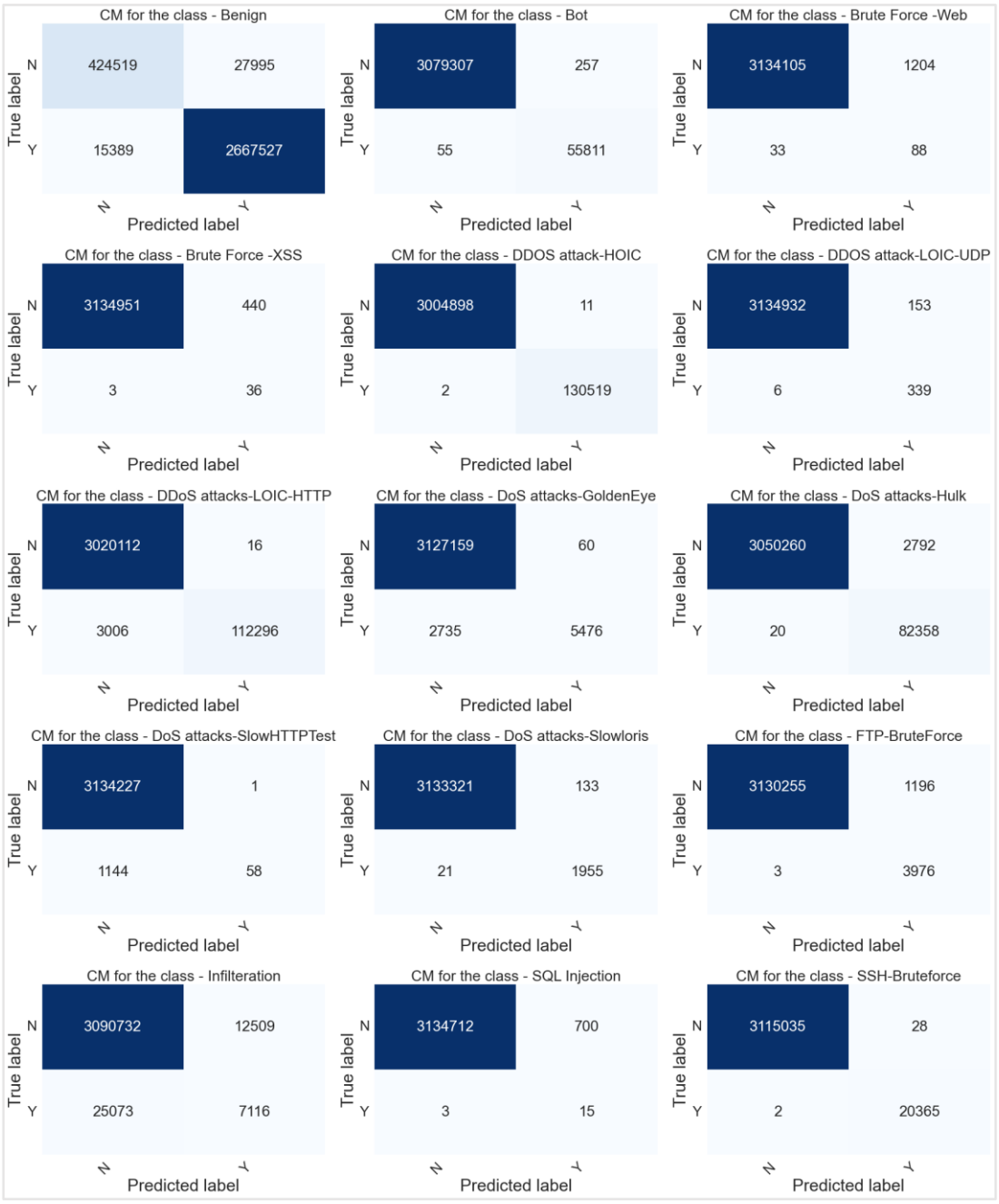

Figure 45: Confusion Matrix – SMOTE #1

The classification report is given in Figure 46. There is an improvement in the macro average value according to multi-class classification which is an expected result for a more balanced dataset. Unlike Recall value, it is seen that there is not so much increase in the Precision value for the over-sampled classes. Since FPs are deterministic for precision value, we conclude that over-sampling caused higher number of false positives. In addition, while there is a significant improvement in the performance of Web attacks, the same does not apply for infiltration. The precision and recall rates have increased but it is still not sufficient for classification.

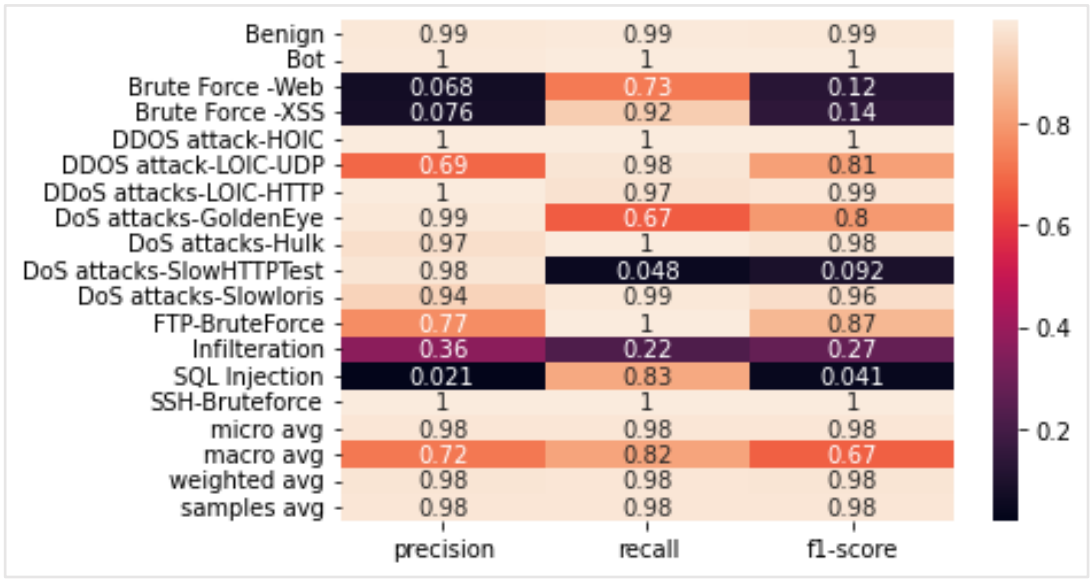

Figure 46: Classification Report – SMOTE #1

The ROC AUC Curve is given in Figure 47. The AUC values have increased according to normal multi-class classification.

