

Developing Flow-Based Intrusion Detection Model for Software Defined Network Using Machine Learning Approach



By: Mohammed Kado Tufa

A Thesis Submitted to Department of Computer Science and
Engineering

School of Electrical Engineering and Computing

Presented in Partial Fulfillment of the Requirement for the Degree of Master's in
Computer Science and Engineering

Office of Graduate Studies
Adama Science and Technology University

February, 2022
Adama, Ethiopia

Developing Flow Based Intrusion Detection Model for Software Defined Networks
using Machine Learning Approach

By: Mohammed Kado Tufa

Advisor: Dr Eng. Frezewud Lemma

A Thesis Submitted to the department of Computer Science and
Engineering

School of Electrical Engineering & Computing

Presented in Partial Fulfillment for the Degree of Masters of Science in Computer
Science and Engineering

Office of Graduate Study

Adama Science and Technology University

February, 2022
Adama, Ethiopia

DECLARATION

I hereby declare that this Master Thesis entitled “**Developing Flow Based Intrusion Detection Model for Software Defined Networks using Machine Learning Approach**” my original work and has not been presented as a partial degree requirement for a degree in any other university and that all sources of materials used for the thesis have been duly acknowledged through citation.

Mohammed Kado

Name

Signature

Date

RECOMMENDATION

I, the advisor of this thesis, hereby certify that I have read the revised version of the thesis entitled **“Developing Flow Based Intrusion Detection Model for Software Defined Networks using Machine Learning Approach”** prepared under my guidance by Mohammed Kado Tufa submitted in partial fulfillment of the requirements for the degree of Master of Science computer science and engineering. Therefore, I recommend the submission of a revised version of the thesis to the department following the applicable procedures.

Dr Eng. Frezewud Lemma

Major Advisor

Signature

Date

APPROVAL PAGE

I/we, the advisors of the thesis entitled “**Developing Flow Based Intrusion Detection Model for Software Defined Networks using Machine Learning Approach**” and developed by Mohammed Kado Tufa, hereby certify that the recommendation and suggestions made by the board of examiners are appropriately incorporated into the final version of the thesis.

_____	_____	_____
Major Advisor	Signature	Date

We, the undersigned, members of the Board of Examiners of the thesis by Mohammed Kado read and evaluated the thesis entitled “**Developing Flow Based Intrusion Detection Model for Software Defined Networks using Machine Learning Approach**” 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.

_____	_____	_____
Chair Person	Signature	Date

_____	_____	_____
Internal Examiner	Signature	Date

_____	_____	_____
External Examiner	Signature	Date

Finally, approval and acceptance of the thesis is contingent upon 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

ACKNOWLEDGMENT

First, I thank the almighty GOD for his blessing me with good health and wellbeing which made me successful in my all-academic pursuits. Then, I would like to express my profound gratitude to Dr Eng. Frezewud Lemma, whose expertise was invaluable in giving the right direction, useful comments, and engagement from the beginning till the end. He continuously encourages me and always has been ready and enthusiastic to assist in any way he could throughout the process. I would also like to thank all my colleagues and classmates for that they shared their knowledge with me. Their advice and encouragement are vital to keeping me on track. Also, I would like to thank my beloved and missed mother, whom I lost amid my writing, for wishing me a bright future even in her worst days. And for my dad, for being light in my doomsday. Furthermore, I could not have completed this paper without the support of my friends, relatives, and neighbors of mine, I thank them all very much!

Contents

ACKNOWLEDGMENT	i
List of Figures	v
List of Table	vii
List of Acronomy	viii
ABSTRACT	x
CHAPTER ONE	1
1.1. Background	1
1.2. Motivation	2
1.3. Statements of the problem	3
1.4 Research Question	4
1.4.1. General Objective	4
1.4.2 Specific objectives	4
1.5. Significance of the Study	5
1.6. Scope and Limitations of the Study	5
1.6.1 The Scope of the Study	5
1.6.2 Limitation of the Study	5
1.7. Organization of the Thesis Work	5
CHAPTER TWO	7
LITERATURE REVIEW	7
2.1 Overview of Intrusion and Flow-based IDS	7
2.2 Categories of IDS	9
2.2.1 Anomaly-Based IDS	10
2.2.2 Signature-Based IDS	11
2.2.3 Network-Based IDS (NIDS)	11
2.2.4 Host-Based IDS (HIDS)	12
2.3 Types of Attack	13
2.4 Software defined network (SDN) based Network intrusion detection system (NIDS)	15
2.4.1 Software Defined Network Architecture and Applications	15
2.5 Artificial Intelligence	16
2.5.1 Machine Learning (ML) in Network Intrusion detection system	17
2.6. Some Popular Classification Algorithms	21

2.7 Feature Selection and Reduction in Machine Learning.....	24
CHAPTER THREE	28
METHODOLOGY	28
3.1 General Approaches	28
3.2 Research Methodology	29
3.3 DeepIDS System Architecture	30
3.4 DeepIDS Module Types.....	32
3.5 Material and Tools.....	33
3.5.1 Software Tools.....	33
3.5.2 Hardware Tools.....	35
3.6 Initialization.....	35
3.7 Research Methodology development.....	36
3.7.1 Deep Learning	36
3.7.2 Notations	36
3.7.3 Deep Neural networks	37
3.7.4 SimpleRNN-Based Models	39
3.7.5 Convolution Neural Networks Based Models (CNNs).....	39
3.8 Dataset.....	40
3.8.1 NSL-KDD Dataset.....	40
3.8.2 Dataset Preprocessing.....	44
3.9 Deep Learning Experimental Setup	45
3.10 Network Performance Evaluation Setup	49
3.11 Preprocessing.....	52
3.12 Model Evaluation	53
3.13 Hyperparameter Tuning	54
3.14. Evaluation Metrics to Evaluate the Accuracy of Model.....	54
CHAPTER FOUR.....	57
IMPLEMENTATION	57
4.1. Introduction.....	57
4.2. Environment.....	57
4.3 System Architecture.....	57
4.3.1. Preprocessing for NSL-KDD Dataset.....	58
4.3.2. Feature Extraction and Selection	60
4.3.3. Feature Set for Anomaly NIDS.....	60

4.3.4. Data Split	69
4.4. Attack Types Category	72
4.6. Build Intrusion Detection Net Model -	82
CHAPTER FIVE	85
RESULTS AND DISCUSSION	85
5.1. RESULTS AND DISCUSSION	85
5.1.1 DeepIDS: Deep Learning Approach for Intrusion Detection in Software Defined Networking	85
5.1.2 Performance of a Classification	86
5.1.2.1 Confusion matrix of Deep Learning Algorithms	86
5.2. Effect of the Learning Rate	92
5.3. Efficiency Evaluation	94
5.4. Network Performance Evaluation	94
5.4.1 Throughput Evaluation	95
5.4.2 Latency Evaluation	96
5.5. Experiment Testbed Setup on Mininet Emulators	97
CHAPTER SIX	98
6.1 Conclusion	98
6.2. Summary	99
6.3. Recommendation	100
6.4. Future Work	100
Reference	101
Appendix A	108
Python Code	108

List of Figures

Figure 2.1 Packet-Based Intrusion detection [11].	8
Figure 2.2 Flow-Based Intrusion Detection [11].	9
Figure 2.3 Categories of IDS from four different points of view [69].	10
Figure 2.4 A flow-based IDS is installed at three LANs [42].	12
Figure 2.5 Host Intrusion Detection System [40].	13
Figure 2.6 SDN Architectures [74].	16
Figure 2.7 Deep learning [90].	21
Figure 2.8 Support vector machine for linearly separable data [54].	23
Figure 2.9 Random Forest [54].	24
Figure 3.1 Research Methodology	29
Figure 3.2 Proposed System Architecture	31
Figure 3.3 Network Intrusion Detection Framework [81].	32
Figure 3.4 Digital Library for the proposed research	33
Figure 3.5 Some open-source library and our thesis Environment [33].	35
Figure 4.1 NSL KDD dataset set preprocessing architectures.	59
Figure 4.2 Feature Selection	60
Figure 4.3 NSL-KDD dataset Preprocessing	61
Figure 4.4 Dataset before Preprocessing	62
Figure 4.5 Review of the Feature Type	62
Figure 4.6 Label distribution in train and test set	64
Figure 4.7 Dataset preprocessing.	65
Figure 4.8 Transform categorical feature.	66
Figure 4.9 One-hot-encoding for categorical data	66
Figure 4.10 Final NSL-KDD dataset	67
Figure 4.11 Feature Scaling	68
Figure 4.12 RNN model Accuracy	69
Figure 4.13 Split the training set and testing set.	70
Figure 4.14 Splitting the Data frame.	71
Figure 4.15 Feature Scaling for preprocessing data.	71
Figure 4.16 Attack Classification Distribution	72
Figure 4.17 DOS Accuracy Comparison Jupyter Code	73
Figure 4.18 U2R Accuracy Comparison Jupiter Code	74
Figure 4.19 U2R figure plotting jupyter Code.	74
Figure 4.20 U2R Accuracy Comparison graph.	75
Figure 4.21 U2R Attack Category jupyter Code	75
Figure 4.22 Confusion matrix	75
Figure 4.23 Creating Neural Network	76
Figure 4.24 ANN Model fitting Output	76
Figure 4.25 Creating Decision tree Classifiers	77
Figure 4.26 Confusion Matrix for decision tree.	77
Figure 4.27 Performance Evaluation of decision tree.	78
Figure 4.28 Performance Evaluation results for decision tree	78
Figure 4.29 Performance Evaluation results for K-nearest neighbor	79
Figure 4.30 Performance Evaluation results for naïve Bayes.	79
Figure 4.31 Performance Evaluation results for support vector machine.	80

Figure 4.32 Intrusion Detection CNN Model	82
Figure 4.33 Model Accuracy	83
Figure 4.34 Model Loss	84
Figure 5.1 Confusion matrix for SimpleRNN.....	87
Figure 5.2 Confusion matrix for LSTM.....	87
Figure 5.3 Confusion matrix for GRU	88
Figure 5.4 Confusion matrix of DNN3	88
Figure 5.5 Confusion matrix of DNN4	89
Figure 5.6 Confusion matrix of CNN	89
Figure 5.7 AUC-ROC of Model	91
Figure 5.8 Multiple ROC in Single Plot	92
Figure 5.9 Throughput Evaluation (log scale on x-axis)	95
Figure 5.10 Latency Evaluation (log scale on x-axis)	97
Figure 5.11 The Emulated Network Topology	97

List of Table

Table 2.1 Comparison of packet-based and flow-based intrusion detection system [36].	9
Table 2.2 Categories of different types of attacks [88].	14
Table 2.3 Contribution and gaps of the related work.....	26
Table 3.1 The NSL-KDD Dataset Distribution	42
Table 3.2 NSL-KDD Dataset Features [71].	42
Table 3.3 The NSL-KDD Dataset Example	44
Table 3.4 Attack in the NSL-KDD Dataset	44
Table 3.5 Network parameter	50
Table 3.6 Confusion matrix	56
Table 4.1 Accuracy Evaluation on U2R attack.....	81
Table 4.2 Efficiency Evaluation	81
Table 5.3 Running Time Evaluation	94

List of Acronomy

Adam	Adaptive Moment Estimation
AI	Artificial Intelligence
ACC	Accuracy
AE	Autoencoder
AIDS	Anomaly Intrusion Detection System
AUC	Area under Cover
CIA	Confidentiality, Integrity, and Availability
CNN	Convolutional Neural Network
CFS	Correlation based Feature Selection
CPU	Central Processing Unit
CSV	Comma-Separated Values
DAE	Denoising Auto encoder
DBNs	Deep Belief Nets
DR	Detection rate
DNN	Deep Neural Network
DL	Deep Learning
DoS	Denial of Services
DDoS	Distribution Denial of Services
DPI	Deep Packet Inspection
ELU	Exponential Linear Unit
FAR	False Alarm Rate
FPR	False Positive Rate
FS	Feature Selection
FTP	File Transfer Protocol
GR	Gain Ratio
GRU	Gated Recurrent Unit
TF	Tensor flow
U2R	User to remote
HIDS	Host-Based Intrusion Detection
HTTP	Hypertext Transfer Protocol

IDS	Intrusion Detection System
IG	Information Gain
IP	Internet Protocol
IUT	International Telecommunication Union
LAN	Local Area Networks
MAC	Media Access Control
ML	Machine Learning
ANN	Artificial Neural Network
NIDS	Network Intrusion Detection System
NSL-KDD	Network Security Laboratory-knowledge- Knowledge discovery in databases
MSE	Mean Square Error
NN	Neural Network
NIDS	Network Intrusion Detection System
OOPs	Object-Oriented Programming
P	Precision
PCA	Principal Component Analysis
R	Recall
R2L	Remote to user Attack
ROC	Receiver Operating Characteristics
RNN	Recurrent Neural Network
SOM	Self-Organizing Map
SGD	Stochastic Gradient Descent
SDN	Software Defined Network
SIDS	Signature Based Intrusion Detection Systems

ABSTRACT

With the rapid growth use of information technology today, hacking and other unauthorized activities have dynamically increased than ever before. Private and organization data and various resources found with the software-defined network controllers are vulnerable to malicious attacks. An intrusion detection system using Machine learning models plays a very important role in protecting computer network security. Various machine learning and Deep learning algorithms have been applied in network intrusion detection systems. Based on the literature review, many intrusions detection-related papers are specific to machine learning-based IDSs are developed. However, the presents IDSs still have their own drawbacks such as low processing speed, low detection rate, high training time, and relatively high false alarm rate (FAR). Most SDN-based IDS use the signature-based approach that detects only known attacks and needs continuous updating of the database to detect new attacks in the SDN network controllers. To overcome these challenges, we propose efficient flow-based IDS using Machine learning. The proposed research work focused on developing of an intrusion detection model with better detection rates, less training time, and improved performance by performing the training in parallel and selecting the most effective parameter for the model that can improve the performance of the models. The model learned the abstract and high dimensional feature representation of the NSL-KDD by passing them into many hidden layers. CNN and its variant architectures have significantly performed well in comparison with different classical machine learning classifiers. The experimental results show that the performance of the proposed flow-based anomaly intrusion detection system for software defined network using Deep learning models is significantly better than other network intrusion detection models, which achieved the best detection accuracy 99.61%, ROC 99%, short training time, low model loss, low reconstruction error, low false-positive rate, and low overfitting. Through rigorous experimental testing, it is confirmed that the flow-based anomaly intrusion detection system for software defined networks using a Machine Learning and Deep Learning algorithm performs well in comparison to the previous studies.

Keyword: Deep Learning, Machine Learning, Intrusion Detection System, Convolutional Neural Network, Deep Neural Network.

CHAPTER ONE

INTRODUCTION

1.1. Background

Software-Defined Networks (SDN) is a developing area that potentials to exchange the manner we layout and build function network architecture for this reason making it best for the high-bandwidth, Dynamic nature of today application and networks. SDNs are presently being deployed in several network environments, from small office home office (SoHo) and enterprise network to datacenter. The capability of SDN (Logically centralized controller and universal network overview) helps to solve numerous security issue in traditional network and products.

The ability to govern traffics at an excellent-grained degree, open networking foundation (ONF) defines SDN as an evolving architecture that is dynamic, controllable, cost-powerful, and adaptable [1]. The tendency to shift from traditional network structure of proprietary based to Openflow switch and programmable network structure is very useful in terms of security, resource utilization, manageable, adaptable and etc.

However, this new innovative and enhanced technology has its own limitation of network security [2]. Several attacks can have an impact on the SDN architectures, for instance, and Distributed Denial of Service (DDOS) attacks can overwhelm an SDN controller and a communications channel with synthetic service calls. A Man-in-the-Middle attack can break connections between the controller and the Openflow switch and claim controller of the network. Because of the extensive type styles of SDN deployments, SDN security is an extreme issue and has currently been notably researched. For that reason, there is a need to develop an effective network intrusion detection system (NIDS). Several machine learning (ML) approaches have been proposed to secure SDNs. Given a set of training data, ML algorithms learn a non-linear pattern of training data and then detect the attack insides the SDNs.

The network weakness has become more open to intruders: the emphasis is now shifted to a single point of failure where the central controller could be a prime target. The combination of the flow-based intrusion detection system (IDS) into the SDN architecture is vital to deliver a network with attack countermeasure. Signature-based Snort flow-based IDS is installed for traffic monitoring and attack detection, by mirroring the traffic destined to the servers [9]. The weakness assessment shows possible attacks threat exist within the network structure and efficiently contain by Snort flow-based IDS except for the few which the suggestion is created for possible mitigation. To

provide scalable threat detection in the SDN controller's architecture, a flow-based IDS model is developed. A flow-based anomaly detection system is implemented with machine learning to overcome the limitation of signature-based IDS. Finally, we conclude the problem of detecting SDN network traffic. By using some machine learning algorithms such as supervised and unsupervised learning methods to detect the network traffic.

Machine Learning (ML) algorithms have been implemented in the SDN-based Network intrusion detection system (NIDS) to protect computer networks and to overcome network security issues. We reviewed various recent works on machine learning (ML) methods that leverage SDN to implement NIDS. We evaluated techniques of deep learning in developing SDN-based NIDS, covered tools that can be used to develop the NIDS model in an SDN environment. Organizations deploy a firewall, antivirus software, and intrusion detection system. The critical success factors for NIDS are abnormality detection speed, accuracy, and reliability. Deep learning approach can be implemented in the SDN controllers to enhance network monitoring and security.

1.2. Motivation

It is always difficult to detect traffic from the system log file server unless by using some machine learning/Deep learning algorithm to detect the traffic whether it is normal or abnormal. Software defined networking (SDN) is developing an architecture that is dynamic, manageable, cost-effective, and adaptable, therefore making it ideal for the high bandwidth, dynamic natures of today's applications and networks.

It helps to solve numerous security problems in an outdated network and brings the ability to control network traffic at a fine-grained level. The SDN architectures themselves also introduce some new attack threats and security vulnerability's [1] introduce threat vectors in software defined network.

Because of the wide of types of SDN deployments, SDN security is a serious concern and has recently been extensively researched –see [2] and [3] for more detail. Therefore, there is a need to improve an efficient network intrusion detection system (NIDS). Several Machine learning (ML) approaches have been proposed to secure the SDNs. Given the training dataset, ML algorithms learn a non-linear pattern of training data and then detect attacks inside the SDNs. These algorithms can be statistical [4], [5], probabilistic [6], or support vector machines [7].

Despite the high accuracy and performance obtained in several fields, ML algorithms used in anomaly intrusion detection system tend to have some limitation.

Machine Learning/ Deep Learning (ML/DL) has emerged to improve the intrusion detection rate and achieved significant success in the field of speech recognition [8], image recognition [9], and natural language processing [10]. Motivated by the development of ML, we extended this research trend to a DL-based intrusion approach for SDN context. We believe the DL approach is a promising method for the next generation of anomaly intrusion detection systems in SDN. The Deep Learning approach can be used, and so we can achieve a high detection Accuracy rate. The flow-based NIDS and programmable natures of SDNs also help the development of NIDS.

1.3. Statements of the problem

There are problems for flow-based anomaly intrusion detection systems in the SDN architecture as flows, network dynamic, varied, and constantly changing [3]. In addition, network attacks keep evolving and become smarter and more aggressive. The dynamic nature of network traffics makes intrusion detection enormously challenging. Traditional NIDS (network intrusion detection system) uses a huge number of hand-crafted features to improve intrusion detection accuracy.

Therefore, it is a challenge to improve intrusion detection accuracy with these limited raw flow features. NIDS (network intrusion detection system) is supposed to improve real-time intrusion detection and mitigation, therefore computational complexity and network overheads also need to be considered. The software defined network architectures are still new technology, so network datasets for it are either quite rare and/ or unpublished [23], [24].

As a result, it is challenging to train and evaluate a model in a supervised manner to detect intrusion in the SDN network. This is to investigate Machine Learning/Deep learning methods that can provide agile and versatile characteristics to network Intrusion detection systems. These datasets are used for training and testing systems for intrusion detection. The availability of categorized network datasets is a big problem. One of the significant challenges of developing a NIDS is ensuring Robustness and effectiveness [74].

Based on the literature review, many intrusions detection-related papers specific to machine learning-based IDSs have been developed. However, the existing IDSs still have their own shortcomings. Some of the shortcomings are low detection rate, high training time, low processing speed, relatively high false alarm or positive rate (FAR), and most SDN-based IDS use the signature-based approach that detects the only known attack and need continuous updating of the database to detect new attacks in the SDN network controllers. To overcome these challenges, in recent years, there have been various attempts to propose efficient SDN-based IDS. The proposed

research work focused on developing an intrusion detection model with better detection rates, less training time, and improved performance by performing the training in parallel and selecting the best parameter for the model that can improve the performance of the models.

1.4 Research Question

RQ1. How the flow-based Network Intrusion Detection facilitates the development of a Software defined network?

RQ2. How to improve the intrusion detection accuracy with the limited raw feature in software defined network architecture using Machine Learning and Deep Learning?

RQ3. How to evaluate the performance of the system in terms of throughput and latency?

1.4. Objective

1.4.1. General Objective

The general objective of the study is to improve the detection accuracy and reduce the false alarm rate of flow-based Intrusion Detection Model for Software Defined Network using a Machine Learning approach.

1.4.2 Specific objectives

The following are the research specific objectives that help to achieve the general objective of the study.

- ❖ To improve intrusion detection for software defined network using machine learning and deep learning algorithms
- ❖ To use machine learning and Deep learning techniques to detect the Intrusion detection type.
- ❖ To design model for intrusion detection of the software defined network.
- ❖ To evaluate performance of the system in terms of throughput and latency.

1.5. Significance of the Study

The result of this research can be applied to any software defined network-based systems which their data and service are prone to intrusion, hackers such as online shopping system, commercial banks, and cloud infrastructure, military, different online service providers, etc. The proposed system stays on the main network line, learn normal network flow behavior, developed a model, examine incoming network traffic then identify the attack or anomaly that has been known or zero-day attack which is unknown to the model, and finally, the alarm be generated.

1.6. Scope and Limitations of the Study

1.6.1 The Scope of the Study

This study mainly focuses on developing a flow-based anomaly intrusion detection system for software defined network. Flow-based anomaly intrusion detection system model implements for the classification of network traffic into normal and attack based on the anomaly approach using machine learning and Deep learning techniques. The classification uses both ML and DL algorithms that are selected from the PySpark software for the implementation and compared their result based on different classification metrics. To train and test these models, feature selection methods are utilized to determine the most discriminant features for the collected dataset. The trained models Classify the input data into binary (Normal or attack) and then if the class label is attacking the input data can be categorized into five classes.

1.6.2 Limitation of the Study

Even though a flow-based anomaly intrusion detection system has the capacity of identifying zero-day and know attacks, it doesn't identify outliers within the payload packet. One of the limitations of this research work is that only one dataset (which contains network traffic dataset) is used to build models that can classify the input data into normal or attack. Also, one feature selection method is applied to the dataset to select relevant features for model building and testing.

1.7. Organization of the Thesis Work

This section of the document summarizes all the number of chapters used and the overall contents of the thesis work as follows:

Chapter One- Introduction: this chapter covers the background of the study, statement of the problem, objective of the study, methodology (which is discussed under chapter three), Significance of the study, scope, and limitation of the study.

Chapter Two- Literature Review: This chapter covers over view of intrusion and flow-based IDS, intrusion detection system category, feature selection, anomaly or signature-based IDS, network or host-based IDS, types of network attacks, centralized and distributed IDS, passive and active IDS, the component of IDS, SDN-based NIDS and application, AI, ML, and DL, 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, materials be discussed.

