Network Intrusion Detection Model Using TabNet

Samuel Eshetu Workneh

A Thesis Submitted to

The Department of Computer Science and Engineering

School of Electrical Engineering and Computing

Presented in Partial Fulfillment of the Requirements for the Degree of Master's in

Computer Science and Engineering

Office of Graduate Studies

Adama Science and Technology University

September 2022

Adama, Ethiopia

Network Intrusion Detection Model Using TabNet

Samuel Eshetu Workneh

Advisor: Prof. B.M.Thippeswamy (Ph.D.)

A Thesis Submitted to

The Department of Computer Science and Engineering

School of Electrical Engineering and Computing

Presented in Partial Fulfillment of the Requirements for the Degree of

Master's in Computer Science and Engineering

Office of Graduate Studies

Adama Science and Technology University

September 2022

Adama, Ethiopia

Approval Page of M.Sc. Thesis

I, the advisor of the thesis entitled "**Network Intrusion Detection Model Using TabNet**" and developed by Samuel Eshetu, hereby certify that the recommendation and suggestions made by the board of examiners are appropriately incorporated into the final version of the thesis.

Prof. B.M. Thippeswamy (Ph.D.) _____ _____

Major Advisor Signature Date

We, the undersigned, members of the Board of Examiners of the thesis by Samuel Eshetu have read and evaluated the thesis entitled "**Network Intrusion Detection Model using TabNet**" and examined the candidate during the open defense. This is, therefore, to certify that the thesis is accepted for partial fulfillment of the requirement of the degree of Master of Science in Computer Science and Engineering (CSE).

_____ _____ _____

Chairperson Signature Date

_____ _____ _____

Internal Examiner Signature Date

_____ _____ _____

External Examiner Signature Date

Finally, approval and acceptance of the thesis are contingent upon the submission of its final copy to the Office of Postgraduate Studies (OPGS) through the Department Graduate Council (DGC) and School Graduate Committee (SGC).

_____ _____ _____

Department Head Signature Date

_____ _____ _____

School Dean Signature Date

_____ _____ _____

Office of Postgraduate Studies, Dean Signature Date

Declaration

I hereby declare that this Master Thesis entitled" **Network Intrusion Detection Model Using TabNet**" is my original work. That is, it has not been submitted for the award of any academic degree, diploma, or certificate in any other university. All sources of materials that are used for this thesis have been duly acknowledged through citation.

Samuel Eshetu _____ _____ _____

Name of the Student Signature Date

Recommendation

I, the advisor of this thesis, hereby certify that I have read the revised version of the thesis entitled "**Network Intrusion Detection Model Using TabNet**" prepared under my guidance by Samuel Eshetu submitted in partial fulfillment of the requirements for the degree of Mater's of Science in Computer Science and Engineering (CSE). Therefore, I recommend the submission of the revised version of the thesis to the department following the applicable procedures.

Prof. B.M Thippeswamy. (Ph.D.) _____ _____

Major Advisor Signature Date

Acknowledgments

First of all, I want to thank the Almighty God for making things possible and helping me in all aspects of my life. This study would not have been possible without his grace and blessings

I would like to express my gratitude to Prof. B.M Thippeswamy. (Ph.D.), my advisor, for his guidance, his knowledge, and his willingness to discuss the subject in depth.

I would like to thank the computer science and engineering staff, Dr. Mesfin Abebe, Dr. Biruk Yirga, Genet Shankoo (Ph.D. candidate), Endris Mohamed (Ph.D. candidate), SIG supervisor, and PG coordinator who deserve special thanks for their insightful comments, and helpful assistance with my thesis.

I would also like to thank the entire Data Science Special Interest Group (DS Sig) and friends who shared their knowledge with me, as well as everyone else who helped with this research.

Table of Contents

List of Tables

List of Figures

List of Acronyms and Abbreviations

AE	Autoencoder
AI	Artificial Intelligence
AUROC	Area under precision-recall curve and time measure.
CIA	Confidentiality, Integrity, Availability
CNN	Convolutional neural network
DDOS	Distributed denial of service
DL	Deep learning
DNN	Deep neural network
DOS	Denial of service
EDA	Explanatory Data Analysis
ELU	Exponential Linear Unit
EM	Expectation maximization
FC	Fully Connected
FN	False negative
FLN	Fast learning network
FP	False positive
KDD	Knowledge discovery and data mining
HIDS	Host-based intrusion detection system
IDS	Intrusion detection system
ML	Machine learning
NDAE	Non-symmetric deep Autoencoder
NIDS	Network-based intrusion detection system
PSO	Particle swarm optimization
RNN	Recurrent neural network
SVM	Support vector machine
SMOTE	Synthetic Minority Oversampling Technique
TP	True Positive
LSTM	Long short-term memory
TN	True negative
XAI	Explainable Artificial Intelligence
XGBoost	Extreme Gradient Boosting

Abstract

Cyber-attacks are increasing worldwide in their complexity, coverage, and damaging impacts. Every victim or target of cyber-attacks must strengthen their defense, mitigation, and recovery mechanism to be immune from future cyber-attacks. One such mechanism is an intrusion detection system. An intrusion detection system is a software or firmware system that monitors intrusion activities in the network or host devices. To defend well from cyber-attacks, an intrusion detection system must be efficient in its detection performance, processing speed, and response time. And also, an intrusion detection system must balance interpretability with high detection performance to be trusted with its detection result and to be deployed in real-world critical application areas. But existing intrusion detection systems especially network-based intrusion detection systems that are implemented using machine learning and deep learning techniques are suffered from some challenges such as low detection performance, complex and opaque detection models, high model training, and test time. Therefore, this research study developed high detection performance and interpretable network intrusion model using TabNet deep neural network and sequential attention encoder model. The model was trained and tested using a recent, more realistic, and representative CICIDS2017 dataset. TabNet encoder's performance to transform the input data into a useful representation of features and its ability to choose salient features sequentially in each step resulted in a high-performance and locally as well as globally interpretable network intrusion detection model. Performance evaluation of the developed model shows a 0.00049 False Positive Rate value and 99.98% of accuracy value on the balanced dataset experiment. A comparative study with state-of-the-art XGBoost model and Autoencoder models on detection performance, model interpretability, scalability, and robustness criteria also shows a promising result of the proposed model.

Keywords: *Intrusion Detection Systems, Cybersecurity, TabNet, Sequential Attention, Encoder Model, XGBoost, Autoencoder, CICIDS2017*

CHAPTER ONE

INTRODUCTION

1.1 Background

After the invention of the computer, the first computer network, the Advanced Research Project Agency Network (ARPANET) was established by the US Department of Defense in the early 1960s (Leiner et al., 2009). These two important Technological inventions (the computer and network) laid the foundations for the creation of the internet and let people share resources using the internet easily. For the last six decades, the internet and related technologies revolutionized dynamically and exponentially (Akpakwu et al., 2018). Currently, the Internet and its infrastructure in the world create huge digital cyberspace. This rapid change in cyberspace improves human lifestyle in different sectors (Li & Liu, 2021a).

But with the advancement of digital technologies such as the internet and computer, cyber security has become a more challenging and critical issue due to the complexity and sophistication of cyber-attacks (Li & Liu, 2021a). Cyber-attacks lead to violation of the five major goals of cyber security which are Confidentiality, Availability, Authentication, Integrity, and Non-repudiation. Attackers or Intruders compromise the normal traffic of the network using different attack techniques such as Denial of Service (DoS), Brute Force attacks, SQL injection, Cross-Site Scripting, Cross-Site Request Forgery, Buffer Overflow attack, Remote File Inclusion attacks, Network Probing, Host Header attack, Malware attack, Ransomware attack, Crypto-Hijacking, Session Hijacking, and many other attack techniques (Pawar & Anuradha, 2015) (Hindy et al., 2020).

The above several attacks led to financial loss, damages, workflow disruption, loss of resources, privacy violation and other irreversible damages to internet reside resources and assets. For these reasons, different users of the internet implement different security policies, defense techniques, and security assessment mechanisms. Among these Antiviruses, Firewalls, Threat Intelligence software, Secure Protocol Systems, and other security tools are used. In addition, most network users use intrusion detection systems (IDS) and intrusion prevention systems (IPS) by integrating with other security mechanisms for defending and preventing intrusion activities (Hindy et al., 2020).

Intrusion detection systems (IDSs) are software systems or software and hardware combined systems that automatically monitor intrusion events occurring in a computer system or network, analyzing them for signs of security problems or intrusion activities and sending reports to the network administrator or security information and event management systems.[1] IDSs are complementary technologies to firewalls and perform deep packet inspection, gaining visibility into detail often missed by firewalls (Surana et al., 2017).

Intrusion detection systems can be classified based on the information source, analysis approach, time aspect, architecture, and response method. Based on information sources IDS is classified as host-based IDS (HIDS) and Network-based IDS (NIDS). Based on the analysis approach it is classified as signature-based IDS and anomaly-based IDS. Based on the time aspect it is classified as real-time detection and off-time detection. Based on architecture it is classified as centralized, distributed, and heterogeneous IDS. Based on the response method it is classified as active and passive IDS (Rajasekaran & Nirmala, 2020). In network anomaly detection systems, their decision model can be implemented using the statistical method, knowledge-based method, and Machine Learning/Deep Learning based methods. Currently, ML/DL-based network anomaly detection methods attract many researchers in the domain area and become state-of-the-art techniques because of the capabilities of machine learning and deep learning algorithms to solve such kinds (classification) of problems (Halimaa & Sundarakantham, 2019; Z. el Mrabet et al., 2019). In addition, intrusion detection systems can be hybridized by combining the advantages of the above-listed IDS types.

All of the above IDS types have their pros and cons with their detection process and methods. For instance, a signature-based intrusion detection system detects any previously known intrusions by comparing signature records. But it is unable to detect previously unknown intrusion activity. Whereas an anomaly-based intrusion detection system can detect previously unknown intrusions. But it is suffering from many false positive detections. These weaknesses of current available IDSs and the growing dynamism and complexity of intruders' activity are challenges to the efficiency of intrusion detection

[1] https://en.wikipedia.org/wiki/Intrusion_detection_system

systems. The goal of all intrusion detection systems is to efficiently detect any intrusion activity and to notify these activities network administrators or any monitoring component.

Both Machine Learning and Deep Learning methods show their performance and capabilities in different domain areas such as image processing natural language processing and online recommender systems (Thakkar & Lohiya, 2021; Uikey & Gyanchandani, 2019). That is why they become state-of-the-art methods in these domain areas. but if they are not studied properly, they can't be the solution for all domains. For example, when we look at Machine learning-based Network intrusion detection models are simple and easy to implement but have less detection performance when compared with deep learning-based network intrusion detection models (Ahmed et al., 2016; M. A. el Mrabet et al., 2021; Khraisat et al., 2019; Surana et al., 2017; Ugurlu & Dogru, 2019). Whereas Deep learning-based network intrusion detection models are complex, opaque, and difficult to deploy in a real-world environment (Ansari et al., 2020; Hodo et al., 2017). We can conclude from this there is a trade-off between detection performance and model interpretability or model explainability(Dosilovic et al., 2018; F. Xu et al., 2019). In real-world applications, it is important to balance detection performance with model interpretability to get the trust of the end users.

A lot of research work has been conducted to mitigate the challenges faced by intrusion detection systems. Researchers suggest improvement ideas on IDS architectures, models, detection approaches, and deployment areas of intrusion detection systems that enhance the efficiency of existing Intrusion Detection Systems. This research study develops an efficient Network Intrusion detection model using TabNet deep neural network-based sequential encoder architecture(Arik & Pfister, 2019). According to the paper (Arik & Pfister, 2019), TabNet shows high-performance values in classification and regression tasks that are tested on numerous kinds of structured or tabular datasets. The paper also mentioned that the TabNet model performance outperforms state-of-the-art Decision tree-based algorithms such as XGBoost and Random Forest models. This research study also identifies the use of old, nonrealistic, and unrepresentative datasets, the use of datasets with class imbalance problems, and improper model performance evaluation metrics as research gaps from previous related research studies (Ashoor, 2019; Goel & Nussbaum, 2021; Halimaa & Sundarakantham, 2019; Sharafaldin et al., 2018; Surana et al., 2017). Therefore, the research study considers these issues in the proposed solution.

1.2 Motivation

A lot of research work on intrusion detection systems has been done in recent years. Due to these research works intrusion detection systems have been improved from time to time (Surana et al., 2017). Like other researchers, the researcher in this research study is motivated to work in this area for the following main reason. Cybersecurity attacks have been increasing from time to time throughout the world. Political, economic, and social impact also increases from time to time (Akpakwu et al., 2018; Khraisat & Alazab, 2021; Leiner et al., 2009). The defense mechanisms such as intrusion detection systems must be up to date with research and development to monitor these damaging attacks. That is why large researcher communities participated in this research domain area. Therefore, the researcher of this research study wants to contribute some knowledge by identifying the research gap in the domain area.

1.3 Statement of the problem

Intrusion detection systems are critical systems since they are concerned with cyber security. Critical systems must gain the trust of users to be deployed in the real-world operating environment. Intrusion detection systems can get trust if they have high performance of detection for any kind of intrusion activities as well as if their decision process is easily understandable and traceable for anyone who can use it.

As we discussed in the previous section, the revolution of cyber-attacks in their number, complexity, impact, and, intruder's activity such as obfuscation, fragmentation, and flooding are major challenges for existing intrusion detection systems. The inefficiency of existing intrusion detection systems in their detection accuracy, detection coverage, and response time are also other challenges (Aljanabi et al., 2021; Khraisat & Alazab, 2021). Specifically, existing anomaly-based intrusion detection systems implemented using machine learning and deep learning techniques for monitoring network traffic also suffer from some limitations.

In general detection model complexity, opaqueness, low detection performance[2], and disability to handle the ever-growing cyber security attack sophistication and dynamism are challenges for existing Network Intrusion detection models (Aziz et al., 2021).

[2] Low detection performance can be characterized with high False Positive or False Negative rate and low True Posttive or True Negative rate.

Therefore, this research study proposed an efficient Network Intrusion Detection model in terms of detection accuracy and model interpretability using the TabNet model. TabNet model is a deep neural network-based high-performance and interpretable model which works based on a sequential attention mechanism (Arik & Pfister, 2019). Then the model will be trained and tested using the latest and more realistic intrusion dataset named the CICIDS2017 dataset (Maseer et al., 2021a; Sharafaldin et al., 2018).

1.4 Research Questions

Q1. How to improve the performance of the network intrusion detection model with the proposed TabNet encoder?
Q2. How to make a global and local interpretable network intrusion detection model using TabNet?
Q3. On what criteria does the proposed TabNet-based model outperform the state-of-the-art XGBoost-based and Autoencoder-based models?

1.5 Objectives

1.5.1 General Objectives

The general objective of this research study is to develop an efficient network intrusion detection model in terms of detection performance and model interpretability.

1.5.2 Specific Objectives

To achieve the above general objective, this research study identifies the following specific objectives.

- To properly study existing Intrusion Detection Systems with an extensive literature review.
- To select appropriate Intrusion Datasets from publicly available popular Intrusion datasets.
- To develop Intrusion Detection Models using the TabNet Deep Neural Network Architecture.
- To study the performance of the proposed model on various test conditions.
- To compare the developed TabNet model with state-of-the-art XGBoost and Autoencoder-based Intrusion Detection Models.

1.6 Scope and limitations of the Study

1.6.1 Scope of the Study

The scope of this research study is to develop a Network Intrusion Detection model using TabNet, a Deep Neural Network Architecture-based encoder model.

1.6.2 Limitations of the Study

This research study did not complete the following tasks due to time, budget, and resource constraints.

- The research study doesn't implement training time optimization techniques.
- The developed intrusion detection model doesn’t detect zero-day attacks since the decoder part of the TabNet model is not implemented.
- The proposed solution only uses a publicly available dataset.

1.7 Significance of the Study

The significance of this research study can be expressed in terms of its importance, critical area of application, and the research outcomes after implementing the output of this research study (Aljanabi et al., 2021)(Ashoor, 2019).

- High-performance Network intrusion detection model that can be integrated into intrusion detection systems in the real world to enhance detection accuracy.

- Reliable and Trustable Network intrusion detection model because of its interpretability behavior.

- As mentioned above, the intrusion detection system will continue to improve with research and development whenever there are security attacks. Therefore, this research study adds some contribution to the existing Intrusion detection system domain research study and will be a reference for future research work in this domain area.

- Any commercial intrusion detection system vendor can integrate this efficient intrusion detection model into their IDS decision-making component.

1.8 Application Areas of Network Intrusion Detection System

The most attack vector in cyberspace is a Network. Usually, intruders or attackers start from the reconnaissance phase to launch a series of cyber attacks. The reconnaissance phase is the first attacking phase that attackers gather as much as possible information about their target using active and passive footprinting. The network contains a lot of information about the devices used in the network, application systems deployed and running in the network, and user agents in the network. Intruders may use this information to get initial access to the target system and also for pivoting or privilege escalating inside the network. In general, networks without cyber defense mechanisms create potential attack surfaces for attackers.

Cyber security is not a single feature. And it is also impossible to give a 100% guarantee of cyber security, instead combined defense mechanisms are implemented to maximize protection. Network Intrusion Detection System is one defense mechanism usually combined with Firewalls, Web Application Firewalls (WAF), and Intrusion Prevention systems (IPS) to create a demilitarized zone (DMZ) of the network. The deployment of NIDS depends on the requirements of the organization or the network owner. If NIDS is needed to monitor the LAN network, it is best practice to deploy it on a parallel server to the gateway switch or router behind the firewall. The parallel configuration helps the NIDS to get a copy of all network traffic from the gateway. Network Test Acess Point (TAP) or switch port analyzer (SPAN) ports of the switch are used to get the exact copy of network traffic for the NIDS. Large organizations may have two or more Local Area Networks (LANs) for web servers, mail servers, and database servers. In this case, different NIDSs can be deployed in each LAN, or only the data capture component of NIDS can be distributed over each LAN and the remaining NIDS components may reside in one of those LANs.

The network intrusion detection model developed in this research study can be integrated with other IDS components: data capture, data preprocessing, and alert components to create a complete network intrusion detection system. The complete network intrusion detection system can be deployed in the local area networks of banks, military institutions, hospitals, universities, government organizations, private business organizations, and in the network infrastructure of other companies. The deployment of this efficient network intrusion detection system within the organization's network will strengthen the defense

mechanism of that network by detecting all possible malicious activities directed at the assets residing in that network.

1.9 Organization of the Thesis Work

This section of the document summarizes the number of chapters used and the overall contents of the research study as follows:

Chapter One: - Introduction: This chapter covers the background of the study, statement of the problem, the objective of the study, Significance of the study, scope, and limitations of the study.

Chapter Two: - Literature Review: This chapter covers intrusion detection systems, intrusion detection system category, feature selection, feature extraction, types of network attacks, application areas of intrusion detection systems, deep learning, machine learning, Network intrusion detection system, and TabNet model will be discussed.

Chapter Three: - Methodology of the research: This chapter includes Dataset, data integration, encoding categorical data, data transformation, data dimensional reduction, missing and duplicate value handling, split dataset, Model development, model evaluation, and materials will be discussed.

Chapter Four: Proposed solution: This chapter explains in detail the architecture of the proposed solution with necessary diagrams.

Chapter Five: - Implementation: In this chapter preprocessing and modeling concepts will be discussed in detail.

Chapter Six: - Result and Discussion: under this chapter, the result of the experiment will be discussed and compared to the other research work.

Chapter Seven: - Conclusions and Recommendations: this is the last chapter of this research study; it contains the conclusion of the study and recommendations forwarded by the study.

CHAPTER TWO

LITERATURE REVIEW

2.1 Theoretical Concepts

2.1.1 Cyber Security and Attacks

Cyber security is a branch of security that focuses on protecting systems, networks, computers, and data, from unauthorized access (intentional or unintentional), modification, or destruction[3]. It is estimated that approximately 1 million potential cyberattacks are attempted per day, and with the evolution of mobile and cloud technologies, this number has the likelihood to increase in the future (Jena et al., 2021). Defensive cyber security's range of operations involves protecting systems, applications, and data from major cyber threats. However, cyber threats have become increasingly innovative and the need for proper cyber defense mechanisms becomes more important than ever (Li & Liu, 2021b).

A cyberattack can be defined as a malicious and deliberate attempt to gain unauthorized access to a computer system, application, or network. Below are some types of common cyberattacks (Goel & Nussbaum, 2021).

Injection attacks[4]: - It allows an attacker to inject malicious code into a network, program, query, and fetch data from the database to the attacker. This type of attack also allows an attacker to inject malware to execute commands remotely.

DNS Spoofing attacks[5]: - DNS spoofing is a type of computer security hack. This includes inserting data into the DNS resolver's cache. This will cause the nameserver to return incorrect IP addresses, redirecting traffic to the attacker's computer or another computer. DNS spoofing attacks can go undetected for long periods and can pose a serious security problem.

[3] https://www.cisco.com/c/en/us/products/security/what-is-cybersecurity.html
[4] https://www.acunetix.com/blog/articles/injection-attacks/
[5] https://www.kaspersky.com/resource-center/definitions/dns

Session Hijacking[6]: - This is a type of cyberattack in which attackers hijack user sessions over a protected network. The attacker steals user web cookies and accesses user data collected in the session.

Phishing attacks[7]: - Phishing is a type of attack in which an attacker impersonates a trusted entity in an attempt to steal sensitive information such as user credentials or credit card numbers.

Denial of Service (DoS)[8]: - This is an attack designed to flood the target's traffic with information and render a server or network resource unavailable to the user, thereby causing a system crash. Attacks servers using a single system and internet connection.

Man in the Middle Attacks[9]: - This is a type of attack that allows an attacker to intercept connections between clients and servers and act as a bridge between them. This allows attackers to read, insert, and modify data in intercepted connections.

Virus Attacks[10]: - This is a malicious software program that, once executed, spreads through computer files without the user's knowledge. It can also run commands that harm your system.

Worm Attacks[11]: - This is malicious software that resembles a computer virus. However, the main difference between the two is that viruses activate when triggered by a host, whereas worms spread and replicate independently.

Trojan Horse Attacks[12]: - This is a malicious program that is often disguised as legitimate software, but when executed, it runs malicious Code in the background, giving cybercriminals backdoor access to the user's system.

[6] https://owasp.org/www-community/attacks/Session_hijacking_attack
[7] https://www.cisco.com/c/en/us/products/security/email-security/what-is-phishing.html
[8] https://en.wikipedia.org/wiki/Denial-of-service_attack
[9] https://en.wikipedia.org/wiki/Man-in-the-middle_attack
[10] https://www.fortinet.com/resources/cyberglossary/computer-virus
[11] https://www.cisco.com/c/en/us/products/security/what-is-a-worm.html
[12] https://www.kaspersky.com/resource-center/threats/trojans

Backdoors[13]: - This is a type of Trojan horse that facilitates unauthorized remote access to a user's computer system or network. A backdoor may have a legitimate purpose, such as granting access for troubleshooting.

Bots[14]: - A bot is a script or software program that performs automated tasks while replacing the actions of a human user. Some bot programs run automatically, while others execute commands only when certain inputs are given.

Figure 2.1 shows the total economic damage cost that has happened to start from 2013 to 2020.

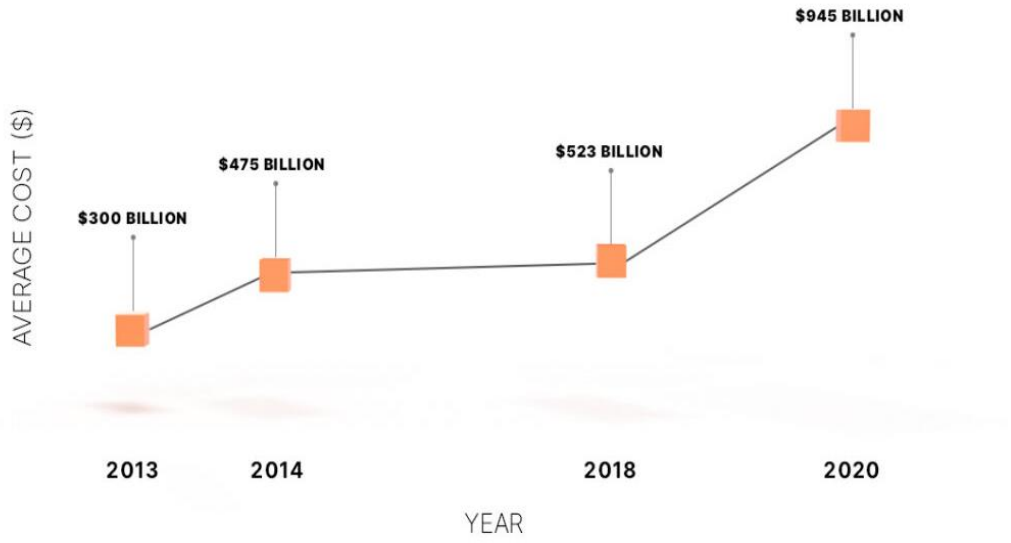


Figure 2.1 Financial loss annual statistics caused by cyber-attacks

Source: adopted from (*Business Cost of Cybercrime | Cobalt*, n.d.)