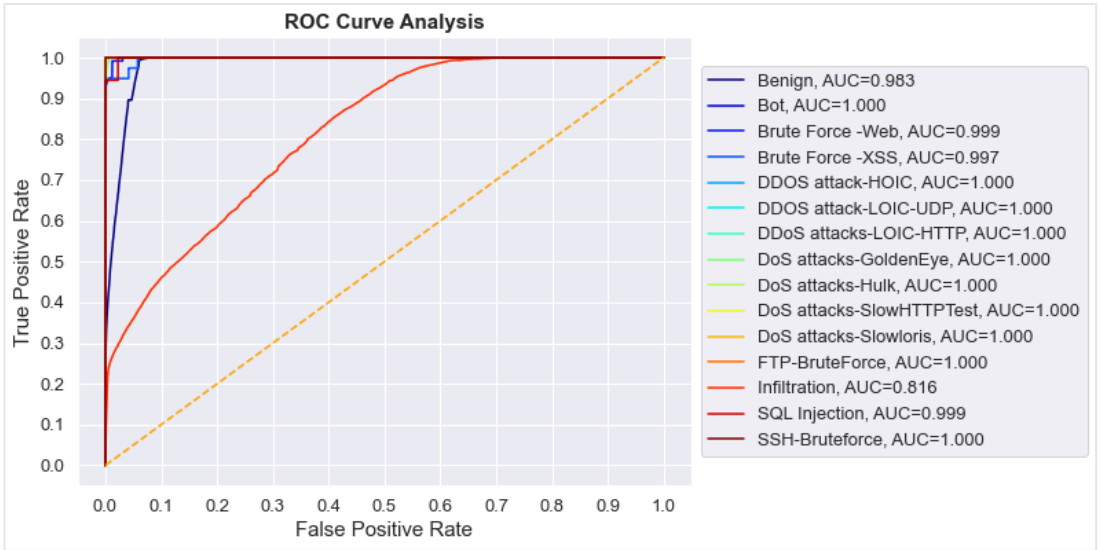

Figure 47: ROC AUC Curve – SMOTE #1

The PR-AUC Curve is given in Figure 48. Due to increase in FP rates, there isn't any improvement in PR-AUC values compared to multi-class classification.

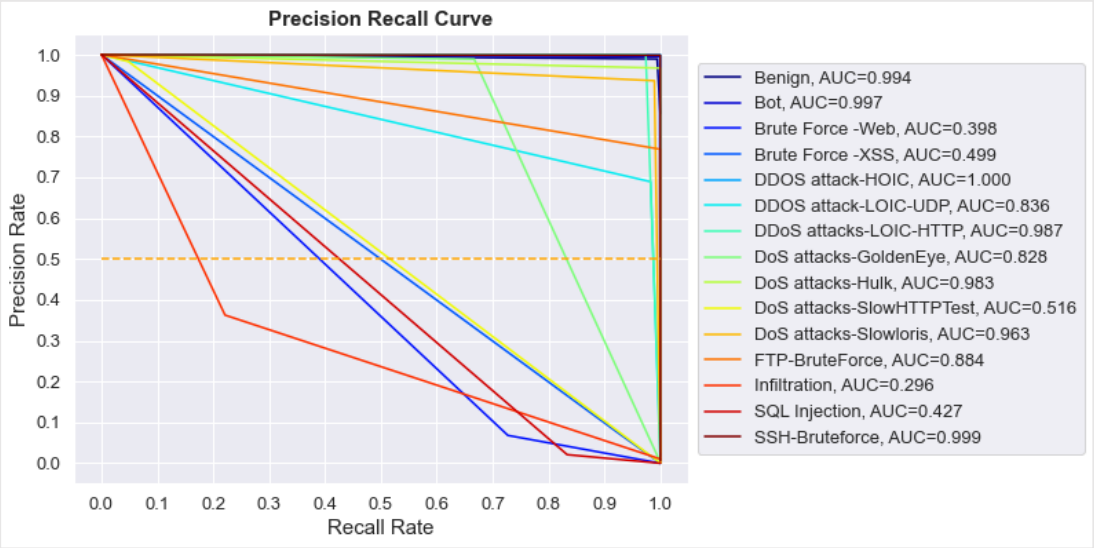

Figure 48: Precision Recall Curve - SMOTE #1

4.2.4.2. Training with Custom Class Weights

The main advantage of this method is that there is no necessity to modify the distribution of the dataset. It is seen that the SMOTE method has introduced pros and cons regarding increasing FP rates with TPs. This is an outcome of creating synthetic inputs which do not completely reflect the real scenario. For this reason, we investigated alternative solutions. Like other ML models, our model also tends to give equal weight to each class regardless of the distribution with its default settings.

As mentioned in (Lin et al., 2019), class weights can be modified during training the model. Since we are using categorical cross entropy in our loss function for multi-class classification, by changing the default class weights in order to optimize the majority and minority classes, there is another weight parameter added as a multiplier in the cross-entropy formula (38).

Since we are using Keras with Tensorflow Platform (Martin Abadi et al., 2015), class weights parameter is supported in LSTM.

Several different methods are tried. In all approaches, we evaluate the dataset by only attack traffic at first, after calculating weights for each attack type, we include the benign traffic by applying one-vs-rest approach. The class weight for the majority Benign class is very deterministic on the overall performance. As (Huang et al., 2013) depicts, in order to make the majority and minority class weights closer to each other, when we give much larger weights to minority classes, we see that our model has become incapable of detecting the majority class which decreases the accuracy.

Different methods tried for determining the class weights are listed below.