Chapter Four- Implementation: Under this chapter preprocessing and modeling concept discussed in detail.

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

Chapter Six- Conclusion and Recommendation: this is the last chapter of the study, it contains the conclusion of the study, recommendations forwarded by the researchers

CHAPTER TWO

LITERATURE REVIEW

The Overview of Flow-based IDS, SDN and Machine learning especially DL, common approaches of IDS along with their strengths and weakness, the basic component of IDS, network attacks, general concepts of machine learning, review some of the classification and intrusion detection of the algorithm are discussed. Then flow-based feature selection methods in machine learning are discussed. Finally, some classification and intrusion detection algorithm evaluation metrics and parameter optimization metrics are given. These extensive works of literature have been reviewed to understand the problems with these works and to identify a solution in this work.

2.1 Overview of Intrusion and Flow-based IDS

Intrusion is defined as malicious activity that compromises the security of a software define network controller or computer system either external or internal without the mandatory privilege access [31]. It causes loss of the CIA (Confidentiality, Integrity and Authentication) of a network and computer system. For instance, an intrusion may compromise the CIA of a network by gaining root-level access and then modifying and stealing the information of the network.

A flow-based Network Intrusion Detection System is developed in a network to detect threats from monitoring packets transmitted through. IDSs detect anomalous and malicious activities from inside and outside intruders [32]. A flow-based IDS needs to deal with problems of such network traffic volumes and highly uneven data distribution.

The primary function of a flow-based IDS is to monitor information sources, such as computers or networks, for unauthorized access activities. IDSs collect data from different systems and network sources and analyze the data for possible threats [32]. In today's world of ever-increasing internet connectivity, there is an ongoing threat of intrusion, denial of service attack, or other abuses of computer and network resources [32]. To overcome these problems, there is various network security measure such as firewalls, encryption, and antivirus and so on. A flow-based IDS is another mechanism to secure computer systems or networks. Flow-based IDSs monitor the events occurring in a computer system and network for analyzing the pattern of intrusion [32].

Flow-based Intrusion Detection System: Based on the drawback of Packet-based intrusion detection mentioned below flow-based intrusion detection systems come up with a counter solution. Flow-based intrusion detection is based on information on network interaction rather than

payload and it is also aggregated information. For many attacks flow information is enough to identify them [86]. According to [86, 87] flow-based intrusion detection uses only information of network interaction, it's enough to identify many attacks [86]. Also, flow-based intrusion detection can detect an anomaly-based attack.

There is an attack that can be detected based on flow, such as Denial of Service (DoS), computer worms, network probing, and flooding [87]. Flow-based anomaly intrusion detections have been proposed by researchers who have a great grasp of the domain, existing issues with current networks, and packet-based intrusion detection systems. Flow-based anomaly detection depends on the modeling of the normal behavior of flows and detecting deviations from normal behavior. According to [87] flow-based intrusion detection system has different components and processes. The comparison between the two detection systems is illustrated by Table 2.1 below.

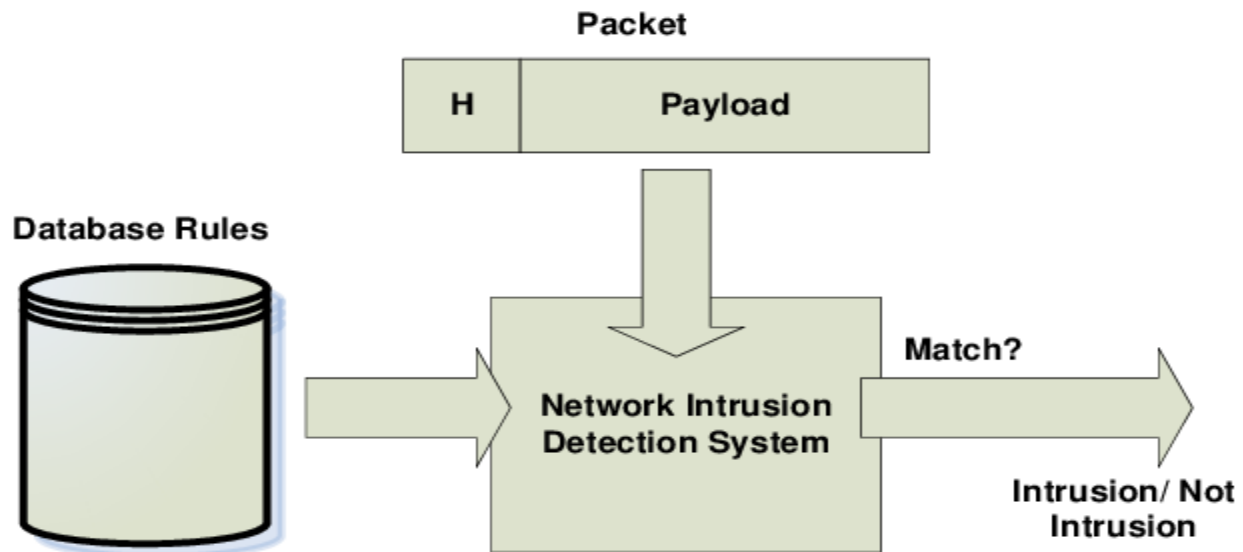


Figure 2.1 Packet-Based Intrusion detection [11].

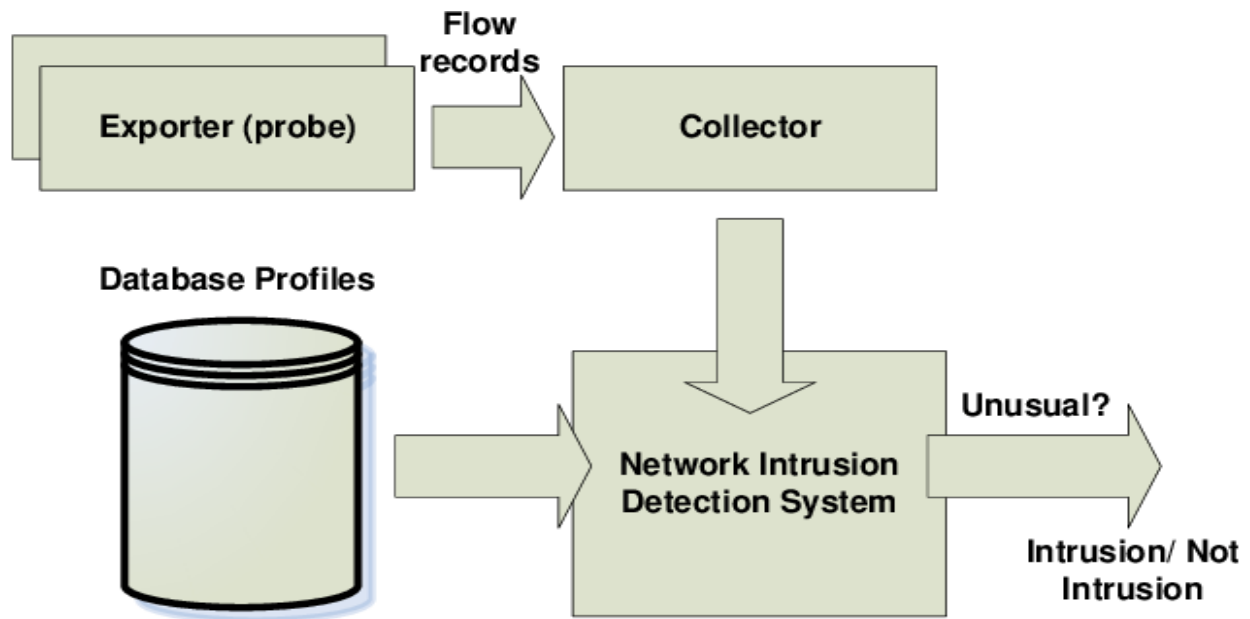


Figure 2. 2 Flow-Based Intrusion Detection [11].

Table 2. 1 Comparison of packet-based and flow-based intrusion detection system [36].

Flow-Based NIDS	Packet Based NIDS
Inspect information of network flow (Information about connections)	Inspect the whole payload besides headers
Mostly anomaly-based NIDS	Mostly Signature-based NIDS
Known as “Network Behavior Analysis”	Known as “Deep Packet Inspection” (DPI) or Knowledge base

2.2 Categories of IDS

IDS can be categorized into various categories based on different criteria as shown in Figure 2.3 [24], [33]. Based on the intrusion detection approach, IDS is classified as anomaly-based and signature-based IDS; based on the source of information as host-based and network-based IDS, and based on structure (or operations) IDS can be categorized as centralized and distributed IDS. In centralized IDS, all operations are performed at a central location while in distribution IDS operations are distributed among the host of the system.

Based on the response after the detection of attacks IDS can also be classified into passive and active IDS. Passive IDS only raise an alarm and does not take any correctives actions after the

detection of attack whereas active IDS raise an alarm and can take corrective action after detection of the attack. The detailed discussion of anomaly-based, signature-based, network-based, and host-based IDSs is presented in the next sections.

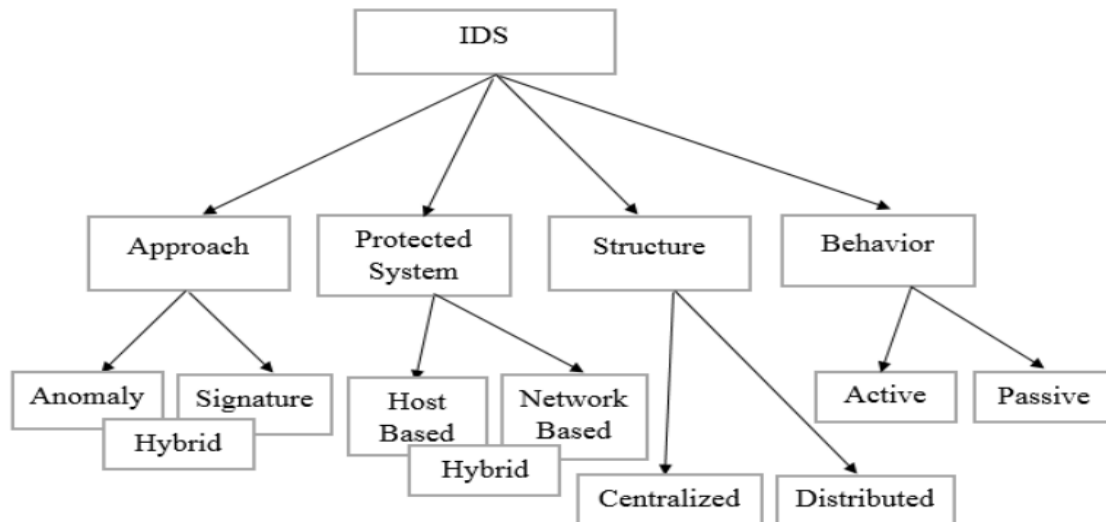


Figure 2. 3 Categories of IDS from four different points of view [69].

2.2.1 Anomaly-Based IDS

Anomaly attempt to break a computer system or a network by applying a malicious activity to gain access to the system or compromise it [34]. Such events must be detected proactively to guarantee the CIA of the resource of a computer system. Anomalies may be caused due to many reasons such as malicious activities that are initiated by attackers or intruders, misconfiguration of network or a host, overloading of network infrastructure, and malfunctioning of network device [35].

Therefore, detecting network attacks is an important step in preventing these dangerous events.

Anomaly-based intrusion detection is one of the approaches to flow-based IDS that enable us to detect and classify network anomalies.

An anomaly detection approach makes use of profiles that, is type of intrusion detection approach could also be feasible even if there is a lack of signature pattern and also works in the condition of the non-regular pattern of traffic. One of the drawbacks of anomaly-based intrusion detection is defining the rules sets. The efficiency of the system depends on the effective implementation and testing of the rules sets on all the protocols. The various protocols can make rule defining complex. However, with the proper incorporation of rules and protocols, the flow-based anomaly detection approach can be performed more efficiently [36].

2.2.2 Signature-Based IDS

In contrast to the anomaly detection system, which triggers alarms based on deviation from the normal network behavior, the signature detection system, also known as misuse detection, triggers alarm based on the characteristic signature of known attacks [37]. Here, a signature means a set of characteristics such as IP numbers, TCP flags, port numbers, logons to a network, and other events on the network. This approach identifies instances of attacks by comparing current activity against the signatures of known attacks stored in a database and it generates an alarm if any malicious activity is found.

It is more efficient to detect attacks whose patterns are already maintained in the signature-pattern database. The model usually doesn't generate many false alarms since it is specifically designed to detect known attacks. However, this approach cannot detect unknown attacks that are not in the model which results in a low detection rate [38]. Another problem with this approach is that the system needs to be continuously updated to add other attacks. Even if there exist well-designed signatures, signature-based IDS can't detect unknown attacks due to the lack of an adequate signature database.

This approach can be used by organizations that are more concerned with monitoring known attacks from intruders trying to access internal hosts from the internet.

A few researchers such as [39, 40, and 41] utilized a hybrid intrusion detection (the combination of an anomaly and signature detection).

2.2.3 Network-Based IDS (NIDS)

A network-based IDS (NIDS) is a software application that monitors and detects network traffic for malicious activity attempted to break into or compromise a network [35]. A NIDS is placed at planned points within the software defined network to monitor traffic to and from all devices on the SDN controllers. It performs analysis of passing traffic on the entire SDN controllers and matches the traffic that is passed on the SDN controllers for identification attacks. Once an attack is recognized, an alert can be sent to the controllers.

For example, as can be from Figure 2.4 [42]. A flow-based IDS is installed at three LANs in the network. The flow-based IDS installed at each LAN attempt to determine unauthorized access to the LANs by analyzing traffic that traffic that passes through the network (LAN) to several hosts. The flow-based IDS read both inbound and outbound packets and analyze the packets for

suspicious pattern and if any malicious activity is found, an alarm is generating is generate to inform administrator something is wrong the network and then to take corrective action [44].

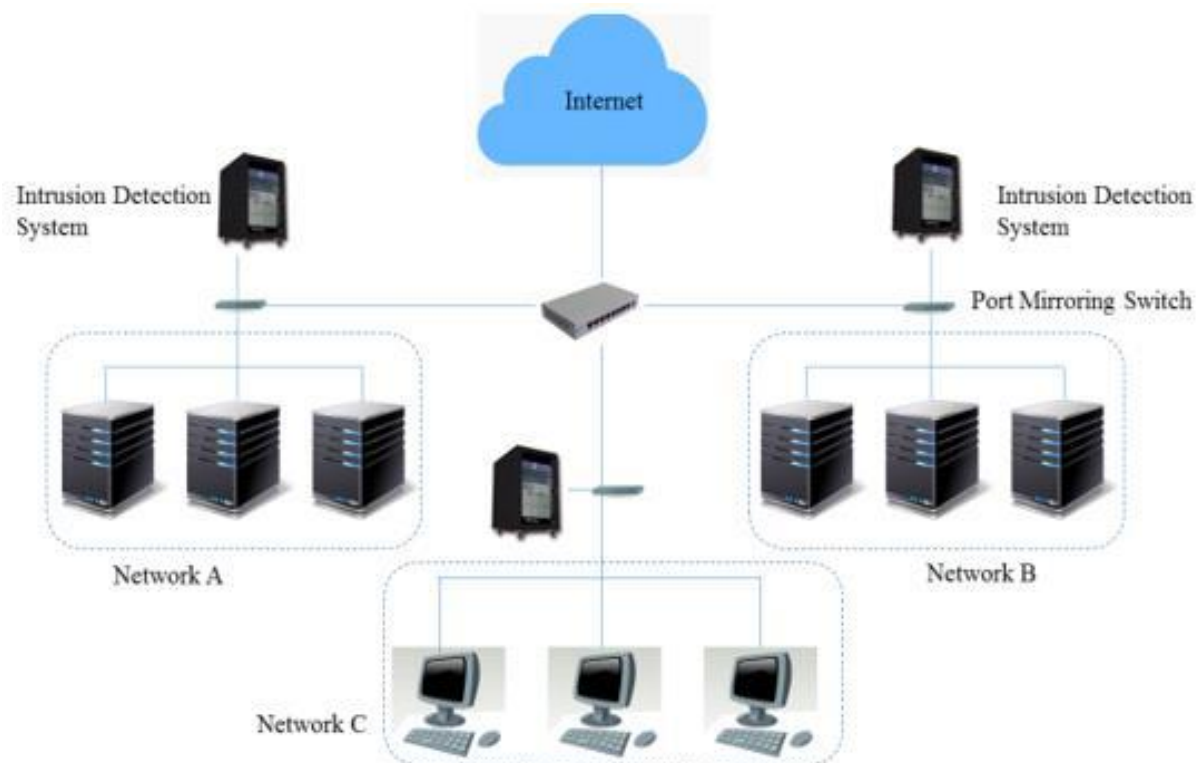


Figure 2. 4 Flow-based IDS is installed at three LANs [42].

NIDS is composed of a set of components that are specialized for network use such as sensors, management servers, a command console, and a database of signatures to examine network traffic on segments [44]. The part of the sensor is to try to filter data and remove any irrelevant information obtained from the occurrence set associated with the protected system, in that way detecting suspicious activities.

2.2.4 Host-Based IDS (HIDS)

A host-based IDS (HIDS) is an IDS that operates within a computer, node, or device. It mainly monitors and analyzes the internals of a computer or system to check if a computer or system is compromised by malicious activity [43]. Once suspicious activity is detecting, the HIDS can either generate an alarm or place the encountered suspicious activity in the logs.

Unlike NIDS where it is designed to monitor and detect attacks flowing over the network, in HIDS a trained model or agent is placed on every host (could be a web server, computer, firewall, SDN controllers, or router) to classify intrusion by examining system calls, application logs, file-system change such as password file and other host actions. Host-based IDS can detect internal attacks as well [45]. HIDS is deployed only on sensitivity or critical hosts because it is expensive to place on every host in the network.

In summary, HIDS provides only data relating to the host on which it is installed, not the network as a whole. The drawback of this IDS is that installing the software on several hosts on the network takes time and can be more expensive.

NIDS and HIDS can be combined to give better performance by utilizing the strength of the approaches integrated to overcome the limitations of individual techniques. In hybrid IDS, the NIDS collected and processes network traffic to detect attacks and the HIDS processes the packets addressed on each host. Even though this integration is useful, getting a disparate system to work in a coordinated manner and analyzing data gathered from multiple systems is somewhat difficult [45].

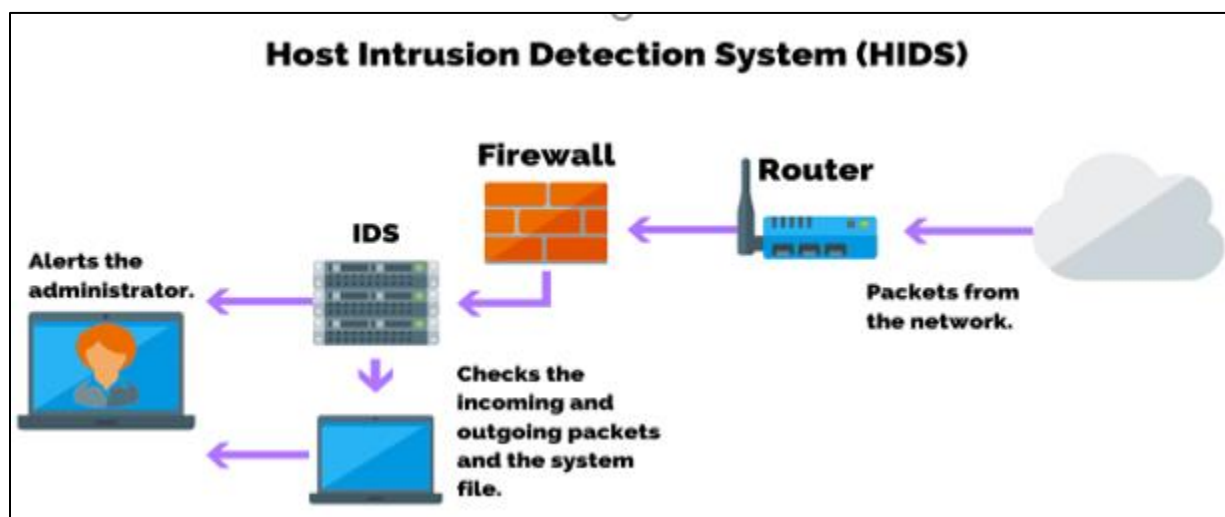


Figure 2. 5 Host Intrusion Detection System [40].

2.3 Types of Attack

As the internet is open for anyone, this creates opportunities for unauthorized users to perform malicious activities within it. A lot of threats are created every day by individuals and organizations to attack computer networks to steal private data and information. External intruders are an unauthorized user of the system they attack, whereas internal intruders are legitimate users of the

organization which has permission to access the system, but do not have the privilege to root or super-users.

Flow-based intrusion detection, since it relies only on header information, can address the only subset of the attacks present above, in particular, the research community currently provides an approach to detect the following class of attacks: such as (Denial of service, Scans, Worms, and Botnets). There are many kinds of attacks for example the few common types [50]: denial of service (DOS), a user to root (U2R), remote to local (R2L), and probe attacks.

Denial of Service (DOS) Attack: This class of attack blocks access to a certain resource (computing and memory) by making unavailable the network services so that legitimate users cannot access the resource allowed to them. The attackers create overloading traffic through the malicious bot to the targeted IP address and flood the network with more requests than the server can process.

User to Root (U2R): Users to Root attack attempts to gain the system super-user privilege by trying to access normal users' accounts either by sniffing or by a dictionary attack. After that, it steals a legitimate user's password to get super user's privilege.

Remote to Local (R2L): This attacker is similar to U2R in that both of them try to get other's legitimate password without the legitimate user's knowledge, unlike U2R it attempts to send packets to a remote machine or a network. After gaining a password, the attacker tries to access the machine learning either as a root or as a normal user.

Probe: probe attack aims at gaining useful information about hosts, their valid IP address, and the services they offer that helps them later to identify the vulnerability of the system to launch an attack against the system services.

From the above description of the four attack categories, the DOS attack acts differently from the three attacks where the DOS attempts to shut down a system to stop traffic flow, whereas the other three slowly penetrate the system.

Table 2. 2 Categories of different types of attacks [88].

Attack Type	
Category	Description
U2R	Buffer_overflow, HTTP Tunnel, loadmodule, Perl, ps, rootkit, SQL attack, xterm
DoS	Aapacha2, back, land, mailbomb, netpune, pod, processtable,

	smurf, teardrop, udpstorm
R2L	ftp-write, guess the password, IMAP, multihop, named, phf, Sendmail, Snmpgetattack, Snmpguess, spy, warezclient, warezmaster, worm, Xlock, Xsnoop
Probing	ipsweep, Mscan, Namp, port sweep, saint, satan

2.4 Software defined network (SDN) based Network intrusion detection system (NIDS)

One of the features in the Software Defined Networking architecture is the separation of the control plane and data plane, which makes easy packet forwarding simple [2]. The Centralized Controller of SDN has the real-time reaction control skill [51], and open interfaces which offer modular plug-in basic features which offer the modular plug-in basic feature. The Centralized controllers provide an abstract network view, critical task by API better programmed of the network [21], [9]. It can integrate security devices within the network topology [52], which can lead to an increase in accuracy, detecting security incidents, and simplify management. In the beginning, they introduce the architecture of Software defined network and application layers, followed by SDN-Based NIDS observation using ML/DL.

2.4.1 Software Defined Network Architecture and Applications

Open Networking Foundation (ONF) [7] is one of the most appropriate architectures for SDN. SDN categorize into three main functional layers. These are the infrastructure layer, control layer, and application layer. Figure 2.6, illustrate the overview of SDN architecture, as shown in Figure. 2.6, the upper layer is the application layer; the control plane is in the middle and the data plane is the

a lower layer which is also known as the infrastructure layer.