2.1.2 Intrusion Detection System

An intrusion detection system (IDS) is a software, hardware, or combined system that detects malicious traffic on the network. To do this task, IDS accesses all the network traffic to monitor the network traffic properly. Intrusion detection systems are deployed with other defense systems such as intrusion prevention systems (IPS), firewalls, and honeypots to strengthen immunity from cyber-attacks (Aljanabi et al., 2021; Aziz et al., 2021).

[13] https://www.malwarebytes.com/backdoor
[14] https://www.kaspersky.com/resource-center/definitions/what-are-bots

2.1.3 Type of Intrusion Detection System

Although it is ambiguous to identify the real taxonomy of Intrusion detection systems, existing IDS can be classified based on different approaches. These approaches are based on the data source, analysis strategy, time aspect, architecture, deployment, and response (Ahmad et al., 2022; Elsadai et al., 2019; Hindy et al., 2020; Li & Liu, 2021b).

Based on the data source

- *Host-based IDS*[15]: - input data for the IDS system is collected from the host computing system. And also, HIDS monitors and analyzes activities in the host. The limitation of HIDS is it doesn't monitor the network part.
- *Network-based IDS*[16]: - input data for the IDS is captured from the network traffic and it monitors the entire network topology for any malicious activities. The limitation of NIDS is that it loses some traffic because it has a workload due to its extensive monitor infrastructure.

Based on the analysis strategy

- *Signature-based IDS*[17]: - SIDS is a type of IDS that detects malicious activities by looking at the signature and comparing it to previously known attack signatures. SIDS is best for previously known attacks. But it has limitations to detect zero-day attacks.
- *Anomaly-based IDS*[18]: AIDS is a type of IDS that works by creating a baseline for normal activities and detecting any activity that deviates from the normal baseline. AIDS are best for detecting zero-day attacks although they suffer from repeated false positive alarms.

Based on the Time Aspect

- *Real-time IDS*: - The detection module component is already deployed in a real-time environment and operated on real data input of the monitored component.
- *Off-line IDS*: - Usually IDS are under the research and development phase using simulated data as input for the detection model. This type of IDS is called offline IDS.

[15] https://www.sciencedirect.com/topics/computer-science/host-based-intrusion-detection-system
[16] https://www.sciencedirect.com/topics/computer-science/network-based-intrusion-detection-system
[17] https://www.geeksforgeeks.org/intrusion-detection-system-ids
[18] https://www.sciencedirect.com/topics/computer-science/anomaly-based-detection

Based on Architecture

- *Centralized IDS*: - In this type of architecture the whole IDS components are deployed in a single host. These centralized ids may monitor remote components.
- *Distributed IDS*: - The IDS components especially the data capture component and decision engines are deployed in a distributed environment as distributed agents.

Based on Deployment

- *On-premises IDS*: - On this type of deployment the whole component is deployed within the monitored infrastructure.
- *Cloud-based IDS*: - This type of deployment is when the heavy computing component of the intrusion detection system is deployed in the cloud environment.

Based on Response

- *Active IDS*: - Is a type of IDS that actively blocks when it encounters any malicious activity.
- *Passive IDS*: - This type of IDS simply logs or sends notifications to monitoring agents when it detects malicious activity instead of blocking the traffic.

But when it comes to real-world implementation, we don't find each of the above intrusion detection systems separately or independently, as they have limitations. Hybrid approaches are solutions to improve the above-mentioned limitation of each intrusion detection system.

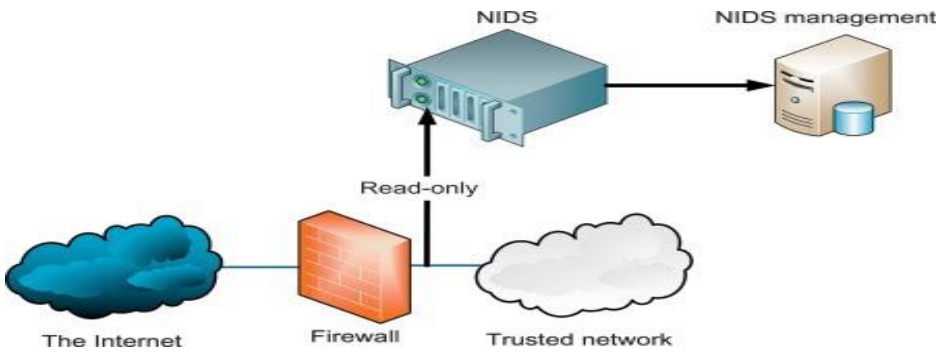


Figure 2.2 Network-based intrusion detection system

Source: adopted from (NIDS - an Overview | ScienceDirect Topics, n.d.)

2.1.4 Components of Intrusion Detection System

Usually, existing NIDS has four main common components in real-world operation (Aljawarneh et al., 2018; Zhou et al., 2020).

Data capture component: - This is the first component in IDS and it continuously captures network traffic data. Capturing fast and dynamic data with high speed is a challenge in this component. Currently, hardware solutions and distributed parallel processing are proposed to improve challenges in this component.

Data preprocessing components: - This component preprocesses the captured data from the previous component to fit the input criteria of the next decision-making model component. For example, if the decision model is a deep learning model the captured data must be converted into a feature vector. Speed of preprocessing is a challenge in this component.

Decision-making component: - This is the main component of the intrusion detection system which classifies the incoming traffic as malicious or benign based on its embedded model. The model can be implemented using different approaches such as signature-based, anomaly-based, machine learning-based, knowledge-based, and so on.

Alert system components: This component works based on the decision model component result. If the decision model component detects malicious activity, it will notify the monitoring agent based on its alert configuration.

2.1.5 Machine Learning

Machine learning is a subfield of artificial intelligence that studies how machines are learned from their previous experience. In traditional programming, we give machine data and rules as input and generate some outputs. Whereas in machine learning we give the machine output data (their previous experience) and input data as input and generate rules as output. This learning ability of machines to learn from their previous experience data and generate rules based on the learning helps to solve complex problems that are hard to write rules manually. There are three learning approaches in the machine learning field of study namely supervised learning, unsupervised learning, and reinforcement learning.

Supervised learning is a type of learning in which a label or class is predefined in the learning data before the training phase. During the training phase, the mode learns the

relation between the class and the features. Some supervised learning algorithms are support vector machine (SVM), Bayesian network, decision tree, k-nearest neighbor, and random forest (RF)(Burkart & Huber, 2020; M. A. el Mrabet et al., 2021).

Unsupervised learning is a type of learning in which the model does not need human supervision during its learning time or predefined class or labels are not needed in the training process. k- mean cluster, hidden Markov models, Hierarchical clustering, and ant clustering are some examples of unsupervised learning (Reis et al., 2021; Usama et al., 2019).

Reinforcement learning is somehow different from the above two methods which use a trial-and-error approach. In reinforcement learning, there is an agent which does an action. Based on its action the agent perceives a change of state in the environment. Then the agent will receive a reward or punishment based on the goal or policy. This helps the reinforcement agent to learn from its live experience. Q-learning, state action reward state action (SARSA), and the Markov decision process are some of the reinforcement algorithms (Alavizadeh et al., 2021; Arulkumaran et al., 2017; Padakandla, 2020).

All of the above classical machine learning algorithms are applied to different problems such as classification, regression detection, and generation problems. Machine learning techniques are applied to different sectors such as agriculture, health care, entitlements security, and so on. Classical Machine learning techniques face problems when the representation of the problem is more complex when more nonlinear data is there and performance stack when experience data become big and big. In supervised learning, the goal is to find a mapping function between the input features and output label whereas in unsupervised learning the goal of the learning is to learn the underlying structure of the input features. the goal of reinforcement learning is to maximize the future reward through many time steps.

2.1.6 XGBoost

The name XGBoost is an abbreviation for extreme gradient boosting algorithm. It is not only an algorithm but also a system or framework. XGBoost is a fast and high-performance tree-based algorithm that can be applied for classification and regression problems usually with a structured dataset. XGBoost is built with an ensemble method called boosting. Boosting combines weak learner models sequentially so that each new tree corrects an error of the previous model. XGBoost uses gradient descent to minimize the loss function.

parallelization concept and cache optimization make it a very fast algorithm. Regularization technique, auto pruning of the tree, and self-handling of missing values also make XGBoost a high-performance model. It is best suitable for small to medium-structured data. XGBoost algorithm has a lot of hyperparameters to be tuned during the experiment to get a good result (Chen et al., 2018; Chen & Guestrin, 2016).

2.1.7 Deep Learning

Classical machine learning models sometimes cannot learn a complex representation of data, that needs nearly human-level understanding. A major source of difficulty in many real-world artificial intelligence applications is that many of the factors of variation influence every single piece of data we can observe. Deep Learning represents complex representations by introducing simple representations. build complex concepts from simple concepts. Deep in deep learning tells the depth of learning. Deep learning is the step-by-step learning of simple features to learn the overall complex representation in depth. Deep learning is an approach to machine learning that has drawn heavily on our knowledge of the human brain, statistics, and applied math as it developed over the past several decades (Dong et al., 2021; Menghani, 2021; Pouyanfar et al., 2018).

2.1.8 Autoencoder Architecture

Autoencoder models are unsupervised feedforward neural network models, in which the number of neurons in the input layer is the same as the number of neurons in the output layers. Autoencoder layer stacks can be partitioned into encoder part, Code part or bottleneck part, and decoder part as shown in figure 2. 3 the encoder part tries to compress the input dimension to a low-dimensional representation called latent space. The Code part is a layer in the Autoencoder with a low dimensional input size or latent space representation. The decoder part tries to decompress the low-dimensional latent space representation to reconstruct the input dimension in the output layer. The training of the Autoencoder model continues until the reconstruction error becomes minimal, thus the difference between input and output values becomes less and less.

Denoising Autoencoder, variational Autoencoder stacked Autoencoder (deep Autoencoder), nonsymmetric Autoencoder, sparse Autoencoder, and convolutional Autoencoder are some variants of Autoencoder architecture scheme. Most of the time

Autoencoder models are applied for dimensionality reduction problems and anomaly detection problems (Baldi, 2012; Lopez Pinaya et al., 2020; Zhai et al., 2019).

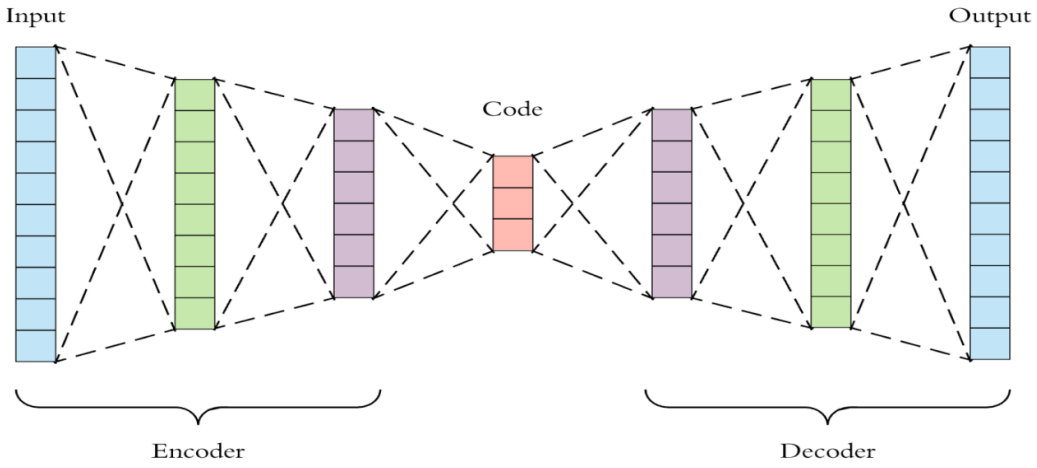


Figure 2.3 Autoencoder architecture

Source: adopted from (*Applied Deep Learning - Part 3: Autoencoders | by Arden Dertat | Towards Data Science*, n.d.)

2.1.9 TabNet Architecture

The TabNet model was first proposed by google cloud AI researchers in 2019 to efficiently apply deep neural network concepts to tabular data (Arik & Pfister, 2019). TabNet is high performance and interpretable deep neural network architecture which is highly suitable for tabular data. A decision tree-like model dominates tabular data-based problems. Their interpretability, fast learning ability, and efficiency for decision manifolds with approximately hyperplane boundaries make them dominant in tabular data tasks. On another side, deep neural network bases models become state-of-the-art models in many domains like computer vision, natural language processing, and speech processing because of their performance for large-size datasets, gradient decent-based ended-to-end learning mechanism, their ability in automatic feature engineering, and their ability in non-linear operations. But deep neural network architectures are not well studied for tabular data-based tasks. Therefore, the TabNet model comes with interpretability and high performance by combining the good side of decision tree-based models and deep neural network models. TabNet is based on a sequential attention mechanism (Vaswani et al., 2017).

TabNet architecture contains sequential multi-steps which pass inputs from one step to another as shown in figure 2.1. Steps are blocks of components that correspond to layers in other deep neural network architectures. Step mimics ensemble learning of the decision tree-like models. Each step has an equal vote for the final decision (classification or regression). Feature selection differs from one step to another (Arik & Pfister, 2019).

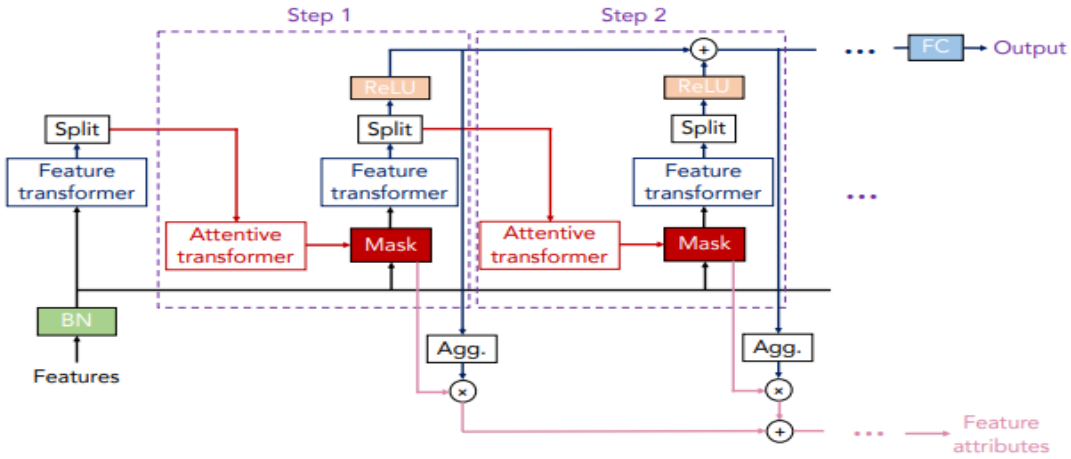


Figure 2.4 TabNet architecture

Source: adopted from (Arik & Pfister, 2019)

Each step mainly contains a Feature Transformer, an Attentive Transformer, and feature mask components.

Feature Transformer

A Feature Transformer is a block of components made up of a network of fully connected layers followed by a batch normalization layer followed by a gated linear unit as shown in figure 2.3. A Feature Transformer has more than 4 blocks of components. Some blocks are independent and other blocks are shared with other steps. According to the paper (Arik & Pfister, 2019) sharing of blocks leads to parameter-efficient and robust learning with high capacity. And the normalization with root five helps to stabilize learning by ensuring that the variance throughout doesn't change dramatically. The input for the Feature Transformer component is" B" batch size and "D" dimension features. The output of the Feature Transformer is "n(d)" the output decision and "n(a)" is the feature that is going to be an input for the next Attentive Transformer (Arik & Pfister, 2019).

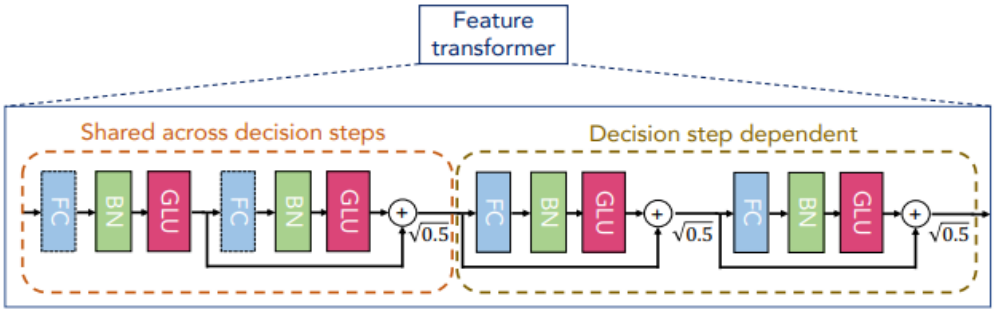


Figure 2.5 Feature Transformer architecture

Source: adopted from (Arik & Pfister, 2019)

Attentive Transformer

The Attentive Transformer contains a fully connected layer, Batch Normalization layer, Prior scale layer, and Sparsemax layer. n(a) inputs from the previous step Feature Transformer pass through the fully connected layer followed by Batch Normalizer layer then multiplied with the Prior scale layer as shown in figure 2.4. The advantage of the Prior scale layer is to determine the importance of features from the previous steps and to hold information about features in the current steps. As shown in equation 2.2 first Prior scale for all features is 1 means they have equal importance. And gamma or relaxation factor is the value to determine how to use features in each step. If we use the small value of gamma leads to the diverse use of features in the next steps and if we use the large value of gamma it leads to the use of the same features in the next steps. Sparsemax is like SoftMax but it contains sparsity (a lot of dimension zeros). The Sparsemax layer is used to drive the mask (Arik & Pfister, 2019).

$$P_i = \Pi_{j=1}^{i}\left(\gamma - 1m_j\right) \quad \text{eq. (2.2).}$$

$P_0 = 1$ all features are equally important.

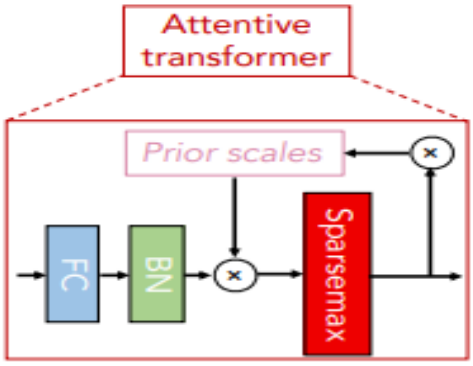


Figure 2.6 Attentive Transformer architecture

Source: adopted from (Arik & Pfister, 2019)

Mask

Each step has a distinct mask for instance-wise feature selection. The mask is driven from an Attentive Transformer block. The interpretability of the TabNet model is achieved by the mask component since it selects salient features in each step (Arik & Pfister, 2019).

TabNet let neural network model architecture be able to outperform the leading tree-based models such as XGBoost, Random Forest, and fast Gradient Boosting on several tasks. TabNet has local and global level interpretability. In general, the TabNet model is an interpretable high-performance model which uses a sequential attention mechanism to choose silent features in each step for tabular dataset-based tasks (Arik & Pfister, 2019).

2.1.10 Class Imbalance Problem

In a supervised classification task, there is a target or label column which contains the class values of the dataset. These class values are associated to feature values and become instances in the dataset. If the difference between the number of instances for each class value in the dataset becomes very large or if some class values are represented by many rows and other class values are represented by small rows, this problem is called a class imbalance problem[19] [20] [21]. The class imbalance problem in the dataset creates a bias in the classification accuracy. The model gets difficulty classifying the minority classes since it was trained with fewer numbers of instances and has less experience with those minority class values (Johnson & Khoshgoftaar, 2019). To handle this problem there are generally under-sampling and over-sampling techniques (Hasib et al., 2020; Johnson & Khoshgoftaar, 2019; Yu & Zhou, 2021). In under-sampling techniques, samples from the majority class are dropped randomly or based on some criteria. In over-sampling techniques, the size of the minority class is increased by sampling to duplicate some values from the minority class or artificially by adding some more representative instances for minority classes. Both of these techniques have their strength and weakness. For example, oversampling techniques add computation costs by maintaining the representation of the dataset. Whereas under-sampling techniques resulted in a loss of valuable information while they are efficient in computation cost (Yu & Zhou, 2021).

[19] https://machinelearningmastery.com/what-is-imbalanced-classification/
[20] https://link.springer.com/referenceworkentry/10.1007/978-0-387-30164-8_110
[21] https://towardsdatascience.com/class-imbalance-d90f985c681e

2.1.11 Min Max Scaler

Min Max Scaler is a type of scaler that put data values between 0 and 1. Usually feature values in the dataset may have different distributions. This different distribution of features affects the training time and the performance of deep neural network-based models. To handle this problem there are many strategies like standardization scaling, normalization, and so on (Ahsan et al., 2021; Gopal et al., 2015). The equation for the Min-Max Scaler is given below in equation 2.3.

$$z = \frac{x - min(x)}{max(x) - min(x)} \quad \text{eq. (2.3)}$$

2.1.12 Batch Normalization

Normalization or Standardization is part of the data preprocessing phases which transform data into some standard scales. Usually when working with neural network data normalization will help the gradient descent to converge easily and quickly resulting in the training process faster. Batch Normalization is a kind of Normalization that is applied to the input of each layer by using the mean and variance values of the current batch in use. The advantage of Batch Normalization is to reduce the internal covariant shift and make weight initialization easy (Ioffe & Christian Szegedy, 2015; Ioffe & Szegedy, 2015).

2.1.13 Model Interpretability

Model interpretability is a term to define how machine learning or deep learning models arrive at to result or the output after a certain learning process. Model interpretability refers to how transparent the Machine Learning and Deep learning models are to humans about the internal processes and how they arrive at the result or output. Traditional machine learning models like decision trees have interpretability properties, it is possible to trace the decision process by tracing all decision nodes of the tree. In rule-based systems also it is easy to identify the rule combination that gives the result. Currently, deep learning techniques attract researchers from a lot of domains since there is an expectation of performance improvement, especially with a large dataset. This assumption makes deep learning models state-of-the-art models for numerous tasks. But there is a tradeoff between model interpretability and high-performance achievement. That is why model interpretability or in general (Dosilovic et al., 2018; Samek et al., 2017)terms Explainable AI[22] become an issue nowadays (F. Xu et al., 2019).

[22] https://www.ibm.com/watson/explainable-ai

Model interpretability has mainly two advantages. First, the ML/DL model must be transparent in its inference process and result generation to be trusted by the user. It is difficult to deploy black box models in critical application areas such as intrusion detection systems with full of confidence. Second, it helps the developer to improve the model performance if the internal working process is transparent. Therefore, it is good practice to balance the interpretability of the model with its high performance.

2.2 Related Works

2.2.1 Machine Learning based Related works

(Dhaliwal et al., 2018) Proposed XGBoost-based intrusion detection model. NSL-KDD was used for training and testing the proposed XGBoost-based model. The performance of the model was evaluated with accuracy, precision, recall, and F1-score which are driven by the confusion matrix. The experiment results achieved 98.70% of accuracy, 98.41% of precision, 99.11% of recall, and 98.76% of F1-score for the proposed model. Although the research study archives high values of performance metrics on the proposed model, it uses an old dataset and the paper didn't use any dataset balancing technique.

(Deepak, 2020) Proposed XGBoost-based network intrusion detection system deployed using Apache spark big data platform. The performance of the proposed model was compared with ANN, random forest, and gradient-based intrusion detection models. The proposed model was trained and tested using the CICIDS2017 dataset. The proposed XGBoost-based model gave 99.8% accuracy on the validation set and nearly 99.1% accuracy on the test set from k-fold cross-validation. Although the research study used a recent dataset, it didn't mention any data preprocessing and class balancing techniques in the paper.

XGBoost and DNN-based network intrusion detection model was proposed as efficient intrusion detection model. XGBoost was used as a feature selector and DNN as a classifier model. The NSL-KDD dataset was used for model training and testing. Adma optimizer and SoftMax activation function was used for learning rate optimization and classification. The performance of the model was compared with SVM, logistic regression, and the Naive Bayes classifier. The performance of the proposed model was evaluated using accuracy, precision, recall, and F1-score metrics. 97.6% of accuracy, 97% of precision, 97% of Recall, and 97% of F1-score values were achieved by the proposed model.(Devan & Khare, 2020)

In (Othman et al., 2018) researchers introduced a new idea which was named spark-chi-SVM. The researcher used an SVM model classifier for classification, ChiSqSelector for feature selection and Apache spark big data platform. He used the KDD99 dataset for training and testing the model. Then he compared the Chi-SVM model classifier with the Chi-logistic regression classifier. The result shows a high performance of 99.55% AUROC and less training time.