- ***Custom:*** In this method, the minority class weight is calculated with formulas (48) and (49).
  Here;
  μ is a constant value to optimize the result.
  i is the attack class number.
  a is the sample count in the corresponding attack traffic.
  n is the total number of classes, which is 15 in our case.
  b is the benign traffic sample count.
  While calculating majority class weight, we change the denominator as 2 because in this case, there are only two classes as benign and attack. Then we take the logarithm of the result for normalization.
  (Assume that k=1 denotes to benign class, k=2…n denotes to attack classes and i is in range(2,n))

$$Minority\ Class\ Weight_i = \log(\mu * \frac{1}{(n-1)} \frac{\sum_{k=1}^{n} x_k}{a_i}) \quad (48)$$

$$Majority\ Class\ Weight = \log(\mu * \frac{1}{2} * \frac{\sum_{k=1}^{n} x_k}{b}) \quad (49)$$

- ***Mean:*** In this method, we calculate the minority class weight by using Formula (48). Then, we take the mean of the minority class weights and give the same value to majority class weight.
  After determining the weights, we train our model by applying these new weights and evaluate the results.
  (Assume that k=1 denotes to benign class, k=2…n denotes to attack classes and i is in range (2, n))

$$Minority\ Class\ Weight_i = \log(\mu * \frac{\sum_{k=1}^{n} x_k}{a_i}) \quad (50)$$

$$Majority\ Class\ Weight = \log(\mu * \frac{\sum_{i=2}^{n} Minority\ Class\ Weight_i}{(n-1)}) \quad (51)$$

When we compare the results with the original multi-class classification, we see that there has not been so much change occurred. Therefore, we decided to combine this approach with SMOTE as applied in (Lin et al., 2019).

Experiment No #2 – SMOTE with Class Weights

The main purpose of this experiment is to make the four classes for which our model has shown the worst performance the majority class among all attack classes. For this reason, we apply SMOTE #1 - Majority Class Referenced with our custom class weight solution. We increase the four problematic attack class samples with SMOTE #1. Afterwards, we apply class weight by choosing our custom strategy considering the updated dataset.

The training has lasted for 40 Epochs. Learning Rate is 0.0002 and batch size is 1024. Also, the same LSTM model with the multi-class classification is used.
The Training Accuracy and Loss of the model is shown in Figure 49 and Figure 50.

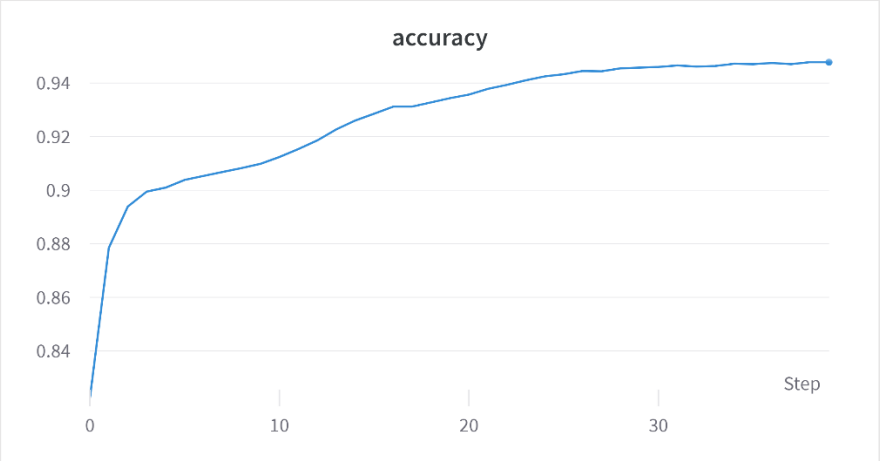

Figure 49: Training Accuracy – SMOTE #2

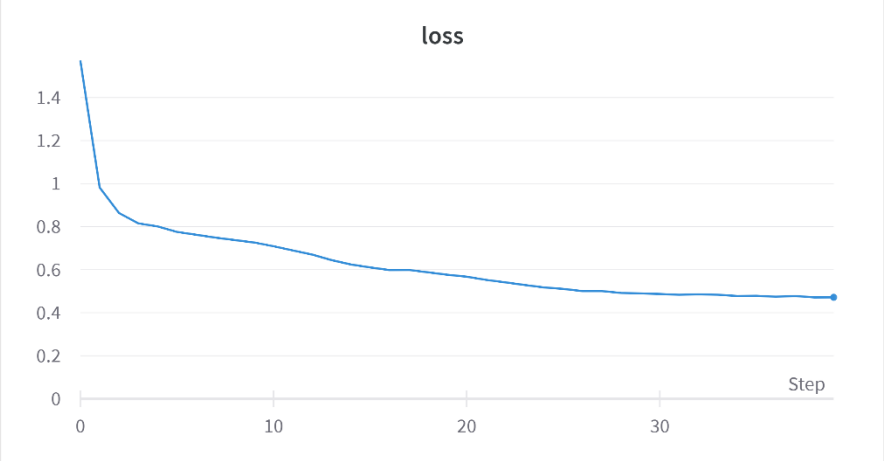

Figure 50: Training Loss – SMOTE #2

Overall performance results are shown in *Table 14*. Comparing with the multi-class classification results, while there is a small decrease in overall accuracy and recall, there is an increase in overall precision and F1 Score. Moreover, the balanced accuracy and g-mean vales have increased according to both multi-class classification only and multi-class classification with SMOTE #1 methods. This indicates a more balanced classification.

Table 14: Overall Performance Results – SMOTE #2

Metric	Result
Accuracy	0.985
Recall	0.985
Precision	0.983
F1 Score	0.983
G-Mean	0.683
Balanced Accuracy	0.816

The confusion matrix is given in *Figure 51* for each class based on one-vs-rest. When we compare the results with SMOTE #1, the FN ratio of Brute Force Web has increased. Moreover, there is a significant decrease in FP values of Brute Force Web and SQL Injection. Also, the TP rate of Infiltration has increased. When we investigate Brute Force Web, Brute Force XSS, Infiltration and SQL Injection classes, we conclude that there has been an improvement for detecting these attacks. Especially, the true positive rates have significantly increased for each of these attacks.