1. Infrastructure layers: infrastructure layer is also known as the data plane. It mainly consists of forwarding devices such as physical switches which interconnected through wired and wireless media. Example of physical switches is juniper switches, HP switches and etc. and virtual switches such as OpenvSwitch.

2. Control Plane: The control plane is the part of a network that controls how data packets are forwarded — meaning how data is sent from one place to another. The process of creating a routing table, for example, is considered part of the control plane. Routers use various protocols to identify network paths, and they store these paths in routing tables.

3. Application layer: The upper layer, application layer consists of the end-user business application such as network monitoring and security applications. Using extended features of SDN, several SDN applications have been developed to increase the flexibility of a network, decrease the total time to market and total cost of ownership of future IT network infrastructures. Additionally, due to the recent increase in the number of cyber-attacks, SDN architecture has been used for the rapid growth and deployment of new services. In this section, some of the key application layers of Software Defined Network.

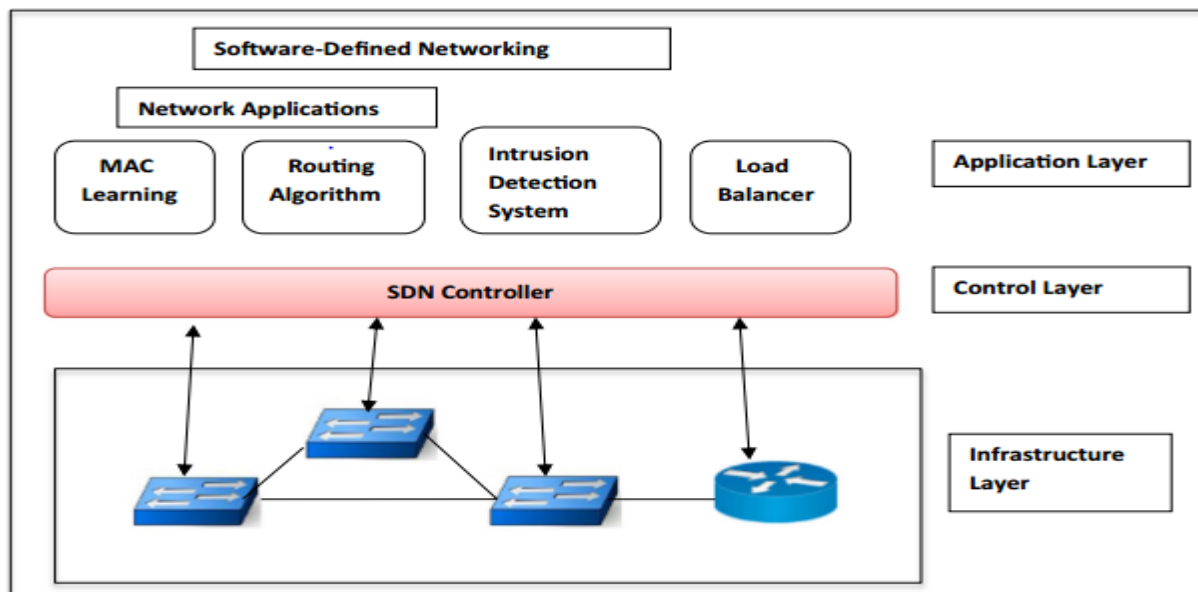


Figure 2.6 SDN Architectures [74].

2.5 Artificial Intelligence

Artificial Intelligence (AI) was born in the 1950s when a handful of pioneers from the promising field of computer science started asking whether the computer could be made to the link which is still being explored today. AI is a general field that encompasses machine learning and deep learning, but it also includes many more approaches that don't involve any learning.

2.5.1 Machine Learning (ML) in Network Intrusion detection system

Machine learning (ML) was born from the theory that computers can learn without being programmed to perform a specific task. It came into prominence perhaps in the 1990s when researchers and scientists started giving it more prominence as a sub-field of AI. In ML, input data and output data are fed to the ML algorithm to generate a function or program that can be used to predict the out of input according to the learning done on the input/output dataset fed to the ML algorithm.

Research reviewed numerous recent works on machine learning (ML) methods that impact SDN to implement NIDS. Rather than ML, deep learning (DL) approaches have been accepted in the field of NIDS. The recent development focuses on leveraging a new network architecture, namely, the software defined network (SDN) to implements NIDS with a machine learning approach [23], [6]. Machine learning and deep learning (ML/DL) approaches can be implemented in the SDN controllers to enhanced network monitoring and security [17], [6]. Research focuses on describing SDN as a platform for implementing NIDS with ML/DL approaches beyond the reach of existing review works.

We look to ML/DL-based NIDS observations. There are Research challenges related applying to ML/DL to SDN-based NIDS. Machine learning methods are examples of anomaly-based intrusion detection techniques. The area of Machine learning (ML) is dedicated to developing systems that can automatically learn from the information [12], [21] and identify hidden patterns without being explicitly programmed to do so [10].

ML algorithms are classified by the learning style they employ and by the functional similarity of how they work [10], [14]. The research focused on reviewing the state-of-the-art machine learning algorithms in implementing NIDS. Machine learning techniques are observed as efficient methods to improve the detection rate, reduce false alarm rate, decrease computation and communication cost [13], [24].

An ML system is trained rather than explicitly programmed and it needs to learn from the past data, optimize for better computations, and also generalize itself to deliver good results. Nowadays, machine learning algorithms are employed for classification, regression clustering, or dimensionality reduction tasks of large datasets with high-dimension. As a result, the task of detecting network intrusions comes under ML as it involves the classification and into normal and abnormal behaviors.

2.5.1.1 Supervised Learning

Supervised learning is a process by which a function is derived with the help of labeled data samples also known as training data. Examples of supervised machine learning algorithms are support vector machine (SVM) for classification problems and random forest for classification and regression problems [12], [15]. Support vector machine (SVM) algorithms are widely used in NIDS research due to their influence on classification power and computation. The algorithm analyzes the training data and produces a function, which will be for input to output mapping of a new example.

This learned knowledge can be then used in the future to predict an output y' for any new input data samples x' which was previously unknown or unseen during the model training process. One of the challenges of supervised ML is the labeling of the large number of data samples which is often done manually and is time-consuming. But if we have enough labeled data samples this type of ML produces good prediction models. A supervised learning model has two major tasks to be performed classification and regression.

Classification: is concerned with predicting output labels or responses that are categorical for input data for what the model has learned in the training phase. Output labels also as classes or class labels are categorical which means they are unordered and discrete values. A classification problem can be binary classification (with two classes) or multiclass classification when there are three or more classes. A common algorithm for performing classification include LR, NNs, Support vector machine, ensembles like RF and gradient boosting, k-nearest neighbors, DTs Recurrent neural network (RNN), Gated Recurrent Unit Recurrent Neural network (GRU-RNN), Deep Autoencoder (DAE) and son on.

Regression: in contrast to classification, regression is concerned with predicting the numeric value for the class label with this knowledge, the regression model can predict output response for new unseen data instance similar to classification but with continuous numeric outputs.

Linear regression and multivariate regression are some of the supervised regression algorithms.

2.5.1.2 Unsupervised Learning

Unsupervised learning deals with how systems can learn to represent a particular input vector in such a way that it reflects the statistical structure or pattern of a collection of input. Examples of unsupervised learning algorithms are feature reduction techniques like principal component analysis (PCA) and clustering techniques, for example, self-organizing map (SOM). Principal

Component Analysis (PCA) is an algorithm that is used to knowingly speed up unsupervised feature learning. The clustering algorithms such as K-means and other distance-based learning algorithms are used for the misuse detection system.

A self-organizing map (SOM) is an artificial neural network that was used to reduce payload in NIDS, in contrast to supervised learning, there are no explicit target output labels associated with each input. Clustering is one of the unsupervised learning methods which helps to cluster or group data points into different groups or categories, without the availability of any output label in the input/training data. The disadvantage of using clustering algorithms in misuse detection is that the clustering algorithms are subjective to initial conditions, for example, centroid, and may produce a high false-positive rate.

2.5.1.3 Semi-supervised Learning

A semi-supervised learning method is another type of ML that combines supervised and unsupervised learning and is used when a smaller number of labeled data is identified for a particular application [43]. It makes a function mapping from inputs of both labeled data and unlabeled data. The goal of this learning mechanism is to classify some of the unlabeled data using the labeled set of information. The quantity of unlabeled data should be higher than the number of labeled data. The Semi-supervised support vector machine was used to enhance the accuracy of NIDS.

2.5.1.4 Deep Learning Algorithms

Internet is continuously changing how people live, study, and work [91], in contrary security threats, becoming more serious than ever because hackers continuously changing inventing new techniques to exploit the vulnerability and to make themselves undetectable [89]. Identifying various network attacks is especially the zero-day attack is a big issue.

Intrusion Detection System is vital for defending network infrastructure and information security which helps to detect attacks that already accorded or the ongoing one. Intrusion detection thought of as a classification problem whether it is multi-classification or a binary classification. Denial of Service (DOS), User to Root (U2R), Probe (Probing), and Root to Local (R2L) can be examples of multiclass classification problems. And identifying network traffic as normal or anomalous can be binary classification problem [47]. Intrusion detection is expected to adapt itself to the dynamic changing of network behavior and attacks.

To detect anomalies or threats many supervised and unsupervised techniques have been studied and advised by the discipline of machine learning and data mining to make intrusion detection more reliable. However, most of the traditional machine learning methodologies belong to shallow learning and they are often emphasized.

Feature engineering and selection [88]. Machine learning methodologies exhibit some drawbacks: In the real network environment, the presence of high dimensional data make ML weak to solve intrusion classification problem [88]. With the dynamic growth of data sets, multiple classification tasks lead to decreased accuracy. Shallow learning is couldn't handle intelligent analysis and high dimensional data. Deep learning is a branch of Machine Learning, can create a better representation of large data which helps to build a better model compared to ML [89]. Deep learning not only creates a better model but also exhibits high accuracy in the classification and extraction of features from incomplete data [89].

Deep-learning is a type of machine learning and artificial neural network with a multilayer structure made up of an input-output layer, which mimics human brain nerves systems and functions like one. Deep means having more than one hidden layer. The deep learning algorithm is not linear like ML as a result can model non-linear and complex relationships. Each layer applies a nonlinear transformation onto its input and creates a statistical model as output from what it learns. The output layer returns the output data. Until the output has reached an acceptable level of accuracy, epochs are continued [88]. Each layer of DNN extracts features and from low-level features, the output layer develops high-level features. DNN has a better ability of modeling complex mapping and feature compression [89].

Deep learning models are increasingly used to solve complicated big data problems which are often not well addressed by the conventional machine learning algorithm. Deep learning requires Fewer hand-engineered features and expert knowledge. Driven by the emergence of big data and hardware acceleration, the intricacy of data can be extracted with higher and more abstract level representation from raw input features. Some recent work showed that deep learning-based IDS has striking learning capability or outperforms traditional ML-based counterparts. Deep learning architecture has three parts such as input, one or more hidden, and output layers.

Each neuron is connected to the neurons of adjacent layers. Each connection has a weight value. The inputs are Multiplied by the respective weights and summed at each unit. The sum then undergoes a transformation based on the activation function. Deep Neural Networks (DNNs) have

attracted the attention of the research community and multiple DNN structures including Convolutional neural networks (CNNs), Recurrent Neural networks (RNN), and GRU have been proposed.

These CNN structures have been successfully applied to devise state-of-the-art solutions in multiple disciplines. Neural Networks have been around for many decades and have been gaining and losing the favor of the research community. The latest rise of this technology is attributed to Alexnet, a Convolutional Neural Network, which won the ImageNet classification challenge. Alexie achieved top-1 and top-5 error rates of 37.5 % and 17.0% on ImageNet Dataset which were considerably better than the previous state-of-the machine [87].

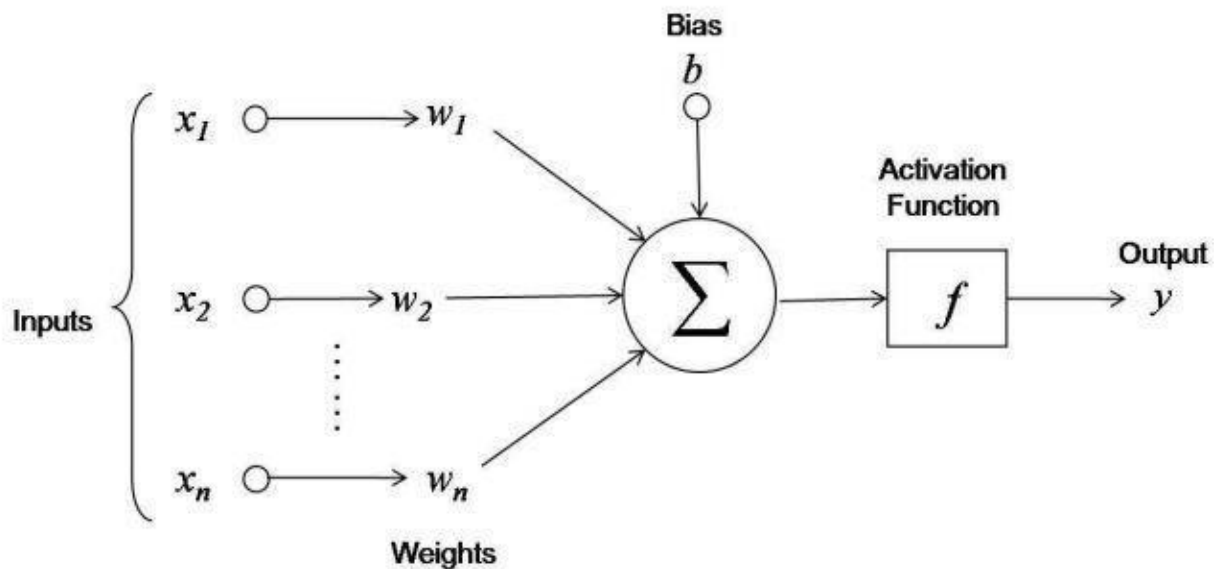


Figure 2. 6 The working process of Neural Network [90].

2.6. Some Popular Classification Algorithms

This section presents some of the common ML classification algorithms with their strengths and weaknesses.

Logistic Regression (LR): The logistic regression algorithm is considered a linear model that measures the relationship between the target variable and other variables by estimating the probability using a logistic function. It was first developed by statisticians to describe the properties of population growth in ecology. In LR, input values (χ) are combined linearly using 17 weights or coefficient values to predict an output value (y). The output of LR is a value between 0 and 1. The Outcomes closer to 1 indicate that the input variable more obviously fits within the

category while results closer to 0 indicate that the input variable likely does not fit within the category.

The ease of mathematical interpretation of the model makes this method suitable. Where Y is the predicted output, B_0 is the bias or intercept term, and B_1 is the coefficient for the single input value (x). Making predictions with an LR model is as simple as plugging numbers into the LR equation and calculating the result. However, one challenge with this algorithm is that we cannot understand the predictions as a linear combination of the inputs.

Support Vector Machine (SVM): SVM works by constructing hyperplanes for linearly separable data in multidimensional space using a kernel function [47], [52]. The hyperplanes separate the sample data into different groups. Objects are rearranged for optimal separation through the use of this kernel function. A kernel function provides a nonlinear mapping from inputs x to feature space $\phi(x)$. During mapping from the input space to the feature space, the mapped object can be separated by a straight line with the new space.

The SVM is one of the most popular algorithms in the field of intrusion detection, but it is not fast enough for real-time detection, especially under SDN architecture. The SVM was proposed for DDoS detection. SVM also achieves quite a high AUC, but it also has some drawbacks like high FPR and low computational efficiency. The NB and DT algorithms have really low training and testing times in ms as compared to the SVM and our CNN. But Our CNN works much faster than the SVM and it also gives a better detection accuracy.

Represented as vectors in high-dimensional space and the SVM finds the closest two points from the two classes that support the best separating line or hyperplane as shown in Figure 2.8

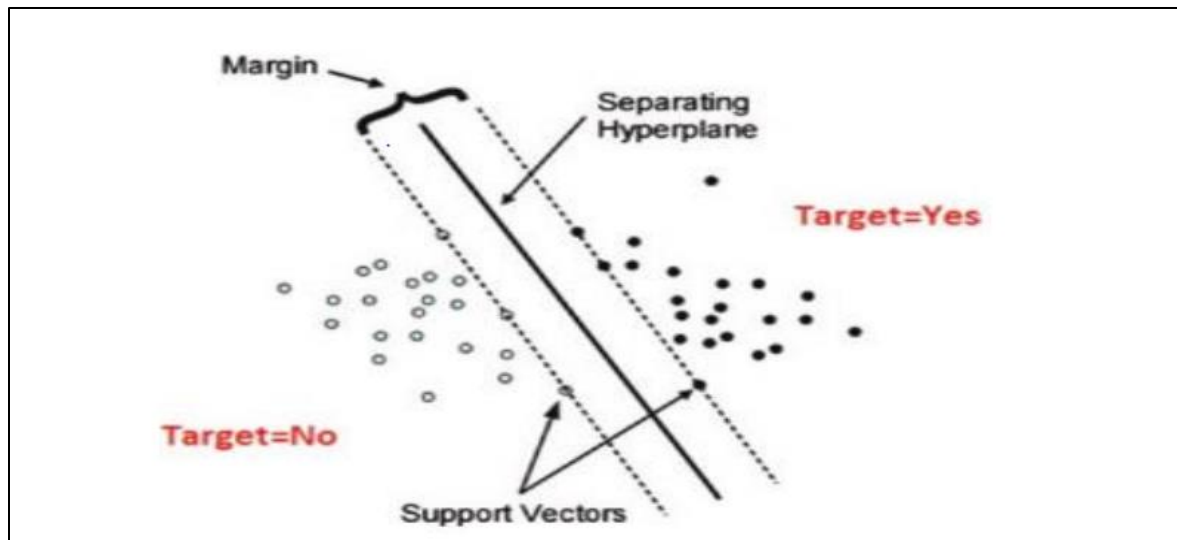


Figure 2. 7 Support vector machine for linearly separable data [54].

Although SVM was initially designed to classify two classes of linearly separable data, today it is used to classify data that are composed of more than two classes that cannot be separated linearly. In multi-class problems, datasets are classified using more than one binary SVM. SVM is capable of training very quickly and performs classification and regression tasks. We also compare the network performance of our DeepIDS with the SVM algorithm

Random Forest: Random Forest is an ensemble classifier that fits a large number of DTs to a dataset and then combines the prediction from all trees. Collaborative methods are methods that combine multiple ML models to make more influential models. Random forest works on the voting mechanism and predicts the output class that received the maximum votes from all individual DTs [49], [52]. RF algorithm starts by randomly selecting a subset of training data and a single DT is generated with the sub-sample.

The random selection of examples is repeated many times, and a single DT is generated with each sub-sample which finally forms a forest. Figure 2.9, shows RF on a sample of the IDS dataset where random samples are selected from the original dataset and trees are constructed based on these sub-samples. After the forest is constructed, individual trees make a classification decision or prediction and then the final predicted class of observation is made by majority vote. One of the main advantages of RF is that it reduces the risk of overfitting. With RF, it is easy to build and train, and often it gives very good results with less chance of overfitting than DTs.

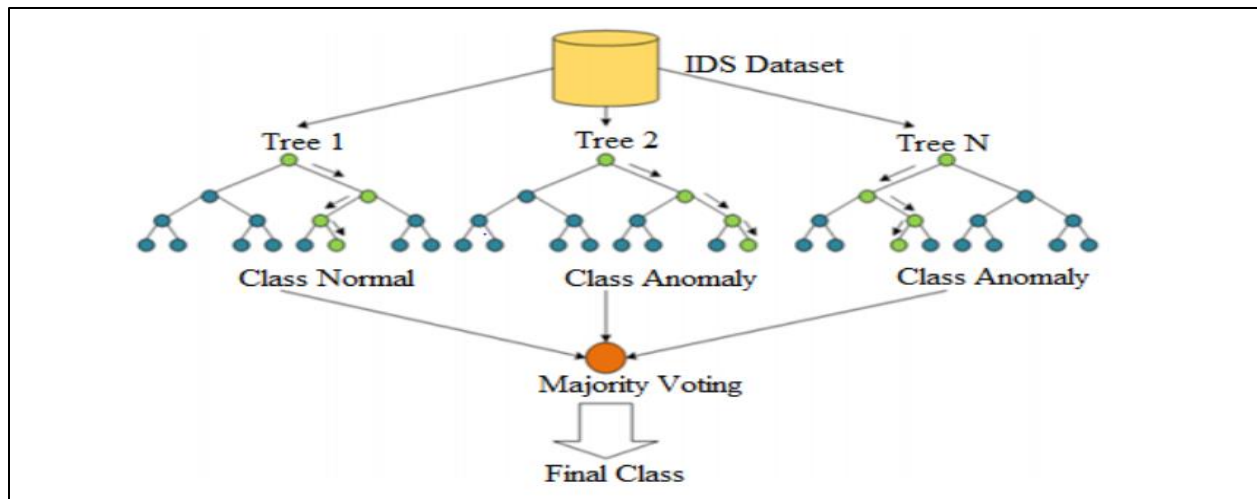


Figure 2. 8 Random Forest [54].

The ensemble scheme of RF makes it enough to overcome the problems of overfitting and hence there is no need to prune the trees. Besides, its high prediction accuracy, efficiency, and interpretability make it preferable for various types of datasets [57].

Neural Networks: An artificial neural network is an information processing system that is inspired by the way biological nervous systems such as the brain work [58]. It is composed of a large number of highly interconnected processing elements (called neurons) working with each other to solve a specific problem. Each neuron can make decisions and feed those decisions to other neurons, organized in interconnected layers. The output of each neuron (after applying a weight parameter) is fed as an input to all of the neurons in the next layer. During training, the NN parameters are improved to subordinate outputs with compatible input patterns. The processing elements of a NN are organized in different layers. A typical NN has an input layer, output layer, and one or more hidden layer with the connection.

2.7 Feature Selection and Reduction in Machine Learning

Usually, the data used for training and evaluating ML models can contain redundant features, irrelevant features, weakly relevant and strongly relevant features. Some features may be optimally correlated with each other and thus only one feature is sufficient in describing the data and no need to use multiple features that work the same. In another way, some features do not correlate with the predictions required and regarded as noise.

Such features don't contribute to the enhancement of the model's performance rather negatively affect the performance of the model. So, it is better to remove these features. In addition, the longer

the length of features, the more computation time is needed for training the models. Feature selection is one of the main tasks in ML which is used for dimensionality reduction by removing irrelevant, noisy, and redundant features from the dataset to obtain an optimal feature subset [58]. This helps to speed up ML algorithms, improving classification accuracy, and enhancing a model's generalizability.

There are several papers published on feature selection [55], [59, 60, and 61]. To classify features as relevant or irrelevant, some metrics are vital to show the relevance of each feature with the output classes or how well each feature predicts the looked-for outputs. Here feature relevance means the ability of the feature to discriminate against the different classes. Feature selection methods are classified into supervised and unsupervised [62]. Supervised methods include filter and wrapper, and unsupervised methods include embedded.

It includes popular methods such as correlation feature selection (CFS), information gain (IG), attribute ratio GR, and principal component analysis (PCA). Nowadays, as big data with high dimensionality are evolving, the filter approach has attracted more attention than before [62].

2.8. Related Work

There are so many related works that has been done in the area of Machine Learning to Network Intrusion Detection, the researcher focused on the best and recent research papers are described on the next table.