The researcher (Malhotra & Sharma, 2019) applies 10 machine learning algorithms by using the NSL-KDD dataset. The algorithms were Bayes Net, Naïve Bayes, Random tree, Random Forest, J48, Bagging, PART, OneR, zeroR, and Logistic. The researcher tests these algorithms with feature selection and without feature selection methods. The result shows 99.83 % of high accuracy with the PART algorithm by using feature selection methods.

The researcher (Hippe et al., 2017) used 12 machine learning algorithms and use three datasets which are the CICIDS-2017, UNSW-NB15, and ICS dataset. The algorithms were logistic regression, gaussian naïve Bayes, k-nearest neighbor, decision tree, adaptive boosting, random forest, convolutional neural network, deep neural network, long short-term memory, gated recurrent unit, and simple recurrent neural network. The result showed that in all three datasets random forest model classifier shows high accuracy.

(Gao et al., 2019) By adjusting the proportion of training data and setting up multiple decision trees, the study investigated the overall detection effectiveness of several basic classifiers such as decision trees, random forests, ANNs, and adaptive ensemble voting algorithms. The research study used the NSL-KDD dataset. The result shows 85.2 % of accuracy for an adaptive voting algorithm.

(Maseer et al., 2021b) By applying ten machine learning algorithms which are ANN, KNN, DT, NB, RF, SVM, CNN, EM, k-mean and self-organizing map on the CICIDS-2017 dataset the research study tries to enhance the efficiency of detection. The result shows that KNN, NB, and DT show high detection accuracy.

2.2.2 Deep learning-based Related Works

(Farahnakian & Heikkonen, 2018)propose a deep Autoencoder-based intrusion detection model for network anomalies. KDD cup99 was used for model training and testing. The proposed model achieved a 94.71% of accuracy score.

(Haggag et al., 2020) by using three Deep learning algorithms which are long short-term memory, recurrent neural network, and multilayer perceptron intrusion detection system models are trained and tested using the NSL-KDD dataset. The research study also uses the Apache spark big data platform to minimize training time. The result shows LSTM algorithm shows 83.574 % of accuracy with two layers.

(Laghrissi et al., 2021) A Long Short-Term Memory algorithm (LSTM) was implemented for detecting intrusions with principal component analysis (PCA) and mutual information is used for dimensionality reduction and feature selection with the KDD dataset. The result shows that PCA-based LSTM active accuracy of 99.44 and 99.39 accuracies for binary classification and multi-class classification respectively.

In (Shone et al., 2018) Deep learning-based intrusion detection system was proposed with a novel idea, a non-symmetric deep Autoencoder (NSAE) for unsupervised feature learning. The proposed method was tested using the KDD99 and NSL-KDD datasets. The result shows 98.81 % of accuracy.

(Azizjon et al., 2020) The 1D-CNN-based intrusion detection system was proposed for improving existing IDSs. The study used the UNSW NB15 IDS dataset. For performance comparisons, the researcher uses a support vector machine and random forest classifiers. The result shows that 1D-CNN outperforms RF and SVM classifiers.

(Belarbi et al., 2022) Propose Deep Belief Network-based and multilayer perceptron-based intrusion detection models. The CICIDS2017 dataset was used for model training and performance evaluation. The model was applied to the multi-class classification problem. Under-sampling, oversampling, class weight strategy and sample weight strategy were tested to handle the class imbalance problem of the CICIDS2017 problem. F1-score, precision, and recall are used as performance metrics. MLP-based and DBN-based NIDS achieved an F1 score of 87.3% and 94% respectively using a combination of SMOTE and random under-sampler.

(K. Singh et al., 2021) The research study used the 1D-CNN model with the CICIDS2017 dataset to build an intrusion detection model. The model was trained first for multiclass classification and then, for binary classification. The result showed more than 99% of accuracy.

(Mendonça et al., 2021) Proposed a novel model which is a tree-CNN hierarchical algorithm with a soft root sign activation function. The new proposed model achieved 98 % of accuracy with less model training time. The research study uses the CICIDS2017 dataset for model training and performance evaluation.

Table 2.1 Related works summary

Ref.	**Datasets**	**Algorithms**	**Results**	**Gap identified**
(Dhaliwal et al., 2018)	NSL-KDD	XGBoost	98.7% of accuracy	Use of an old dataset
(Deepak, 2020)	CICIDS2017	XGBoost	99.1% of test accuracy	Use of Imbalance dataset.
(Devan & Khare, 2020)	NSL-KDD	DNN XGBoost for feature selection	97.6 % accuracy	Low accuracy when compared to other related works.
(Farahnakian & Heikkonen, 2018)	KDD99	Deep Autoencoder	94.71% accuracy	Low accuracy when compared to other related works.
(Lee et al., 2020)	Unsupervised	Sparse Autoencoder	99.0%	Complex and opaque model.
(W. Xu & Fan, 2022)	UNSW-NB15 and CICIDS2017	XGBoost, Logarithm Autoencoder for feature selection	95.11% and 99.92 %of accuracy for the UNSW-NB15 and CICIDS2017 datasets respectively.	Use of Imbalance dataset with a complex model.

2.2.3 Research Gaps Identified from Related Works

The following performance-related and issue consideration-related research gaps were identified by this research study from previous related works.

- **Low Detection performance: -** In classification tasks, it is difficult to minimize false positive and false positive rates. Previous research studies related to the intrusion detection domain specifically anomaly-based intrusion detection systems suffer from this problem(Sardana & Singh, 2018). Accurate detection is a critical criterion for intrusion detection systems since it is related to security. An intrusion detection system

with a high false positive and false negative rate in real-world deployment harms the defense mechanism (Aljanabi et al., 2021; Aziz et al., 2021). In general, low detection performance can be defined as high False positive or False Negative rates and low True Positive and True Negative rates.

- **Use of old, unrealistic, and outdated datasets: -** It is very expensive to prepare intrusion datasets for intrusion detection-related research since it needs a large infrastructure. Therefore, most research works use publicly available datasets. But most public datasets are old, outdated, and unrepresentative of real-world network traffic(Dwibedi et al., 2020; Hindy et al., 2020; Li & Liu, 2021a; Maseer et al., 2021b; Surana et al., 2017). And also, the dataset doesn't have a recent attack scenarios definition. In general, most of the intrusion detection-related research studies were conducted using these inappropriate datasets. Therefore, the research studies' proposed model was biased by these unsuitable datasets (Aljanabi et al., 2021; Aziz et al., 2021; Khraisat & Alazab, 2021).

- **Class Imbalance: -** In classification tasks, most of the time the class label value that is to be predicted (to be classified as an attack) is usually found with a smaller number of instances or rows in the dataset. This phenomenon is defined as a class Imbalance Problem. An unbalanced representation of each class in the dataset tends the model to bias since most ML/DL models work with the assumption of there is equal distribution of class values in the dataset. Most publicly available intrusion datasets are characterized by class imbalance problems. Normal traffic in the dataset is much higher in number than attack traffics (Ahmed et al., 2016; Khraisat et al., 2019; Thakkar & Lohiya, 2021).

- **Use of Improper Performance Evaluation Metrics: -** after training the Detection Model, it is important to evaluate the performance of the trained Model with appropriate performance evaluation metrics. Most research studies related to intrusion detection systems use accuracy as model performance evaluation metrics. But the problem of the use of accuracy metrics rises when the training dataset was an Imbalanced dataset (Shyla et al., 2021)(Thakkar & Lohiya, 2021). The accuracy metric is not a good performance evaluation metric for an imbalanced dataset.

- **Model training time: -** Although the deep learning-based intrusion detection model shows high performance they need a much longer time for training. This challenge

remains unsolved not only for deep learning-based IDS but also for other deep learning-based models in other domains (Lansky et al., 2021; Thirimanne et al., 2022a).

- **Unrealistic Intrusion Detection Models for Real-world Environment Deployment: -** Most intrusion detection models developed in the academic study couldn't work in a real-world environment since they are developed and tested in an unrealistic simulated environment (Al-Yaseen et al., 2016; Nayak et al., 2019; Thirimanne et al., 2022b).

- **Model Interpretability: -** Deep Learning models are complex and opaque in their internal working and structure. Humans cannot trace their behavior on how they update weight parameters, which features are selected, and which neurons are participated in computation since they are complex and deep. This is one of a drawback of deep learning models. If the internal working of the deep learning model is understood by the DL engineer it is easy to maintain and tune parameters to archive even better performance of the models (Alatwi & Morisset, 2021; Arya et al., 2019; Wang et al., 2020). With the expectation of performance improvement, researchers proposed deep and complex neural network-based intrusion models. But they didn't balance the interpretability of the model by maintaining its high accuracy.

- **Zero-day Attacks Detection: -** zero-day attack is a kind of attack that doesn't have a previous occurrence example or signature. As we discuss in chapter one cyber security attacks increasing rate and complexity grow from time to time. So, currently, there is a high probability of the occurrence of zero-day attacks. Usually, ML/DL algorithms especially supervised learning needs predefined attack definition for training. And for unsupervised learning also the model must have high performance to cluster attacks and normal traffic in the given dataset. But currently, existing ML/DL-based intrusion detection models are unable to detect zero-day attacks (Aziz et al., 2021; Hindy et al., 2020; Sardana & Singh, 2018).

One of the research gaps identified from previous related studies was they did not balance high detection performance with interpretability results in their proposed intrusion detection models.

The model's detection performance means its ability to classify all normal traffic as normal and all attack traffic as attacks in the given data set. Model performance evaluation should also be done with appropriate performance evaluation metrics. The network intrusion detection models proposed in previous related studies lack comprehensive performance evaluation by considering all scenarios or the performance of the models was not measured with appropriate model evaluation metrics (Ahmad et al., 2022; Aljanabi et al., 2021; Aziz et al., 2021; Dhaliwal et al., 2018; Elsadai et al., 2019; Gao et al., 2019; Halimaa & Sundarakantham, 2019; Khraisat et al., 2019; Malhotra & Sharma, 2019; Sardana & Singh, 2018; Shyla et al., 2021; Surana et al., 2017). In the performance evaluation, the model's detection performance is evaluated using true negative, true positive, false positive, false negative, and a combination of these metrics. But previous related research studies did not evaluate their model using these metrics. For example, a model may have high detection accuracy with a high false positive rate. It is difficult to generalize such kinds of models as high-performance models. The use of accuracy metrics for performance evaluation of the model that is trained using an imbalanced dataset was also another issue identified in previous related research studies.

With the expectation of performance improvement in the intrusion detection models complex and hybrid architectures were repeatedly proposed in the previous related works (Belarbi et al., 2022; Farahnakian & Heikkonen, 2018; Hodo et al., 2017; Laghrissi et al., 2021; Lansky et al., 2021b; Mendonça et al., 2021; Z. el Mrabet et al., 2019; Rajasekaran & Nirmala, 2020; Shone et al., 2018; Thirimanne et al., 2022a; Ugurlu & Dogru, 2019; W. Xu & Fan, 2022). Although these proposed studies improved the detection performance they could not maintain the interpretability features in their proposed model. The necessity of model interpretability in such kind of critical application area was not considered in the previous related works. If the proposed models were not interpretable it is difficult to get the trust of the end-user to be deployed in the real world environment. This research study focuses on how to build a network intrusion detection model that balances high detection performance with interpretable results by using TabNet encoder architecture.

CHAPTER THREE

METHODOLOGY

3.1 Research Design Overview

This research study was first started with an extensive literature review in the domain area. The research study conducts the literature review mainly for three reasons. The first one is to identify research gaps. A lot of research work has been done in this research domain area for decades. The performance of intrusion detection systems and other issues are improved from time to time because of those research studies in the past. But still, there are gaps and challenges in the intrusion detection systems research study domain area. So, the purpose of the literature review is to identify those gaps and challenges and to propose mitigation mechanisms through research studies. The second one is to understand the theoretical concepts for this research study. Usually, knowledge can be acquired through research. So, after identifying the research gap questions like what to do, how to do it, and when to do are answered with a literature review. The third one is the proposed solution is also derived by carefully studying other related research studies. After identifying the research gap then followed by developing an appropriate research study methodology.

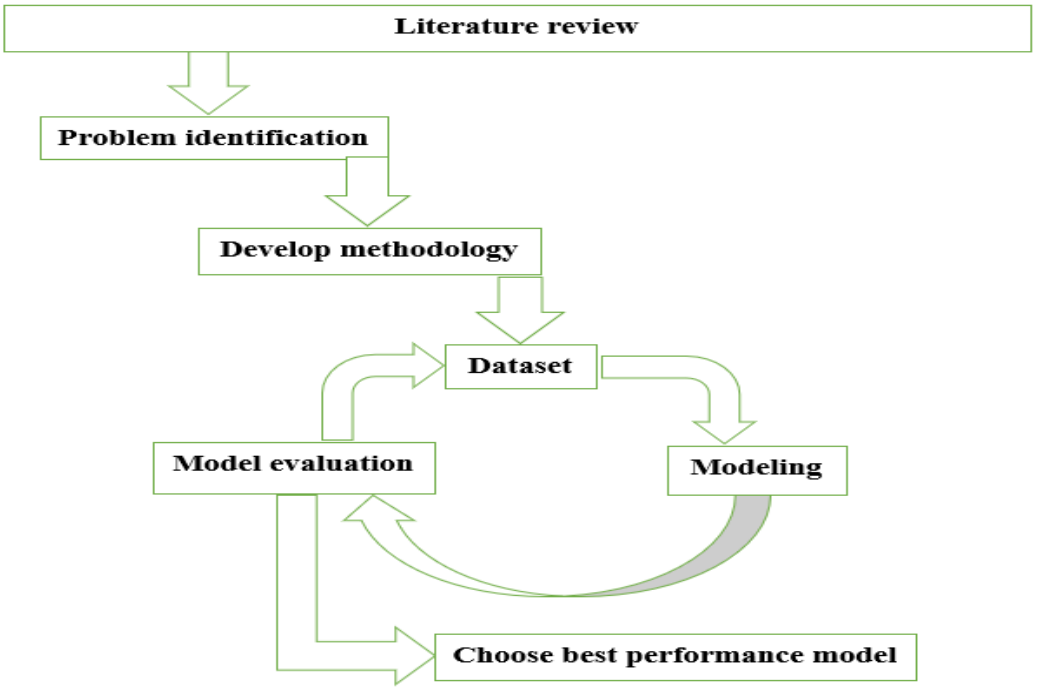


Figure 3.1 Research Design Diagram

3.2 Research Methodology

The research methodology is a framework that guides the process of the research study. This research study is a type of Quantitative Research study and needs clear steps to follow and to conduct experiments according to it. In the following sections, we will see the Research Methodology starting from dataset preparation.

3.3 Dataset and Data Preprocessing

Preparing network intrusion datasets by individuals is very costly, time-consuming, and needs complex infrastructure to simulate real-world network traffic. Some private datasets are also not shared with the general public due to privacy issues (Ferriyan et al., 2021). For this reason, most of the research works in the domain use a publicly available dataset. There are a lot of publicly available network intrusion datasets. Among these is KDD cup 99, NSL KDD, DEFCON, LBN, CDX, Kyoto CICIDS2017, and CICIDS2018 are some of them (Thakkar & Lohiya, 2020a). These datasets are different in their attributes, diversity of the network, extraction method, volume, and so on. (Sharafaldin et al., 2018) put 11 criteria to choose the best network intrusion detection dataset. These criteria are Labeling, Set of Features, Complete Traffic, Complete Network Configuration, Complete Capture, Attack Diversity, Anonymity, Available Protocols, Complete Interaction, Heterogeneity, and Metadata. Based on these criteria this research study chooses the CICIDS2017[23] dataset since it fulfills the above list of criteria. But most of the rest datasets do not fulfill these criteria (Guven et al., 2022; Maseer et al., 2021a; Tavallaee et al., 2009; Thakkar & Lohiya, 2020b; Umer et al., 2020).

3.3.1 Preprocessing

Handle Duplicate Values: - Duplicate values in the dataset must be removed before further preprocessing steps since they affect the quality of data and affect model generalization performance (Cheng, 2017).

Handle Missing Values: - Missing values are other challenges in data preprocessing tasks. The problem arises because Many Machine learning algorithms usually require numeric values, not null values. Therefore, the missing values must be handled properly even though there is no best missing value handling strategy (Choudhury & Kosorok, 2020; Emmanuel et al., 2021).

[23] https://www.unb.ca/cic/datasets/ids-2017.html

Handle Infinity Values: - Sometimes columns contain the “inf” symbol to represent infinity values. The existence of the infinity value in the dataset must be dropped or substituted by other values to avoid mathematical operation errors (Alasadi & Bhaya, 2017).

Categorically Feature Transformation: - Usually, neural network models are working well with numerical values instead of categorical values. So, if there is a feature with categorical values it must be transformed into numeric values using any appropriate techniques (Hancock & Khoshgoftaar, 2020; Potdar et al., 2017).

Feature Scaling: - Features in the dataset may have values with extremely different scales. One feature may have values between 1 and 100 and another feature may have values between 1000 and 100000. This different range of features creates a challenge for gradient descent to converge quickly and makes the training process very slow. Therefore, dataset value scaling improves this problem by putting all the dataset values in the same scale range (Ahsan et al., 2021).

Class Imbalance Problem Handling: - There is no best mechanism to handle class imbalance problems. For example, if we choose the under-sampling strategy, we lose information from dropped dataset instances. And also, if we chose oversampling strategy, it will add computation cost (Hasib et al., 2020; Johnson & Khoshgoftaar, 2019). So, class imbalance problems will be handled by trial and error.

3.4 Model

The proposed network intrusion detection model will be implemented using TabNet deep neural network architecture. For performance comparison, this research study also implements the state-of-the-art XGBoost machine learning algorithm and Autoencoder's state-of-the-art deep learning algorithm. In the following sections, the high-level architecture of the three algorithms will be revised.

3.4.1 TabNet model

TabNet is a deep neural network-based architecture for classification and regression problems that select important features using a sequential attention mechanism. TabNet is a stepwise model with the same number and kinds of components in each step. The TabNet model can have any number of steps. But when step size increases it tends to overfit although the performance of the model is improved. Every step has the following three main components (Arik & Pfister, 2019).

- Feature Transformer: - This component is also an architecture by itself holding three sub-components namely a fully connected layer, batch normalization layer, and gated linear unit activation. The main function of this component is to output every step decision and to produce transformed features for the next step Attentive Transformer component.
- Attentive Transformer: - this component also contains sub-components of fully connected layer batch normalizer prior scale and Sparsemax layer. Feature selection and model interpretability are achieved with this component.

- The Mask: - the mask is derived from an Attentive Transformer component and it helps the model to focus on important features and bring interpretability to the model

As mentioned in the paper TabNet is a high-performance explainable model for the structured dataset that beats the state-of-the-art ML and DL models on tabular dataset-based classification and regression tasks. Therefore, in this research study, we develop a TabNet-based network intrusion detection model and compare the performance of the model with state-of-the-art deep learning and machine learning classifier models.

3.4.2 XGBoost Model

For comparison purposes, XGBoost is selected from machine learning algorithms since it shows high performance on structured data-based tasks such as classifications and regressions in the past few years. XGBoost is a kind of ensemble decision tree algorithm based on boosting ensemble technique. XGBoost first builds a weak learner decision tree model and continues to build sequentially by minimizing the error from the previous weak learner model (Chen et al., 2018; Chen & Guestrin, 2016).

3.4.3 Autoencoder Model

Autoencoder is a deep neural network-based unsupervised learning architecture that mainly has three components namely encoder, decoder, and bottleneck. When training the Autoencoder-based model first the input feature with dimension d is fed to the encoder component. The encoder component enCodes the input features and reduces the input features to the minimum dimensional feature representation. The bottleneck component is the minimum dimension representation of the input feature. Finally, the decoder component tries to decode the minimum representation of features to restore the first input dimension. During the training phase, the reconstruction error is calculated and the weight is updated to minimize the reconstruction error (Baldi, 2012; Lopez Pinaya et al., 2020; Zhai et al., 2019).

3.4.4 Dataset Splitting

Splitting the dataset is a process of creating distinct sub-datasets from the original datasets usually into a training set, validation set, and test set. After splitting all the subsets are mutually exclusive subsets. There is no common instance in any of the two or the three subsets.

- Training set: - it is a set of data used to train the model.
- Validation set: - is a set of data used to validate the model during the training phase.
- Test set: - is a set of data used to test the performance of the model after training.

There is no standard dataset splitting mechanism. Most of the research studies use a 70:15:15 or 80:10:10 0r 75:10:15 splitting strategy for the training set, validation set, and test set sizes respectively (Rácz et al., 2021; Y. Xu & Goodacre, 2018).

3.4.5 Model performance evaluation metrics

Every ML/DL-based model must measure its performance by different evaluation metrics to implement the model in real-life uses. There are different kinds of performance evaluation metrics based on the task of the model. The task of the ML/DL-based IDS model is to classify each incoming traffic into benign or attack. ML/DL-based classifier models have the following performance measure metrics (Canbek et al., 2020; Ghori et al., 2020; M & M.N, 2015).

Confusion Matrix: - In a binary class classification problem a 2 by 2 table is prepared from the class labels. The table field is filled with true positive, true negative, false positive, and false negative value counts. This table is called a confusion matrix. Precession, recall, and other metrics can be derived from the confusion matrix.

True Positive (TP): - Defines the correct classification of attack packets as attacks.

True Negative (TN): - This value is the correct classification of normal packets as normal.

False Negative (FN): - This value is an incorrect classification of attack as normal.

False Positive (FP): - This value is an incorrect classification of normal as an attack.

Accuracy: - This is a simple metric that shows how often the model correctly classifies a given piece of data. It is given as the ratio of the number of correct predictions to the total predictions. Accuracy is a good metric when the data set is balanced, but not when the data set is unbalanced.

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN} \qquad eq.\ (3.1)$$

Precession: - Precision is defined as the number of true positives divided by the number of predicted positives. precision describes the number of correctly predicted cases that are truly positive.

$$precession = \frac{TP}{TP+FP} \qquad eq.\ (3.2)$$

Recall: - Recall is defined as the number of true positives divided by the total number of actual positives. Recall describes the actual number of positive cases that the model correctly predicted.

$$Recall = \frac{TP}{TP+FN}\ eq.\ (3.3)$$

F1 score: - The F1 score is a number between 0 and 1 and is the harmonic mean of precision and recall.

$$F1\ score = 2 * \frac{precession * Recall}{Precession + Recall}\ \ eq.\ (3.4)$$

3.5 Materials

3.5.1 Hardware Tools

The hardware used in this research study is HP ProDesk with 8GB RAM intel core I7 and Dell Laptop with 8 GB RAM processor intel core I5 processor.

3.5.2 Software Tools

- **Python Programming Language**: - Currently python is a popular object-oriented programing that is frequently used by the AI community. It has a lot of modules that are used for machine learning and deep learning developments.
- **Jupiter Notebook**: - Jupiter Notebook is a browser-based Code editor usually used for research purposes.
- **NumPy**: - NumPy is a popular and extremely fast module in the python programming language that is used for numeric operation-related problems.
- **Pandas**: - Panda's module is a python module that is built on top of NumPy. Panda's support for multi-dimension array operation capability makes it a preferable module to work to handle data.
- **Matplotlib**: - it is a data visualization module for python programing language.
- **Seaborn**: - it is also a visualization module built on top of the Matplotlib module.
- **Pytorch**: - it is an open-source library owned by Meta AI and used for machine learning and deep learning-related tasks.
- **Keras**: - Keras is a high-level API built on top of TensorFlow and used for deep learning-based operations.
- **Sklearn**: - It is an open-source module but not only a module it is also a framework used for machine learning-related tasks and data preprocessing tasks.

Table 3.1 Summary of used software tools with their version

Software tools	**Used version**
NumPy	1.23.2
Pandas	1.4.3
Matplotlib	3.5.3
Seaborn	0.11.2
TensorFlow	2.9.1
Sklearn	1.1.2
Pytorch	1.12.1
Jupyter notebook	6.4.12
XGBoost	0.9

CHAPTER FOUR
PROPOSED SOLUTION

4.1 Overview

The objective of this research study is to develop high performance and interpretable network intrusion detection model. To achieve this objective the research study design proposed a solution that is guided by the research methodology. The proposed solution mainly contains dataset selection, dataset preprocessing, model development, model training, and model evaluation tasks. As we discussed in chapter three dataset selection has a great impact on the proposed solution. Since one of the gaps identified in the previous related study is the use of old, unrepresentative and unrealistic datasets to train and test intrusion detection models (Khraisat et al., 2019; Sharafaldin et al., 2017; Thakkar & Lohiya, 2020a). By considering the above gap this research study conducted a literature review and selected suitable datasets from publicly available datasets. Based on the study CICIDS2017 dataset was selected as a training and testing dataset since it fulfilled the eleven characteristics of a good intrusion dataset that are Attack Diversity, Anonymity, Available Protocols, Complete Capture, Complete Interaction, Complete Network Configuration, Complete Traffic, Feature Set, Heterogeneity, Labelling, and Metadata according to the paper (Sharafaldin et al., 2017).