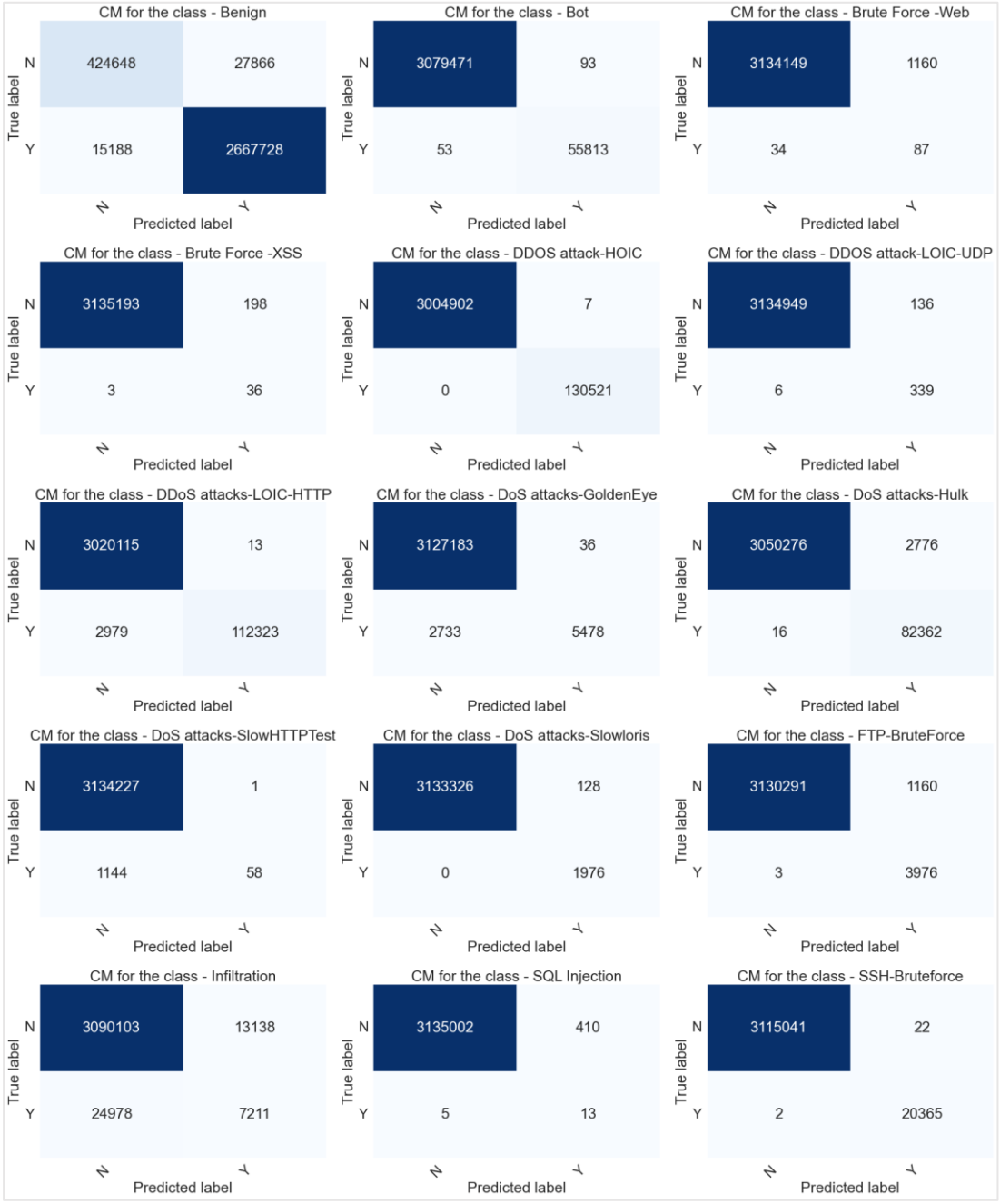

Figure 51: Confusion Matrix - SMOTE #2

The classification report is given in Figure 52. There is an improvement in the macro average values of precision, recall and F1 Score according to SMOTE #1 which is an expected result for a more balanced training.

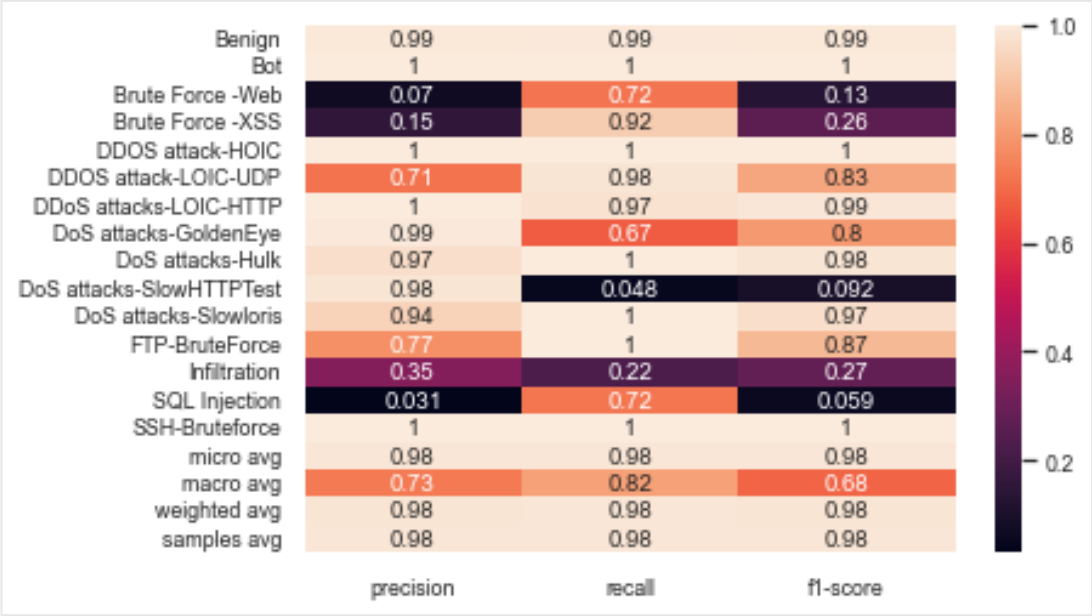

Figure 52: Classification Matrix - SMOTE #2

The ROC AUC Curve is given in Figure 53. The ROC AUC values have increased according to SMOTE #1.