Table 2. 3 Contribution and gaps of the related work.

Paper	Title	Datas et Used	Algori thm used	Results		Gaps
Tuan A Tang	Deep Learning Approach for Network Intrusion Detection in Software Defined Networki ng	NSL- KDD	. J48 . NB . RF . MLP . SVM . DNN	ML Algorithm	Accuracy	The obtained accuracy is high, it is not sufficient enough to predict or differentiate b/n normal of intrusion attack.
				J48	81.05	
				NB	76. 56	
				RF	80.67	
				MLP	77.41	
				SVM	69.52	
				DNN	75.75	
Juma Ibrahim	SDN- Based Intrusion Detection System	NSL- KDD	J48 SVM DNN	J48=81.05% SVM=69.52% DNN=7575%		-Overfitting in case of noisy data -the amount of control that the user has over the model is relatively less.
Atiku Abubak ar and Bernardi Pranggo no*	Machine Learning Based Intrusion Detection System for SDN	NSL- KDD	ANN	Accuracy= 97%		-Problem face the research during implementation of NIDS using many ML and DL practice, the techniques make of DL in making SDN based NIDS and the tools which can be used to develop NIDS model in SDN architecture that will help to detect various network related problem.

Des McLern on	Intrusion Detection in SDN-Based Networks : Deep Recurrent Neural Network Approach	NSL-KDD	GRU and RNN	Accuracy=89%		-The anomaly –based IDS is a potential method to detect previously unseen attacks, even though it suffer from less accuracy and a high PFR than a signature based IDS.
Nathan Shone, Tran Nguyen Ngoc	A Deep Learning Approach to Network Intrusion Detection	NSL-KDD	. SVM . RNN .LST M	Algorithm	Accuracy	The researcher only done the implement various other DL models evaluate their performance Comprehensively With current state of the art.
				SVM	90.7%	
				RNN	99%	
				LSTM	99%	
Lotfi Mhamdi , Des McLern on	Deep Learning Approach for Network Intrusion Detection in SDN	NSL-KDD	. NB . SVM . DT .DNN	Algorithm	Accuracy	It provides very good detection results for specified low known attacks, but it’s not capable to detect knew and unknown attacks.
				NB	45%	
				SVM	70.9%	
				DT	74%	
				DNN	75%	

CHAPTER THREE

METHODOLOGY

3.1 General Approaches

Here are the general procedures followed by the proposed work to develop the Most Net-Model

- ❖ Review literature on previous studies made on Intrusion detection systems in software defined network detection by using feature set preprocessing and deep learning techniques.
- ❖ They have been necessary data collected to train the model from different three different Data Science best sites to find dataset.
- ❖ The necessary tools used to develop the models are downloaded from their sites and set up on the computer to be trained.
- ❖ Data Augmentation, preprocessing, and splitting are used to generated a large amount of training data from a small amount of data since it is a must condition to have a lot of data in Deep Learning.
- ❖ The feature set used to train the models is preprocessed using different preprocessing techniques like feature selection including feature encoding, feature scaling to uniform standards so that the model can get normalized and uniform data to train and validate the dataset.
- ❖ Design a Convolutional Neural Network (CNN) model to detect and classify the attack types.
- ❖ Train the proposed model by using the collected dataset.
- ❖ Validate and test the proposed model with a validation dataset.
- ❖ Evaluating the effect of learning parameters including learning rate, batch size, number of epochs as well as train, validation, and test split ration.

3.2 Research Methodology

The Research Methodology followed in this study is shown in Figure 3.1 below

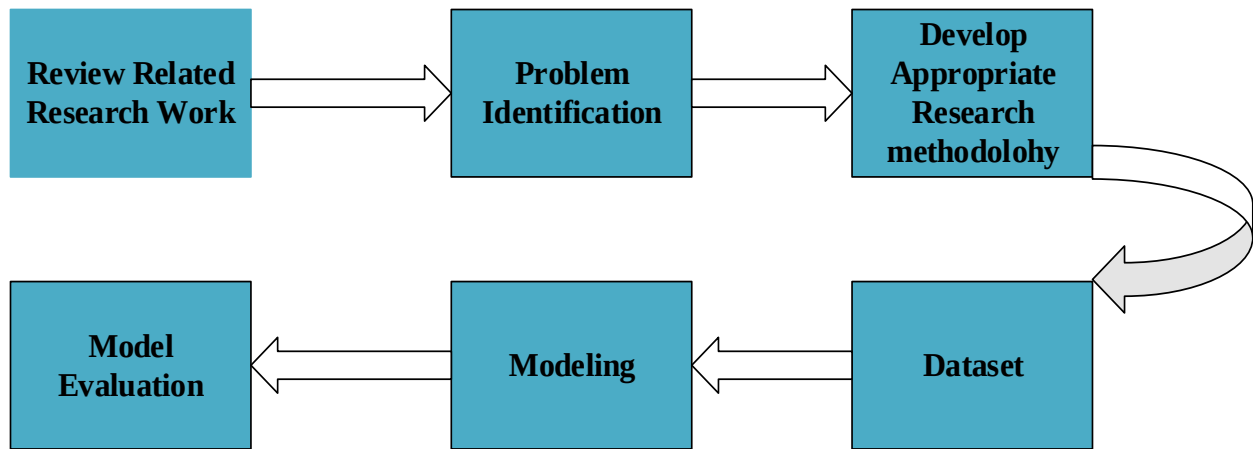


Figure 3.1 Research Methodology

The first of this research was to review related work survey papers work to find the gap of knowledge about the domain. In this study, better results that present research work focused on developing an intrusion detection model with better detection rates, less training time, and improved performance by performing the training in parallel and selecting the best parameter for the model that can improve the performance of the models. These results were achieved by utilizing machine learning with a different approach.

But those work has a drawback, as a result, cybersecurity is still a concern. According the current network intrusion detection system suffer from the following challenges [67, 68]. The rate of new cyber-attack exponentially increased every year, the speed of network lines growth, a large amount of network traffic flow over the network traffic flow over the network which is difficult to monitor and analyze, identify and detect the attack. Besides storage and computational capacity required is [67, 68]. According to A [69] some of the drawback of current software defined network-based network intrusion detection system are listed below [43].

- ❖ The performance of software defined network-based of the intrusion detection system is poor in the real environment.
- ❖ The learning capacity of software defined network based intrusion detection systems is poor when data size increases.
- ❖ Accuracy of SDN-based intrusion detection system decrease while dataset size increase.

- ❖ SDN-based intrusion detection works well with a specific dataset.

3.3 DeepIDS System Architecture

In this section, we propose an SDN-based NIDS architecture (DeepIDS) as depicted in Figure 3.2. Our DeepIDS consists of three main modules: the Collector, the Anomaly Detector, and the Counter Measure Deployment. The DeepIDS is written in Python programming language and deployed as an application in the SDN controller.

The DeepIDS is designed to fulfill the following properties:

- ❖ **Flexibility:** DeepIDS is developed as an application for ease of deployment, configuration, and interaction. It can be implemented on top of any SDN controller. Our DL models can be modified and optimized based on network requirements. New threat models can also be easily added/updated.
- ❖ **Scalability:** The DeepIDS is designed to facilitate not only small scale networks but also large-scale networks. The overhead of our approach does not degrade the performance of the whole network. The overhead on the SDN controller is evaluated with different network sizes.

The below diagram show that after the dataset is processed through (Data Collect→Data Preprocessing→Feature Extraction→Evaluation→Training, Validation, and Testing→Build Model) then passes through our Deep IDS application. We take the DeepIDS as application layers.

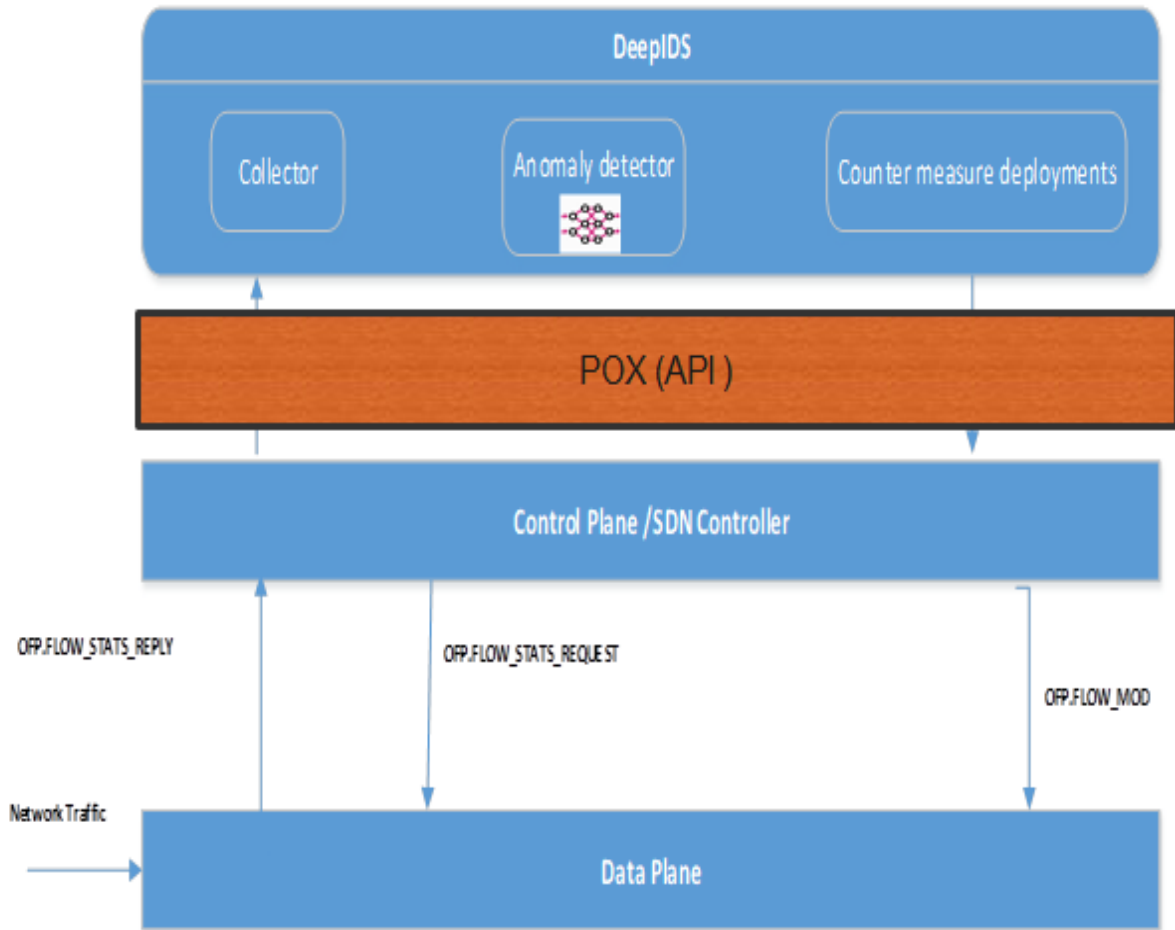


Figure 3.2 Proposed System Architecture

For ease of understanding, details of the network intrusion detection framework are presented in Figure 3.3. This framework describes a complete cycle of data flow in the DeepIDS. At the first stage of the intrusion detection process. The Collector collects network information and then extracts flow features from them. These features set are inputs of the DL algorithm. Some examples of features set are shown in Table 3.1. For the DL algorithm training purpose, these flow features are cleaned, labeled, and saved as a network dataset.

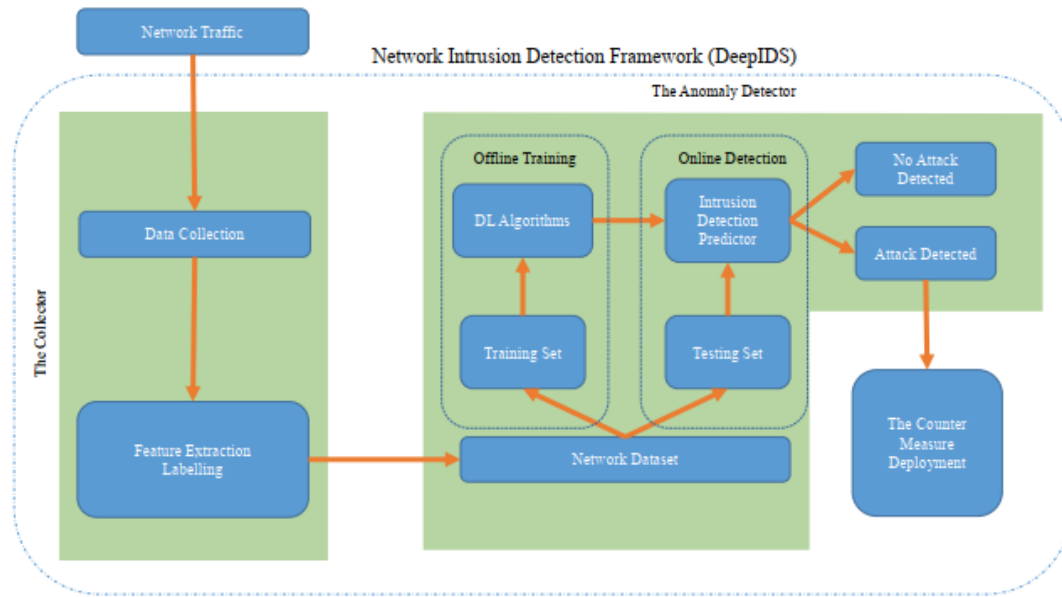


Figure 3.3 Network Intrusion Detection Framework [81].

After preparing the network dataset, we move to the second stage which is the most critical part of intrusion detection and also is the primary research of this thesis. At this stage, we train the DL algorithm to detect all types of network attacks. The network dataset is separated into two subsets: Training Set and Testing Set. The training used for training, and the testing set used for evaluating trained models. If there is any attack detected, the Counter Measure Deployment module activated to mitigate the attack

3.4 DeepIDS Module Types

- ❖ The Collector
- ❖ The Anomaly Detector
- ❖ The Counter Measure Deployment

Solution Formation: This part identifies the main limitation in the current research related developing flow-based intrusion detection model for software defined network using deep learning approach detection of attack and classification and finds out the possible solutions for the identified problem. The research is focused on software defined controller intrusion detection, classification, and analysis of the log files. The main aim of this study is to detect the condition of anomaly detection in the software defined controllers which have a short life span. To achieve this, a deep learning-based deep learning neural network model is trained on the feature set, collected from different data science GitHub, and labeled according to the researchers' visual perspective of the

feature set. The performance of the deep learning model was evaluated based on the confusion matrices and classifications accuracy.

Review searching methods: By searching different related literature (Internet, Books, Journals, etc.) reviewed to understand the concepts of feature set processing and CNN how they applied to solve the related problem in the prior research. To achieve this research objective, the most recent and current literature was reviewed. During the version of this literature, the gap of the previous solutions identified to use input for this proposed solution. This is the important and necessary content where all possible reference and journal related to research is investigated and analyzed.

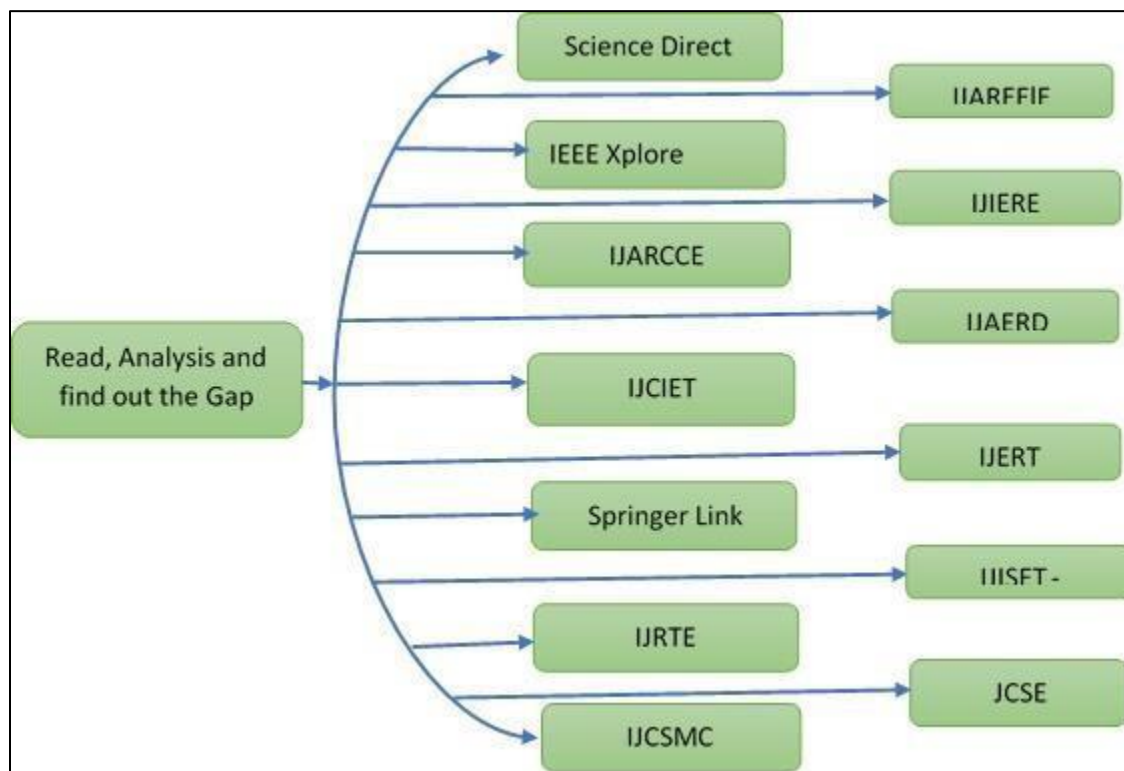


Figure 3.4 Digital Library for the proposed research

3.5 Material and Tools

3.5.1 Software Tools

Investigation of available software tools with their libraries is conducted to select the appropriate tool for the implementation of the DL algorithm for SDN-bases NIDS detection. During the investigations, we have seen that there are tools that are general for both deep learning and machine

learning algorithm and specific only for one of them. Before selecting the tools, we have considered some criteria which are helpful to select the appropriate software tools with their corresponding libraries. The other criteria are to select tools with enough learning materials such as free video tutorials, existing, existing experience, and other one is the tools that must be used in machines with a limited resource (like CPU only). Software tools that we have used to implement the DL algorithm are python as a programming language with TensorFlow and Keras libraries on an anaconda environment. These tools fulfill all the consideration criteria and they are used in python which is familiar to us.

Anaconda is used for the implementation of the model and it is free and open-source distribution of the python and R programming languages for feature set processing, data science, machine learning, and related applications that aim to add and simplify package management and deployment. It contains different IDE's which are used to write the coding part such as Qt Console, Jupyter Notebook, Visual studio, and Spyder. We have used the Jupyter notebook and visual studio to implement the coding part. It is easy to use and run in a web browser.

TensorFlow is a free and open-source library developed by Google and it is currently the most famous and fastest deep learning library [70]. It can be used on any desktop which runs Windows, macOS, Linux, in the cloud as a service, and mobile devices like iOS and Android. The architecture of TensorFlow works for the preprocessing of the data, building the model, train the model, and estimate the model..

Keras is a high-level neural network API that is written in python which runs on the top of TensorFlow either Theano or Microsoft Cognitive Toolkit (CNTK). It is very simple to develop a Model, user-friendly, easily extensible with python and most importantly it contains pre-trained CNN models such as VGG16 and Inception that we during the experiment. It allows easy and fast prototyping, and support both CNN and DNN or the combination of the two [70].

Visio 2016 is used for designing the system architecture. This tool was used to design a diagram and to create a Gant chart easily with ready-made templates and helping to simplify complex information.

Mininet is used for the emulated SDN Network.

Snort is used as signature-based NIPS on SDN for packet-based detection and Artificial Neural Network is used for anomaly-based detection of the flow table entries.

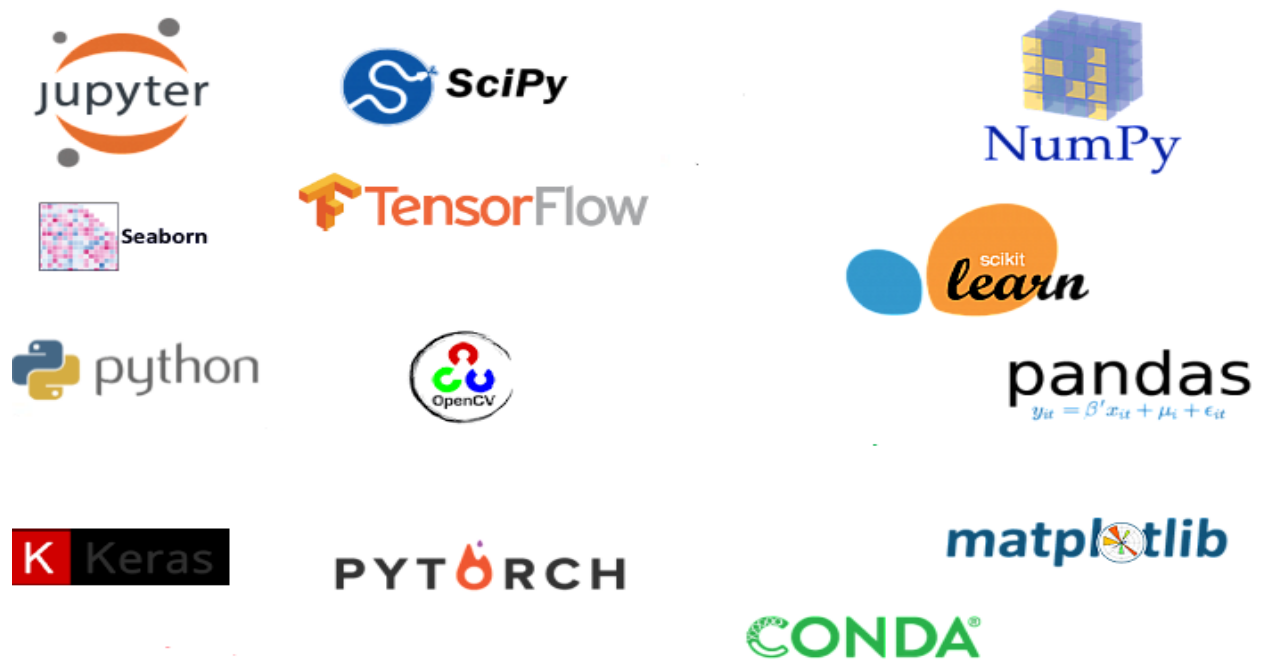


Figure 3.5 Some open-source library and our thesis Environment [33].

3.5.2 Hardware Tools

Any feature set of attack group and NSL-KDD dataset are used. To implement the DL algorithm with the selected software tools a very slow machine with CPU Intel(R) Core(TM) i7-7500 CPU @ 2.70GHz 2.90GHz processor, memory 8 GB.

3.6 Initialization

Choosing the proper initialization for each layer is the key to achieving high performance. Appropriately initializing the weight can be significantly influential to how easily the network learns from the training as it effectively preselected the initial position of the model parameter for the loss function to optimize in designing the models to experiments with the network parameter are initialized various dataset. The key idea behind this initialization scheme is to make it easier for a signal to pass through the layer during forward as well as backward propagation, for a linear activation this also works nicely for sigmoid activation because the interval where they are unsaturated is roughly linear as well.

3.7 Research Methodology development

3.7.1 Deep Learning

The Deep learning overview is introduced for easy understanding of this thesis. In addition, we also introduce the evaluation methodology and the network dataset used in this thesis.