After collecting the dataset, maintaining the quality of the dataset is also an important task. By conducting explanatory data analysis, the research study discovered dataset quality issues such as duplicate values, infinity values, missing values, and unexpected values. The proposed solution incorporated a module to handle these issues. Categorical values encoding is also part of the proposed solution since the proposed classifier model work with only numerical values. Another issue that is related to the dataset was a class imbalance problem. The class imbalance problem is a condition in the dataset when there is no balanced representation of instance values for each class in the dataset[24]. This problem biased the classifier model since it has no balanced experience of learning for both minority class and majority class (Hasib et al., 2020). This proposed solution must handle this issue. Mainly there are three approaches to handling class imbalance problems namely oversampling, under-sampling, and class weight-based sampling (Yu & Zhou,

[24] https://machinelearningmastery.com/what-is-imbalanced-classification/

2021). There is no standard way to handle the class imbalance problem (Yu & Zhou, 2021). Therefore, the proposed solution experimented with all mechanisms to overcome the problem and select the suitable handling strategy according to the context of this research study.

It is expected that the Intrusion detection model to have high detection performance and interpretability to be deployed in the real world with trust and confidence. The proposed solution must hold these properties. According to the paper (Arik & Pfister, 2019) TabNet deep neural network architecture balance high performance with model interpretability. TabNet uses sequential attention mechanisms to select important features in every sequence of steps and to make decisions based on selected features. This research study used the encoder part of TabNet to make the network intrusion detection model interpretable with high performance. In the next coming section, we will see the encoder part of the TabNet architecture in detail. Finally, the solution design includes model performance evaluation mechanisms to assure that the developed model has a good performance. Figure 4.1 present the proposed solution in a flow diagram.

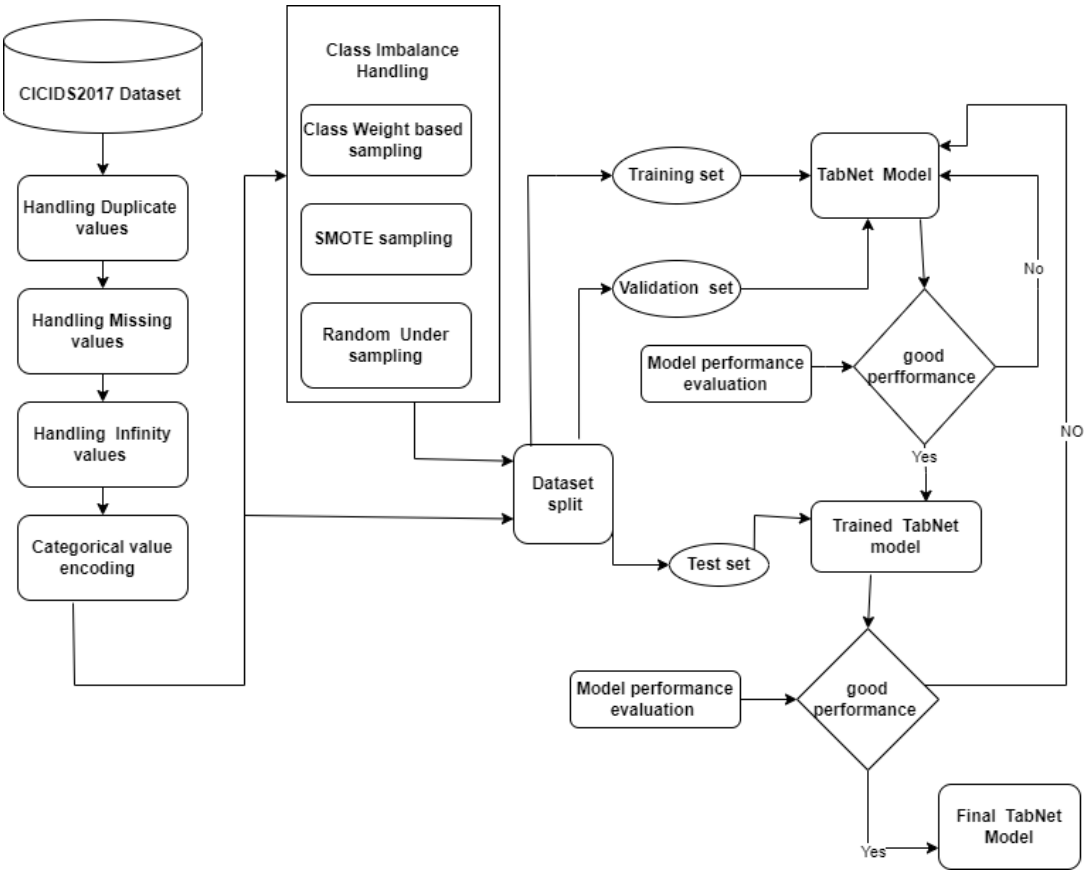


Figure 4.1 Proposed solution diagram

4.2 CICIDS2017 dataset

The CICIDS2017[25] dataset was released in 2017 by the Canadian Institute of cybersecurity. The dataset was collected for 5 consecutive days from a well-prepared real-world look-like network traffic infrastructure. The lab infrastructure contains the attack network profile, victim network, and network devices such as a router, firewall, switches, and nodes. The dataset was collected using a software tool known as CICFlowMeter. The dataset contains 2,830,743 rows and 79 columns or features in 8 separate CSV files. All the columns have numeric values except the label column. The label column contains 15 unique categorical values namely BENIGN, DoS Hulk, Port Scan, DDoS, DoS Goldeneye, FTP-Patator, SSH-Patator, DoS slow loris, DoS Slowhttptest, Bot, Web Attack-Brute Force, Web Attack-XSS, Infiltration, Web Attack-SQL Injection, and Heartbleed as shown in figure 3.2 with their percentage in the whole dataset(*IDS 2017 | Datasets | Research | Canadian Institute for Cybersecurity | UNB*, n.d.; Maseer et al., 2021a; Sharafaldin et al., 2017, 2018).

```
Class Value                    count          Percentage
benign.......................:  2273097 ..........  80.3
Bot..........................:  1966 ..........  0.07
DDoS.........................:  128027 ..........  4.52
DoS GoldenEye................:  10293 ..........  0.36
DoS Hulk.....................:  231073 ..........  8.16
DoS Slowhttptest.............:  5499 ..........  0.19
DoS slowloris................:  5796 ..........  0.2
FTP-Patator..................:  7938 ..........  0.28
Heartbleed...................:  11 ..........  0.0
Infiltration.................:  36 ..........  0.0
PortScan.....................:  158930 ..........  5.61
SSH-Patator..................:  5897 ..........  0.21
Web Attack - Brute Force.....:  1507 ..........  0.0
Web Attack - Sql Injection...:  21 ..........  0.0
Web Attack - XSS.............:  652 ..........  0.0
```

Figure 4.2 Attack types, counts, and percentage in the CICIDS2017 dataset

4.3 Dataset preprocessing

Explanatory data analysis in the data preprocessing phase gives the ability to understand the dataset properly. Based on the explanatory data analysis the research study identified the following lists of issues that affect the quality of the dataset.

[25] https://www.unb.ca/cic/datasets/ids-2017.html

Handling Duplicate values: - Based on the EDA there were 308381 Duplicated rows and 1 duplicated column. Therefore, the solution dropped these duplicated rows and columns.

Handling Infinity values: - Based on the EDA there were 2775 infinity values in the dataset. Therefore, these infinity values were changed to Null values for further processing.

Handling Missing Values: - There was a total of 3128 or 1.24% missing values in the dataset. All the missing values were from two columns. After studying the description of the two columns all the rows with missing values were dropped since they account for small proportions of the whole dataset.

Handling class Imbalance problem: - Usually classifier model works with the assumption of there is balanced or equal distribution of each class value in the dataset. So, the proposed solution must show the performance change of the proposed model with the class imbalance problem and after handling the class imbalance problem with the following mechanisms.

Random Under Sampling: - This is a technique in which randomly select and drop samples from the majority class.

Synthetic minority over-sampling technique (SMOTE): - Random oversampling technique Usually resulted in overfitting problems since it introduces duplicate values in the dataset. SMOTE technique improves this problem with data augmentation by creating synthetic data points from the original data points.

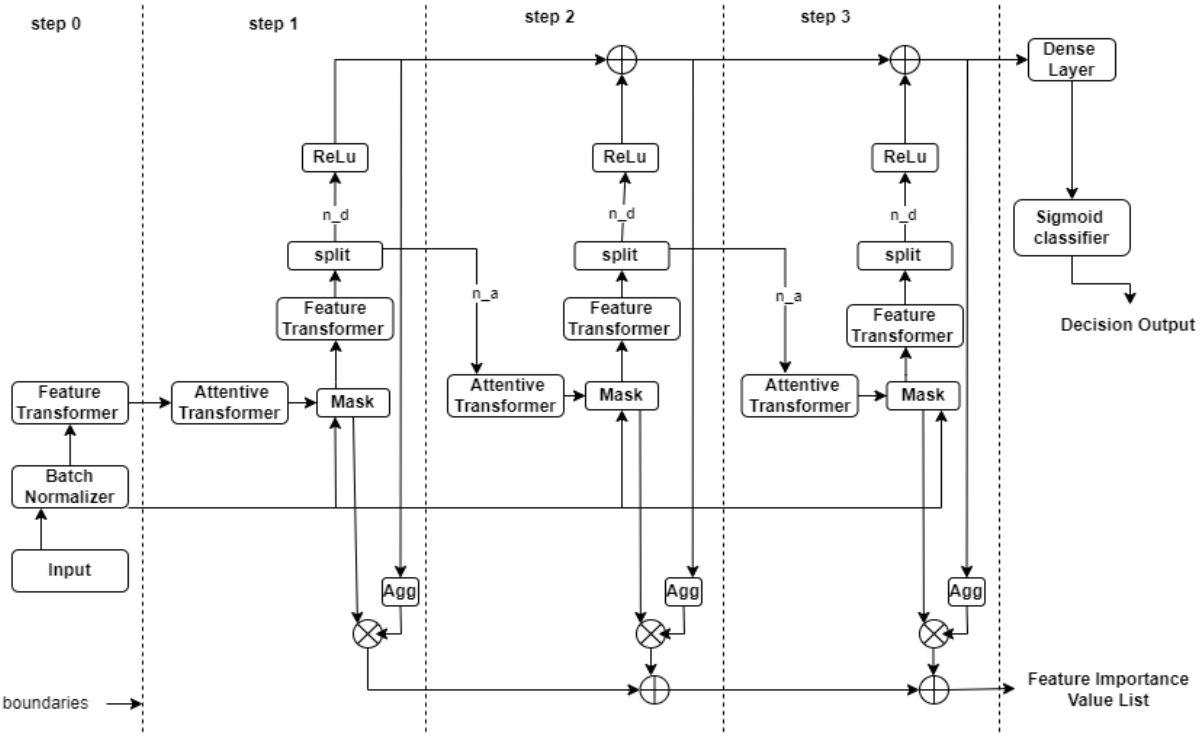
Categorical value Encoding: - CICIDS 2017 dataset only contains one categorical column which is the target or label column. There are 15 unique values for this column. But the task of this research study is binary classification. Therefore all 14 values were encoded as integer values "1" and the remaining "Benign" values were encoded as integer values "0" to represent intrusion and normal traffic respectively.

4.4 Proposed Model Architecture

TabNet is a deep neural superset of components architecture that works based on a sequence of attention mechanisms. TabNet architecture is mainly represented with "n" number of steps as shown in Figure 4.2 below. Feature selection and output prediction are based on the decision of these individual steps. According to the paper (Arik & Pfister, 2019), TabNet is a high-performance and interpretable architecture. The proposed solution includes the encoder part of the TabNet architecture to transform the input features into important feature representations and to get the importance score of those features locally

and globally. An important representation of the features helps the sigmoid classifier to classify the representation efficiently. Getting feature importance scores at the global level and the local level or instance-wise is used to show model interpretability.

As we mentioned before TabNet is a superset architecture that mainly contains a Feature transformer, an Attentive transformer, and the Mask. This research study used three-step TabNet encoder architecture. The Three-step usage is based on the paper's recommendation and decision after trial and error.



[26]

Figure 4.3 TabNet architecture

Source: adopted (Arik & Pfister, 2019)

The preprocessed feature is first fed to the Batch Normalizer layer. The purpose of this layer is to scale input values between 0 and 1. The normalized values help the gradian decent to converge quickly and make the training process faster (Gopal et al., 2015). Batch Normalization also speeds up the training by reducing internal covariant shift (Ioffe & Christian Szegedy, 2015).

[26] + sign in a circle represents element wise addition of two metrices.
× sign in a circle represents element wise multiplication of two metrices.
Agg symbol represents aggregation operation.

4.4.1 Feature Transformer

As shown in Figure 4.4 below Feature transformer is an architecture with a block of components. The block of components is composed of a Fully Connected (FC) layer, Batch Normalizer (BN), and Gated Linear Unit (GLU). The main task of this Architectural component is to transform the input data into a useful representation of feature values.

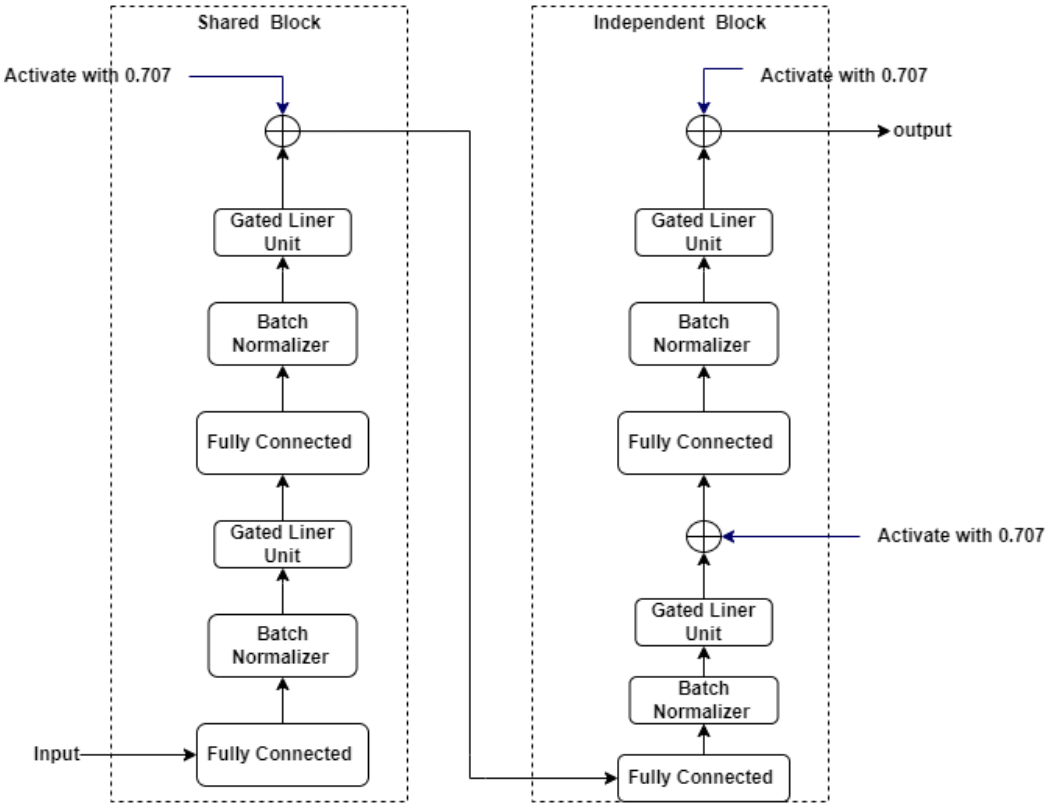


Figure 4.4 Feature Transformer

Source: adopted from (Arik & Pfister, 2019)

Feature transformer has two shared and two independent blocks of a component which are composed of a Fully Connected, Batch Normalizer, and Gated Linear Unit. The shared blocks are shared for every three steps. According to the paper (Arik & Pfister, 2019), sharing some blocks between steps help to share learning pattern among all decision steps and to make the learning parameter efficient and robust. Normalization with 0.707 also stabilize the learning process by controlling the dynamic change of variance of the feature values. Skip connection between the block of components was used to handle the performance degradation problem when the network becomes deep (He et al., 2015).

Gated Linear Unit: - As the name indicate Gated Linear Unit is a gate for the flow of information. Necessary information is filtered out with the GLU component (Yann N. Dauphin et al., 2016). The equation of GLU is given in equation 4.1 below.

$$GLU = \sigma(x) \cdot x \qquad \text{eq. (4.1)}$$

4.4.2 Attentive Transformer

The attentive transformer is also architecture by itself that contains a Fully connected Layer Batch Normalizer Layer and Sparsemax activation Layer as shown in Figure 4.5 below. The advantage of this component is to select salient features at each decision step and to drive the Mask component.

Entmax: - Entmax is a version of the sparsemax activation function that only differs by its ability to output more sparse probability. With entmax, it is possible to output zero values since it rounds up the smallest values to zero by keeping the probability of higher values(Martins & Astudillo, 2016).

Prior scale: - This term denotes how much a particular feature is used in the previous steps. The equation for the Prior scale is given in equation 4.2 where *M[j]* is the mask of the previous step and gamma (γ) is the relaxation parameter that determines the repetition usage of features in each step. When the value of gamma is close to one diverse features are used in different steps.

$$P[i] = \prod_{j=1}^{i} (\gamma - M[j]) \qquad \text{eq. (4.2)}$$

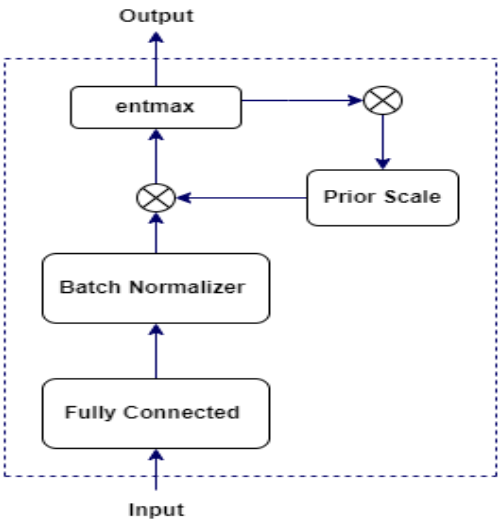


Figure 4.5 Attentive Transformer

Source: adopted from (Arik & Pfister, 2019)

4.4.3 Mask

Feature selection is achieved by the Mask component of each step. The mask is a sparse matrix of size *"B"* by *"D"* where *"B"* is batch size and *"D"* is the dimension of features. The matrix value of the Mask component at step *"i"* is given as equation 4.3. where *a[i-1]* represents the use of the previous step attention transformer output to create the Mask. *"hi"* is a trainable function in the Fully Connected Layer. *P[i-1]* is the previous step priority scale term and the equation is given in equation 4.1.

$$M[i] = entmax\big(P[i-1] \cdot h_i(a[i-1])\big) \qquad \text{eq. (4.3)}$$

Sigmoid: - sigmoid activation or logistic activation function is a type of activation which puts every input value between 0 and 1. For binary classification tasks, it is recommended to use the sigmoid activation function at the output layer. The equation of the sigmoid activation function is given below in equation 4.4.

$$sigmoid(x) = \frac{1}{1+e^{-x}} \qquad \text{eq. (4.4)}$$

Rectified Linear Unit (ReLu): - Relu is a non-linear activation function usually used in the hidden layer of neural network architecture as well as the output layer for regression tasks. The implementation of ReLu is very simple as shown in equation 4.5 if the value is less than zero it is replaced by zero otherwise it gives the value itself.

$$Re\,L \cup (x) = Max(x, 0) \qquad \text{eq. (4.5)}$$

4.5 Improvements from TabNet Encoder Architecture

An ablation study[27] was conducted to customize the TabNet encoder architecture for this research study. A factor of skip connection (Kaiming He et al., 2015), square root of 0.5 stability coefficient were examined with an ablation study.

The skip connection used in the TabNet architecture was adopted from the ResNet architecture (He et al., 2015b). The skip connection in ResNet was proposed to avoid performance degradation issues when the architecture gets deeper and deeper by introducing parameter reuse (He et al., 2015b). The TabNet architecture also uses skip connection to avoid the same performance degradation issue when multiple GLU blocks are stacked together (Arik & Pfister, 2019). But with the ablation study, skip connection

[27] In general, an ablation study is a set of experiments in which components of a machine learning system are removed/replaced in order to measure the impact of these components on the performance of the system.

in the proposed TabNet-based network intrusion detection model does not prevent performance degradation issues or has no impact on performance improvement. Therefore, the proposed TabNet-based intrusion detection model removes skip connection from the proposed architecture.

Sparsemax activation function was adopted by TabNet architecture in the attentive transformer component. The sparsemax activation function is a variant of softmax activation which was proposed by (Martins & Astudillo, 2016) to add sparsity (0 values) to the input features. Sparsity activation used in the TabNet architecture used to generate the Mask component. The mask component is a matrix with some zero values that are resulted from sparsemax activation. The mask component is used for feature selection in sequential steps since some zero values turn off values in the input features. Efficient feature selection was achieved by Entmax activation which has a high degree of sparsity than the previous sparsemax or softmax activation (Peters et al., 2019). This research study uses entmax activation for more efficient feature selection with a high degree of sparsity.

As shown in figure 4.6 below the encoder part of TabNet architecture is only used in this research study. Encoder models are usually used for dimensionality reduction or feature selection tasks. In this research study, the encoder part is used to select the most important features sequentially in different decision steps. The final task of the model is to classify the instance as intrusion or normal. So, a classifier network or model is required at the end of the encoder model. Based on the proposed solution dense layer and sigmoid layer are stacked with the encoder model to do the binary classification task.

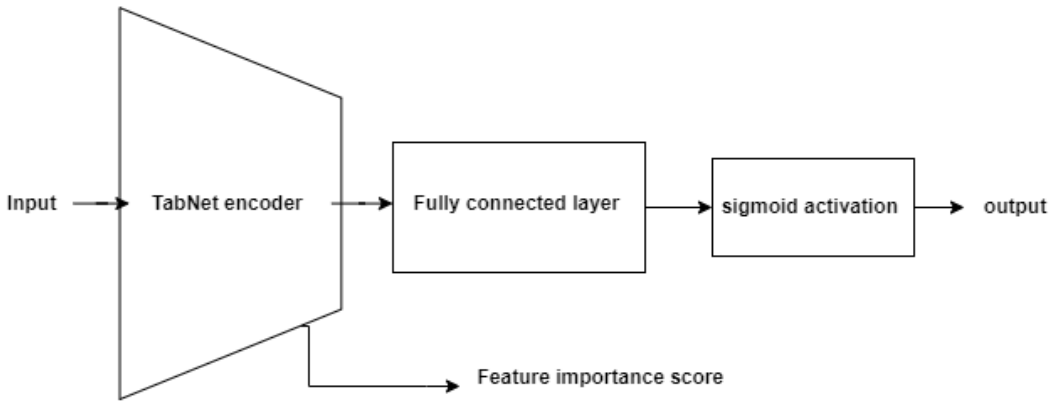


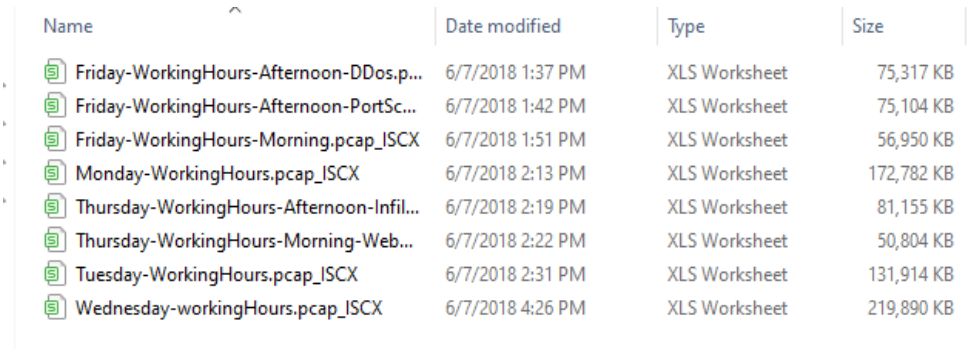
Figure 4.6 The proposed Network Intrusion Detection model high-level diagram

CHAPTER FIVE

IMPLEMENTATION

5.1 Preprocessing

Integrate the Dataset: - As mentioned in the previous chapter CICISD 2017 dataset is provided by the Canadian institution of cyber security with 8 separated CSV files. So, the first step is to merge these separated files by checking their consistency. After merging, the dataset contains 2,830,743 rows and 79 columns in a single CSV file.