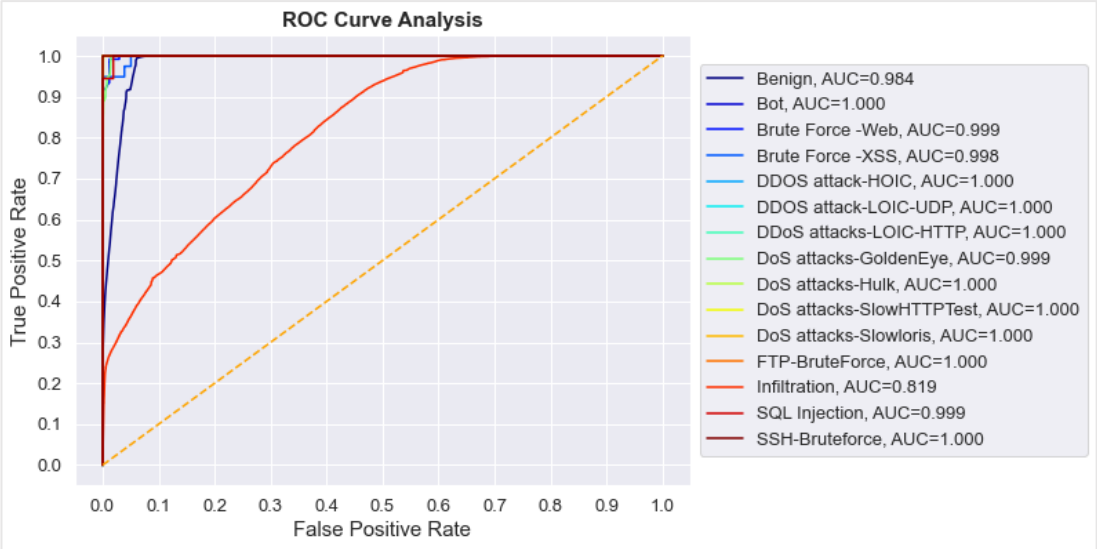

Figure 53: ROC AUC Curve - SMOTE #2

The PR-AUC Curve is given in Figure 54. There is an important increase in the Brute Force XSS PR AUC value. While it was below the threshold 0.5 in SMOTE #1, now it is over this average. Similar to SMOTE#1, due to increase in FP rates, there isn't any improvement in PR-AUC values compared to multi-class classification.

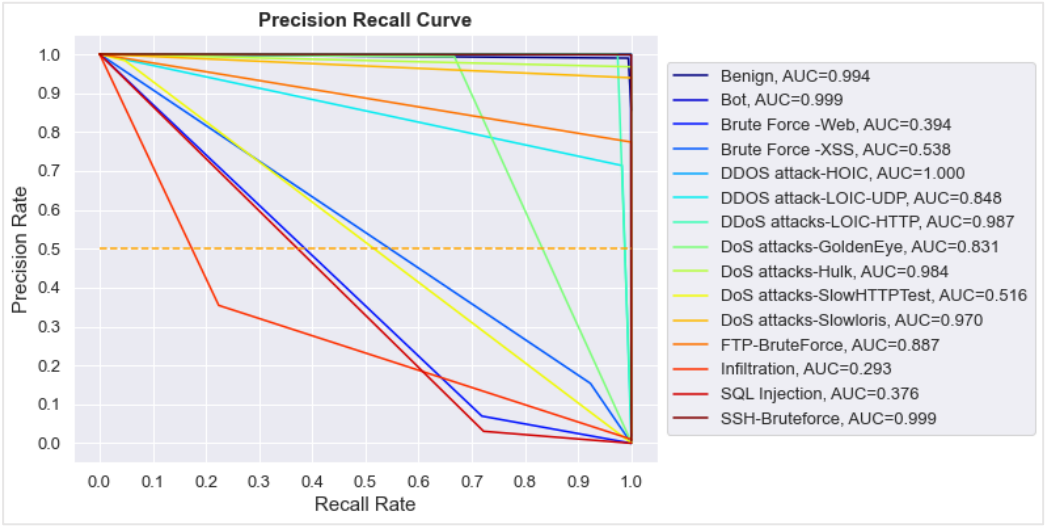

Figure 54: Precision Recall Curve - SMOTE #2

CHAPTER 5

CONCLUSION

In this thesis, we evaluated the effect of deep learning approach for intrusion detection by analyzing network flow data. We implemented a deep learning solution by using *CSE-CIC-IDS2018* dataset containing samples which are suitable for real world scenarios.

We investigated the effect of using only network flows against raw traffic or deep packet inspection on intrusion detection. The results obtained were very promising that showed us it is possible to detect a certain amount of attack traffic by network flow analysis. This is very useful to apply in real world scenarios because according to raw traffic analysis, it is lighter on resource consumption and also considering especially SSH traffic and most of the web traffic is encrypted, if you are not doing analysis inline, it is not possible to perform deep packet inspection. For these cases, network flow analysis becomes more useful.

We applied different approaches by using LSTM. After conducting feature engineering, we built out LSTM model by selecting the most efficient activation functions, optimizer and loss functions specific for our dataset and aim of prediction. By taking advantage of the LSTM's capability of time series analysis, we fit the processed data into our model. Binary classification results show that our model is capable of detecting the attack vs benign traffic. However, this model does not generalize on real world scenarios.

We examined possible solutions to overcome the class imbalance problem. Then, we applied the two of them (SMOTE & adjusting class-weights) and compared the improvement on the results. After several experiments, it is concluded that applying both of the strategies gives the most efficient solution.

During this research, we tried to find answers to the following questions;

1. *Due to the fact that traditional signature-based intrusion detection systems (IDS) are no longer sufficient to detect today's attack types, do Machine Learning based IDS perform better in detecting these attacks?*

Yes, the results depict that by using deep learning approaches, there is a considerable amount of decrease in FP and FN ratios from which most signature-based IDS suffer.