3.7.2 Notations

Throughout this thesis, we used the following mathematical notations. In general, the following rules are used for number and arrays:

- Non-bold letters (e.g., x ; y ; I) are for scalars.
- Bold small letters (e.g., w) denote column vectors. A row vector is denoted by its transpose (e.g., w^T).
- Bold capital letters (e.g., W) denote matrices

In recent years, DL has emerged as one of the hottest technologies and somewhat of the buzzword. So, what is DL? And what is the difference between Artificial Intelligence (AI), ML, and DL? Figure 3.6 shows the relationship between these terms.

As we can see, DL is a subset of ML, and then ML is a subset of AI. Figure 3.6: The AI, ML and DL Relationships [5]

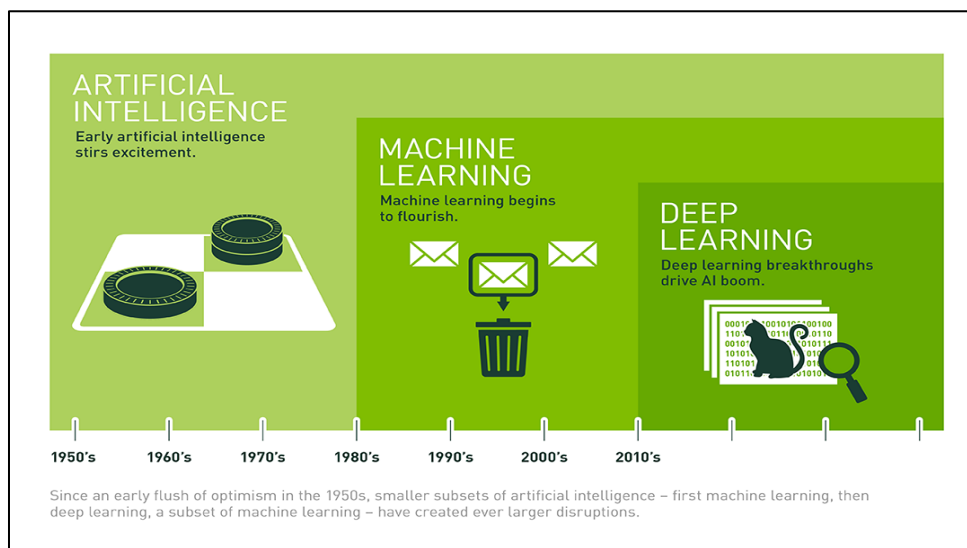


Figure 3.6 The AI, ML, and DL Relationship [30]

AI is a broad concept that means incorporating human intelligence into machines. ML intends to give the machine the ability to learn and make the decision by themselves using the provided data. ML was inspired by the structure and function of the human brain which is a highly interconnected

system of neurons. A neuron is a basic computation unit of the human brain. In general, DL is an ML approach. DL algorithms use a neural network (NN) to learn associations between inputs and outputs. DL refers to creating a deep NN (DNN) that has a huge amount of hidden layers. The NN consists of several artificial neurons arranged in a series of layers. The most common type of NN is a fully connected NN in which neurons between two adjacent layers are fully connected.

3.7.3 Deep Neural networks

In [90] flow-based anomaly detection system by constructing a deep neural network (DNN) architecture is proposed to secure the SDN network. The structure of the Deep learning system includes three basic layers. An input layer with six dimensions, three hidden layers contain twelve, six, and three neurons respectively, and an output layer with two dimensions. To achieve a better detection rate, the same authors propose another intrusion detection system for SDN networks [89] the proposed approach is based on a deep recurrent neural network. The authors advocate a Gated Recurrent Unit.

The recurrent neural network (GRU-RNN) model to build an IDS contains input and output layers with six and one dimensions respectively, and hidden layers which have six, four, and two dimensions. The system architecture is composed of Flow Collector, Anomaly detector, and Anomaly Mitigator. For the test results and training phases, they use the NSL-KDD dataset. The first solution [90] achieves an accuracy of 75.75% using six basic features, and the second one [62], [89] had an accuracy of 89 %. While traditional machine learning algorithms are linear, deep neural networks are stacked in an increasing hierarchy of complexity as well as abstraction. Each layer applies a nonlinear transformation onto its input and creates a statistical model as output from what it learns.

In simple terms, the input layer is received by the input layer and passed onto the first hidden layer. These hidden layers perform mathematical computations on our inputs. One of the challenges in creating neural networks is deciding the hidden layers' count and the count of the neurons for each layer. Each neuron has an activation function that is used to standardize the output from the neuron. The "Deep" in Deep learning refers to having more than one hidden layer. The output layer returns the output data. Until the output has reached an acceptable level of accuracy, epochs are continued.

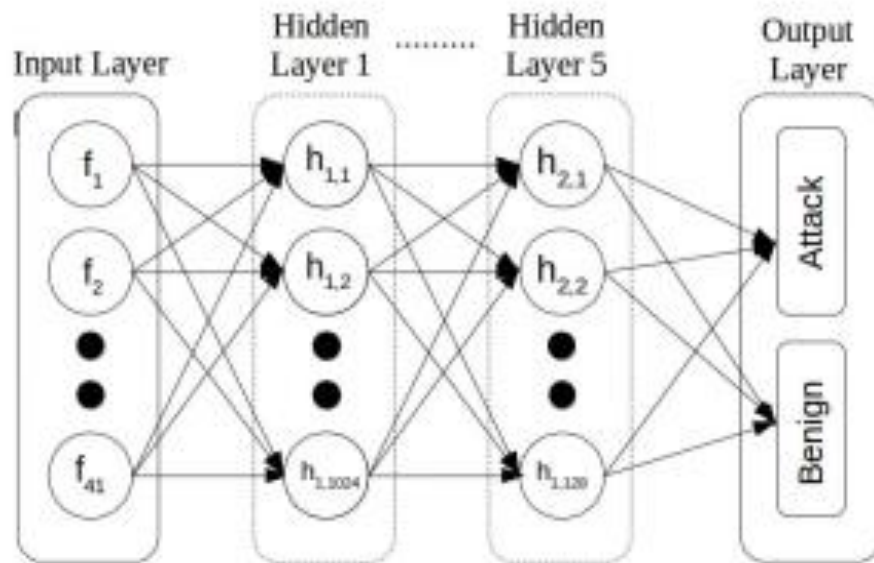


Figure 3.7 DNN Architecture [32].

Gaussian Noise: Gaussian noise layer is applied as a regularization layer and only active at training time. This layer randomly corrupts input layers of the neural network. Gaussian noise layer is a technique used to avoid system overfitting and identity function [64], [83].

Activation Function: A neural network consists of multiple neurons that are clustered into several layers, neurons in the previous layers connected to the neurons of the next layer, each connection have weight W , bias b , and activation function corresponding to it. To perform complex tasks neural network needs an activation function. A neural network without activation a function is a linear regression model.

The output of the single neuron is calculated and passed to the next layers of neutrons using activation function this process is continuously repeated. The activation function is used to introduce non-linearity into the output of a neuron besides decide whether the neurons need to be activated or not [84].

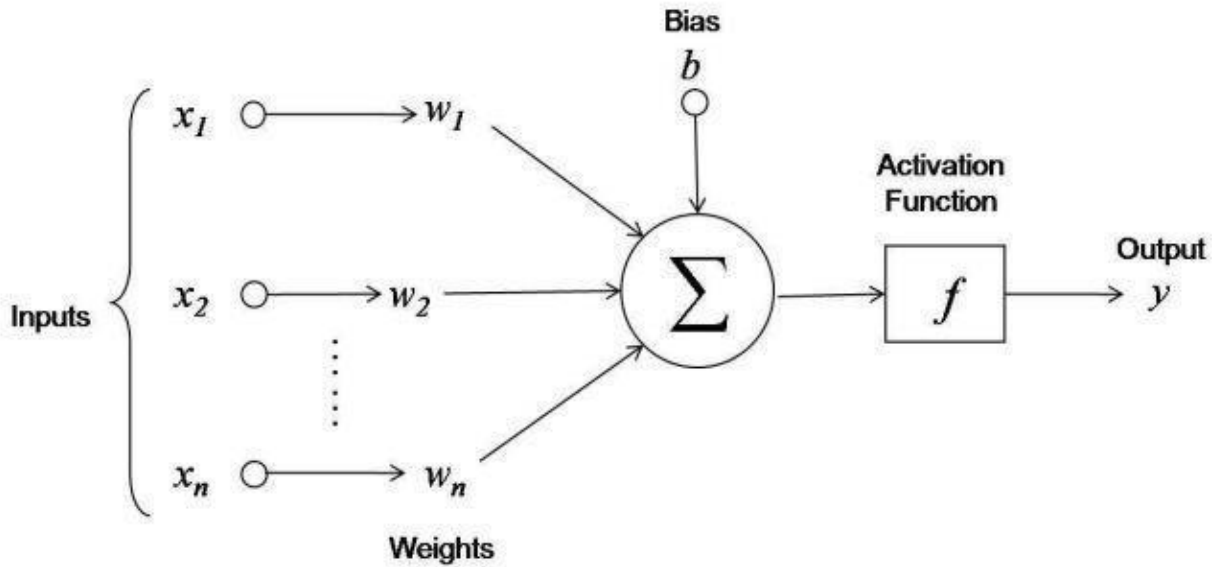


Figure 3.8 Activation Function in Neural Network [56].

3.7.4 SimpleRNN-Based Models

The well-entertained approach to NLP in Deep Learning is based on a recurrent model. RNN-based models are good in interpreting the text as a sequence of words, which makes them perfect for capturing word dependencies and text categorization. Nevertheless, vanilla RNN models do not perform admirably and may sometimes understand a simple feed-forward neural network. Among numerous variations of RNNs, Long Short Terms Memory (LSTM) is the foremost prevalent variant, which is outlined to superior captures long terms conditions. The secret behind the LSTM model is its capability to address the gradient vanishing and exploding issues endured by the vanilla RNNs.

By presenting a memory cell to keep values over arbitrary time intervals, and three gates (input gate, forget gate, output gate), LSTM can control the stream of information in the cell state. The other competent variant is Gated recurrent units (GRU). GRUs are fast to train and do well on a considerably small amount of training data. Besides LSTM and GRUs, in this study, an extension of these traditional models such as bidirectional LSTM and GRU is accessed to see if the performance on intrusion detection in the software defined network is improved.

3.7.5 Convolution Neural Networks Based Models (CNNs)

Convolution neural network (CNN), originally invented for computer vision, has shown super performance on text classification tasks as well as other traditional Natural language processing

(NLP) tasks, even when considering relatively simple one-layer models. Besides its performance, researchers also have tried to understand the inner working of the network has makes it preferable in some NLP tasks, especially in text classification. In a paper by [29] the researchers examined a hypothesis that claims filters accompanied by global max-pooling could be as ngram detectors. From their finding, they able to conclude that different semantics of ngrams can be captured by filters with different activation patterns and global max-pooling.

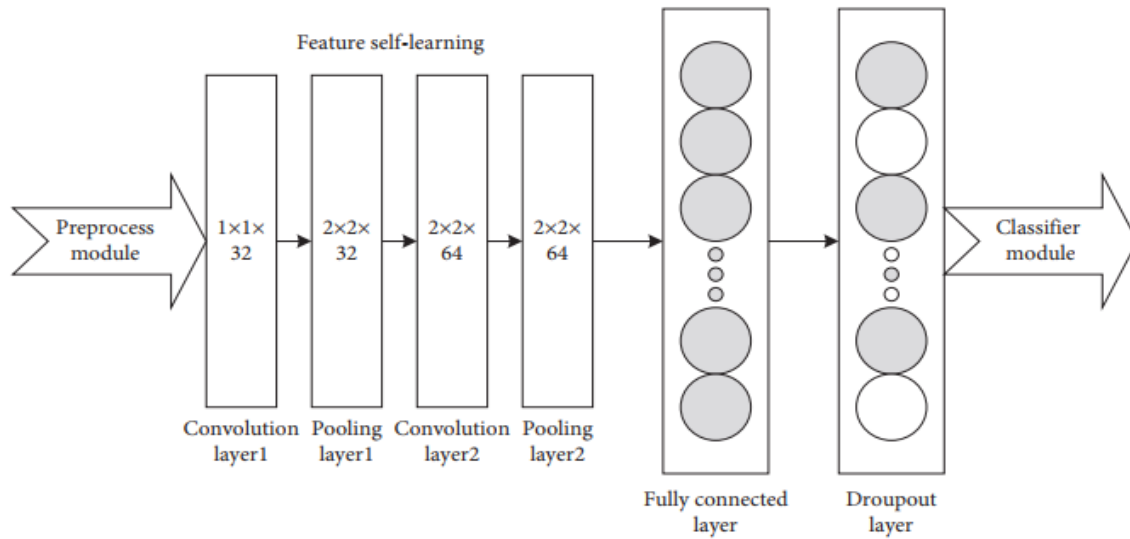


Figure 3.9 CNN model with different convolutions for feature extraction [65].

3.8 Dataset

3.8.1 NSL-KDD Dataset

To solve cybersecurity challenges of our times, Network Security Laboratory and Knowledge Discovery and Data Mining (NSL-KDD) dataset [71] are used. In this work an effective Benchmark dataset is convenient because it helps the researcher to compare different intrusion detection methods. NSL-KDD is a dataset suggested to solve some of the inherent problems of the KDD'99 datasets.

Although this new version of the KDD dataset still suffers from some of the problems discussed by McHugh and may not be a perfect representative of existing real networks, because of the lack of a public dataset for network-based IDS, we believe it still can be applied as an effective benchmark dataset to help researcher compare different intrusion detection methods. Furthermore, the number of records in the NSL-KDD train and test sets is reasonable. This advantage makes it

affordable to run the experiment on the complete set without the need to randomly select a small portion. Consequently, evaluation results of different research work consistent and comparable. It builds Intrusion Detection System (Host-based or network-based), doing some experiments in the cybersecurity area likewise there are so many advantages. And also, there are a lot of datasets (ADFA-ID, ISCX-UNB, etc.) which is in this field.

The KDD Cup 99 dataset is one of the most popular datasets and is widely applied to evaluate the performance of intrusion detection systems. However, this dataset suffers from the redundancy of records that makes the classifier fail to deliver better accuracy. The NSL-KDD dataset is introduced by [71], to resolve redundancy issue. SDNs are a new environment, and so the dataset for them is still very rare and unpublished. The NSL-KDD dataset is still considered a state-of-the-art dataset by many researchers to evaluate their approaches, and so we also choose it for our experiment.

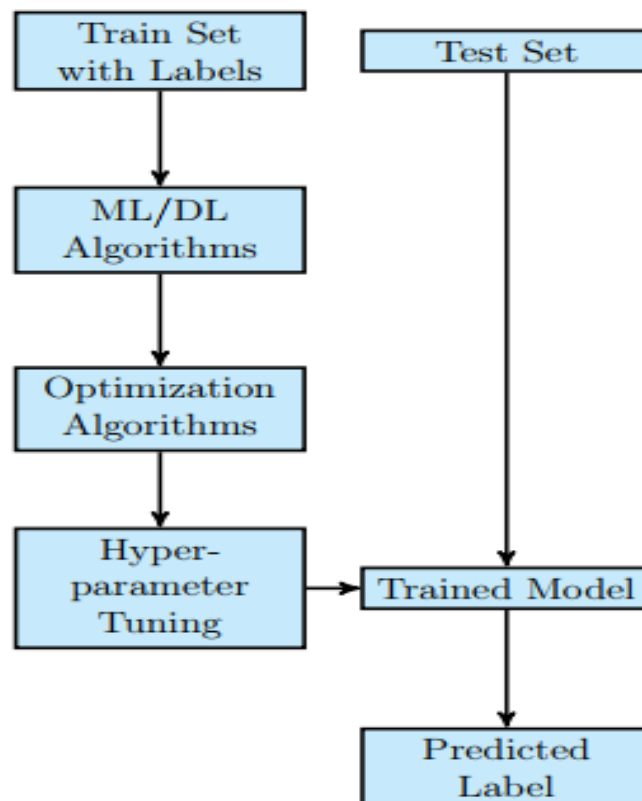


Figure 3.10 ML/DL Training and Testing Phase [63].

This dataset contains 125,973 records for training and 22,544 records for testing. Details about the distribution of this dataset are presented in Table 3.1. The number of legitimate and anomaly traffic is balanced to make sure that the ML/DL algorithms can be trained properly.

Table 3. 1 The NSL-KDD Dataset Distribution

Dataset	Legitimate Traffic	Anomaly Traffic
KDDTrain+	67,343	58630
KDDTest+	9,711	12833

Each traffic sample in this dataset has 41 network features (see Table 3.2) that are categorized into three types of features: basic features (1 - 9), content-based features (10 - 22), and traffic-based features (23 - 41). The details of these features can be found in [71].

Table 3. 2 NSL-KDD Dataset Features [71]

No.	Feature Name	No.	Feature Name
1	Duration	22	Is_guest_login
2	Protocol_type	23	Count
3	Service	24	Serror_rate
4	Flag	25	Rerror_rate
5	Src_bytes	26	Same_srv_rate
6	Dst_bytes	27	Diff_srv_rate
7	Land	28	Srv_count
8	Wrong_fragment	29	Srv_serror_rate
9	Urgent	30	Srv_rerror_rate
10	Hot	31	Srv_diff_host_rate
11	Num_failed_logins	32	Dst_host_count
12	Logged_in	33	Dst_host_srv_count
13	Num_compromised	34	Dst_host_same_srv_rate
14	Root_shell	35	Dst_host_diff_srv_rate
15	Su_attempted	36	Dst_host_same_src_port_rate
16	Num_root	37	Dst_host_srv_diff_host_rate
17	Num_file_creations	38	Dst_host_serror_rate
18	Num_shells	39	Dst_host_srv_error_rate
19	Num_access_files	40	Dst_host_error_rate
20	Num_outbound_cmds	41	Dst_host_srv_rerror_rate
21	Is_host_login		

Attacks in the dataset are divided into four categories according to their characteristics. The details of each category are described in Table 3.4. Some specific attack types in the testing set do not appear in the training set, and that makes the detection task more realistic. The CNN requires each

record in the input data to be represented as a vector of real numbers. Thus, every symbolic feature in a dataset is first converted into a numerical value. The NSL-KDD dataset contains both numerical and symbolic features. These symbolic features include the type of protocol (TCP, UDP, and ICMP), the service type, and the TCP status flag. After converting all symbolic attributes into numerical values, every feature within each record is normalized by the respective maximum value and therefore falls into the same range of [0-1] by Min-Max scaling. Its mathematical equation is given as where x' is the normalized value, and x is the original value.

$$x' = \frac{x - \min(x)}{\max(x) - \min(x)} \quad \text{-----3.1}$$

Where x' is the normalized value, and x is the original value. In Table 3.6, we give some examples of the NSL-KDD dataset for more understanding. The reason we choose the min-max is the network traffic have a lot of variable features.

Good practice usage with the *MinMaxScaler* and other scaling techniques is as follows:

- ❖ **Fit the scaler using available training data.** For normalization, this means the training data will be used to estimate the minimum and maximum observable values. This is done by calling the *fit ()* function.
- ❖ **Apply the scale to training data.** This means you can use the normalized data to train your model. This is done by calling the *transform ()* function.
- ❖ **Apply the scale to data going forward.** This means you can prepare new data in the future on which you want to make predictions.

The default scale for the *MinMaxScaler* is to rescale variables into the range [0, 1], although a preferred scale can be specified via the “*feature range*” argument and specify a tuple including the min and the max for all variables.

Table 3.3 The NSL-KDD Dataset Example

Duration	Protocol Type	Service	Flag	Src.Bytes	Dts Bytes	Land	...	Dst_host_srv_rerror_rate	Label
0	TCP	ftp_data	SF	491	0	0	...	0	normal
0	UDP	other	SF	146	0	0	...	0	normal
0	TCP	private	S0	0	0	0	...	0	neptune
0	TCP	http	SF	232	8153	0	...	0.01	normal
0	ICMP	ecr_i	SF	1032	0	0	...	0	smurf

Table 3.4 Attack in the NSL-KDD Dataset

Category	Training Set	Testing Set
DoS	back, land, neptune, pod, smurf, teardrop	back, land, neptune, pod, smurf, teardrop, mail-bomb, processtable, udpstorm, apache2, worm
R2L	fpt-write, guess-passwd, imap, multihop, phf, spy, warezclient, warezmaster	fpt-write, guess-passwd, imap, multihop, phf, spy, warezmaster, xlock, xsnoop, snmpguess, snmpgetattack, httptunnel, sendmail, named
U2R	buffer-overflow, loadmodule, perl, rootkit	buffer-overflow, loadmodule, perl, rootkit, sqlattack, xterm, ps
Probe	ipsweep, nmap, portsweep, satan	ipsweep, nmap, portsweep, satan, mscan, saint

3.8.2 Dataset Preprocessing

The public collected dataset should pass through some data filtering, cleaning, and consolidation processes to alleviate the data quality issue, which may have an undesirable effect on building classification models.

The data items that need to be filtered and cleaned are:

- ❖ Entries that miss some fields.
- ❖ Poorly formatted entries.
- ❖ Remove Duplicated entries.
- ❖ Non- Intrusion posts/ entries

After cleaning, all intrusion from different organization pages are integrated into one file to be annotated.

3.9 Deep Learning Experimental Setup

We implement a fully connected CNN model and is an effective algorithm of deep learning; CNN is designed to process multidimensional array data, and its greatest advantage is to be able to accurately extract the local correlation of features and improve the accuracy of feature extraction. Using CNN algorithm, combined with mainstream deep learning technology such as Dropout and ADAM and Softmax classifiers, this thesis presents a binary classification network intrusion detection model based on convolutional neural network and implements the code based on keras. As a semi supervised neural network, convolutional neural networks have the ability to abstractly represent low-level intrusion traffic data features as high-level features and outstanding feature learning capabilities, so they have been gradually applied to the field of network intrusion detection in recent years.

Convolutional neural networks are neural networks that use convolution operations in place of ordinary matrix multiplication operations in at least one layer of the network [28]. The deep learning technology used in the feature self-learning module mainly includes convolution operations and pooling operations, dropout, activation functions, and ADAM optimization algorithms [31]. The common activation functions of convolutional neural networks are sigmoid, tanh, ReLU.

To prevent overfitting the methods dropout, and batch normalization are used. This thesis uses the method of inserting a dropout layer between the feature self-learning module and the classifier to prevent overfitting. This article uses the Softmax classifier as a classification module for convolutional neural networks. The thesis presents a new network intrusion detection model using convolutional neural network (CNN), using CNN to automatically select traffic features from the original data set, and set the cost function weight coefficient according to the number of categories to solve the problem of balance.

The CNN is trained with flow features, and it then classifies network traffic as legitimate traffic or anomaly traffic. The anomaly traffic includes all types of network attacks appearing in the training set. The number of convolutional layers and the dimension of each pooling layers can be varied to find the best structure for intrusion detection. In our experiment, forty-one 41 input features were used, so we define an input layer with forty-one neurons. The KDDTrain++ and KDDTest++ of

the NSL-KDD dataset are used for training and testing the CNN respectively. We use Keras [88] to implement the CNN model. Details of all initiated parameters of the CNN model.