Name	Date modified	Type	Size
Friday-WorkingHours-Afternoon-DDos.p...	6/7/2018 1:37 PM	XLS Worksheet	75,317 KB
Friday-WorkingHours-Afternoon-PortSc...	6/7/2018 1:42 PM	XLS Worksheet	75,104 KB
Friday-WorkingHours-Morning.pcap_ISCX	6/7/2018 1:51 PM	XLS Worksheet	56,950 KB
Monday-WorkingHours.pcap_ISCX	6/7/2018 2:13 PM	XLS Worksheet	172,782 KB
Thursday-WorkingHours-Afternoon-Infil...	6/7/2018 2:19 PM	XLS Worksheet	81,155 KB
Thursday-WorkingHours-Morning-Web...	6/7/2018 2:22 PM	XLS Worksheet	50,804 KB
Tuesday-WorkingHours.pcap_ISCX	6/7/2018 2:31 PM	XLS Worksheet	131,914 KB
Wednesday-workingHours.pcap_ISCX	6/7/2018 4:26 PM	XLS Worksheet	219,890 KB

Figure 5.1 Dataset in separated CSV files

Drop Duplicates: - The CICIDS2017 dataset has 308381 duplicate rows. Therefore, the dataset must be cleaned by dropping these duplicated values. And there is also one duplicate column called “Fwd Header Length”, so this duplicate column also must be dropped.

```
[30]  1 df.duplicated().sum()
17s

      308381
```

Figure 5.2 Duplicate count Code snippet

Handle Infinity Values: - CICIDS2017 dataset contains a total of 2775 “inf” values in two columns namely "Flow Bytes/s": 1211 and "Flow Packets/s": 1564. Therefore these “inf” values must be changed to null values for further processing.

```
[54]  1 nm.isinf(df.iloc[:,:-1]).values.sum()
2s

      2775
```

Figure 5.3. Infinity values count Code snippet

Handling Missing Values: - CICIDS2017 has a total of 3128 missing values in the 'Flow Bytes/s' and 'Flow Packets/s' columns of the dataset. The description of the two columns is presented in figure 2.1. This research study conducts an experiment to fill the null values by mean value, fill by the most frequent value in the column, fill by median value and drop missing values strategies to handle the missing values. According to the experiment dropping missing values strategy was selected because it gives a better accuracy value for the model performance when compared to other strategies and the ratio of missing values to the total size of the dataset is very less.

```
[23]  1 df['Flow Bytes/s'].describe()

count    2.520798e+06
mean     1.410707e+06
std      2.657084e+07
min     -2.610000e+08
25%      1.194308e+02
50%      3.715038e+03
75%      1.071429e+05
max      2.071000e+09
Name: Flow Bytes/s, dtype: float64
```

Figure 5.4 (a). Description of "Flow Bytes/s" feature column

```
[24]  1 df['Flow Packets/s'].describe()

count    2.520798e+06
mean     4.729188e+04
std      2.026366e+05
min     -2.000000e+06
25%      2.023326e+00
50%      6.974224e+01
75%      1.785714e+04
max      4.000000e+06
Name: Flow Packets/s, dtype: float64
```

Figure 5.4 (b). Description of "Flow packets/s" feature column

Categorically Features Encoding: - The only categorical column in the dataset is the "Label" column. As previously mentioned, the column contains 15 unique values. In the label encoding step first, these 15 unique values are grouped into two classes since the task of the research study is binary classification. All attack types are represented by integer value *1* and all benign traffics are represented by integer value *0*.

```
1 df["Label"].unique().astype(str).tolist()
```

```
['BENIGN',
 'DDoS',
 'PortScan',
 'Bot',
 'Infiltration',
 'Web Attack \x96 Brute Force',
 'Web Attack \x96 XSS',
 'Web Attack \x96 Sql Injection',
 'FTP-Patator',
 'SSH-Patator',
 'DoS slowloris',
 'DoS Slowhttptest',
 'DoS Hulk',
 'DoS GoldenEye',
 'Heartbleed']
```

Figure 5.5 (a). Label column categorical values

```
1 clean_df["Label"].value_counts()
```

```
0    2095057
1     425741
```

Figure 5.5 b). Label column Distinct values

After all missing values, duplicate values, and infinity values are handled, the dataset resulted in 2520798 rows and 78 columns.

Handle Class Imbalance Problem: - As shown in figure 4.7, (a) the CICIDS2017 dataset is affected by the class imbalance problem. This problem challenges most machine learning algorithms since they are designed with the assumption of there is equal distribution of class values in the dataset (Johnson & Khoshgoftaar, 2019; Yu & Zhou, 2021). This problem leads to poor performance of the classifier model, especially for the minority class. So, this research study conducts three experiments with three class imbalance handling strategies namely under-sampling, oversampling, and class weight-based sampling. All sampling techniques can be found in the "Imblearn" python module. Figure 4.6 shows a Code snippet for sampling the dataset using different techniques.

```
1 from imblearn.over_sampling import RandomOverSampler
2 ros = RandomOverSampler(random_state=0)
3 X_resampled, y_resampled = ros.fit_resample(x, y)
```

Figure 5.6 (a). Code snippets for random oversampling

```
from imblearn.over_sampling import SMOTE
sm = SMOTE(random_state=0)
resampled_x, resampled_y= sm.fit_resample(x,y)
```

Figure 5.6 (b). Code snippets for SMOTE sampling

```
from imblearn.under_sampling import RandomUnderSampler
rus = RandomUnderSampler(random_state=0)
x_resampled, y_resampled = rus.fit_resample(features, y)
```

Figure 5.6 (c). Code snippets for under-sampling

In this research study, the dataset under-sampling technique was selected for the sampling technique of its optimality for training time, size, and impact on the performance of the model. Figure 4.7 (b) shows the percentage of attack instances and normal instances in the dataset using a pie chart.

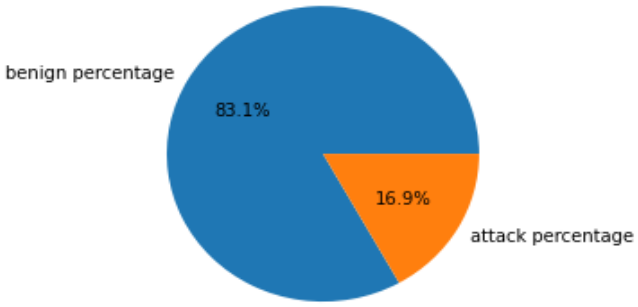


Figure 5.7 (a). Pie chart of Label column values with class Imbalance problem

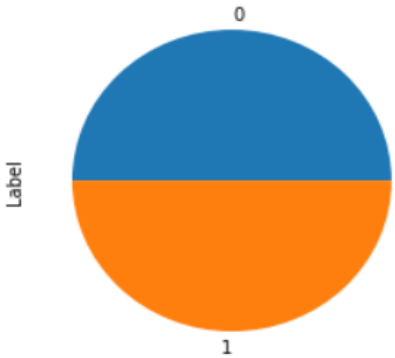


Figure 5.7 (b). Pie chart of Label column after class imbalance problem handled

Scaling the Dataset: - Neural network-based models work well for feature values ranging from 0 to 1 or from -1 to 1. So, it is good practice to scale down all feature values between these ranges to make the training process fast and robust. This research study uses the min max scaler feature scaling technique (Alasadi & Bhaya, 2017). The implementation is already present in the sklearn module "Preprocessing" class.

```
from sklearn import preprocessing
min_max_scaler = preprocessing.MinMaxScaler()
X = min_max_scaler.fit_transform(x)
features=pd.DataFrame(X)
```

Figure 5.8 Code snippet for feature values scaling

Dataset Split: - as we mentioned earlier the dataset must be split into a distinct training set, validation set, and test set to train, validate and test the developed model respectively. This research study first uses the 80:10:15 ratio dataset splitting technique for the training set, validation set, and test set respectively. The ratio was selected with trial and error since it shows good results from previously related research studies (M. A. el Mrabet et al., 2021; Khraisat et al., 2019; Surana et al., 2017). For splitting the dataset this study uses the train test split method of the sci-kit learn module.

```
from sklearn.model_selection import train_test_split
train_x, test_x, train_y, test_y = train_test_split(x, y,test_size=0.15,random_state=42,stratify=y)
train_x,valid_x,train_y,valid_y= train_test_split(train_x, train_y,test_size=0.1, random_state= 42,stratify=train_y)
```

Figure 5.9 Code snippet for training, validating, and testing data split

But this splitting technique has its drawback. If we run the train test split method of the sklearn method again and again by changing the random state value, we will get different accuracy results when we train the model with those splits. The problem arises since the distinct split into train and test sets prohibit the model from experiencing all possible scenarios during the training time. The solution for this is cross-validation. The cross-validation technique lets the model have all experience on the full dataset. This research study uses the k-fold cross-validation technique the implementation is already found in the sklearn module.

5.2 Model Development

5.2.1 TabNet Model

There is no official implementation of the TensorFlow-based TabNet model even though it was first released by Google. Instead, there is PyTorch Implementation of the TabNet model. The implementation is Sklearn compatible[28]. And also, the research study implements each TabNet component using python modules such as pandas and NumPy for ablation[29] study.

Network and fit parameters for PyTorch TabNet Model implementation

- n_d: - it is an integer value of the width of the decision prediction layer. From the paper, it is advised to use the range between 8-64. A large value gives the model more performance with the risk of overfitting.
- n_a: - it is an integer value of the width of the attention embedding for each mask. According to the paper, it is advisable to use n_a=n_d for a good result.
- n_steps: - an integer value of the Number of steps in the architecture. According to the paper, it is advisable to use values between 3 and 10.
- gamma: - it is the float value coefficient used to determine the feature usage in the mask. The value ranges from 1.0 to 2.0. If the value is close to 1 the next step mask will use different features.
- n_independent: - integer value to represent the number of independent Gated Linear Units layers at each step. Usual values range from 1 to 5.
- n_shared: - integer value to represent the number of shared Gated Linear Units at each step Usual values range from 1 to 5.
- mask_type: - string values of Mask type for the masking function to use for selecting features.
- eval_metric: list of metrics strings for model performance evaluation.
- max_epochs: - integer value for a maximum number of epochs for training.
- batch_size: - integer value to determine the number of examples per batch. Large batch sizes are recommended.

[28] https://github.com/dreamquark-ai/tabnet

[29] Ablation is removing of certain components in complex system in order to study performance change. https://en.wikipedia.org/wiki/Ablation_(artificial_intelligence)

- virtual_batch_size: - integer value to determine the size of the mini-batches used for "Ghost Batch Normalization".

Hyperparameter tuning

The TabNet model has a relatively small number of hyperparameters to be tuned to achieve higher performance. But it is difficult to tune every parameter manually. So, the research study uses the RandomSearchCV hyperparameter tuning method from the Sklearn module. Table 4.1 shows all possible parameter values for the TabNet model that are tuned to get maximum accuracy results.

Table 5.1 Summary of hyperparameter values list for the experiment

hyperparameter name	**Values list**
n_d	8, 16,32,48,64,77
n_a	8, 16,32,48,64,77
epoch	20, 50, 70,100
gamma	0.9, 1.2, 1.5
mask type	entmax, Sparsemax
Learning rate=	0.002,0.02, 0.05
n_steps	3,5,6,7
n_independent	1,2,4
n_shared	1,2,4

```
1 tb_cls.explain
```

```
<bound method TabModel.explain of TabNetClassifier(n_d=48, n_a=48, n_steps=7, gamma=1.3,
cat_idxs=[], cat_dims=[], cat_emb_dim=1, n_independent=2, n_shared=2, epsilon=1e-15,
momentum=0.02, lambda_sparse=0.001, seed=1, clip_value=1, verbose=1, optimizer_fn=<class
'torch.optim.adam.Adam'>, optimizer_params={'lr': 0.02}, scheduler_fn=<class
'torch.optim.lr_scheduler.StepLR'>, scheduler_params={'step_size': 50, 'gamma': 1.2},
mask_type='entmax', input_dim=None, output_dim=None, device_name='auto')>
```

Figure 5.10 TabNet model configuration randomly selected

5.2.2 Autoencoder Model

As we discussed before Autoencoder architecture consists of three main components namely encoder, bottleneck, and decoder. Both the encoder and decoder layers consist of the same number of nodes in most cases. In practice, Autoencoders are created from multiple layers where the outputs of the preceding layer are connected to inputs of the following layer. Learning tasks are accomplished by capturing the most significant features of training data at the hidden layer. The identity Function is one of the drawbacks of Autoencoder, it occurs when nodes of the hidden layer are directly copied at the output node. To get rid of the identity function problem Denoising Autoencoder (DAE) was proposed by (Vincent et al., 2008). for this research study Denoising, Autoencoder is implemented.

Denoising Autoencoder is a type of Autoencoder that receives the corrupted output instance and is trained to recover the original uncorrupted input as its output. Denoising Autoencoder does not replicate input at output rather it captures the most significant features. Inputs are randomly corrupted and the Denoising Autoencoder is expected to reconstruct pure original input as its output aims to produce a robust representation. Some advantages of Denoising Autoencoder are high capability of dimensional reduction, low overfitting, and its non-identity function. components of the proposed Denoising Autoencoder model are listed below.

Gaussian Noise: - Gaussian noise layer is carried out as a regularization layer and it is only activated at the training time. This layer randomly corrupts the input layers of the neural network. This behavior of the gaussian noise layer prevents the model from overfitting and identity function.

Activation function: - exponential linear unit and the Tanh activation function were tested in the model layers.

Optimizer: - The optimization method helps the algorithm to achieve the optimum solution for nonlinear problems, a better success rate, and frequently and quickly update parameter values.

Reconstruction Error: - DEA corrupts and encodes the input into a compressed representation at the bottleneck layer then recover the original data at the decoding layer. The loss function explains the difference between the original input and the output at the

decoded layer. If the loss function of a model is small it implies the Autoencoder is well performed because the original input and predicted output are almost similar.

```
autoencoder=tf.keras.models.Sequential([
  tf.keras.layers.Dense(input_dim0, activation="elu",input_shape=(input_dim0,)),
  tf.keras.layers.Dense(input_dim1, activation="elu",input_shape=(input_dim1,)),
  tf.keras.layers.Dense(input_dim2, activation="elu",input_shape=(input_dim2,)),
  tf.keras.layers.Dense(input_dim3, activation="elu",input_shape=(input_dim3,)),
  tf.keras.layers.Dense(input_dim4, activation="elu",input_shape=(input_dim4,)),
  tf.keras.layers.Dense(input_dim5, activation="elu",input_shape=(input_dim4,)),
  tf.keras.layers.Dense(input_dim6, activation="elu",input_shape=(input_dim4,)),
  #tf.keras.layers.Dense(input_dim7, activation="elu",input_shape=(input_dim4,)),
  tf.keras.layers.GaussianNoise(0.01),
  tf.keras.layers.Dense(input_dim7, activation="elu"),
  #tf.keras.layers.Dense(input_dim7, activation="elu"),
  tf.keras.layers.Dense(input_dim6, activation="elu"),
  tf.keras.layers.Dense(input_dim5, activation="elu"),
  tf.keras.layers.Dense(input_dim4, activation="elu"),
  tf.keras.layers.Dense(input_dim3, activation="elu"),
  tf.keras.layers.Dense(input_dim2, activation="elu"),
  tf.keras.layers.Dense(input_dim1, activation="elu"),
  tf.keras.layers.Dense(input_dim0, activation="elu"),
])
```

Figure 5.11 Configuration of Denoising Autoencoder model

The Denoising Autoencoder model was developed with encoder and decoder networks with 7 hidden layers in each network. The number of input neurons at the encoder network and the number of output neurons at the decoder layer were 77. The number of neurons decreased while the layer gets deeper down. A gaussian layer is applied before the bottleneck to randomly corrupt the inputs. The Tanh activation function and Adams optimizer are applied in every layer of the network. After configuring the Denoising Autoencoder with the above parameters, it was trained for different 100 epoch sizes with a train set and validated using the validation set. This configuration of the model is based on previous related works (Lee et al., 2020; A. Singh & Jang-Jaccard, 2022; W. Xu & Fan, 2022).

Table 5.2 Initial configuration of the Autoencoder model

hyperparameter name	**Values used**
Number of hidden layers	7
epoch	100
Learning rate	10e-7
Bach size	9056
Optimizer	Adams
Activation function	ELU

5.2.3 XGBoost Model

XGBoost algorithm is usually chosen for classification and regression tasks because of its speed and high performance. But besides these advantages, it requires proper parameter setups. And also, it has more parameters to be tuned when compared to other models. The following list of parameter values was tuned to achieve high accuracy values.

Table 5.3 Summary of XGBoost model hyperparameter value lists to be tuned

hyperparameter name	**Values list**
Booster	Gbtree, gblinear
Max_depth	3,4,5,6,8,10,12,15
min_child_weight	1,3,5,7
gamma	0.1,0.2,0.3,0.4
Colsample_bylevel	0.3,0.4,0.5,0.7
Learning_rate	0.05,0.10,0.15,0.20,0.25,0.30

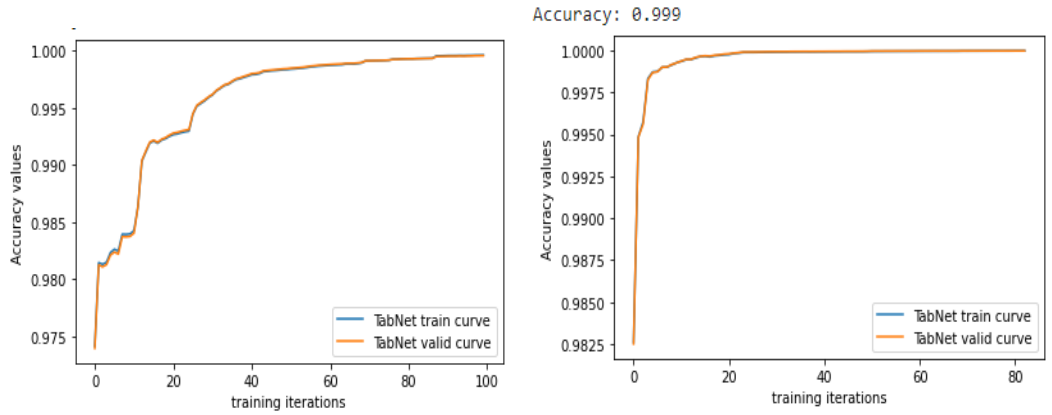
CHAPTER SIX

RESULTS AND DISCUSSION

6.1 Results

6.1.1 Performance Results

Figure 6.1. and Figure 6.2. shows the accuracy curve and loss curve of the TabNet model respectively for different dataset sizes. Based on the figure and the experiment result from the TabNet model achieve a 99.9 % of high accuracy value for a balanced dataset experiment. Table 6.1. shows the model configuration setup after iterative tuning to achieve this maximum accuracy. Figure 6.3. also shows a confusion matrix with True Positive, False Positive, False Negative, and True Positive entry values.



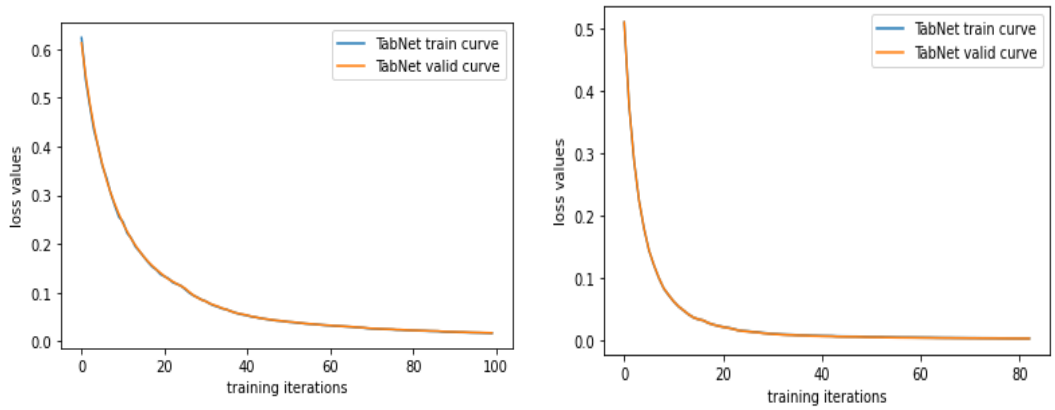
(a) Accuracy curve (original dataset)

(b) Accuracy curve (balanced dataset)

(c) Accuracy curve (small dataset)

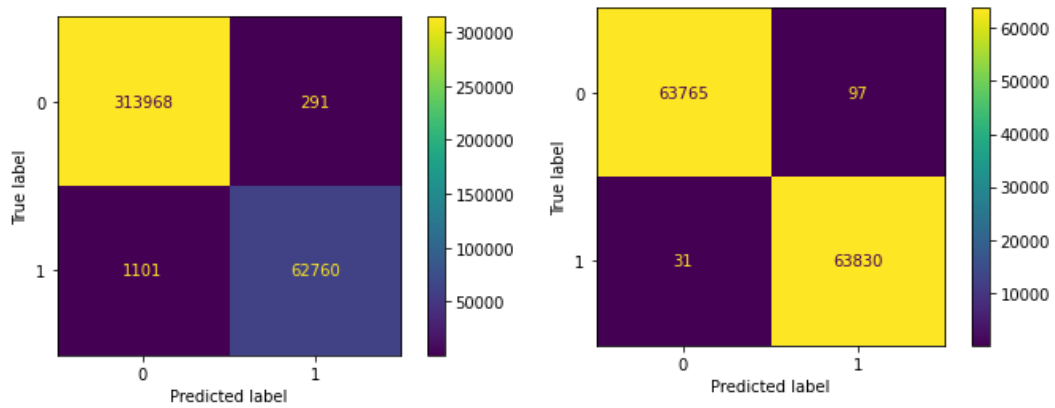
(d) Accuracy curve (large dataset)

Figure 6.1 (a), (b), (c), (d). Accuracy curves of Tabnet-based Model.

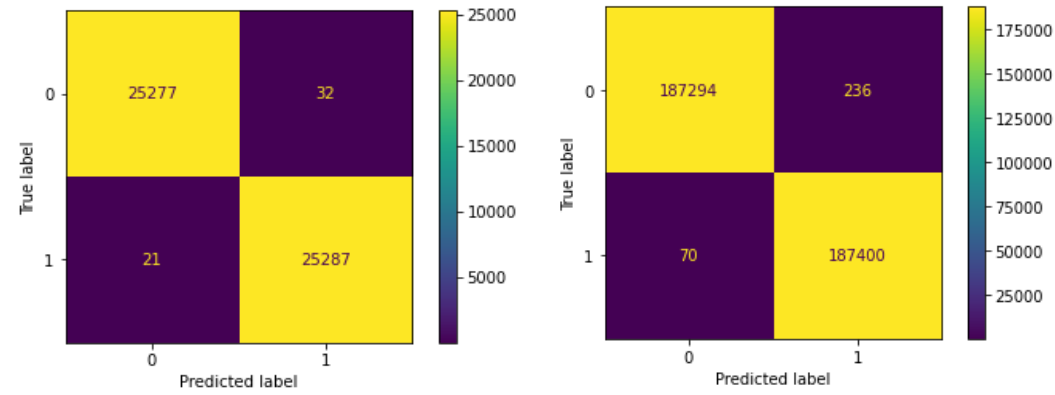


(a) loss curve (unbalanced dataset) *(b) Loss curve(balanced dataset)*

Figure 6.2 (a), (b) . loss curves of Tabnet-based Model.



(a) Confusion Matrix (unbalanced dataset) *(b) Confusion Matrix(balanced dataset)*



(a) Confusion matrix (small dataset) *(b) Confusion matrix(large dataset)*

Figure 6.3 (a),(b).Confusion Matrix of Tabnet-based Model.

Table 6.1. Optimal hyperparameter values for the good accuracy result

hyperparameter name	Optimal value
n_d	48
n_a	48
n_sahred	2
n_independent	2
epochs	80
gamma	1.5
n_steps	3
mask_type	entmax

```
from sklearn.metrics import precision_recall_fscore_support
from sklearn.metrics import accuracy_score
preds = tabnet_model.predict(test_x)
test_acc=accuracy_score(test_y, preds)
a,b,c,d=precision_recall_fscore_support(test_y, preds, average='weighted')
print(f"Accuracy  :",test_acc)
print(f"Precesssion  :",a)
print(f"Recall  :",b)
print(f"F-1 score  :",c)
```

```
Accuracy  : 0.999184
Precesssion  : 0.9991843913837575
Recall  : 0.999184
F-1 score  : 0.9991839998978963
```

Figure 6.4 Sample Precession, Recall, and F1- score values for TabNet-based model.