2. *Is network flow information sufficient for detecting different kinds of attacks?*
   The answer to this question is both yes and no. The deep learning model was able to detect most of the attacks introduced in our dataset. However, although several methods tried to overcome the imbalanced feature of our dataset, there has been no sufficient improvement on the detection performance of "Infiltration" traffic. Since Infiltration traffic is more sophisticated than the others, our model tended to classify this traffic as benign or other attack types. We conclude that the main reason is that we are not looking for certain patterns in the payload of the traffic. For example, if we were able to analyze the payload of the traffic, we would have recognized that the user is downloading a suspicious pdf file from Dropbox. Since we are just trying to analyze features extracted from network flow, these are not sufficient to detect such kind of traffic.
3. *Which network flow extracted features are the most effective for identifying each attack type?*
   By doing feature engineering on our dataset, we have determined the most effective features as explained in Section 3.1.
4. *Can RNNs give more efficient results in analyzing sequential information of network activity that occurs in timely manner?*
   Yes, we have obtained very promising results by using LSTM. By reordering the dataset according to timestamp data and using not so small batch sizes for our stateless LSTM helped us to achieve applicable results.
5. *What is the difference in performance between binary and multi-class classification by using LSTM?*
   We have concluded that the overall performance does not differ for both of the classification methods. However, for real world scenarios, in order to detect specific kind of attacks and develop mitigation against those attacks, multi-class classification is evaluated as more effective.

5.1. Contributions

Most of the studies either conduct their experiments in a subset of the dataset or perform only binary classification or train their model by only the separate csv files of the dataset including only specific type of attack and benign traffic. However, this doesn't reflect the real world scenario. The attacks do not come at a time as one by one. Also, benign traffic of all days are included so that the model will also be capable of learning user behavior per day/week basis. Moreover, fitting whole dataset into our model for training helps our model to learn to decompose specific attack types from others.

When we examine the obtained results and compare them with the existing studies, this approach is evaluated as the most efficient for intrusion detection.

However, for multi-class classification, due to applying several class imbalance solutions, we see that our model still performing unsufficiently for detecting certain classes. The common feature of these classes is that their sample percentage in the

dataset is comparedly much smaller than the others. Whether we generate syntetic data or not, the imbalance ratio is too large that while we try to produce mode data, we become distant form the original samples. It is determined that this can be a reason for the results. Moreover, when we change the class weights while learning, it is seen that it also affects other classes since in that way, we decrease the weight of the other classes and the resultant distribution does not reflect the real-world scenario.

In addition to the previous findings, as explained in Section 2.4.4, it is seen that most of the related works did not include detailed information about the preprocessing and the used hyperparameters during their experiments. This seriously affects the reproducibility of these studies. Therefore, in this thesis, we added detailed explanation about our approach for the pre-processing and hyperparameter tuning. The source code used during the experiments also can be reached via GitHub. Moreover, used metrics for evaluating the performance of the model is another important issue that most of the previous works suffer. Since our dataset is highly imbalanced, including only accuracy for the evaluation is not sufficient. For this reason, we included additional metrics such as balanced accuracy, g-mean, PR-AUC and ROC-AUC which give more realistic results for our case.

5.2. Future Works

As an improvement of our solution, it is evaluated that besides using class imbalance solutions, a more up-to-date and worth-to-try solution such as “Attention Mechanism” can be used as a future work for increasing the performance. (Li et al., 2018; Yang et al., 2020; Yu et al., 2018)

Moreover, the minority classes in our dataset can be extended by feeding CICFlowMeter with own generated captures.

In order to move one step further, now that our model has shown better performance according to many studies, after improving the performance of the model, a comparison is expected to be done with the traditional S-IDS under real traffic. For this reason, in order to convert raw traffic into flow based extracted features, CICFlowMeter can be integrated with the Snort (Lashkari, 2020). Afterwards, the detection performance of the model vs Snort detection with its flow preprocessor features can be compared. In that way, we can test our model under different conditions.

The adaptability of our model to the real world scenario is one of the other issues to be considered. As a new research topic, the integration of our model with signature based IDS can be investigated.

REFERENCES

Abdellah Af. (2021). *DDoS Hulk Github Page*. https://github.com/Mr4FX/HULK

Afshine Amidi, S. A. (n.d.). *Recurrent Neural Networks cheatsheet*. https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks

Amini, M., Jalili, R., & Shahriari, H. R. (2006). RT-UNNID: A practical solution to real-time network-based intrusion detection using unsupervised neural networks. *Computers and Security*, *25*(6), 459–468. https://doi.org/10.1016/j.cose.2006.05.003

Anderson, J. P. (1980). Computer security threat monitoring and surveillance. *Technical Report James P Anderson Co Fort Washington Pa*, 56. https://doi.org/citeulike-article-id:592588

Atefinia, R., & Ahmadi, M. (2021). Network intrusion detection using multi-architectural modular deep neural network. *Journal of Supercomputing*, *77*(4), 3571–3593. https://doi.org/10.1007/s11227-020-03410-y

Basnet, R. B., Shash, R., Johnson, C., Walgren, L., & Doleck, T. (2019). Towards detecting and classifying network intrusion traffic using deep learning frameworks. *Journal of Internet Services and Information Security*, *9*(4), 1–17. https://doi.org/10.22667/JISIS.2019.11.30.001

Canadian Institute of Cyber Security. (n.d.). *CICFlowMeter (formerly ISCXFlowMeter)*. https://www.unb.ca/cic/research/applications.html#CICFlowMeter

Cedric Nabe. (2021). *Impact of COVID-19 on Cybersecurity*. https://www2.deloitte.com/ch/en/pages/risk/articles/impact-covid-cybersecurity.html

Chawla, N. v., Bowyer, K. W., Hall, L. O., & Kegelmeyer, W. P. (2002). SMOTE: Synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research*, *16*(June), 321–357. https://doi.org/10.1613/jair.953

Cisco. (2020). *Future of Secure Remote Work Report*. https://www.cisco.com/c/dam/en/us/products/collateral/security/secure-remote-worker-solution/future-of-secure-remote-work-report.pdf

Claise, B. (2008). Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of IP Traffic Flow Information (Issue 5101). RFC Editor. https://doi.org/10.17487/RFC5101

CSE-CIC-IDS2018 on AWS. (n.d.). Retrieved September 19, 2021, from https://www.unb.ca/cic/datasets/ids-2018.html

Drewek-Ossowicka, A., Pietrołaj, M., & Rumiński, J. (2021). A survey of neural networks usage for intrusion detection systems. Journal of Ambient Intelligence and Humanized Computing, 12(1), 497–514. https://doi.org/10.1007/s12652-020-02014-x