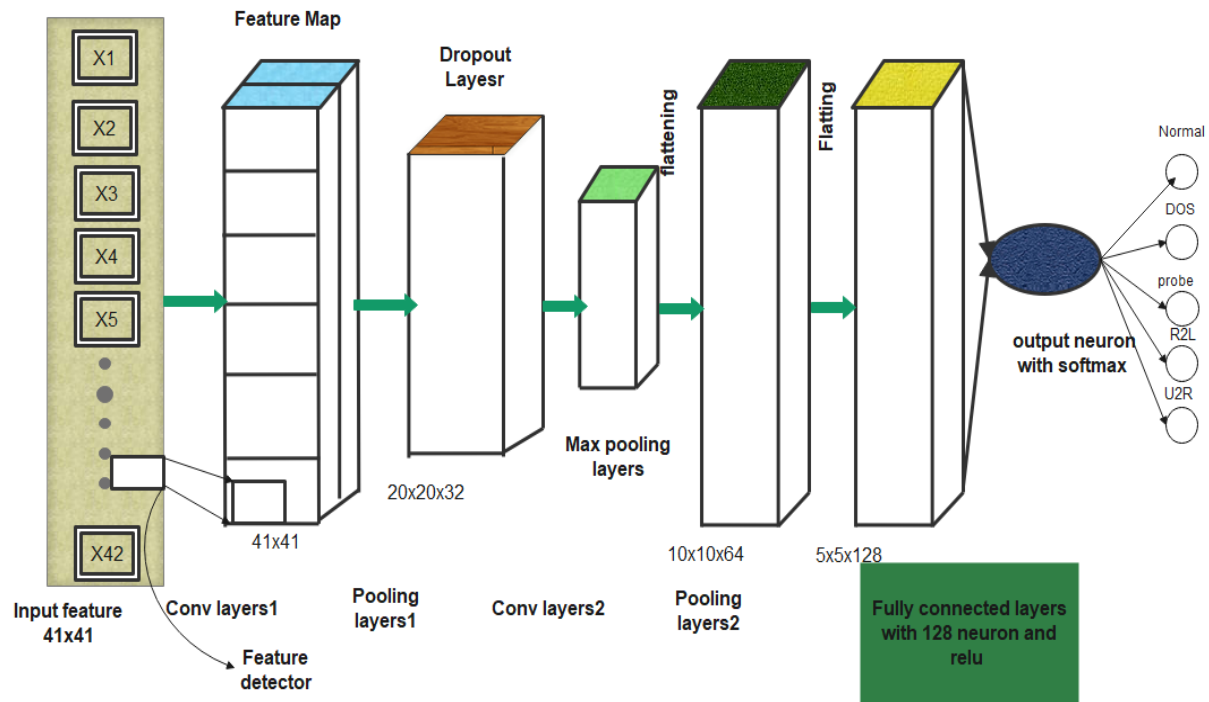


Figure 3.11 Convolutional Neural Network Design

The unique convolution and pooling process of CNN allow the model to learn complicated pattern of features from a high-dimensional data space while maintaining reasonable storage and computation time overhead. For any SDN network IDS, it is exceptional crucial to have the ability to capture the complicated features of network traffic within a short amount of time. Therefore, it is believed that CNN is the priority NN model for developing the prototype of the proposed model. Fig 3.11 show the design of a CNN model with the input size of 41x41 matrix. The whole model could be separated into three main sections: convolution layer, pooling layer, and full connected layer.

In terms of functionality, the convolution layer and pooling layer together handle the feature extraction of the training samples, and the fully connected layer process the final classification. At the convolutional layer, CNN model convolves the input sample with some specific matrices called “feature detector” or “kernel map”. The result of the convolution is called “feature map”. Feature

detector are matrices used to extract, from the input sample, specific features, such as patterns, shape and lines. Their value is initially randomized and gradually updated by the optimizer during the training process. After the input sample is convolved with feature detector, the feature maps are generated.

The feature map calculated by the mapped to a nonlinear activation function to break the linearity of the model. In the Figure 3.11 the whole process of generating the features maps and mapping them to the activation function is represented by the green connection at the convolution layer. The second section of the model is the pooling layer. At this layer, the feature maps will be pooled to bring down their dimension. The purpose of this process is to eliminate noise and models dependent on the spatial location of the learned patterns. To eliminate noise, pooling brings down the feature maps, while preserving the valid information carried by them. This also easy for the calculation of any further processing to the feature maps.

After the feature maps are calculated and pooled, they undergo a “flattening” process converts the feature maps from 2-D matrices to 1-D arrays in row-major order, which is the format expected by the fully connected layer. The fully connected layer is composed of input neurons, optional hidden neurons and output neurons. Each neuron is connected to every neuron at the adjacent layers with distinct weights and biases as the parameter. The output neurons, which are also the final output of the CNN model, determine to which class each input sample belongs. In the case of cybersecurity, the model can use the output neurons to represent the benign class and every class of attack that the model is trained to classify.

In other words, the feature detector, which contain the weight vectors, are replicated at every convolution process at the convolutional layers. This means the optimizer can potentially converge the model to optimal loss with less time overhead. The constraint on weight also enables the model to achieve better generalization on the task assigned to it. Briefly, CNN has advantages that are suitable for developments of IDS for cybersecurity in the modern era. CNN require less time overhead at the testing process. Besides with its convolution mechanism, CNN could potentially learn the much more complex characteristics of some modern cyberattacks, which other NN model struggle to captures.

Lastly, CNN can achieve better generalization on the classification of cyberattack samples. This enables the IDS to potentially detect innovative attacks that share similar traits with the known attacks. A novel data-preprocessing method is proposed in the model design section to generate a

relatively balanced sub-dataset, which still includes all the available attacks classes from the NSL-KDD dataset, for the training and test of the proposed IDS model. And the proposed IDS is trained to perform binary classification on every type of attacks that are known in the NSL-KDD dataset. Each matrix has a size of 41x41, which is composed 41 extracted features from the network traffics samples and zero padding for format concern.

The decision of hyperparameters, such as the number of layers, the kernel size for convolution, and the number of neurons in fully connected layer, are all determined by exhaustive grid search. In addition, the number of output feature maps is selected to be 41, and the activation function to be RELU function at this layer.

Accordingly, for each matrix in a training batch, the model generates 41 feature maps at the first convolution layer. Each feature map has a size of 41x41 and is mapped to ReLU function. Then, the feature maps are pooled at the first pooling layer by the Max pooling algorithm with kernel size of 2x2, stride of 2 and padding of 1. The pooled feature maps are passed to the second's convolution layers and pooling layers which both have the same kernel size, stride and padding configuration as the first two layers.

The second convolution layer is designed to output 20 feature maps, and they are also individually mapped to ReLU activation function. Eventually, after the two convolution layers and two pooling layers, the proposed model generates 10 pooled to 3x3 feature maps for each input matrix. At the flattening layer, all the elements of the pooled feature maps are converted into a 1-D array in row-major order, which is the format expected by the fully connected layer. This generates an array of 1681 elements for each input matrix. Then the array becomes an input sample of the fully connected layer, which has 1681 input neurons, a hidden layers and nine output neurons. The Adam optimizer is used in the training process.

The learning rate is set to 0.001 for the first 100 epochs and decrease for refine tuning. The convolution and pooling process of CNN allow the preparation of SDN IDS model to learn complicated pattern of features form network traffic, while maintaining reasonable storage and computation overhead. Under the SDN context, we just focus on the basic features and traffic-based features and content-based features. For traditional networks, an anomaly-based NIDS is trained with a huge number of features including packet content-based features. This consumes a lot of computer resources and time. Many researchers have developed feature selection algorithms to reduce the feature dimension and gain higher detection accuracy. In our experiment, we created

three distinct sub-datasets that contain traffic samples with features extracted from the NSL-KDD dataset:

- ❖ Basic Feature Set: contains basic features of individual TCP connections
- ❖ Traffic Feature Set: contains features of network traffic
- ❖ Content based feature: is trained with a huge number of features.

The main purpose of these subsets is to evaluate the role of each type of feature in the detection performance.

3.10 Network Performance Evaluation Setup

Cbench [76] is a standard tool used for evaluating SDN OpenFlow controllers. In this chapter, Cbench is used to evaluate the overheads of the DeepIDS on the SDN controller. We measured the performance of the controller in terms of

Throughput and latency. Cbench tests can run in either throughput or latency mode.

In throughput mode, Cbench sends a stream of PACKET IN messages to the controller for a specified period and then records the number of responses (PACKET OUT messages) for the request that it has sent to the controller. Throughput data reflects the average number of flows the controller could treat per second in each switch.

In latency mode, Cbench sends a PACKET IN message to the controller and waits for the response (PACKET OUT message) before sending another packet. The latency results represent the average number of milliseconds that a flow consumes to be installed in each switch.

Figure 3.12 shows the flow diagram of the network performance evaluation process. At the beginning of the evaluation process, Cbench connected to the SDN controller. Cbench emulates a network topology by itself, then sends PACKET IN messages to the SDN controller and waits for PACKET OUT messages from the SDN controller. We tested the controller's throughput and latency with a different number of virtual OpenFlow switches emulated by Cbench.

We evaluate the network performance in six scenarios. Details about the network parameters of each scenario are described in Figure 3.12 below.

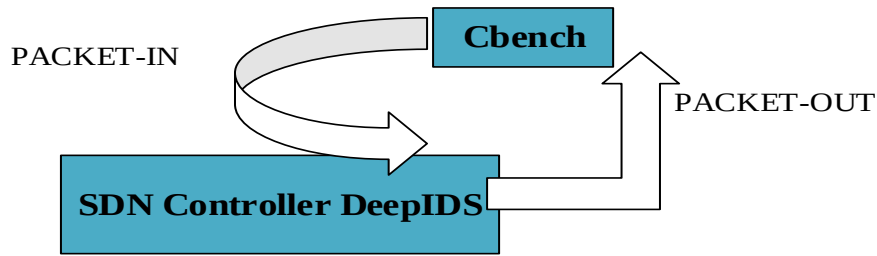


Figure 3.12 Network Evaluation Process

In all the testing scenarios, each switch is connected with an eighty thousand hosts in a star topology. Then all the switches are connected to the SDN controller.

Table 3. 5 Network parameter

Number of Switches	Number of Host in switch
8	1000
16	1000
32	1000
64	1000
128	1000
256	1000

Each Cbench run consists of 10 test loops with a duration of 10 seconds. We ran the controller with a typical layer 2 learning switch application. The obtained results are the partially values of the 10 tests. We evaluate the performance of the controller in the following scenarios.

- ❖ **The SDN controller runs without DeepIDS.** This scenario serves as a baseline for our evaluation.
- ❖ **The SDN controller** runs with DeepIDS or other ML algorithms being enabled.

Encoding the Categorical Data: Encoding categorical data value is encoded to a numeric value. Machine learning requires all input values must be numeric before fit to the model and evaluation. The NSL-KDD dataset contains 3 categorical values, thirty-nine of them are attack type, and one is normal software defined network feature set? The proposed model in this study

identifies whether the Software defined network feature set is normal or attack, so encoding all attack categories to number one and normal network feature set to number zero was applied.

Data Transformation: Each feature in the dataset may contain a different range of values, magnitude, data types, and scale. To make all features contribute equally to the result data transformation is applied, besides normalization. Is also used to rescale to range values b/n 0 and 1. Data transformation can have achieved through data normalization [72].

Normalization: Deep learning algorithm trains and develop a model than expected output compared with actual output in hopping the [72] difference is minimum (minimum loss function) Gradient Descent techniques scale the magnitude of parameters. Small incremental update results in converging on the optimal value on the other hand drastic update results in divergent behavior. The mentioned drawback was overcome by using learning rates.

The loss function must be at the lowest value while finding the optimal value of parameters with the help of the learning rate. MinMax normalization is a type of normalization it changes the range of data, scaled to a fixed range. The normalized dataset is proportionally distributed and slightly uniform. Normalized input helps the neural network layer to learn the parameters at the beginning of the layer. The relationship between the original values of the dataset is preserved after normalization [66], [72]. This study selected min-max normalization because it intended not to lose the relationship of original dataset values.

Missing Value and Duplicate Values Handling: According to R.B Kline there are three methods to handle missing values in data science [80]. Ignoring method [77], imputation method, and last one is using the modeling method [78]. According to [17] ignoring techniques is widely used when the dataset is big [78]. Duplicate values in the dataset result in accuracy decreasing and consumption of computational resources so removing redundancy is vital beside it helps to make the dataset consistent [78, 79]. Ignoring techniques is choosing for this study because the NSL-KDD dataset contains over one million instances of huge network flow.

Data Dimensional Reduction: A large number of features increase computation complexity, increase resource usage, and time consumption is high [82]. To solve the mentioned drawback finding relevant and significant features of large network traffic is need besides it increases the accuracy of the neural network [80]. Even though there are several feature reductions machines learning algorithms Autoencoder is the ideal one. Autoencoder is a deep learning algorithm that is known for its capability of dimensional reduction and outperforms other dimensional reduction

algorithms [81]. This study uses a big dataset size that needs to be reduced and Auto encoder is chosen for its capability to reduce large input to small representation.

Training Dataset: a subset of the normal network traffic flow dataset used to train deep learning networks. Autoencoder train only on the training dataset and try to create a model by studying it. Any deviation from the developed model is considered an outlier or anomaly.

Validation Dataset: is a subset of the dataset which is attack-free instance network flows. Validation dataset used to tune the parameter of the developed model, such as find an optimal the number of hidden units or determine the stopping point.

Test Dataset: the subset of the dataset which includes an instance of Benning and attack network flows. Used to assess the performance of fully trained deep learning.

Optimizer: Good models have minimum loss function while finding the optimal value of parameters with the help of the learning rate [86, 87]. The learning rate of deep learning must be perfect for finding the optimal value of parameters quickly. It should be between zero and one. Smaller learning rates changes the parameters slowly, requires more training epoch even the the process may stack. In reverse, if the learning rate is too large models converge quickly to a suboptimal solution [86, 87].

The optimization method helps the algorithm to achieve the optimum solution for nonlinear problems, better success rate, and frequently and quickly update parameter values [68], [85]. Such as Stochastic Gradient Descent (SGD), Adadelata, Adam, RmsProp, Nadam optimization methods were used in the study on Keras.

Adaptive Moment Estimation (Adam): This is an adaptive optimization method, have the advantages of Popular methods AdaGrad and RMSProp AdaGrad works well with sparse gradients and RMSProp works in on-line and non-stationary setting [87, 88] during the training of deep learning on large scale data Adaptive Moment Estimation helps to optimize parameters [89].

Adaptive Moment Estimation have many pros such as memory requiems is little, efficient computationally, for a problem with large data and parameter is well suited, also suited with a problem with very noisy and spare gradients, responds, fast, work with squares of gradients and the average of the Second-order moment [87, 88, 89].

3.11 Preprocessing

Even though preprocessing steps are essential, applying all the commonly used steps may put the detection model in trouble. For example, stop-word removal could be advantageous in

documentation classification as the text's central idea can be recognized even if the stop-word is removed; and which leads to improved performance. But as the problem under study is different and as the data visualized in the word cloud above shows; stop-words appeared to be common in authentic news peculiarly, which could be an important feature in fitting the deep learning models for better classification results. So, by considering such an adverse effect of some preprocessing steps, we adhered to apply only other preprocessing techniques such as cleaning, normalization, and tokenization.

3.12 Model Evaluation

To see if the proposed model is performing well on unseen data and trust its prediction, a model evaluation should be done. Evaluating a model is an important step in the machine learning pipeline to assess whether the model is merely memorizing the data or not. And more importantly to evaluate the generalization capability of the model on new out-of-sample data, which the model doesn't see in the process of fitting/training the model. There are two approaches to evaluate the given model performance.

The first and straightforward method is the holdout method, where the evaluation strategy is based on testing a model on data that was not seen by the model at training time. Even though the approach is advantageous in terms of simplicity, speed, and flexibility; it has a downside in resulting difference in estimating accuracy as a result of its high variability among the training, validation, and test sets.

The second and widely used approach is K-Fold Cross-validation. This method is known for its skill in the less biased or less optimistic estimation over the previous simple train/split approach. And also, significantly reduce variance. As a result, k-fold cross-validation is selected as a model evaluation method in this research.

After Convolutional Neural Network trained on the training dataset, tune the parameters of the model on the validation dataset and develop the model. Finally, the developed model performance was evaluated on a test dataset. Model evaluation helps to understand how the model was performed. Below are used to determine the performance of the model. Accuracy: Total number of correct identification for attack/benign data concerning total made. If the accuracy is higher, the machine learning model is better [68], [87, 88].

Loss: Model loss of the proposed system evaluated. This model loss is expressed with two model

Loss variations that are a model loss against the test dataset and model loss on the validation dataset [89, 90]. The goal of training a model is to find a set of weights and biases that have low loss [89]. Model loss measured using Mean Squared Error (MSE). MSE is useful when the dataset contains a lot of noise, the dataset contains outliers, or unexpected values (too high or too low values) [89].

Area under Curve (AUC): indicates the performance of the system which fail to distinguish Outliers or attacks from normal network traffic. AUC tells how much the model is capable of Distinguishing between attack and normal network flows. Higher the AUC, better the model Identifying network flows of attack and normal [73], [75], [87].

Receiver Operating Characteristic Curve (ROC): is a graph that tells the performance of a model at all classification thresholds with a true positive rate against the false-positive rate at all possible threshold [69], [84], [85].

3.13 Hyperparameter Tuning

Although deep learning models may increase the training time due to their complexity, they outperform the traditional machine learning algorithms in terms of performance. Nevertheless, their performance relies extremely on the selected values of hyper parameters. Hence, the critical concern when using one of these deep learning models is how to adjust its hyperparameters properly.

The hyperparameters of a deep learning model are the set of fundamental settings that control the behavior, architecture, and performance of the model in the underlying task. The problem is that the values of these hyperparameters are varying from task to task, and prior knowledge of these values is required to set them just before the learning process begins.

Hyperparameter tuning is a technique to find hyperparameter values for an ML model that performs the best on a validation dataset for a problem [73]. Most of the time people use the terms parameter and hyperparameter interchangeably, however, they are different.

Model parameters are variables that are learned during training whereas model hyperparameters are variables set by the programmer before training and help to control the implementation of the model. For example, the weights learned during training a NN Model are parameters while the maximum number of iterations, regularization parameter (RegParam), and elastic net.

3.14. Evaluation Metrics to Evaluate the Accuracy of Model

This experiment uses the F1 score and the accuracy to evaluate the performance of the model. The model is trained on 70 % of the data while 15% is used for validation and 15% for testing.

The metrics used to evaluate the model in this classification task are

- ❖ Accuracy of classification accuracy (CA).
- ❖ Precision
- ❖ Recall
- ❖ F1-measure

Accuracy: - the model calculated as the percentage of correct prediction of the top class (the class having the highest probability as indicated by the CNN model) and the target class assigned by the author beforehand is the same. It can be represented by the below e

$$\text{Accuracy} = \frac{TP+TN}{TP+FP+FN+TN} \dots\dots\dots \text{Equation 3.1}$$

Where TP (True positive) represents the positive instances that are correctly classified as positive, TN (True Negative) represents the negative instance that is correctly classified as negative, FP (False positive) represents the negative instance that is wrongly classified as positive, FN (False Negative) represents the positive that is wrongly classified as negative.

Precision: - is calculated as the fraction of true positives (TP) from the sum of the relevant classes, i.e. the sum of the true positives and the false positives. It can be represented by the following formula.

$$\text{Precision} = \frac{TP}{TP+FP} \dots\dots\dots \text{Equation 3.2}$$

Recall: - is calculated as the fraction of true positives from the sum of True positives and False Negatives. It can be represented by the below

$$\text{Recall} = \frac{TP}{TP+FN} \dots\dots\dots \text{Equation 3.3}$$

F1 score:-we used in this experiment as the dataset is imbalanced. F1 score is interpreted as the harmonic mean of precision and Recall. It can be represented by the below

$$\text{F1 Score} = \frac{2 * \text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}} \dots \dots \dots \text{Equation 3.4}$$

A confusion matrix: - is also an evaluation metric that is used to describe the performance of a classification task. Precision, Recall, and Accuracy can all be calculated with the help of a confusion matrix

Table 3. 6 Confusion matrix

	Positive(Actual)	Negative(Actual)
Positive(Prediction)	True Positive(TP)	False Negative(FP)
Negative(Prediction)	False Positive(FP)	True Negative(TN)

CHAPTER FOUR

IMPLEMENTATION

4.1. Introduction

The section provides detail on how the entire implementation of the research was done for Creating an efficient intrusion detection system for software defined network using Machine Learning/Deep learning and classification model. It also discusses how the detected intrusion was suggested based on the classification.

4.2. Environment

The research was implemented using python 3.7.7 using Jupyter notebook as the main integrated development environment (IDE). Python language was selected as there is a lot of support from an active community for intrusion detection for software defined network using Machine learning/Deep learning using Tensorflow with Keras.

The study was initially started on a local windows system but was eventually moved to the Google cloud platform (GCP) due to a shortage of resources. A Machine Learning/ Deep Learning intrusion instance was created on the cloud, which is optimized for Machine Learning/ Deep Learning and has all the required packages pre-installed.

4.3 System Architecture

The architecture of the proposed network IDS has different components. The system architecture has preprocessing, train-test split, validation split, training with a default parameter, and grid search phases. The preprocessing phase consists of sub-tasks: string indexing to convert string data to numeric, one-hot encoding to convert string indices into 0s and 1s, feature selection to select relevant features, normalization for standardizing range of values, and vector assembling for merging features into a single feature vector. In the train-test split phase, the whole preprocessed data is split into a training set (for training machine learning algorithms) and a testing set (for evaluating models' performance).

In the validation split phase, the training set is further split into actual training and validation set. In the training with default parameters phase, the machine learning algorithm is trained using the training data on their default parameters the architecture of the proposed network IDS has different components. The system architecture has preprocessing, train-test split, validation split, training

with a default parameter, and grid search phases. The preprocessing phase which consists of sub-tasks: string indexing to convert string data to numeric, one hot encoding to convert string indices into 0s and 1s, feature selection to select relevant features, normalization for standardizing range of values, and vector assembling for merging features into a single feature vector.

In training with default parameters phase, and produced a trained model. The trained model has then evaluated its performance on the separated preprocessed testing data and evaluation result 1 is generated. The grid search phase is used to find the best parameters for a model by constructing a parameter grid. In this phase first, the training and validation data is passed to the grid search. Then, the parameter grid of hyper parameters and classification algorithm is specified and passed to the cross-validation.

The cross-validation searches the whole parameter grid and running the training set for selecting the best parameters. Then, the models are evaluated on the validation set. The selected best parameters are then used for building the final trained model which is then evaluated on the testing set. The result of this evaluation is evaluation result 2. A detailed description of each component of the system architecture is presented in the following sections.

4.3.1. Preprocessing for NSL-KDD Dataset

Dataset integration: Dataset NSL-KDD is coming with eight separate data frames. Each data frame has the same number of features that is 41 features. Out of 42 features, 42 features data each traffic sample in this dataset has 41 network features that are categorized into three types of features: basic features (1 - 9), content-based features (10 - 22), and traffic-based features (23 - 41). Machine learning requires all input values must be numeric before fit to the model and evaluation.

The NSL-KDD dataset contains 3 categorical values, thirty-nine of them are attack types and one is normal software defined network feature set. Data Preprocessing is one of the critical steps in the data mining process which does the preparation and transformation of the original dataset. The various steps are included in Data preprocessing, they are Data cleaning, Feature reduction, Feature construction. Feature Reduction includes Feature extraction and Feature selection. Feature extraction, selection, and construction all are independent methods in data preprocessing. They can be combined depending on the problem analyzed like feature extraction followed by feature selection, feature construction followed by feature selection. In this paper, we follow the combination of feature extraction followed by feature selection.

To develop an intrusion classification model, a huge amount of network traffic data is needed to provide enough data to the classifiers to train them properly. The publicly available network intrusion detection dataset is collected from the Web [89]. Even though there are many publicly available datasets for simulating network intrusions, all of them may not help develop effective IDSs. For example, the NSL-KDD dataset which is available at [90] is the most widely used and freely available network intrusion detection evaluation dataset. However, the major limitation of this dataset is the presence of redundant and irrelevant records in the training and testing set, and using it reduced the quality of system performance.