Based on the experiment result from the XGBoost model and Autoencoder-based model also shows closely high Accuracy, Precession, Recall, and F1-score values for the experiment conducted using different dataset sizes and balanced and imbalanced dataset. Using the RandomSearchCV hyperparameter tuning technique XGBoost gets its optimal hyperparameter combination value as shown in Table 6.2 below. The Autoencoder model also shows good performance results based on previous related studies on hyperparameter configurations (Lee et al., 2020; A. Singh & Jang-Jaccard, 2022; W. Xu & Fan, 2022).

Table 6.2. Hyperparameter values for the good accuracy result of the XGBoost model.

hyperparameter name	Optimal value
colsample_bytree	0.7
gamma	0.1
Learning_rate	0.25
Max_depth	5
min_child_weight	3

Table 6.3 shows a summary of performance results for the three models with four kinds of performance evaluation metrics. From the result, we can see that the proposed TabNet model outperforms most of the criteria.

Table 6.3 Summary of performance values for the three models

Criteria	**Model name**	**Accuracy**	**Precession**	**Recall**	**F1 score**
Imbalanced Dataset	TabNet%	99.62%	98.14%	99.63%	98.88%
	Autoencoder	99.63%	98.28%	99.54%	98.90%
	XGBoost	99.61%	98.17%	99.49%	98.83%
Balanced Dataset	TabNet	99.98 %	99.94%	99.94%	99.98%
	Autoencoder	98.43%	97.25%	99.68%	98.45%
	XGBoost	99.90%	99.95%	99.85%	99.90%
10,000 Balanced Samples	TabNet	99.90%	99.92%	99.87%	99.90%
	Autoencoder	99.63%	99.73%	99.73	99.63%
	XGBoost	99.82%	99.84%	99.79%	99.82%
100,000 Balanced samples	TabNet	99.52%	99.47%	99.47%	99.47%
	Autoencoder	98.77%	98.33%	99.22%	98.77%
	XGBoost	99.90%	99.87%	99.92%	99.79%
3,000,000 Balanced samples	TabNet	99.92%	99.92%	99.92%	99.92%
	Autoencoder	99.46%	99.28%	99.65%	99.46%
	XGBoost	99.23%	98.97%	99.49%	99.23%

Table 6.3 summarizes the performance evaluation result of the TabNet, Autoencoder, and XGBoost network intrusion detection models for the CICIDS2017 unbalanced dataset and different sample sizes of the balanced CICIDS2017dataset. The proposed TabNet-based

model outperforms the Autoencoder and XGBoost-based models with high accuracy values, as shown in the second, third, and fifth rows of Table 6.3. These high-accuracy results are achieved due to the ability of the proposed TabNet-based network intrusion detection model to encode the input feature values to the most important feature values that are very useful for the detection decision.

The first three rows of Table 6.3 show the results of the three-model experiment for the unbalanced CICIDS2017 dataset. The unbalanced CICIDS2017 dataset contains 2,095,057 instances of normal network traffic and 425,741 instances of attack network traffic. One research gap identified from previous related research studies is the use of accuracy metrics for unbalanced dataset-based experiments. It is difficult to generalize the performance of the model with the evaluation of accuracy metrics (ratio between the correct prediction and the total prediction) since it is biased by the occurrence of the majority class in the data set. Therefore, the F1 score is a good evaluation metric for the imbalanced dataset that contains precession and recall information. According to the result, the Autoencoder model shows 0.02% more accuracy than the proposed TabNet model. This is because the Autoencoder model's effectiveness in learning the pattern in the dataset by sampling the dataset based on the weight of the class values makes it resistant to class imbalance problems (Ger & Klabjan, 2019; Tomescu et al., 2021; Zhang & Bom, 2021). The TabNet model also shows the F1 score value almost close to the Autoencoder model, which is a high and interpretable result. But the result of the Autoencoder model is not interpretable.

The third three rows of Table 6.3 show the result of the experiment of 100,000 balanced sample instances of the CICIDS2017 dataset with the three models. According to the result, the XGBoost model achieves more than a 0.38% of accuracy value over the proposed TabNet-based model. This is because the XGBoost model has high performance on small to medium-sized datasets as mentioned in the original article (Chen & Guestrin, 2016). The proposed TabNet-based model also achieves an accuracy value relatively close to the XGBoost model with locally interpretable results that is missing feature from the XGBoost model. The overall result of the proposed TabNet-based intrusion detection model shows high-performance evaluation results for different performance evaluation metrics in the conducted experiments.

6.1.2 Model Interpretability Results

Figure 6.5, shows the feature importance score values for all 77 features of the dataset. Based on the figure, for example, we can conclude that feature 8 has high importance score and it has a high impact on the decision of the output globally.

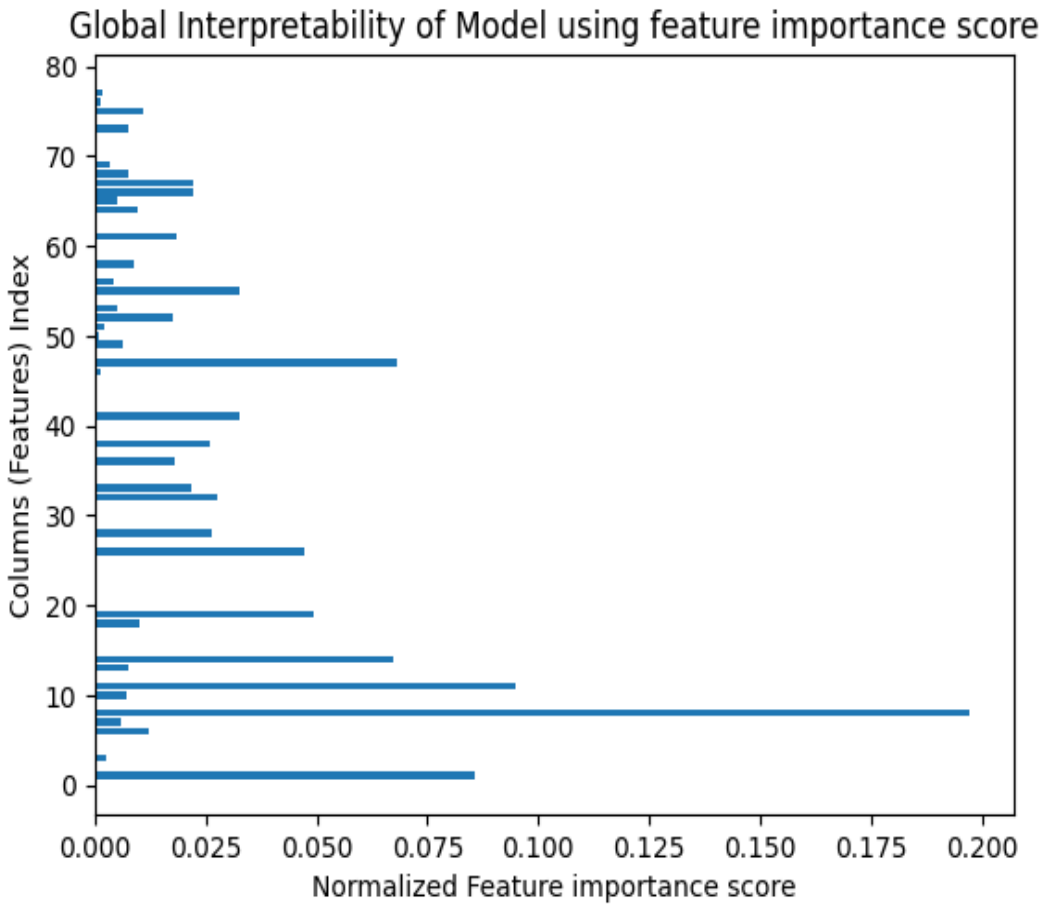


Figure 6.5. Global interpretability of the TabNet model

Figure 6.6, shows two different Masks for the first two steps on 50 training samples. As we discussed before, TabNet follows instant-wise feature selection. In the figure, mask 0 focuses on features 18, 40, 46, 52 60, and 65 for most of the instances to select salient features. Mask 1 also, focuses on features 13, 25, 32,37, and slightly on features 47, and 48 to select the most important features in an instance-wise (row by row) manner.

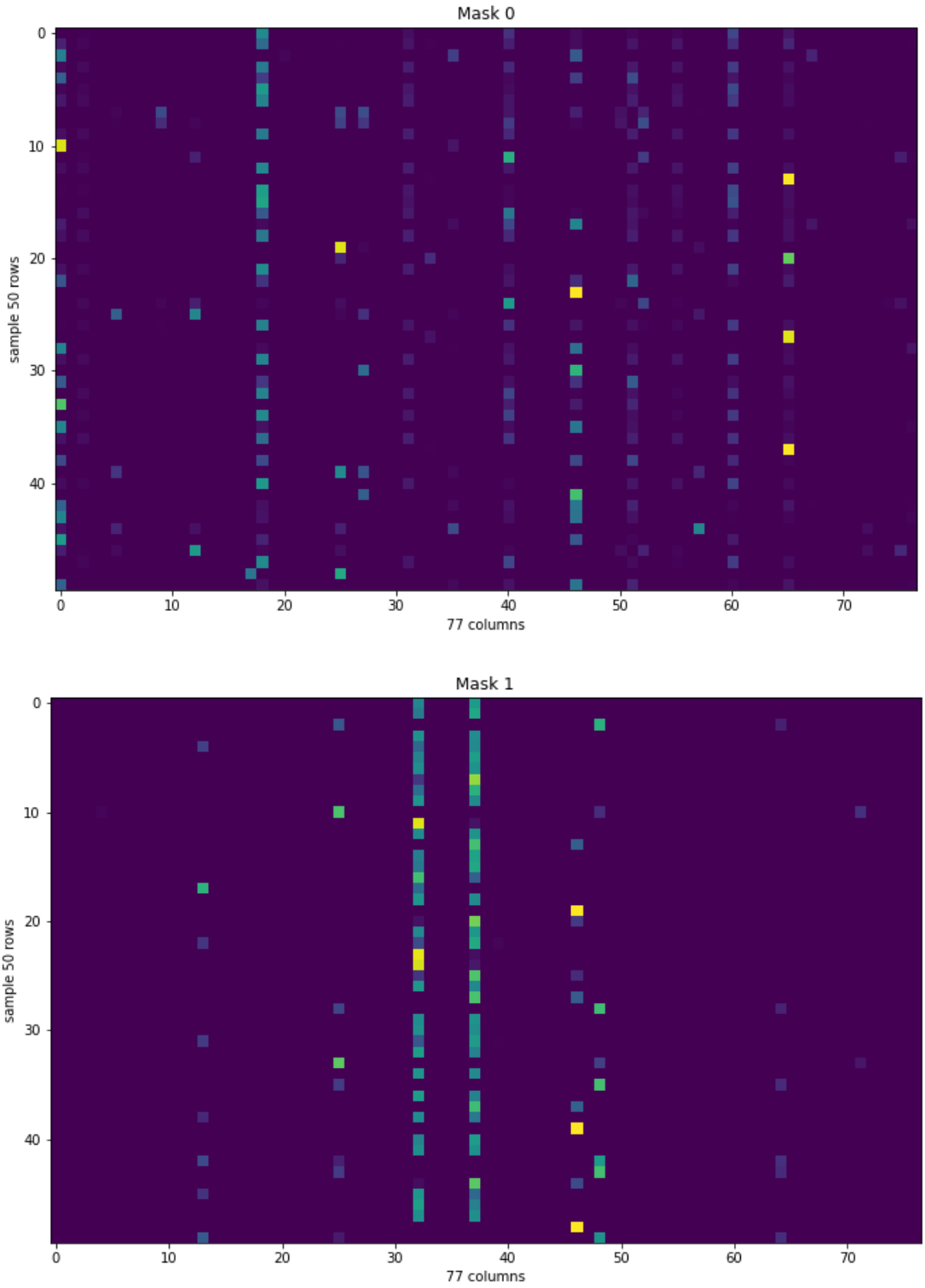


Figure 6.6. Local interpretability of the TabNet model for the first two-step Masks

6.2 Discussion

The TabNet encoder architecture's ability to transform input feature values into useful representations of feature values helps the model to learn the important pattern from the whole dataset efficiently. The feature transformation process as well as the decision process are distributed sequentially in each step. In general, the Attentive Transformer's ability to select the most important features at each step sequentially and the Feature Transformer's ability to encode the input data into a more representative format give the TabNet-based model high performance in the detection task. This performance is proved with ablation studies and repeated experiments. Figure 6.7 and Figure 6.8 High-performance results of the TabNet-based Network Intrusion Detection models for different experiments.

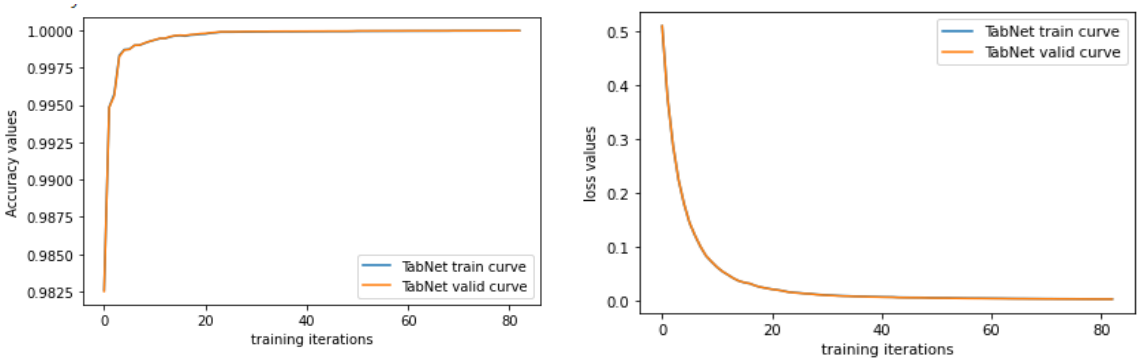


Figure 6.7 Accuracy curve (left) and loss curve (right) for a balanced dataset.

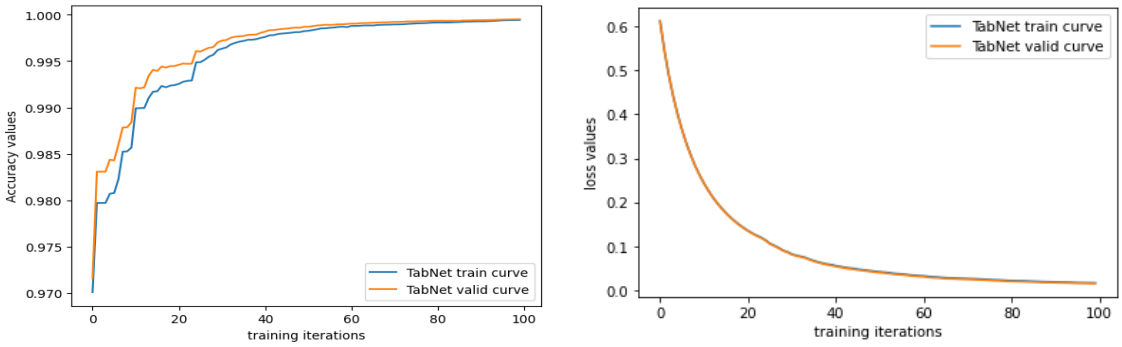


Figure 6.8 Accuracy curve (left) and loss curve (right) for imbalanced dataset.

Figure 6.5 and 6.6 shows global and local feature importance scores respectively. These scores are generated at each step by an attentive transformer and Mask components of the TabNet architecture. Feature Transformer component of the TabNet architecture transforms the input feature values into a useful representation of values. After transforming the feature values, the Feature transformer gives outputs that will be input for the ReLu activation layer and Attentive transformer. When Attentive Transformer gets input from Feature Transformer, it further processes the output from the feature transformer with its Fully Connected and Batch normalizer network. The result from this network is multiplied with the prior scale factor to attach information about the feature repetition usage in the previous steps. Then Sparsemax activation layer creates a sparse matrix from the previous multiplication product values. Then this sparse matrix becomes the Mask component of the TabNet architecture. This process is repeated in every step to create a unique Mask component for every step. Local interpretability, as well as global Interpretability of the model, is achieved because of this Mask component.

Model Local interpretability means the openness of the model to describe its inference process to drive the result at the instance level or single row level. The input features values normalized by the batch normalizer layer are distributed for each sequential step with a certain batch size and dimension. Then the normalized features values matrix is multiplied with the sparse Mask at every step for salient feature selection. The sparsity of the Mask gives the ability to select important features at each step. Figure 6.6 shows this process of row-level feature selection. Knowing Instance level or row level feature selection help to get more insight in-depth into how the model inference process at the instance and gives local interpretability property to the model.

Model Global Interpretability means the model decision process traceability for a certain number of instances or the whole dataset. Model feature importance score at the global level can be obtained by simply aggregating each Masks value from different steps. But before aggregation, each Mask holds information about their voting contribution to the decision output. To achieve this every step decision output from the ReLu layer is multiplied with its corresponding Mask values as a coefficient to quantify the voting contribution. The aggregated feature score values of each mask create global interpretability.

From the above (Performance Results) performance result section, we can see that the accuracies of all the models are almost the same which is above 99%. But from the experiment, we can understand that when the size of the dataset increases XGBoost model tries to build weak learner trees and the model become more complex. Due to this the performance of the model becomes constant or decreases. When we look at the Autoencoder model the performance increases when we increase the size of the dataset. But this makes the model more complex and opaquer. TabNet model on the other side gets good performance and model interpretability even if the dataset size increases or decreases in size. With the TabNet model, it is easy and requires less time to find the most optimal hyperparameter value combinations. Whereas in XGBoost relatively high time is required to get optimal hyperparameter values that configure a high-performance classifier model. Performance stability is also shown from the proposed model for different sizes of the dataset. This gives the TabNet model scalability property. The tabNet-based model also shows almost close training and testing errors under different test conditions. This makes the TabNet model robust to handle unseen data. On contrary, both Autoencoder and XGBoost models show less scalability and robustness property under repeated experiments. Based on the experiment Table 6.4 summarizes a comparison of the three models based on different criteria.

Table 6.4 Summary of Model Comparisons

Criteria	**Models**		
	XGBoost	**Autoencoder**	**TabNet**
Classification Performance (Detection Accuracy)	>99%	~99%	>99%
Scalability (Dataset size)	Less scalable	Less Scalable	Scalable
Robustness (close training error and testing error)	Less Robust	Less Robust	Robust
Global Interpretability	Interpretable	Opaque	Interpretable
Local Interpretability	Opaque	Opaque	Interpretable
Support end-to-end learning	Less support	Support	Support
Hyperparameter tuning	Requires more	Requires less	Requires less
Model training time	Fastest	Slow	Medium

Table 6.5 Comparison of this research study with previous related works

Ref.	**Datasets**	**Algorithms**	**Results**
(Deepak, 2020)	CICIDS2017	XGBoost	99.1% of accuracy
(Lee et al., 2020)	CICIDS2017	Autoencoder	99.0% of accuracy
(W. Xu & Fan, 2022)	CICIDS2017	XGBoost model for classification and Autoencoder for feature selection	99.92 % of accuracy
This Research Study	**CICIDS2017**	**TabNet encoder**	**99.98 % of accuracy**

Table 6.5 shows the comparison between this research study and three previous related studies done on the CICIDS2017 dataset. Based on the comparison the proposed TabNet Based intrusion detection model improves the accuracy result from previous related works.

CHAPTER SEVEN

CONCLUSIONS AND FUTURE WORKS

7.1 Conclusions

The objective of this research study is to develop an efficient network intrusion detection model in terms of detection performance and model interpretability. Based on this objective first, this research study reviewed existing intrusion detection systems' limitations and challenges with an extensive literature study. Low detection performance and model opaqueness were identified as research gaps by this research study from related previous studies. Based on identified research gaps literature review was continued to find a solution for those gaps. Then this research study developed high performance and interpretable model using TabNet deep neural network sequential architecture. The CICIDS2017 intrusion dataset was selected for proposed model training and testing since it is proved with a literature review that it is the most appropriate dataset from publicly available intrusion datasets for this research study (Khraisat et al., 2019; Sharafaldin et al., 2017; Thakkar & Lohiya, 2020a, 2020b). The developed TabNet-based model shows high performance with local and global model interpretability during the training phase as well as the test phase when evaluated with appropriate performance evaluation metrics. The performance of the proposed model is also compared with XGBoost and Autoencoder state-of-the-art ML and DL algorithms respectively. The comparison result shows the proposed TabNet model is better in detection performance, interpretability[30], robustness[31] , and scalability[32] criteria. The finding of this research study is high performance and interpretable network-based intrusion detection model with the following contributions.

Contributions of this research study.

- Develop and test a high-performance network-based intrusion detection model.

Existing network intrusion detection models developed using machine learning and deep learning techniques are suffered from low detection performance as we discuss in chapter two. Low detection performance can be defined either by low true positive and true negative rates or by high false positive and false negative rates. In other words, if the model detects or classifies a certain number of normal traffic as an attack or certain numbers of

[30] How the inference process and the driven result is understandable by humans.
[31] Almost having the same or close training error and testing error.
[32] Shows good performance for small, medium and large size dataset.

attack traffic as normal, we can generalize that the model performance is low. This problem was seen in network intrusion detection models proposed in the previous related works as summarized in Tables 2.1 and 6.5. But the network intrusion detection model proposed in this research study archives high detection performance by maximizing true positive rate and true negative rate and by minimizing false positive rate and false negative rates as shown in the result section. High detection performance was achieved in this research study due to the improvement of sparsity and removal of skip connection from the original TabNet architecture. The entmax sparsity mechanism adopted by this research study leads to better feature selection than the sparsemax mechanism used in the original TabNet architecture. Entmax's high degree of sparsity helps to encode the input features to the most important features. These most important features are then used by the fully connected and sigmoid classifier to predict the class of the input features. A dense multilayer perceptron with the number of neurons equal to the dimension of the output of the TabNet encoder was used as a fully connected layer for the classification task with the sigmoid activation function. Removal of skip connection from feature transformer block result in better detection performance since shallow (only two shared and two independent) GLU component was used.

- Develop locally and globally interpretable deep neural network-based intrusion detection model.

As we discussed in chapter two existing deep learning-based network intrusion detection models are not locally and globally interpretable. And also existing machine learning-based intrusion detection model lack local interpretability features and global interpretability features except for decision tree-like models. Interpretability is a key feature in critical systems to get the trust of users when deployed in a real-world environment and to understand the inner working principle of the model for maintenance and further performance improvement purpose. TabNet architecture introduces learnable and controllable mask components in the architecture for local and global interpretability. The mask component is generated by passing input feature values in the sparsemax activation layer. This research study adopted the entmax sparsity layer to enhance the interpretability of the network intrusion detection model. The ratio of each value of output encoded features to the sum of the output of encoded feature values was multiplied in an element-wise manner with the mask values to give coefficient weight. The weight of the coefficient multiplied by the mask component indicates how much each feature influences

the decision process. This mechanism makes the proposed network intrusion detection model inference process locally and globally interpretable.

- Compare the performance of the proposed model with the existing state-of-the-art models in the domain.

The comparison result shows the proposed TabNet model is better in detection performance, interpretability , robustness , and scalability criteria thean state of the art Autoencoder and XGBoost models.

7.2 Future Works

Although the proposed intrusion detection model shows high-performance results and interpretability characteristics, there are some uncovered issues and challenges that must be studied in feature works. Therefore, the following issues are mentioned for feature work as a recommendation.

- The training time of the proposed model was much longer compared to decision trees like machine learning algorithms. Any training time improvement strategies were not implemented in this research study. So, for the feature study training time improvement strategies must be applied.
- The decoder part of TabNet architecture is not implemented in this research study. Autoencoder with the encoder-decoder full network will help to build unsupervised and semi-supervised learning which may have high significance to handle zero-day attacks. Therefore, the decoder part must be implemented and studied properly.
- Any feature work can test the performance of the proposed model using different publicly available datasets in the domain or by generating a new intrusion dataset.

This research work fund was granted by Adama Science and Technology University.
ASTU/SM-R/488/22

References

Ahmad, I., Haq, Q. E. U., Imran, M., Alassafi, M. O., & Alghamdi, R. A. (2022). An Efficient Network Intrusion Detection and Classification System. *Mathematics*, *10*(3). https://doi.org/10.3390/math10030530

Ahmed, M., Naser Mahmood, A., & Hu, J. (2016). A survey of network anomaly detection techniques. *Journal of Network and Computer Applications*, *60*, 19–31. https://doi.org/10.1016/j.jnca.2015.11.016

Ahsan, M. M., Mahmud, M. A. P., Saha, P. K., Gupta, K. D., & Siddique, Z. (2021). Effect of Data Scaling Methods on Machine Learning Algorithms and Model Performance. *Technologies*, *9*(3), 52. https://doi.org/10.3390/technologies9030052

Akpakwu, G. A., Silva, B. J., Hancke, G. P., & Abu-Mahfouz, A. M. (2018). A Survey on 5G Networks for the Internet of Things: Communication Technologies and Challenges. *Undefined*, *6*, 3619–3647. https://doi.org/10.1109/ACCESS.2017.2779844

Alasadi, S. A., & Bhaya, W. S. (2017). Review of data preprocessing techniques in data mining. *Journal of Engineering and Applied Sciences*, *12*(16), 4102–4107. https://doi.org/10.3923/JEASCI.2017.4102.4107

Alatwi, H. A., & Morisset, C. (2021). Adversarial Machine Learning In Network Intrusion Detection Domain: A Systematic Review. *Undefined*.