Farah, N., Avishek, Md., Muhammad, F., Rahman, A., Rafni, M., & Md., D. (2015). Application of Machine Learning Approaches in Intrusion Detection System: A Survey. International Journal of Advanced Research in Artificial Intelligence, 4(3), 9–18. https://doi.org/10.14569/ijarai.2015.040302

Ferrag, M. A., Maglaras, L., Moschoyiannis, S., & Janicke, H. (2020). Deep learning for cyber security intrusion detection: Approaches, datasets, and comparative study. Journal of Information Security and Applications, 50, 102419. https://doi.org/10.1016/j.jisa.2019.102419

Gamage, S., & Samarabandu, J. (2020). Deep learning methods in network intrusion detection: A survey and an objective comparison. Journal of Network and Computer Applications, 169(July), 102767. https://doi.org/10.1016/j.jnca.2020.102767

Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep Learning. MIT Press. http://www.deeplearningbook.org

Graves, A. (2013). *Generating Sequences With Recurrent Neural Networks*. 1–43. http://arxiv.org/abs/1308.0850

Greff, K., Srivastava, R. K., Koutnik, J., Steunebrink, B. R., & Schmidhuber, J. (2017). LSTM: A Search Space Odyssey [Institute of Electrical and Electronics Engineers Inc.]. In *IEEE Transactions on Neural Networks and Learning Systems* (Vol. 28, Issue 10). https://doi.org/10.1109/TNNLS.2016.2582924

Gsami, Rezos, Thiagoalz, KristenS, D0ubl3 h3lix, Andrew Smith, Jenjava1762, Mtesauro, kingthorin. (n.d.). *Brute Force Attack*. https://owasp.org/www-community/attacks/Brute_force_attack

Guy Bruneau. (2001). *The History and Evolution of Intrusion Detection*. https://www.sans.org/reading-room/whitepapers/detection/history-evolution-intrusion-detection-344

He, H. (2011). Imbalanced Learning. *Self-Adaptive Systems for Machine Intelligence*, 44–107. https://doi.org/10.1002/9781118025604.ch3

Hinton, G. E., Osindero, S., & Teh, Y.-W. (2006). Communicated by Yann Le Cun A Fast Learning Algorithm for Deep Belief Nets 500 units 500 units. *Neural Computation*, *18*(June), 1527–1554. https://doi.org/10.1162/neco.2006.18.7.1527

Hochreiter, S. (1991). Untersuchungen zu dynamischen neuronalen Netzen. *Institut Für Informatik, Technische Universität, Munchen*, 1–71. http://people.idsia.ch/~juergen/SeppHochreiter1991ThesisAdvisorSchmidhuber.pdf

Hochreiter, S., & Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*, *9*(8), 1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735

Huang, W., Song, G., Li, M., Hu, W., & Xie, K. (2013). Adaptive Weight Optimization for Classification of Imbalanced Data. In C. Sun, F. Fang, Z.-H. Zhou, W. Yang, & Z.-Y. Liu (Eds.), *Intelligence Science and Big Data Engineering* (pp. 546–553). Springer Berlin Heidelberg.

Ian Reynolds. (2020). *2020 SANS Network Visibility and Threat Detection Survey*. https://www.sans.org/reading-room/whitepapers/analyst/2020-network-visibility-threat-detection-survey-39490

Jan Seidl. (2020). *GoldenEye Github Page*. https://github.com/jseidl/GoldenEye

Jason Brownlee. (2020). *Tour of Evaluation Metrics for Imbalanced Classification*. https://machinelearningmastery.com/tour-of-evaluation-metrics-for-imbalanced-classification/

Khraisat, A., Gondal, I., Vamplew, P., & Kamruzzaman, J. (2019). Survey of intrusion detection systems: techniques, datasets and challenges. *Cybersecurity*, *2*(1). https://doi.org/10.1186/s42400-019-0038-7

Kim, J., Kim, J., Thu, H. L. T., & Kim, H. (2016). Long Short Term Memory Recurrent Neural Network Classifier for Intrusion Detection. *2016 International Conference on Platform Technology and Service, PlatCon 2016 - Proceedings*, 8–12. https://doi.org/10.1109/PlatCon.2016.7456805

Kingma, D. P., & Ba, J. L. (2015). Adam: A Method for Stochastic Optimization. *CoRR*, *abs/1412.6*, 1–15.

Kwon, D., Kim, H., Kim, J., Suh, S. C., Kim, I., & Kim, K. J. (2019). A survey of deep learning-based network anomaly detection. *Cluster Computing*, *22*(s1), 949–961. https://doi.org/10.1007/s10586-017-1117-8

Lai, H., Cai, S., Huang, H., Xie, J., & Li, H. (2004). A parallel intrusion detection system for high-speed networks. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *3089*, 439–451. https://doi.org/10.1007/978-3-540-24852-1_32

Lanjelot. (2021). *Patator Github Page*. https://github.com/lanjelot

Lashkari, A. H. (2020). *CICFlowMeter*. https://github.com/CanadianInstituteForCybersecurity/CICFlowMeter

Leevy, J. L., & Khoshgoftaar, T. M. (2020). A survey and analysis of intrusion detection models based on CSE-CIC-IDS2018 Big Data. *Journal of Big Data*, *7*(1). https://doi.org/10.1186/s40537-020-00382-x

Lemaître, G., Nogueira, F., & Aridas, C. K. (2017). Imbalanced-learn: A Python Toolbox to Tackle the Curse of Imbalanced Datasets in Machine Learning. *Journal of Machine Learning Research*, *18*(17), 1–5. http://jmlr.org/papers/v18/16-365.html

Levy, J. (2020). *Sophos 2021 Threat Report*. 36. https://www.sophos.com/en-us/medialibrary/pdfs/technical-papers/sophos-2021-threat-report.pdf

Lin, P., Ye, K., & Xu, C. Z. (2019). Dynamic network anomaly detection system by using deep learning techniques. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *11513 LNCS*(November), 161–176. https://doi.org/10.1007/978-3-030-23502-4_12

Li, Y., Zhu, Z., Kong, D., Han, H., & Zhao, Y. (2018). EA-LSTM: Evolutionary attention-based LSTM for time series prediction. *ArXiv*.