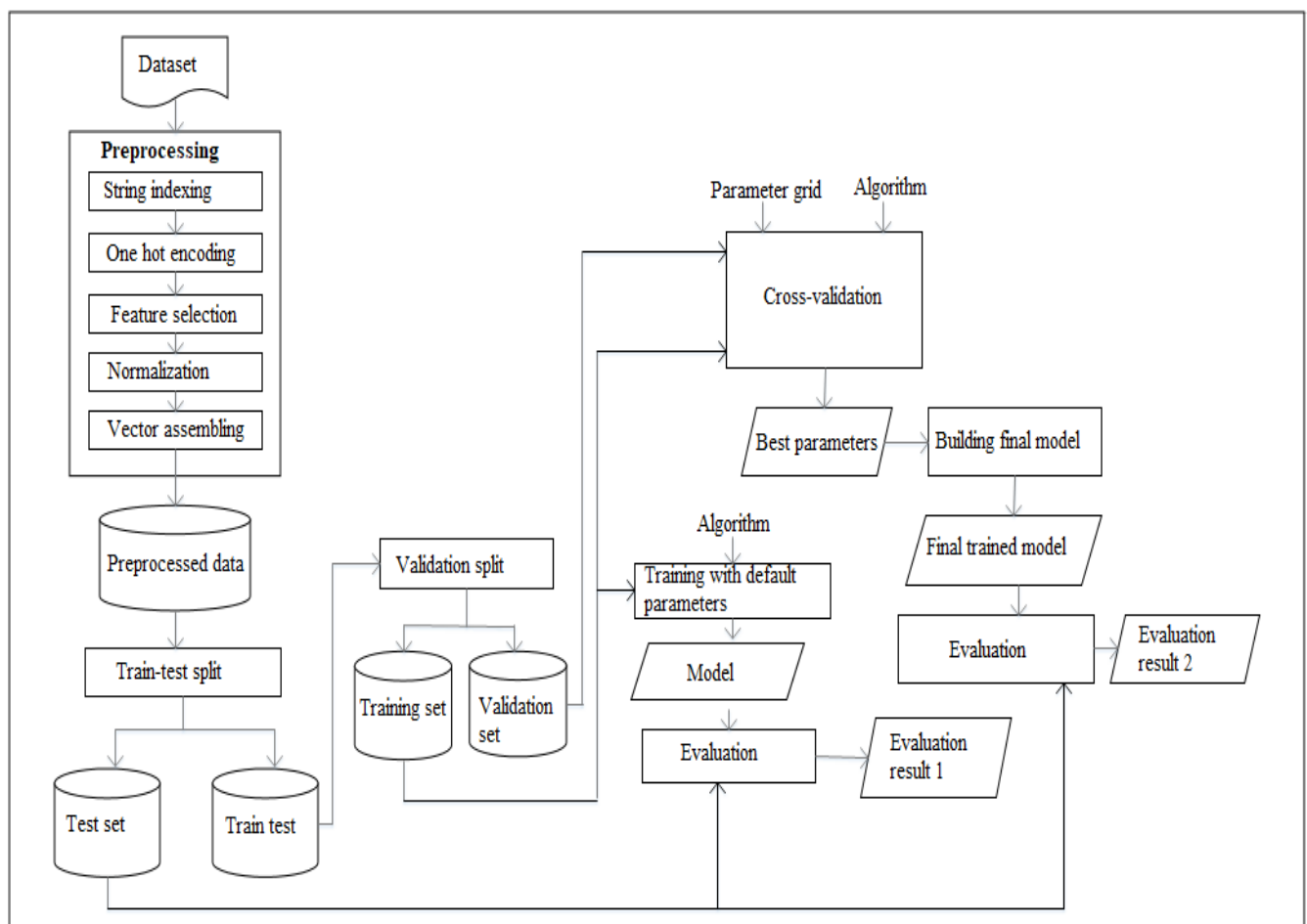


Figure 4.1 Detail NSL KDD dataset set preprocessing architectures

4.3.2. Feature Extraction and Selection

Feature Extraction transforms data from high dimensionality to low dimensionality. Feature extraction is a process that determines what evidence that can be taken from audit data is most useful for analysis in this paper,

The principal Component Analysis method is used for feature extraction. PCA is a linear method in dimensionality reduction for data analysis and compression. It is based on transforming a relatively large number of uncorrelated features by finding an orthogonal linear combination of the original features with the largest variance.

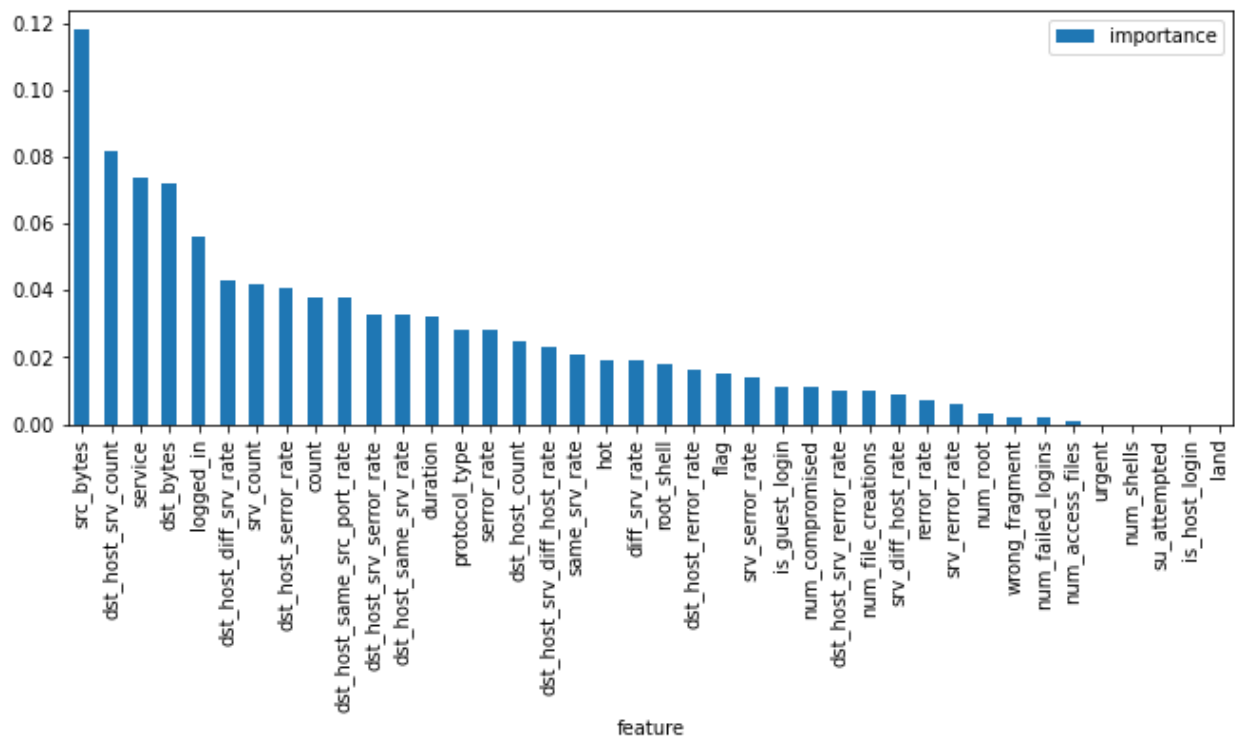


Figure 4.2 Dataset Feature Selection

4.3.3. Feature Set for Anomaly NIDS

In broad anomaly detection, separate feature sets are built for each anomaly detector. For, traffic-based anomaly NIDS, features set of multiple connection derivative features are included, which specifies the count of connection derivative feature is included, which specifies the counts of connection to a particular destination IP address and port. In this paper a

Feature set of 41 features namely protocol, src_bytes, Dst_host_count, Dst_host_same_srv_rate is constructed [10]. The feature set constructed for anomaly detection is used to determine DDOS attack [16].

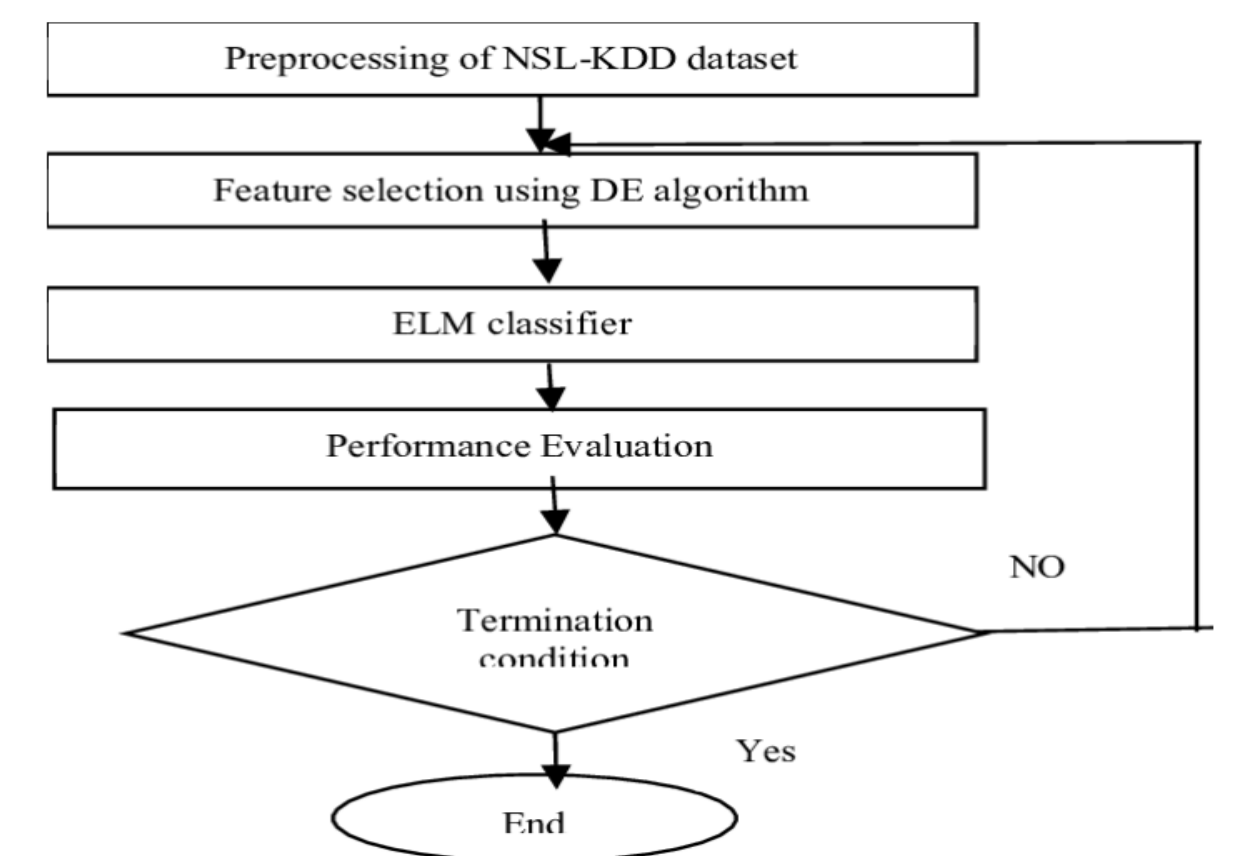


Figure 4.3 NSL-KDD dataset Preprocessing

The Preprocessing Method shown in Figure 4.3 for NSL-KDD dataset the first step is to prepare the data for classification by converting the symbolic features to numeric features during the representation process. For example, the second feature called protocol contains {tcp, udp, and icmp}.

The NSL-KDD dataset contains 3 categorical values, thirty-nine of them are attack type, and one is normal software defined network feature set.

This dataset contains 125,973 records for training and 22,544 records for testing. Details about the distribution of this dataset are presented in Figure 4.4. The number of legitimate and anomaly traffic is balanced to make sure that the ML/DL algorithms can be trained properly.

```
## add the columns' name and read the KDDTrain+ and KDDTest+ datasets
col_names = ["duration", "protocol_type", "service", "flag", "src_bytes",
             "dst_bytes", "land", "wrong_fragment", "urgent", "hot", "num_failed_logins",
             "logged_in", "num_compromised", "root_shell", "su_attempted", "num_root",
             "num_file_creations", "num_shells", "num_access_files", "num_outbound_cmds",
             "is_host_login", "is_guest_login", "count", "srv_count", "error_rate",
             "srv_serror_rate", "rerror_rate", "srv_rerror_rate", "same_srv_rate",
             "diff_srv_rate", "srv_diff_host_rate", "dst_host_count", "dst_host_srv_count",
             "dst_host_same_srv_rate", "dst_host_diff_srv_rate", "dst_host_same_src_port_rate",
             "dst_host_srv_diff_host_rate", "dst_host_serror_rate", "dst_host_srv_serror_rate",
             "dst_host_rerror_rate", "dst_host_srv_rerror_rate", "label"]

#Training set
df = pd.read_csv('C:/Users/muhe/Desktop/Deep-learning-for-intrusion-detection-using-Recurrent-Neural-network-RNN-main /Deep-learning-for-intrusion-detection-using-Recurrent-Neural-network-RNN-main/KDDTrain+.csv')

# Testing set
df_test_21 = pd.read_csv('C:/Users/muhe/Desktop/Deep-learning-for-intrusion-detection-using-Recurrent-Neural-network-RNN-main/Deep-learning-for-intrusion-detection-using-Recurrent-Neural-network-RNN-main/KDDTest+.csv')

print('Dimensions of the Training set:', df.shape)
print('Dimensions of the Test_21 set:', df_test_21.shape)
```

Dimensions of the Training set: (125973, 42)
Dimensions of the Test_21 set: (11850, 42)

Figure 4.4 Dataset before Preprocessing

The review of the feature before preprocessing the 42 NSL-KDD dataset feature

```
# review the features type
df.head(5)
```

	duration	protocol_type	service	flag	src_bytes	dst_bytes	land	wrong_fragment	urgent	hot	...	dst_host_srv_count	dst_host_same_srv_rate	dst_host_
0	0	tcp	ftp_data	SF	491	0	0	0	0	0	...	25	0.17	
1	0	udp	other	SF	146	0	0	0	0	0	...	1	0.00	
2	0	tcp	private	S0	0	0	0	0	0	0	...	26	0.10	
3	0	tcp	http	SF	232	8153	0	0	0	0	...	255	1.00	
4	0	tcp	http	SF	199	420	0	0	0	0	...	255	1.00	

5 rows x 42 columns

Figure 4.5 Review of the Feature Type

We must perform label distribution for both training and testing feature set before preprocessing. Labels distribution in Training, Validation, and Test Sets. In brackets, the respective percentage value. Class ID Training set, Validation set, and Test set.

When carrying machine learning or Deep learning having an (even quick) look at distribution within the data set helps in getting a little understanding of how data has been generated. Most of the time, and for a broad number of professional projects, since you have to carry on with the data you've got, you cannot make a lot of assumptions from this kind of observation. When it comes to distribution between training, validation, and test sets, it's kind of slightly different though. Indeed, it's much recommended to at least have a glance at how data is distributed within the three sets, and even ensure that this distribution is the same.

Put simply, if you build a model using an algorithm and associated parameters using your training set, it seems reasonable to argue that such a model won't be able to describe what's going on within a data set sourced from other processes. We usually think of model parameters as characteristics of the data generating process that gave birth to the data set, we have to work with. Not willing to reinvent the wheel here, so we'd point to [this_answer](#) which explains it quite well, using a model example.

Concerning oversampling and under-sampling as mentioned above, these are pretty useful methods of ensuring that an imbalanced distribution between target labels in your data set would be properly extended to the process of splitting it into training, validation, and test sets while avoiding bias propagation due to the difference in occurrences between labels.

```

print('Label distribution in the Training set:')
print("*****")
print(df['label'].value_counts())
print('\n')
print('Label distribution in the Test set:')
print("*****")
print(df_test_21['label'].value_counts())

```

Label distribution in the Training set:		Label distribution in the Test set:	
*****		*****	
normal	67343	normal	2152
neptune	41214	neptune	1579
satan	3633	guess_passwd	1231
ipsweep	3599	mscan	996
portsweep	2931	warezmaster	944
smurf	2646	apache2	737
nmap	1493	satan	727
back	956	processtable	685
teardrop	892	smurf	627
warezclient	890	back	359
pod	201	snmpguess	331
guess_passwd	53	saint	309
buffer_overflow	30	mailbomb	293
warezmaster	20	snmpgetattack	178
land	18	portsweep	156
imap	11	ipsweep	141
rootkit	10	httptunnel	133
loadmodule	9	nmap	73
ftp_write	8	pod	41
multihop	7	buffer_overflow	20
phf	4	multihop	18
perl	3	named	17
spy	2	ps	15
Name: label, dtype: int64		sendmail	14
		xterm	13
		rootkit	13
		teardrop	12
		xlock	9
		land	7
		xsnoop	4
		kernel_name-python3#	3

Figure 4.6 Label distribution in train and test set

Step 1: Data preprocessing:

The main reason why we need data preprocessing arises from the fact that redundant data and insignificant features may be unclear to the classification algorithm and this may lead to inaccurate and less generalizable models. Usually, the data collected from different sources may not be in a format that is suitable for a machine, may contain invalid and/or missed values, and maybe too large for a specific task.

Machines need specific representations of the input data for both training and testing which are termed features before a feed to an ML algorithm. To prepare the data in that format preprocessing tasks such as string indexing, one-hot encoding, feature selection, normalization, and vector assembling is performed on the training and testing data. Please find the attached sample of codes

for string indexing, one-hot encoding, and vector assembling in Annex A. A detailed discussion of these concepts is presented in the next sections.

One-Hot-Encoding is used to convert all categorical properties to binary properties. One-Hot-Encoding requirement, the input to this transformer must be an integer matrix expressing values taken with categorical (discrete) properties. The output would be a sparse matrix in which each column corresponds to a possible value. It is assumed that the input properties have values in the range [0, values]. Therefore, to convert each category to a number, properties must first be converted with Label Encoder.

Steps in Data Preprocessing

The steps followed in data pre-processing: - Import libraries-> Read data-> Checking for missing values-> Checking for categorical data-> standardize the data-> PCA transformation-> Data splitting. See the following code.

```
: # columns are categorical, not yet binary: protocol_type (column 2), service (column 3), flag (column 4).
print('Training set:')
for col_name in df.columns:
    if df[col_name].dtypes == 'object' :
        unique_cat = len(df[col_name].unique())
        print("Feature '{col_name}' has {unique_cat} categories".format(col_name=col_name, unique_cat=unique_cat))

print()
print('Distribution of categories in service:')
print(df['service'].value_counts().sort_values(ascending=False).head())
```

```
Training set:
Feature 'protocol_type' has 3 categories
Feature 'service' has 70 categories
Feature 'flag' has 11 categories
Feature 'label' has 23 categories
```

```
Distribution of categories in service:
http      40338
private   21853
domain_u    9043
smtp       7313
ftp_data   6860
Name: service, dtype: int64
```

Figure 4. 7 Dataset preprocessing

LabelEncoder: Insert categorical features into a 2D numpy4. 10 array and transform categorical features into numbers using LabelEncoder ().

```
In [11]: df_categorical_values_enc=df_categorical_values.apply(LabelEncoder().fit_transform)
print(df_categorical_values.head())
print('-----')
print(df_categorical_values_enc.head())
# test set
testdf_categorical_values_enc=testdf_categorical_values.apply(LabelEncoder().fit_transform)
```

	protocol_type	service	flag
0	tcp	ftp_data	SF
1	udp	other	SF
2	tcp	private	S0
3	tcp	http	SF
4	tcp	http	SF

	protocol_type	service	flag
0	1	20	9
1	2	44	9
2	1	49	5
3	1	24	9
4	1	24	9

Figure 4. 8 Transform categorical feature

One-Hot-Encoding: Since the output of string indexing is numeric, the model can be confused thinking that a column has data with some kind of order or hierarchy, but these numbers do not have any order. To avoid this, the OneHotEncode of that column is necessary. To do this, an OneHotEncoder transformer that encodes the index of a categorical value as a binary vector is used. OneHotEncoder takes a column that has categorical data, which has been label encoded, and then splits the column into multiple columns. The indices are replaced by 1s and 0s, depending on which column has what value. This encoding allows algorithms that can only operate with numerical valued features, to be applied to categorical features.

```
In [12]: enc = OneHotEncoder(categories='auto')
df_categorical_values_encenc = enc.fit_transform(df_categorical_values_enc)
df_cat_data = pd.DataFrame(df_categorical_values_encenc.toarray(),columns=dumcols)
# test set
testdf_categorical_values_encenc = enc.fit_transform(testdf_categorical_values_enc)
testdf_cat_data = pd.DataFrame(testdf_categorical_values_encenc.toarray(),columns=testdumcols)
```

Figure 4. 9 One-hot-encoding for categorical data

Missing columns in the test set are added and new numeric columns are added to the main data frame:

```
In [14]: newdf=df.join(df_cat_data)
newdf.drop('flag', axis=1, inplace=True)
newdf.drop('protocol_type', axis=1, inplace=True)
newdf.drop('service', axis=1, inplace=True)

# test data
newdf_test=df_test_21.join(testdf_cat_data)
newdf_test.drop('flag', axis=1, inplace=True)
newdf_test.drop('protocol_type', axis=1, inplace=True)
newdf_test.drop('service', axis=1, inplace=True)

print(newdf.shape)
print(newdf_test.shape)

(125973, 123)
(11850, 123)
```

Figure 4. 10 Missing columns in the test set are added

After feature preprocessing using one-hot encoding and label encoder. The protocol type, service, and flag previously are categorical. Then when we implement an encoder to the three feature sets it can be converted to numeric which can be understood by machine learning.

```
Out[16]:
```

	Protocol_type_icmp	Protocol_type_tcp	Protocol_type_udp	service_IRC	service_X11	service_Z39_50	service_aol	service_auth	service_bgp	service_courier
0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

5 rows x 84 columns

Figure 4. 11 Final NSL-KDD dataset

Feature Scaling:

```
In [15]: # step1: apply the logarithmic scaling method for scaling to obtain the ranges of `duration[0,4.77]', `src_bytes[0,9.11]' and `d:
newdf['log2_value1'] = np.log2(newdf['duration'])
newdf['log2_value2'] = np.log(newdf['src_bytes'])
newdf['log2_value3'] = np.log(newdf['dst_bytes'])
newdf=newdf.drop(['log2_value3','log2_value2','log2_value1'], axis=1)

# testing set

newdf_test['log2_value1'] = np.log2(newdf_test['duration'])
newdf_test['log2_value2'] = np.log(newdf_test['src_bytes'])
newdf_test['log2_value3'] = np.log(newdf_test['dst_bytes'])
newdf_test=newdf_test.drop(['log2_value3','log2_value2','log2_value1'], axis=1)

print(newdf)
print(newdf_test)
```

C:\Users\muhe\Anaconda3\lib\site-packages\pandas\core\arraylike.py:358: RuntimeWarning: divide by zero encountered in log2
result = getattr(ufunc, method)(*inputs, **kwargs)
C:\Users\muhe\Anaconda3\lib\site-packages\pandas\core\arraylike.py:358: RuntimeWarning: divide by zero encountered in log
result = getattr(ufunc, method)(*inputs, **kwargs)

	duration	src_bytes	dst_bytes	land	wrong_fragment	urgent	hot	\
0	0	491	0	0	0	0	0	
1	0	146	0	0	0	0	0	
2	0	0	0	0	0	0	0	
3	0	232	8153	0	0	0	0	
4	0	199	420	0	0	0	0	
...	...	...	...	...	...	...	...	...
125968	0	0	0	0	0	0	0	
125969	8	105	145	0	0	0	0	
125970	0	2231	384	0	0	0	0	
125971	0	0	0	0	0	0	0	

Figure 4. 12 Feature Scaling

RNN Model is a special type of deep neural network or deep learning algorithm.