Alavizadeh, H., Jang-Jaccard, J., & Alavizadeh, H. (2021). *Deep Q-Learning based Reinforcement Learning Approach for Network Intrusion Detection*. http://arxiv.org/abs/2111.13978

Aljanabi, M., Ismail, M. A., & Ali, A. H. (2021). Intrusion detection systems, issues, challenges, and needs. *International Journal of Computational Intelligence Systems*, *14*(1), 560–571. https://doi.org/10.2991/ijcis.d.210105.001

Aljawarneh, S., Aldwairi, M., & Yassein, M. B. (2018). Anomaly-based intrusion detection system through feature selection analysis and building a hybrid efficient model. *Journal of Computational Science*, *25*, 152–160. https://doi.org/10.1016/J.JOCS.2017.03.006

Al-Yaseen, W. L., Zakree, M., Nazri, A., & Othman, Z. A. (2016). Real-Time Intrusion Detection System Using Multi-agent System Multi-agent System based Intrusion Detection System View project the Universiti Kebangsaan Malaysia Dana Impak Perdana grant [DIP-2014-039]. View project Real-Time Intrusion Detection System Using Multi-agent System. In *Article in IAENG International Journal of Computer Science*. https://www.researchgate.net/publication/299363900

Ansari, M. S., Bartos, V., & Lee, B. (2020). Shallow and Deep Learning Approaches for Network Intrusion Alert Prediction. *Procedia Computer Science*, *171*, 644–653. https://doi.org/10.1016/J.PROCS.2020.04.070

Applied Deep Learning - Part 3: Autoencoders | by Arden Dertat | Towards Data Science. (n.d.). Retrieved August 26, 2022, from https://towardsdatascience.com/applied-deep-learning-part-3-Autoencoders-1c083af4d798

Arik, S. O., & Pfister, T. (2019). *TabNet: Attentive Interpretable Tabular Learning*. http://arxiv.org/abs/1908.07442

Arulkumaran, K., Deisenroth, M. P., Brundage, M., & Bharath, A. A. (2017). A Brief Survey of Deep Reinforcement Learning. *IEEE Signal Processing Magazine*, *34*(6), 26–38. https://doi.org/10.1109/msp.2017.2743240

Arya, V., Bellamy, R. K. E., Chen, P.-Y., Dhurandhar, A., Hind, M., Hoffman, S. C., Houde, S., Liao, Q. V., Luss, R., Mojsilović, A., Mourad, S., Pedemonte, P., Raghavendra, R., Richards, J., Sattigeri, P., Shanmugam, K., Singh, M., Varshney, K. R., Wei, D., & Zhang, Y. (2019). *One Explanation Does Not Fit All: A Toolkit and Taxonomy of AI Explainability Techniques*. https://doi.org/10.48550/arxiv.1909.03012

Ashoor, A. S. (2019). Importance of Intrusion Detection System (IDS). *International Journal of Scientific Engineering Research*, 61227.

Aziz, R. S., Kareem, D. M., & Jawad, S. F. (2021). Intrusion Detection Systems: Status, Challenges, and Future Trends - A survey. *4th International Iraqi Conference on Engineering Technology and Their Applications, IICETA 2021*, 303–307. https://doi.org/10.1109/IICETA51758.2021.9717859

Azizjon, M., Jumabek, A., & Kim, W. (2020). 1D CNN-based network intrusion detection with normalization on imbalanced data. *2020 International Conference on Artificial Intelligence in Information and Communication, ICAIIC 2020*, 218–224. https://doi.org/10.1109/ICAIIC48513.2020.9064976

Baldi, P. (2012). *Autoencoders, Unsupervised Learning, and Deep Architectures*. *27*, 37–50.

Belarbi, O., Khan, A., Carnelli, P., & Spyridopoulos, T. (2022). An Intrusion Detection System based on Deep Belief Networks. *ArXiv*. https://github.com/othmbela/dbn-based-nids

Burkart, N., & Huber, M. F. (2020). A Survey on the Explainability of Supervised Machine Learning. *Journal of Artificial Intelligence Research*, *70*, 245–317. https://doi.org/10.1613/jair.1.12228

Business Cost of Cybercrime | Cobalt. (n.d.). Retrieved August 26, 2022, from https://www.cobalt.io/blog/business-cost-of-cybercrime

Canbek, G., Taskaya Temizel, T., & sagiroglu, seref. (2020). *Binary-Classification Performance Evaluation Reporting Survey Data with the Findings*. *3*. https://doi.org/10.17632/5C442VBJZG.3

Chen, T., & Guestrin, C. (2016). XGBoost: A Scalable Tree Boosting System. *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, *13-17-August-2016*, 785–794. https://doi.org/10.1145/2939672.2939785

Chen, T., He, T., & Benesty, M. (2018). XGBoost : eXtreme Gradient Boosting. *R Package Version 0.71-2*, 1–4.

Cheng, A. M. (2017). *The Causes, Impact, and Detection of Duplicate Observations*.

Choudhury, A., & Kosorok, M. R. (2020). *Missing Data Imputation for Classification Problems*. http://arxiv.org/abs/2002.10709

Deepak, T. A. (2020). XGBoost Classification based Network Intrusion Detection System for Big Data using PySparkling Water. *International Journal of Advanced Trends in Computer Science and Engineering*, *9*(1), 377–382. https://doi.org/10.30534/IJATCSE/2020/55912020

Devan, P., & Khare, N. (2020). An efficient XGBoost–DNN-based classification model for network intrusion detection system. *Neural Computing and Applications 2020 32:16*, *32*(16), 12499–12514. https://doi.org/10.1007/S00521-020-04708-X

Dhaliwal, S. S., Nahid, A. al, & Abbas, R. (2018). Effective intrusion detection system using XGBoost. *Information (Switzerland)*, *9*(7). https://doi.org/10.3390/INFO9070149

Dong, S., Wang, P., & Abbas, K. (2021). A survey on deep learning and its applications. *Computer Science Review*, *40*, 100379. https://doi.org/10.1016/J.COSREV.2021.100379

Dosilovic, F. K., Brcic, M., & Hlupic, N. (2018). Explainable artificial intelligence: A survey. *2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics, MIPRO 2018 - Proceedings*, 210–215. https://doi.org/10.23919/MIPRO.2018.8400040

Dwibedi, S., Pujari, M., & Sun, W. (2020). A Comparative Study on Contemporary Intrusion Detection Datasets for Machine Learning Research. *Proceedings - 2020 IEEE International Conference on Intelligence and Security Informatics, ISI 2020*. https://doi.org/10.1109/ISI49825.2020.9280519

el Mrabet, M. A., el Makkaoui, K., & Faize, A. (2021). Supervised Machine Learning: A Survey. *Proceedings - 4th International Conference on Advanced Communication Technologies and Networking, CommNet 2021*. https://doi.org/10.1109/COMMNET52204.2021.9641998

Elsadai, A., Ibrahim, J., Hajjaj, F., & Jakić, P. (2019). *The Overview of Intrusion Detection System Methods and Techniques*. 155–161. https://doi.org/10.15308/SINTEZA-2019-155-161

Emmanuel, T., Maupong, T., Mpoeleng, D., Semong, T., Mphago, B., & Tabona, O. (2021). A survey on missing data in machine learning. *Journal of Big Data*, *8*(1). https://doi.org/10.1186/S40537-021-00516-9

Farahnakian, F., & Heikkonen, J. (2018). A deep auto-encoder-based approach for intrusion detection system. *International Conference on Advanced Communication Technology, ICACT*, *2018-February*, 178–183. https://doi.org/10.23919/ICACT.2018.8323688

Ferriyan, A., Thamrin, A. H., Takeda, K., & Murai, J. (2021). Generating network intrusion detection dataset based on real and encrypted synthetic attack traffic. *Applied Sciences (Switzerland)*, *11*(17). https://doi.org/10.3390/app11177868

Gao, X., Shan, C., Hu, C., Niu, Z., & Liu, Z. (2019). An Adaptive Ensemble Machine Learning Model for Intrusion Detection. *IEEE Access*, *7*, 82512–82521. https://doi.org/10.1109/ACCESS.2019.2923640

Ghori, K. M. U., Imran, M., Nawaz, A., Abbasi, R. A., Ullah, A., & Szathmary, L. (2020). Performance analysis of machine learning classifiers for non-technical loss detection. *Journal of Ambient Intelligence and Humanized Computing*, *1*, 1–16. https://doi.org/10.1007/S12652-019-01649-9/FIGURES/11

Goel, S., & Nussbaum, B. (2021). Attribution Across Cyber Attack Types: Network Intrusions and Information Operations. *IEEE Open Journal of the Communications Society*, *2*, 1082–1093. https://doi.org/10.1109/ojcoms.2021.3074591

Gopal, S., Patro, K., & Kumar Sahu, K. (2015). *Normalization: A Preprocessing Stage*. www.kiplinger.com,

Guven, E. Y., Gulgun, S., Manav, C., Bakir, B., & Gurkas Aydin, Z. (2022). Multiple Classification of Cyber Attacks Using Machine Learning. *Electrica*, *22*(2), 313–320. https://doi.org/10.54614/ELECTRICA.2022.22031

Haggag, M., Tantawy, M. M., & El-Soudani, M. M. S. (2020). Implementing a deep learning model for intrusion detection on the apache spark platform. *IEEE Access*, *8*(1), 163660–163672. https://doi.org/10.1109/ACCESS.2020.3019931

Halimaa, A. A., & Sundarakantham, K. (2019). Machine learning-based intrusion detection system. *Proceedings of the International Conference on Trends in Electronics and Informatics, ICOEI 2019*, 916–920. https://doi.org/10.1109/ICOEI.2019.8862784

Hancock, J. T., & Khoshgoftaar, T. M. (2020). Survey on categorical data for neural networks. *Journal of Big Data*, *7*(1). https://doi.org/10.1186/S40537-020-00305-W

Hasib, K. Md., Iqbal, Md. S., Shah, F. M., Mahmud, J. al, Popel, M. H., Showrov, Md. I. H., Ahmed, S., & Rahman, O. (2020). A Survey of Methods for Managing the Classification and Solution of Data Imbalance Problem. *Journal of Computer Science*, *16*(11), 1546–1557. https://doi.org/10.3844/jcssp.2020.1546.1557

He, K., Zhang, X., Ren, S., & Sun, J. (2015). Deep Residual Learning for Image Recognition. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, *2016-December*, 770–778. https://doi.org/10.48550/arxiv.1512.03385

Hindy, H., Brosset, D., Bayne, E., Seeam, A. K., Tachtatzis, C., Atkinson, R., & Bellekens, X. (2020). A Taxonomy of Network Threats and the Effect of Current Datasets on Intrusion Detection Systems. *IEEE Access*, *8*(1), 104650–104675. https://doi.org/10.1109/ACCESS.2020.3000179

Hippe, R., Dornheim, J., Zeitvogel, S., & Laubenheimer, A. (2017). Evaluation of machine learning algorithms for Smurf Detection. *Cerc2017*, 65.

Hodo, E., Hamilton, A. W., & Tachtatzis, C. (2017). *Shallow and Deep Networks Intrusion Detection System: A Taxonomy and Survey Intrusion Detection Systems View project Deep Packet Inspection View project*. https://www.researchgate.net/publication/312170608

IDS 2017 | Datasets | Research | Canadian Institute for Cybersecurity | UNB. (n.d.). Retrieved August 16, 2022, from https://www.unb.ca/cic/datasets/ids-2017.html

Ioffe, S., & Christian Szegedy. (2015). Batch Normalization: Accelerating Deep Network Training by Reducing. *Journal of Molecular Structure*, *1134*, 63–66.

Ioffe, S., & Szegedy, C. (2015). Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *32nd International Conference on Machine Learning, ICML 2015*, *1*, 448–456. https://doi.org/10.48550/arxiv.1502.03167

Jena, S. R., Yadav, A. K., Patel, S., & Saibaba, C. H. M. H. (2021). Analysis on mobile cloud security and comparison of existing models. *Indian Journal of Computer Science and Engineering*, *12*(3), 580–590. https://doi.org/10.21817/INDJCSE/2021/V12I3/211203065

Johnson, J. M., & Khoshgoftaar, T. M. (2019). Survey on deep learning with class imbalance. *Journal of Big Data*, *6*(1), 1–54. https://doi.org/10.1186/S40537-019-0192-5/TABLES/18

Khraisat, A., & Alazab, A. (2021). A critical review of intrusion detection systems in the internet of things: techniques, deployment strategy, validation strategy, attacks, public datasets, and challenges. *Cybersecurity*, *4*(1), 1–27. https://doi.org/10.1186/S42400-021-00077-7/TABLES/10

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

Laghrissi, F. E., Douzi, S., Douzi, K., & Hssina, B. (2021). Intrusion detection systems using long short-term memory (LSTM). *Journal of Big Data*, *8*(1). https://doi.org/10.1186/s40537-021-00448-4

Lansky, J., Ali, S., Mohammadi, M., Majeed, M. K., Karim, S. H. T., Rashidi, S., Hosseinzadeh, M., & Rahmani, A. M. (2021). Deep Learning-Based Intrusion Detection Systems: A Systematic Review. *IEEE Access*, *9*, 101574–101599. https://doi.org/10.1109/ACCESS.2021.3097247

Lee, J., Pak, J. G., & Lee, M. (2020). Network Intrusion Detection System using Feature Extraction based on Deep Sparse Autoencoder. *International Conference on ICT Convergence*, *2020-October*, 1282–1287. https://doi.org/10.1109/ICTC49870.2020.9289253

Leiner, B. M., Cerf, V. G., Clark, D. D., Kahn, R. E., Kleinrock, L., Lynch, D. C., Postel, J., Roberts, L. G., & Wolff, S. (2009). A brief history of the internet. *ACM SIGCOMM Computer Communication Review*, *39*(5), 22–31. https://doi.org/10.1145/1629607.1629613

Li, Y., & Liu, Q. (2021a). A comprehensive review study of cyber-attacks and cyber security; Emerging trends and recent developments. *Energy Reports*, *7*, 8176–8186. https://doi.org/10.1016/J.EGYR.2021.08.126

Lopez Pinaya, W. H., Vieira, S., Garcia-Dias, R., & Mechelli, A. (2020). Autoencoders. *Machine Learning: Methods and Applications to Brain Disorders*, 193–208. https://doi.org/10.48550/arxiv.2003.05991

M, H., & M.N, S. (2015). A Review on Evaluation Metrics for Data Classification Evaluations. *International Journal of Data Mining & Knowledge Management Process*, *5*(2), 01–11. https://doi.org/10.5121/IJDKP.2015.5201

Malhotra, H., & Sharma, P. (2019). Intrusion Detection using Machine Learning and Feature Selection. *International Journal of Computer Network and Information Security*, *11*(4), 43–52. https://doi.org/10.5815/ijcnis.2019.04.06

Martins, A. F. T., & Astudillo, R. F. (2016). From Softmax to Sparsemax: A Sparse Model of Attention and Multi-Label Classification. *33rd International Conference on Machine Learning, ICML 2016*, *4*, 2432–2443. https://doi.org/10.48550/arxiv.1602.02068

Maseer, Z. K., Yusof, R., Bahaman, N., Mostafa, S. A., & Foozy, C. F. M. (2021a). Benchmarking of Machine Learning for Anomaly Based Intrusion Detection Systems in the CICIDS2017 Dataset. *IEEE Access*, *9*, 22351–22370. https://doi.org/10.1109/ACCESS.2021.3056614

Mendonça, R. v, Teodoro, A. A. M., Rosa, R. L., Saadi, M., Carrillo Melgarejo, D., Nardelli, P. H. J., & Rodríguez, D. Z. (2021). Intrusion Detection System Based on Fast Hierarchical Deep Convolutional Neural Network. *IEEE*. https://doi.org/10.1109/ACCESS.2021.3074664

Menghani, G. (2021). *Efficient Deep Learning: A Survey on Making Deep Learning Models Smaller, Faster, and Better*. https://doi.org/10.48550/arxiv.2106.08962

Mrabet, Z. el, Ezzari, M., Abou, B., Majd, E., & Elghazi, H. (2019). Deep Learning-Based Intrusion Detection System for Advanced Metering Infrastructure. *ArXive*, *1*.

Nayak, R., Behera, M. M., Pati, U. C., & Das, S. K. (2019). Video-based Real-time Intrusion Detection System using Deep-Learning for Smart City Applications. *International Symposium on Advanced Networks and Telecommunication Systems, ANTS*, *2019-December*. https://doi.org/10.1109/ANTS47819.2019.9117960

Network-Based Intrusion Detection System - an overview | ScienceDirect Topics. (n.d.). Retrieved August 26, 2022, from https://www.sciencedirect.com/topics/computer-science/network-based-intrusion-detection-system

Othman, S. M., Ba-Alwi, F. M., Alsohybe, N. T., & Al-Hashida, A. Y. (2018). Intrusion detection model using machine learning algorithm on Big Data environment. *Journal of Big Data*, *5*(1). https://doi.org/10.1186/s40537-018-0145-4

Padakandla, S. (2020). A Survey of Reinforcement Learning Algorithms for Dynamically Varying Environments. *ACM Computing Surveys*, *54*(6). https://doi.org/10.1145/3459991

Pawar, M. v., & Anuradha, J. (2015). Network security and types of attacks in the network. *Procedia Computer Science*, *48*(C), 503–506. https://doi.org/10.1016/j.procs.2015.04.126

Potdar, K., Pardawala, T. S., & Pai, C. D. (2017). A Comparative Study of Categorical Variable Encoding Techniques for Neural Network Classifiers. *Undefined*, *175*(4), 7–9. https://doi.org/10.5120/IJCA2017915495

Pouyanfar, S., Sadiq, S., Yan, Y., Tian, H., Tao, Y., Reyes, M. P., Shyu, M. L., Chen, S. C., & Iyengar, S. S. (2018). A Survey on Deep Learning. *ACM Computing Surveys (CSUR)*, *51*(5). https://doi.org/10.1145/3234150

Rácz, A., Bajusz, D., & Héberger, K. (2021). Effect of dataset size and train/test split ratios in qsar/qspr multiclass classification. *Molecules*, *26*(4). https://doi.org/10.3390/molecules26041111

Rajasekaran, K., & Nirmala, K. (2020). *Classification and Importance of Intrusion Detection System*. http://sites.google.com/site/ijcsis/

Reis, I., Rotman, M., Poznanski, D., Prochaska, J. X., & Wolf, L. (2021). Effectively using unsupervised machine learning in next-generation astronomical surveys. *Astronomy and Computing*, *34*, 100437. https://doi.org/10.1016/J.ASCOM.2020.100437

Samek, W., Wiegand, T., & Müller, K.-R. (2017). *Explainable Artificial Intelligence: Understanding, Visualizing and Interpreting Deep Learning Models*. http://arxiv.org/abs/1708.08296

Sardana, H., & Singh, K. (2018). Reducing False Alarms in Intrusion Detection Systems-A Survey. *International Research Journal of Engineering and Technology*, *9001*. www.irjet.net

Sharafaldin, I., Gharib, A., Lashkari, A. H., & Ghorbani, A. A. (2017). Towards a Reliable Intrusion Detection Benchmark Dataset. *Software Networking*, *2017*(1), 177–200. https://doi.org/10.13052/JSN2445-9739.2017.009

Sharafaldin, I., Lashkari, A. H., & Ghorbani, A. A. (2018). Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization. *Undefined*, *2018-January*, 108–116. https://doi.org/10.5220/0006639801080116

Shone, N., Ngoc, T. N., Phai, V. D., & Shi, Q. (2018). A Deep Learning Approach to Network Intrusion Detection. *IEEE Transactions on Emerging Topics in Computational Intelligence*, *2*(1), 41–50. https://doi.org/10.1109/TETCI.2017.2772792

Shyla, Kumar, K., & Bhatnagar, V. (2021). Machine Learning Algorithms Performance Evaluation for Intrusion Detection. *Journal of Information Technology Management*, *13*(1), 42–61. https://doi.org/10.22059/JITM.2021.80024

Singh, A., & Jang-Jaccard, J. (2022). Autoencoder-based Unsupervised Intrusion Detection using Multi-Scale Convolutional Recurrent Networks. *ArXiv*.

Singh, K., Mahajan, A., & Mansotra, V. (2021). 1D-CNN-based Model for Classification and Analysis of Network Attacks. *IJACSA) International Journal of Advanced Computer Science and Applications*, *12*(11), 2021. www.ijacsa.thesai.org

Surana, J., Sharma, J., Saraf, I., Puri, N., Navin, B., & Professor, A. (2017). A Survey On Intrusion Detection System. *International Journal of Engineering Development and Research*, *5*(2). www.ijedr.org

Tavallaee, M., Bagheri, E., Lu, W., & Ghorbani, A. A. (2009). A detailed analysis of the KDD CUP 99 data set. *IEEE Symposium on Computational Intelligence for Security and Defense Applications, CISDA 2009*. https://doi.org/10.1109/CISDA.2009.5356528

Thakkar, A., & Lohiya, R. (2020a). A Review of the Advancement in Intrusion Detection Datasets. *Procedia Computer Science*, *167*, 636–645. https://doi.org/10.1016/J.PROCS.2020.03.330

Thakkar, A., & Lohiya, R. (2021). A survey on intrusion detection system: feature selection, model, performance measures, application perspective, challenges, and future research directions. In *Artificial Intelligence Review* (Issue 0123456789). Springer Netherlands. https://doi.org/10.1007/s10462-021-10037-9

Thirimanne, S. P., Jayawardana, L., Yasakethu, L., Liyanaarachchi, P., & Hewage, C. (2022a). Deep Neural Network Based Real-Time Intrusion Detection System. *SN Computer Science 2022 3:2*, *3*(2), 1–12. https://doi.org/10.1007/S42979-022-01031-1

Ugurlu, M., & Dogru, I. A. (2019). A Survey on Deep Learning Based Intrusion Detection System. *UBMK 2019 - Proceedings, 4th International Conference on Computer Science and Engineering*, 223–228. https://doi.org/10.1109/UBMK.2019.8907206

Uikey, R., & Gyanchandani, M. (2019). Survey on Classification Techniques Applied to Intrusion Detection System and its Comparative Analysis. *Proceedings of the 4th International Conference on Communication and Electronics Systems, ICCES 2019*, *Icces*, 1451–1456. https://doi.org/10.1109/ICCES45898.2019.9002129

Umer, M., Xiaoli, H., & Abdul, S. (2020). Big Data Security Analysis in Network Intrusion Detection System. *International Journal of Computer Applications*, *177*(30), 12–18. https://doi.org/10.5120/ijca2020919759

Usama, M., Qadir, J., Raza, A., Arif, H., Yau, K. L. A., Elkhatib, Y., Hussain, A., & Al-Fuqaha, A. (2019). Unsupervised Machine Learning for Networking: Techniques, Applications, and Research Challenges. *IEEE Access*, *7*, 65579–65615. https://doi.org/10.1109/ACCESS.2019.2916648

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention Is All You Need. *Advances in Neural Information Processing Systems*, *2017-December*, 5999–6009. https://doi.org/10.48550/arxiv.1706.03762

Vincent, P., Larochelle, H., Bengio, Y., & Manzagol, P. A. (2008). Extracting and composing robust features with denoising Autoencoders. *Proceedings of the 25th International Conference on Machine Learning*, 1096–1103. https://doi.org/10.1145/1390156.1390294

Wang, M., Zheng, K., Yang, Y., & Wang, X. (2020). An Explainable Machine Learning Framework for Intrusion Detection Systems. *IEEE Access*, *8*, 73127–73141. https://doi.org/10.1109/ACCESS.2020.2988359

Xu, F., Uszkoreit, H., Du, Y., Fan, W., Zhao, D., & Zhu, J. (2019). Explainable AI: A Brief Survey on History, Research Areas, Approaches and Challenges. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and*

Lecture Notes in Bioinformatics), 11839 LNAI, 563–574. https://doi.org/10.1007/978-3-030-32236-6_51

Xu, W., & Fan, Y. (2022). Intrusion Detection Systems Based on Logarithmic Autoencoder and XGBoost. *Security and Communication Networks*, *2022*. https://doi.org/10.1155/2022/9068724