Martin Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Jia, Y., Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, … Xiaoqiang Zheng. (2015). *{TensorFlow}: Large-Scale Machine Learning on Heterogeneous Systems*. http://tensorflow.org/

McAfee. (2021). *McAfee Threat Report 04.21*. 1–24.

Mcculloch, W. S., & Pitts, W. (1990). A logical calculus nervous activity. *Bulletin of Mathematical Biology*, *52*(l), 99–115.

Ming, G., Kenong, Z., & Jiahua, L. (2006). Efficient packet matching for gigabit network intrusion detection using TCAMs. *Proceedings - International Conference on Advanced Information Networking and Applications, AINA*, *1*, 249–254. https://doi.org/10.1109/AINA.2006.164

Mishra, P., Varadharajan, V., Tupakula, U., & Pilli, E. S. (2019). A detailed investigation and analysis of using machine learning techniques for intrusion detection. *IEEE Communications Surveys and Tutorials*, *21*(1), 686–728. https://doi.org/10.1109/COMST.2018.2847722

Murphy, K. P. (2022). *Probabilistic Machine Learning: An introduction*. MIT Press. https://probml.github.io/pml-book/book1.html

Nsrav. (n.d.). *Denial of Service*. https://owasp.org/www-community/attacks/Denial_of_Service

Olah, C. (2015). *Understanding LSTM Networks*. http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Pascanu, R., Mikolov, T., & Bengio, Y. (2013). On the difficulty of training recurrent neural networks. *30th International Conference on Machine Learning, ICML 2013, PART 3*, 2347–2355.

patator Usage Example. (2021). https://www.kali.org/tools/patator/

Rian Dolphin. (2020). *LSTM Networks | A Detailed Explanation*. https://towardsdatascience.com/lstm-networks-a-detailed-explanation-8fae6aefc7f9

Shah, N. (2020). *The Challenges of Inspecting Encrypted Network Traffic*. https://www.fortinet.com/blog/industry-trends/keeping-up-with-performance-demands-of-encrypted-web-traffic

Shah, S. A. R., & Issac, B. (2018). Performance comparison of intrusion detection systems and application of machine learning to Snort system. *Future Generation Computer Systems*, *80*(November 2017), 157–170. https://doi.org/10.1016/j.future.2017.10.016

Sharafaldin, I., Lashkari, A. H., & Ghorbani, A. A. (2018). Toward generating a new intrusion detection dataset and intrusion traffic characterization. *ICISSP 2018 - Proceedings of the 4th International Conference on Information Systems Security and Privacy*, *2018-Janua*(Cic), 108–116. https://doi.org/10.5220/0006639801080116

Sherstinsky, A. (2020). Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) network. *Physica D: Nonlinear Phenomena*, *404*, 132306. https://doi.org/10.1016/j.physd.2019.132306

Shirey, R. W. (2007). *Internet Security Glossary, Version 2* (Issue 4949). RFC Editor. https://doi.org/10.17487/RFC4949

Singla, R. K., Guleria, A., & Sharma, R. (2018). An overview of flow-based anomaly detection. *International Journal of Communication Networks and Distributed Systems*, *21*(2), 220. https://doi.org/10.1504/ijcnds.2018.10014505

slowhttptest Usage Example. (n.d.). https://www.kali.org/tools/slowhttptest/

Sperotto, A., Schaffrath, G., Sadre, R., Morariu, C., Pras, A., & Stiller, B. (2010). An overview of IP flow-based intrusion detection. *IEEE Communications Surveys and Tutorials*, *12*(3), 343–356. https://doi.org/10.1109/SURV.2010.032210.00054

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, *15*(3–4), 1929–1958.

Steven Vaughan-Nichols. (2018). *The day computer security turned real: The Morris Worm turns 30*. https://www.zdnet.com/article/the-day-computer-security-turned-real-the-morris-worm-turns-30/

SuperDataScience Team. (2018). *Recurrent Neural Networks (RNN) - The Vanishing Gradient Problem*. https://www.superdatascience.com/blogs/recurrent-neural-networks-rnn-the-vanishing-gradient-problem

Tang, T. A., Mhamdi, L., McLernon, D., Zaidi, S. A. R., & Ghogho, M. (2016). Deep learning approach for Network Intrusion Detection in Software Defined Networking. *Proceedings - 2016 International Conference on Wireless Networks and Mobile Communications, WINCOM 2016: Green Communications and Networking*, 258–263. https://doi.org/10.1109/WINCOM.2016.7777224

Urbanowicz, R. J., & Moore, J. H. (2015). ExSTraCS 2 . 0 : description and evaluation of a scalable learning classifier system. *Evolutionary Intelligence*, 89–116. https://doi.org/10.1007/s12065-015-0128-8

Vinayakumar, R., Alazab, M., Soman, K. P., Poornachandran, P., Al-Nemrat, A., & Venkatraman, S. (2019). Deep Learning Approach for Intelligent Intrusion Detection System. *IEEE Access*, *7*, 41525–41550. https://doi.org/10.1109/ACCESS.2019.2895334

Wandb Main Page. (n.d.). https://wandb.ai/

Weber, N. (2017). *Why LSTMs Stop Your Gradients From Vanishing: A View from the Backwards Pass*. https://weberna.github.io/blog/2017/11/15/LSTM-Vanishing-Gradients.html

Yang, S., Tan, M., Xia, S., & Liu, F. (2020). A method of intrusion detection based on Attention-LSTM neural network. *ACM International Conference Proceeding Series*, 46–50. https://doi.org/10.1145/3409073.3409096

Yin, C., Zhu, Y., Fei, J., & He, X. (2017). A Deep Learning Approach for Intrusion Detection Using Recurrent Neural Networks. *IEEE Access*, *5*, 21954–21961. https://doi.org/10.1109/ACCESS.2017.2762418

Yu, Y., Liu, G., Yan, H., Li, H., & Guan, H. (2018). Attention-Based Bi-LSTM Model for Anomalous HTTP Traffic Detection. *2018 15th International Conference on Service Systems and Service Management, ICSSSM 2018*. https://doi.org/10.1109/ICSSSM.2018.8465034