Xu, Y., & Goodacre, R. (2018). On Splitting Training and Validation Set: A Comparative Study of Cross-Validation, Bootstrap and Systematic Sampling for Estimating the Generalization Performance of Supervised Learning. *Journal of Analysis and Testing*, *2*(3), 249–262. https://doi.org/10.1007/S41664-018-0068-2/FIGURES/9

Yu, L., & Zhou, N. (2021). *Survey of Imbalanced Data Methodologies*. https://doi.org/10.48550/arxiv.2104.02240

Zhai, J., Zhang, S., Chen, J., & He, Q. (2019). Autoencoder and Its Various Variants. *Proceedings - 2018 IEEE International Conference on Systems, Man, and Cybernetics, SMC 2018*, 415–419. https://doi.org/10.1109/SMC.2018.00080

Zhou, Y., Cheng, G., Jiang, S., & Dai, M. (2020). An Efficient Network Intrusion Detection System Based on Feature Selection and Ensemble Classifier. *Undefined*, *174*. https://doi.org/10.1016/J.COMNET.2020.107247

Appendices

Appendix A: Sample Codes

Feature transformer Code snippet

```
#Implementation of feature transformer component

class FeatureTransformer(tf.keras.Model):
    def __init__(
        self,
        feature_dim,
        fcs = [],
        n_total = 4,
        n_shared = 2,
        bn_momentum = 0.9,
    ):
        super(FeatureTransformer, self).__init__()
        self.n_total, self.n_shared = n_total, n_shared

        kwrgs = {
            "feature_dim": feature_dim,
            "bn_momentum": bn_momentum,
        }

        # build blocks
        self.blocks = []
        for n in range(n_total):
            # some shared blocks
            if fcs and n < len(fcs):
                self.blocks.append(FeatureBlock(**kwrgs, fc=fcs[n])) # Building shared blocks by providing FC layers
            # build new blocks
            else:
                self.blocks.append(FeatureBlock(**kwrgs)) # Step dependent blocks without the shared FC layers

    def call(self, x, training = None):
        # input passes through the first block
        x = self.blocks[0](x, training=training)
        # for the remaining blocks
        for n in range(1, self.n_total):
            # output from previous block gets multiplied by sqrt(0.5) and output of this block gets added
            x = x * tf.sqrt(0.5) + self.blocks[n](x, training=training)
        return x
```

Gated Linear Unit Code snippet

```
#IMplementation of gated linear unit
def glu(x, n_units=None):
    return x[:, :n_units] * tf.nn.sigmoid(x[:, n_units:])
```

Attentive Transformer Code snippet

```
#implementation of attentive transformer component

class AttentiveTransformer(tf.keras.Model):
    def __init__(self, feature_dim):
        super(AttentiveTransformer, self).__init__()
        self.block = FeatureBlock(
            feature_dim,
            apply_glu=False, # sparsemax instead of glu
        )

    def call(self, x, prior_scales, training=None):
        # Pass input trhough a FC-BN block
        x = self.block(x, training=training)
        # Pass the output through sparsemax activation
        return sparsemax(x * prior_scales)
```

Feature Transformer GLU block Code snippet

```
    Implementation of a FL->BN->GLU block
"""
class FeatureBlock(tf.keras.Model):
        def __init__(
        self,
        feature_dim,
        apply_glu = True,
        bn_momentum = 0.9,
        fc = None,
        epsilon = 1e-5,
    ):
    super(FeatureBlock, self).__init__()
    self.apply_gpu = apply_glu
    self.feature_dim = feature_dim
    units = feature_dim * 2 if apply_glu else feature_dim
    self.fc = tf.keras.layers.Dense(units, use_bias=False) if fc is None else fc #shared-layers
    #get re-used
    self.bn = tf.keras.layers.BatchNormalization(momentum=bn_momentum, epsilon=epsilon)

    def call(self, x, training = None):
        x = self.fc(x) # inputs passes through the FC layer
        x = self.bn(x, training=training) # FC layer output gets passed through the BN
        if self.apply_gpu:
            return glu(x, self.feature_dim) # GLU activation applied to BN output
        return x
```

TabNet using PyTorch implementation

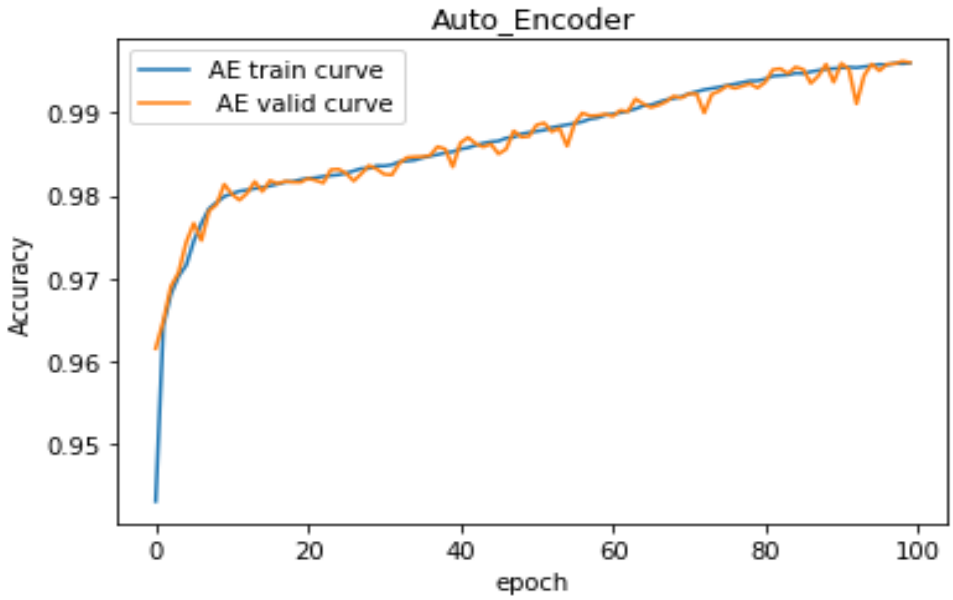
```
pip install pytorch-tabnet

from pytorch_tabnet.tab_model import TabNetClassifier
import torch

tb_cls =TabNetClassifier(seed=42,
                            n_a=77,
                            n_d=77,
                            mask_type='entmax',
                            n_steps=4
                            )

tb_cls.explain

tb_cls.fit(X_train=x_train, y_train=y_train,
              eval_set=[(x_train, y_train),(x_valid, y_valid)],
              patience=10, max_epochs=40,
              eval_metric=['auc','accuracy'],batch_size=9856)



"""**model 2**"""

from sklearn.model_selection import StratifiedKFold
skf=StratifiedKFold(n_splits=21,random_state=42,shuffle=True)

feature=features.to_numpy()

#from sklearn.model_selection import StratifiedKFold
#skf=StratifiedKFold(n_splits=10,random_state=42,shuffle=True)
#predictions_array =[]
CV_score_array=[]
for train_index, valid_index in skf.split(feature,y):
    X_train, X_valid = feature[train_index], feature[valid_index]
    y_train, y_valid = y[train_index], y[valid_index]
    tb_cls = TabNetClassifier(seed=42)
    tb_cls.fit(X_train=X_train, y_train=y_train,
              eval_set=[(X_valid, y_valid)],
              patience=20, max_epochs=20,
              eval_metric=['accuracy'])
    CV_score_array.append(tb_cls.best_cost)
    #predictions_array.append(nm.expm1(tb_cls.predict(X_test)))
nm.mean(CV_score_array,axis=0)
```

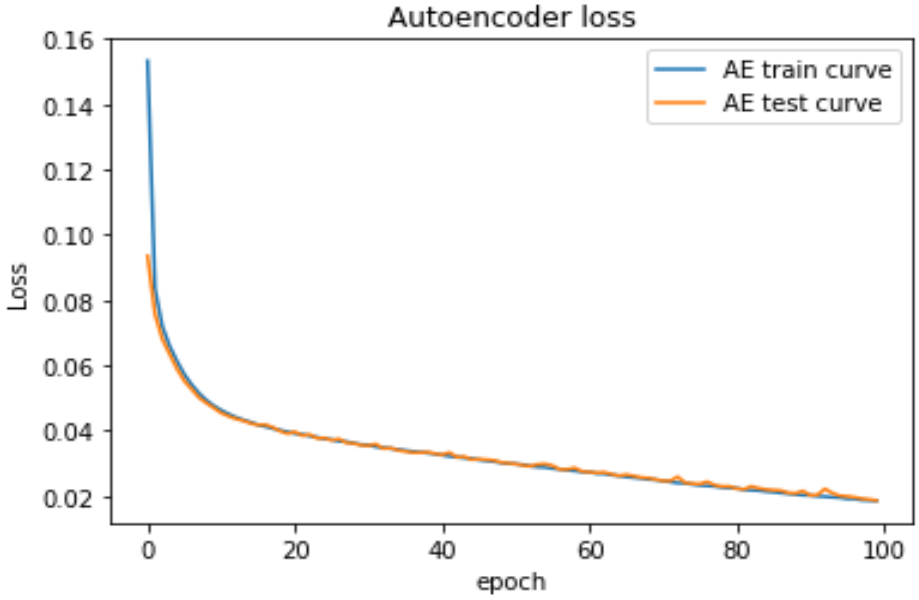
Appendix B: Sample Dataset

	BK	BL	BM	BN	BO	BP	BQ	BR	BS	BT	BU	BV	BW	BX	BY	BZ	CA	CB	CC
49702	1	48	1	48	-1	-1	0	32	0	0	0	0	0	0	0	0	BENIGN		
49703	2	158	2	326	-1	-1	1	32	0	0	0	0	0	0	0	0	BENIGN		
49704	2	104	2	178	-1	-1	1	32	0	0	0	0	0	0	0	0	BENIGN		
49705	3	18	3	43	63463	33372	2	20	50941	0	50941	50941	10000000	0	10000000	10000000	BENIGN		
49706	203	43906	104	72183	29200	1176	101	32	0	0	0	0	0	0	0	0	Web Attack 锘?Brute Force		
49707	7	187	2	92	256	402	6	20	93501	0	93501	93501	58900000	0	58900000	58900000	BENIGN		
49708	2	78	2	288	-1	-1	1	32	0	0	0	0	0	0	0	0	BENIGN		
49709	2	78	2	152	-1	-1	1	32	0	0	0	0	0	0	0	0	BENIGN		
49710	1	0	1	0	227	229	0	32	0	0	0	0	0	0	0	0	BENIGN		
49711	2	12	0	0	613	-1	1	20	0	0	0	0	0	0	0	0	BENIGN		
49712	12	72	12	0	256	80	11	20	23964.091	105.27341	24101	23857	10000000	608.02228	10000000	10000000	BENIGN		
49713	1	0	1	0	227	229	0	32	0	0	0	0	0	0	0	0	BENIGN		
49714	1	0	1	0	256	939	0	32	0	0	0	0	0	0	0	0	BENIGN		
49715	3	0	1	0	29200	28960	0	32	0	0	0	0	0	0	0	0	Web Attack 锘?Brute Force		
49716	1	0	1	0	1444	1174	0	32	0	0	0	0	0	0	0	0	Web Attack 锘?Brute Force		
49717	1	0	1	0	227	229	0	32	0	0	0	0	0	0	0	0	BENIGN		
49718	3	0	1	0	29200	28960	0	32	0	0	0	0	0	0	0	0	Web Attack 锘?Brute Force		
49719	1	48	1	48	-1	-1	0	32	0	0	0	0	0	0	0	0	BENIGN		
49720	3	0	1	0	29200	28960	0	32	0	0	0	0	0	0	0	0	Web Attack 锘?Brute Force		
49721	1	48	1	48	-1	-1	0	32	0	0	0	0	0	0	0	0	BENIGN		
49722	3	0	1	0	29200	28960	0	32	0	0	0	0	0	0	0	0	Web Attack 锘?Brute Force		
49723	1	0	1	0	227	229	0	32	0	0	0	0	0	0	0	0	BENIGN		
49724	3	0	1	0	29200	28960	0	32	0	0	0	0	0	0	0	0	Web Attack 锘?Brute Force		
49725	1	0	1	0	237	235	0	32	0	0	0	0	0	0	0	0	Web Attack 锘?Brute Force		

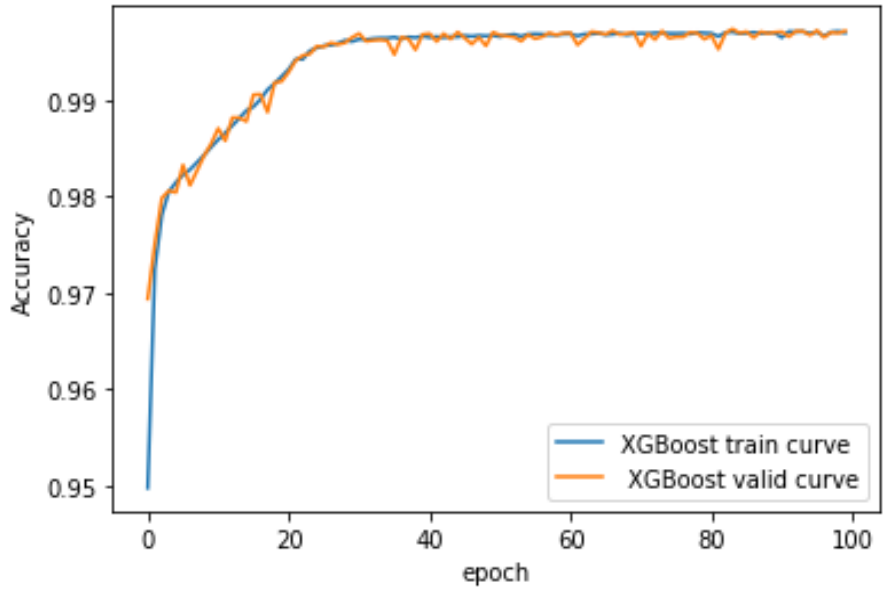
	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
1	n Port	Flow Durat	Total Fwd	Total Back	Total Lengt	Total Lengt	Fwd Packe	Fwd Packe	Fwd Packe	Fwd Packe	Bwd Packet	Bwd Packe	Bwd Packe	Bwd Packe	Flow Bytes,	Flow Packe	Flow IAT M	Flow IAT St
2	389	113095465	48	24	9668	10012	403	0	201.41667	203.54829	923	316	417.16667	231.08095	174.01228	0.6366303	1592893.9	4597264.6
3	389	113473706	68	40	11364	12718	403	0	167.11765	171.91941	1139	126	317.95	208.26129	212.22538	0.9517623	1060501.9	3813685.5
4	0	119945515	150	0	0	0	0	0	0	0	0	0	0	0	0	1.2505678	805003.46	5277836.8
5	443	60261928	9	7	2330	4221	1093	0	258.88889	409.70216	1460	0	603	653.59417	108.70877	0.2655076	4017461.9	15500000
6	53	269	2	2	102	322	51	51	51	0	161	161	161	0	1576208.2	14869.888	89.666667	148.37902
7	123	30352	1	1	48	48	48	48	48	0	48	48	48	0	3162.8888	65.893516	30352	0
8	59135	48	1	1	6	6	6	6	6	0	6	6	6	0	250000	41666.667	48	0
9	5353	89501767	223	0	26257	0	219	45	117.74439	63.991006	0	0	0	0	293.36851	2.4915709	403161.11	2843991.6
10	53	23527	1	1	34	109	34	34	34	0	109	109	109	0	6078.123	85.008713	23527	0
11	53	23978	2	2	46	196	23	23	23	0	98	98	98	0	10092.585	166.81958	7992.6667	13836.777
12	123	107015212	14	14	672	672	48	48	48	0	48	48	48	0	12.558962	0.261645	3963526.4	14300000
13	138	75288199	21	0	4935	0	235	235	235	0	0	0	0	0	65.548121	0.2789282	3764410	16800000
14	137	3378994	205	0	13760	0	68	50	67.121951	3.8868501	0	0	0	0	4072.2179	60.668945	16563.696	166333.52
15	137	3	2	0	124	0	62	62	62	0	0	0	0	0	41300000	666666.67	3	0
16	138	76354195	42	0	9471	0	237	214	225.5	11.639399	0	0	0	0	124.04034	0.550068	1862297.4	11900000
17	53	25117	1	1	45	77	45	45	45	0	77	77	77	0	4857.268	79.627344	25117	0
18	53	25926	1	1	45	106	45	45	45	0	106	106	106	0	5824.2691	77.142637	25926	0
19	53	25525	2	2	68	132	34	34	34	0	66	66	66	0	7835.4554	156.70911	8508.3333	14730.803
20	53	171	2	2	46	196	23	23	23	0	98	98	98	0	1415204.7	23391.813	57	57.419509
21	53	26386	2	2	68	190	34	34	34	0	95	95	95	0	9777.9125	151.59554	8795.3333	15229.634
22	137	1582026	28	0	1400	0	50	50	50	0	0	0	0	0	884.94121	17.698824	58593.556	211129.03
23	53	302	2	2	68	190	34	34	34	0	95	95	95	0	854304.64	13245.033	100.66667	134.7677
24	53	257	2	2	68	132	34	34	34	0	66	66	66	0	778210.12	15564.202	85.666667	106.61301

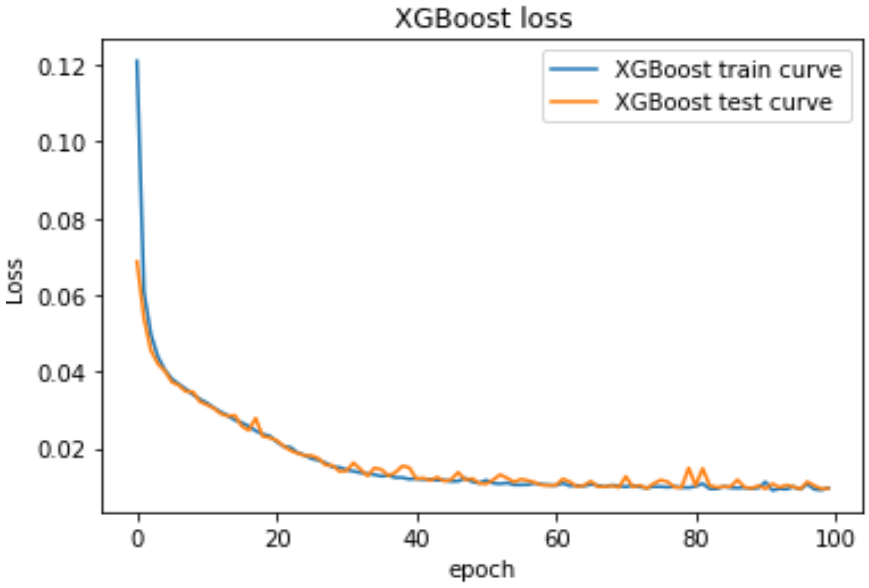
Appendix C: Autoencoder model sample curves





Appendix D: XGBoost sample curves





Appendix E: Sample epochs

```
Epoch 1/100
348/348 [==============================] - 8s 17ms/step - loss: 0.1533 - accuracy: 0.9430 - val_loss: 0.0935 - val_accuracy: 0.9615
Epoch 2/100
348/348 [==============================] - 5s 16ms/step - loss: 0.0835 - accuracy: 0.9643 - val_loss: 0.0757 - val_accuracy: 0.9648
Epoch 3/100
348/348 [==============================] - 5s 13ms/step - loss: 0.0719 - accuracy: 0.9681 - val_loss: 0.0680 - val_accuracy: 0.9690
Epoch 4/100
348/348 [==============================] - 5s 14ms/step - loss: 0.0658 - accuracy: 0.9702 - val_loss: 0.0634 - val_accuracy: 0.9707
Epoch 5/100
348/348 [==============================] - 5s 13ms/step - loss: 0.0612 - accuracy: 0.9716 - val_loss: 0.0589 - val_accuracy: 0.9743
Epoch 6/100
348/348 [==============================] - 5s 13ms/step - loss: 0.0571 - accuracy: 0.9745 - val_loss: 0.0552 - val_accuracy: 0.9766
Epoch 7/100
348/348 [==============================] - 5s 14ms/step - loss: 0.0540 - accuracy: 0.9766 - val_loss: 0.0526 - val_accuracy: 0.9746
Epoch 8/100
348/348 [==============================] - 7s 21ms/step - loss: 0.0514 - accuracy: 0.9784 - val_loss: 0.0501 - val_accuracy: 0.9781
Epoch 9/100
348/348 [==============================] - 11s 31ms/step - loss: 0.0492 - accuracy: 0.9792 - val_loss: 0.0484 - val_accuracy: 0.9789
Epoch 10/100
348/348 [==============================] - 9s 24ms/step - loss: 0.0475 - accuracy: 0.9799 - val_loss: 0.0469 - val_accuracy: 0.9814
Epoch 11/100
348/348 [==============================] - 8s 24ms/step - loss: 0.0462 - accuracy: 0.9802 - val_loss: 0.0454 - val_accuracy: 0.9802
Epoch 12/100
348/348 [==============================] - 7s 20ms/step - loss: 0.0451 - accuracy: 0.9805 - val_loss: 0.0444 - val_accuracy: 0.9795
Epoch 13/100
348/348 [==============================] - 6s 17ms/step - loss: 0.0440 - accuracy: 0.9807 - val_loss: 0.0436 - val_accuracy: 0.9803
Epoch 14/100
348/348 [==============================] - 5s 13ms/step - loss: 0.0432 - accuracy: 0.9809 - val_loss: 0.0430 - val_accuracy: 0.9817
Epoch 15/100
348/348 [==============================] - 5s 13ms/step - loss: 0.0426 - accuracy: 0.9810 - val_loss: 0.0422 - val_accuracy: 0.9805
Epoch 16/100
348/348 [==============================] - 5s 13ms/step - loss: 0.0418 - accuracy: 0.9812 - val_loss: 0.0417 - val_accuracy: 0.9818
Epoch 17/100
```

```
Epoch 85/100
348/348 [==============================] - 5s 13ms/step - loss: 0.0213 - accuracy: 0.9948 - val_loss: 0.0219 - val_accuracy: 0.9956
Epoch 86/100
348/348 [==============================] - 5s 13ms/step - loss: 0.0211 - accuracy: 0.9948 - val_loss: 0.0217 - val_accuracy: 0.9953
Epoch 87/100
348/348 [==============================] - 5s 14ms/step - loss: 0.0209 - accuracy: 0.9950 - val_loss: 0.0216 - val_accuracy: 0.9936
Epoch 88/100
348/348 [==============================] - 5s 14ms/step - loss: 0.0207 - accuracy: 0.9952 - val_loss: 0.0209 - val_accuracy: 0.9945
Epoch 89/100
348/348 [==============================] - 5s 13ms/step - loss: 0.0204 - accuracy: 0.9953 - val_loss: 0.0207 - val_accuracy: 0.9959
Epoch 90/100
348/348 [==============================] - 5s 13ms/step - loss: 0.0202 - accuracy: 0.9954 - val_loss: 0.0215 - val_accuracy: 0.9937
Epoch 91/100
348/348 [==============================] - 5s 13ms/step - loss: 0.0202 - accuracy: 0.9955 - val_loss: 0.0201 - val_accuracy: 0.9960
Epoch 92/100
348/348 [==============================] - 5s 13ms/step - loss: 0.0199 - accuracy: 0.9955 - val_loss: 0.0203 - val_accuracy: 0.9953
Epoch 93/100
348/348 [==============================] - 4s 13ms/step - loss: 0.0197 - accuracy: 0.9955 - val_loss: 0.0221 - val_accuracy: 0.9911
Epoch 94/100
348/348 [==============================] - 5s 13ms/step - loss: 0.0196 - accuracy: 0.9956 - val_loss: 0.0208 - val_accuracy: 0.9946
Epoch 95/100
348/348 [==============================] - 5s 14ms/step - loss: 0.0194 - accuracy: 0.9957 - val_loss: 0.0200 - val_accuracy: 0.9959
Epoch 96/100
348/348 [==============================] - 5s 13ms/step - loss: 0.0191 - accuracy: 0.9959 - val_loss: 0.0198 - val_accuracy: 0.9951
Epoch 97/100
348/348 [==============================] - 5s 13ms/step - loss: 0.0191 - accuracy: 0.9958 - val_loss: 0.0195 - val_accuracy: 0.9959
Epoch 98/100
348/348 [==============================] - 5s 13ms/step - loss: 0.0187 - accuracy: 0.9960 - val_loss: 0.0191 - val_accuracy: 0.9960
Epoch 99/100
348/348 [==============================] - 4s 13ms/step - loss: 0.0186 - accuracy: 0.9960 - val_loss: 0.0189 - val_accuracy: 0.9963
Epoch 100/100
348/348 [==============================] - 5s 13ms/step - loss: 0.0185 - accuracy: 0.9960 - val_loss: 0.0186 - val_accuracy: 0.9961
```