In [90] flow-based anomaly detection system by constructing a deep neural network (DNN) architecture is proposed to secure the SDN network. The structure of the Deep learning system includes three basic layers. An input layer with six dimensions, three hidden layers contain twelve, six, and three neurons respectively, and an output layer with two dimensions. To achieve a better detection rate, the same authors propose another intrusion detection system for SDN networks [89] the proposed approach is based on a deep recurrent neural network. The authors advocate a Gated Recurrent Unit.

Recurrent neural network (RNN) model to build an IDS contain input and output layers with six and one dimensions respectively, and hidden layers which have six, four, and two dimensions. The system architecture is composed of Flow Collector, Anomaly detector, and Anomaly Mitigator. For the test results and training phases, they use the NSL-KDD dataset. The first solution [90] achieves an accuracy of 75.75% using six basics. Input layer LSTM input layer must be 3D the meaning of the 3 input dimensions is: samples, time steps, and features. The number of samples is

assumed to be 1 or more. Reshape () function takes a tuple as an argument that defines the new shape. number_of_rows_to_process_each_loop, the_time_interval_for_next_move (e.g., per day, per month), column

RNN Model

```
In [25]: # multi-class classification
from keras.models import Sequential
from keras.layers import Dense
from keras.wrappers.scikit_learn import KerasClassifier
from keras.utils import np_utils
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from sklearn.pipeline import Pipeline

# define baseline model
def baseline_model():
    # create model
    model = Sequential()
    model.add(Dense(80, input_dim=122, activation='relu'))
    model.add(Dense(5, activation='softmax'))
    # Compile model

    model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
    return model

estimator = KerasClassifier(build_fn=baseline_model, epochs=20, batch_size=5, verbose=0)
kfold = KFold(n_splits=10, shuffle=True)
results = cross_val_score(estimator,xtest,ytest, cv=kfold)
print("Accuracy: %.2f%%" % (results.mean()*100))

Accuracy: 90.87%
```

Figure 4. 13 RNN model Accuracy

4.3.4. Data Split

The dataset was split into three sections: A training portion dedicated to training the classifier a Validation portion that would allow the training process to improve and a testing portion that is Completely hidden from the training process and will be used for validating the classifier on Unknown data. All split create that contain a 70/30 ratio of the class of intrusion detection to avoid the unnecessary bias that would incur if a dataset would contain intrusion dataset from only one category.

Python's random shuffle generator was used to shuffle the order of intrusion detection for software defined networks and avoid having datasets with simpler external conditions. This way, intrusion of attack with different shapes, angles, levels of intrusion detection, main attackers are all included

in all categories to achieve the highest possible variability. Variability in the dataset ensures that the model can be trained in the most generalized fashion possible, and evaluated and tested under all conditions.

The first data split takes place by removing 15% of the total dataset and saving it for later uses as testing. The remaining 85% of the dataset is then split again into a 70/15 ratio, resulting in the following portions, as seen in Figure 4.14.

Filter all the rows with label value of train and test set except itself that is in filters functions.

<pre>In [27]: to_drop_DoS = [0,1] to_drop_Probe = [0,2] to_drop_R2L = [0,3] to_drop_U2R = [0,4] # Filter all rows with Label value except itself # isin filter function DoS_df=newdf[newdf['label'].isin(to_drop_DoS)]; Probe_df=newdf[newdf['label'].isin(to_drop_Probe)]; R2L_df=newdf[newdf['label'].isin(to_drop_R2L)]; U2R_df=newdf[newdf['label'].isin(to_drop_U2R)]; #test DoS_df_test=newdf_test[newdf_test['label'].isin(to_drop_DoS)]; Probe_df_test=newdf_test[newdf_test['label'].isin(to_drop_Probe)]; R2L_df_test=newdf_test[newdf_test['label'].isin(to_drop_R2L)]; U2R_df_test=newdf_test[newdf_test['label'].isin(to_drop_U2R)]; print('Train Set:') print('Dimensions of DoS:',DoS_df.shape) print('Dimensions of Probe:',Probe_df.shape) print('Dimensions of R2L:',R2L_df.shape) print('Dimensions of U2R:',U2R_df.shape) print() print('Test Set:') print('Dimensions of DoS:',DoS_df_test.shape) print('Dimensions of Probe:',Probe_df_test.shape) print('Dimensions of R2L:',R2L_df_test.shape)</pre>	Train Set: Dimensions of DoS: (113270, 123) Dimensions of Probe: (78999, 123) Dimensions of R2L: (68338, 123) Dimensions of U2R: (67395, 123) Test Set: Dimensions of DoS: (6496, 123) Dimensions of Probe: (4554, 123) Dimensions of R2L: (5037, 123) Dimensions of U2R: (2219, 123)
--	--

Figure 4. 14 Split the training set and testing set

The validation set is used during the training process to help the algorithm update its weights appropriately so that it can improve its performance and avoid overfitting. After the model has finished training, we run the testing data and verify if it has classified the test images correctly, thus creating the need to keep the testing set hidden.

```

In [29]: # Split dataframes into X & Y
# X Properties, Y result variable

X_DoS = DoS_df.drop('label',1)
Y_DoS = DoS_df.label

X_Probe = Probe_df.drop('label',1)
Y_Probe = Probe_df.label

X_R2L = R2L_df.drop('label',1)
Y_R2L = R2L_df.label

X_U2R = U2R_df.drop('label',1)
Y_U2R = U2R_df.label

# test set
X_DoS_test = DoS_df_test.drop('label',1)
Y_DoS_test = DoS_df_test.label

X_Probe_test = Probe_df_test.drop('label',1)
Y_Probe_test = Probe_df_test.label

X_R2L_test = R2L_df_test.drop('label',1)
Y_R2L_test = R2L_df_test.label

X_U2R_test = U2R_df_test.drop('label',1)
Y_U2R_test = U2R_df_test.label

```

Figure 4. 15 Splitting the Data frame

Since the column names can be deleted at this stage, we save the column names for later use.

Feature Scaling:

```

In [31]: from sklearn import preprocessing

scaler1 = preprocessing.StandardScaler().fit(X_DoS)
X_DoS=scaler1.transform(X_DoS)

scaler2 = preprocessing.StandardScaler().fit(X_Probe)
X_Probe=scaler2.transform(X_Probe)

scaler3 = preprocessing.StandardScaler().fit(X_R2L)
X_R2L=scaler3.transform(X_R2L)

scaler4 = preprocessing.StandardScaler().fit(X_U2R)
X_U2R=scaler4.transform(X_U2R)

# test data
scaler5 = preprocessing.StandardScaler().fit(X_DoS_test)
X_DoS_test=scaler5.transform(X_DoS_test)

scaler6 = preprocessing.StandardScaler().fit(X_Probe_test)
X_Probe_test=scaler6.transform(X_Probe_test)

scaler7 = preprocessing.StandardScaler().fit(X_R2L_test)
X_R2L_test=scaler7.transform(X_R2L_test)

scaler8 = preprocessing.StandardScaler().fit(X_U2R_test)
X_U2R_test=scaler8.transform(X_U2R_test)

```

Figure 4. 16 Feature Scaling for preprocessing data

4.4. Attack Types Category

As the internet is open for anyone, this creates opportunities for unauthorized users to perform malicious activities within it. A lot of threats are created every day by individuals and organizations to attack computer networks to steal private data and information. N External intruders are an unauthorized user of the system they attack, whereas internal intruders are legitimate users of the organization which has permission to access the system, but do not have the privilege to root or super-users.

Flow-based intrusion detection, since it relies only on header information, can address the only subset of the attacks present above, in particular, the research community currently provides an approach to detect the following class of attacks: such as (Denial of service, Scans, Worms, and Botnets). There are many kinds of attacks for example the few common types [50]: denial of service (DOS), a user to root (U2R), remote to local (R2L), and probe attacks.

Denial of Service (DOS) Attack: This class of attack blocks access to a certain resource (computing and memory) by making unavailable the network services so that legitimate users cannot access the resource allowed to them. The attackers create overloading traffic through the malicious bot to the targeted IP address and flood the network with more requests than the server can process.

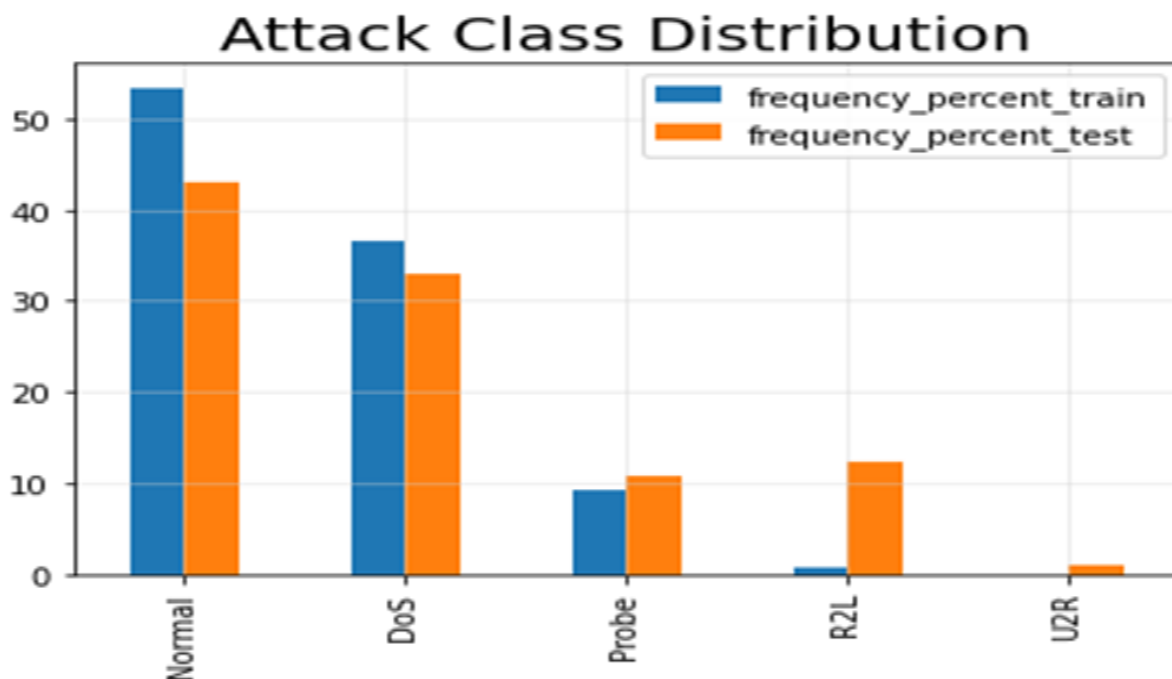


Figure 4. 17 Attack Classification Distribution

```

In [34]: #1. DoS
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
import matplotlib.pyplot as plt
import seaborn as sns

mlp = MLPClassifier(solver='adam', learning_rate_init = 0.0005, learning_rate = 'adaptive', activation="relu", max_iter=2000, ran
dec = DecisionTreeClassifier(criterion="entropy", max_depth=3)
ran = RandomForestClassifier(n_estimators=100)
svm = SVC(random_state=1)
naive = GaussianNB()

models = {"J48" : dec,
         "NB" : naive,
         "RF" : ran,
         "MLP" : mlp,
         "SVM" : svm
        }
scores= { }

for key, value in models.items():
    model = value
    model.fit(X_DoS, Y_DoS.astype(int))
    scores[key] = model.score(X_DoS_test , Y_DoS_test)

# results
scores_frame = pd.DataFrame(scores, index=["Accuracy Score"]).T
scores_frame.sort_values(by=["Accuracy Score"], axis=0 ,ascending=False, inplace=True)
print('DoS')
print(scores_frame)
print("\n")

```

Figure 4. 18 DOS Accuracy Comparison Jupyter Code

If we compare all the attack category, we can get the best result in terms of accuracy and other computation matrices. Therefore, we focus on the attack category that yields the best results **U2R**.

```
In [46]: # 4.U2R
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
mlp = MLPClassifier(solver='adam', learning_rate_init = 0.0005, learning_rate = 'adaptive', activation="relu", max_iter=2000, ran
dec = DecisionTreeClassifier(criterion="entropy", max_depth=3)
ran = RandomForestClassifier(n_estimators=100)
svm = SVC(random_state=1)
naive = GaussianNB()

models = {"J48" : dec,
         "NB" : naive,
         "RF" : ran,
         "MLP" : mlp,
         "SVM" : svm
        }

scores = { }

for key, value in models.items():
    model3 = value
    model3.fit(X_U2R, Y_U2R.astype(int))
    scores[key] = model3.score(X_U2R_test , Y_U2R_test)

# results
scores_frame = pd.DataFrame(scores, index=["Accuracy Score"]).T
scores_frame.sort_values(by=["Accuracy Score"], axis=0 ,ascending=False, inplace=True)
print('U2R')
print(scores_frame)
print('\n')
```

```
plt.figure(figsize=(8,5))
sns.barplot(x=scores_frame.index,y=scores_frame['Accuracy Score'])
plt.ylim(0, 1)
plt.ylabel("Accuracy%")
plt.xlabel("U2R")
```

Figure 4. 19 U2R Accuracy Comparison Jupiter Code

```
# Plot the results
plt.figure(figsize=(8,5))
sns.barplot(x=scores_frame.index,y=scores_frame["Accuracy Score"])
plt.ylim(0, 1)
plt.ylabel("Accuracy%")
plt.xlabel("Binary Classification/ KDDTest+")
```

Figure 4. 20 U2R figure plotting jupyter Code

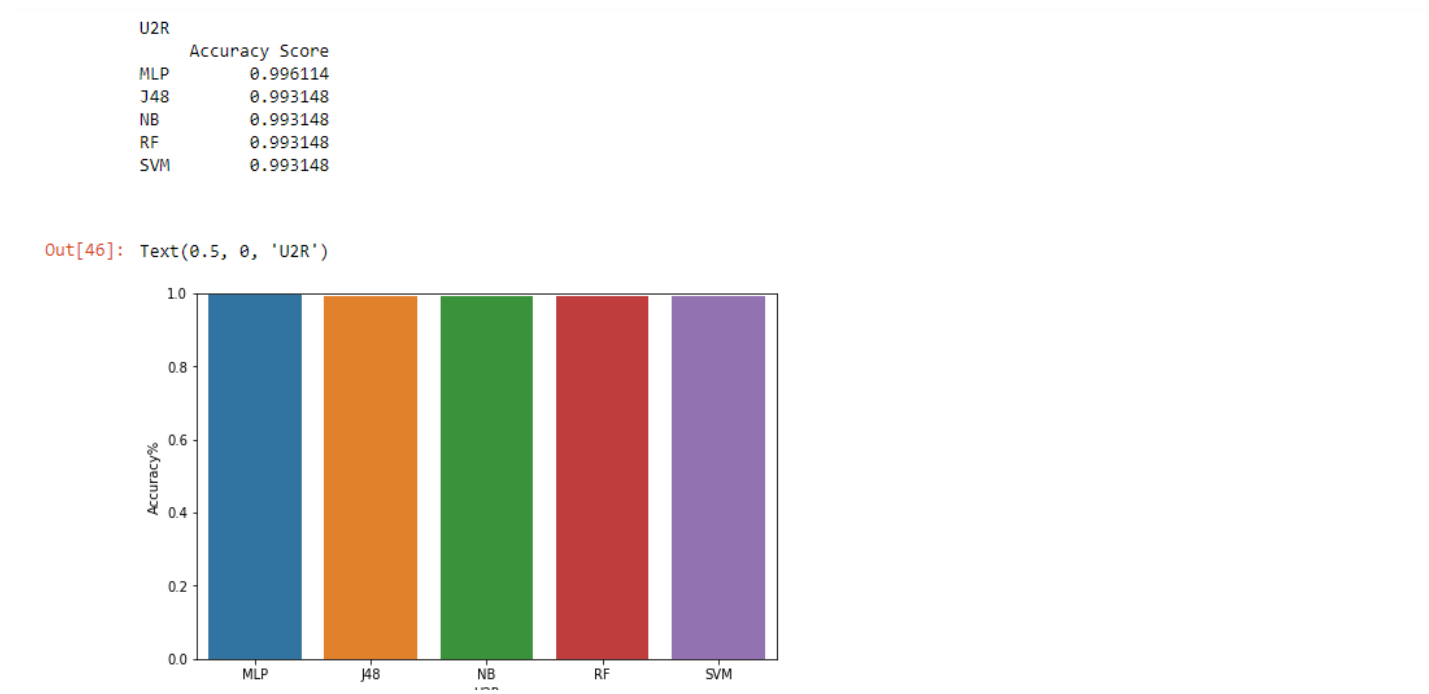


Figure 4. 21 U2R Accuracy Comparison graph

Attack Category Yield the best result is called **U2R** attack types. Prediction & Evaluation (validation): If evaluate the U2R the prediction and validation. It looks like the below Figures. Draw the confusion matrix of U2R.

```
In [43]: #4.U2R
Y_U2R_pred=model3.predict(X_U2R_test)
print('U2R')
# Create confusion matrix
pd.crosstab(Y_U2R_test, Y_U2R_pred, rownames=['Actual attacks'], colnames=['Predicted attacks'])
```

Figure 4. 22 U2R Attack Category jupyter Code

U2R		
Out[43]:	Predicted attacks	
	0	
	Actual attacks	
	0	2152
	4	67

Figure 4. 23 Confusion matrix

Previously check some supervised learning algorithm by creating an RNN model to check some accuracy and computational matrix. Some attack categories and compare the accuracy that yields the best results. Now by creating an ANN model which is a super class of the deep learning algorithm. In chapter two we use some supervised learning algorithms such as, Linear Regression, Random Forest, Support vector machine, Decision tree, Multilayer's perceptron, KNN, and ANN.

```
In [35]: # Create neural net
model = Sequential()
model.add(Dense(512, activation='relu'))
model.add(Dense(256, activation='relu'))
model.add(Dense(10, kernel_initializer='normal'))
model.add(Dense(2, activation='softmax'))

In [36]: model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

In [38]: monitor = EarlyStopping(monitor='val_loss', min_delta=1e-3, patience=3, verbose=1, mode='auto')
checkpointer = ModelCheckpoint(filepath="modelann.h5", verbose=0, save_best_only=True)

In [39]: model.fit(X_Df, y_binary, batch_size=128, epochs=50, verbose=1, validation_split=0.1)
```

Figure 4. 24 Creating Neural Network

```
Epoch 45/50
113375/113375 [=====] - 12s 102us/step - loss: 0.0208 - accuracy: 0.9913 - val_loss: 0.0244 - val_ac
curacy: 0.9908
Epoch 46/50
113375/113375 [=====] - 11s 98us/step - loss: 0.0207 - accuracy: 0.9911 - val_loss: 0.0238 - val_acc
uracy: 0.9908
Epoch 47/50
113375/113375 [=====] - 12s 106us/step - loss: 0.0211 - accuracy: 0.9907 - val_loss: 0.0318 - val_ac
curacy: 0.9898
Epoch 48/50
113375/113375 [=====] - 12s 107us/step - loss: 0.0219 - accuracy: 0.9912 - val_loss: 0.0263 - val_ac
curacy: 0.9899
Epoch 49/50
113375/113375 [=====] - 13s 112us/step - loss: 0.0199 - accuracy: 0.9917 - val_loss: 0.0250 - val_ac
curacy: 0.9907
Epoch 50/50
113375/113375 [=====] - 11s 96us/step - loss: 0.0203 - accuracy: 0.9912 - val_loss: 0.0237 - val_acc
uracy: 0.9907

Out[39]: <keras.callbacks.callbacks.History at 0x153993ea088>
```

Figure 4. 25 ANN Model fitting Output

Decision tree

```
In [47]: # Create Decision Tree classifier object
clf_Tree = DecisionTreeClassifier()
train0 = time.time()
# Train Decision Tree Classifier
clf_Tree = clf_Tree.fit(X_Df, Y_Df.astype(int))
train1 = time.time() - train0
```

```
In [50]: test0 = time.time()
Y_Df_pred=clf_Tree.predict(X_Df_test)
test1 = time.time() - test0

# Create confusion matrix
pd.crosstab(Y_Df_test, Y_Df_pred, rownames=['Actual attacks'], colnames=['Predicted attacks'])
```

Figure 4. 26 Creating Decision tree Classifiers

Confusion matrices.

Out[50]:

Predicted attacks		0	1
Actual attacks			
	0	6805	2906
	1	1289	11544

Figure 4. 27 Confusion Matrix for decision tree

```

In [51]: from sklearn.model_selection import cross_val_score
         from sklearn import metrics
         accuracy = cross_val_score(clf_Tree, X_Df_test, Y_Df_test, cv=10, scoring='accuracy')
         print("Accuracy: %0.5f (+/- %0.5f)" % (accuracy.mean(), accuracy.std() * 2))
         precision = cross_val_score(clf_Tree, X_Df_test, Y_Df_test, cv=10, scoring='precision')
         print("Precision: %0.5f (+/- %0.5f)" % (precision.mean(), precision.std() * 2))
         recall = cross_val_score(clf_Tree, X_Df_test, Y_Df_test, cv=10, scoring='recall')
         print("Recall: %0.5f (+/- %0.5f)" % (recall.mean(), recall.std() * 2))
         f = cross_val_score(clf_Tree, X_Df_test, Y_Df_test, cv=10, scoring='f1')
         print("F-measure: %0.5f (+/- %0.5f)" % (f.mean(), f.std() * 2))
         print("train_time:%.3fs\n" %train1)
         print("test_time:%.3fs\n" %test1)

```

Figure 4. 28 Performance Evaluation of decision tree

```

Accuracy: 0.98155 (+/- 0.00447)
Precision: 0.98348 (+/- 0.00486)
Recall: 0.98301 (+/- 0.00599)
F-measure: 0.98301 (+/- 0.00423)
train_time:1.124s

test_time:0.006s

```

Figure 4. 29 Performance Evaluation results for decision tree

KNN

```
In [55]: from sklearn.model_selection import cross_val_score
from sklearn import metrics
accuracy = cross_val_score(clf_KNN, X_Df_test, Y_Df_test, cv=10, scoring='accuracy')
print("Accuracy: %0.5f (+/- %0.5f)" % (accuracy.mean(), accuracy.std() * 2))
precision = cross_val_score(clf_KNN, X_Df_test, Y_Df_test, cv=10, scoring='precision')
print("Precision: %0.5f (+/- %0.5f)" % (precision.mean(), precision.std() * 2))
recall = cross_val_score(clf_KNN, X_Df_test, Y_Df_test, cv=10, scoring='recall')
print("Recall: %0.5f (+/- %0.5f)" % (recall.mean(), recall.std() * 2))
f = cross_val_score(clf_KNN, X_Df_test, Y_Df_test, cv=10, scoring='f1')
print("F-measure: %0.5f (+/- %0.5f)" % (f.mean(), f.std() * 2))
print("train_time:%.3fs\n" %train1)
print("test_time:%.3fs\n" %test1)

Accuracy: 0.95755 (+/- 0.00444)
Precision: 0.95027 (+/- 0.00593)
Recall: 0.97654 (+/- 0.00939)
F-measure: 0.96322 (+/- 0.00396)
train_time:51.142s

test_time:30.831s
```

Figure 4. 30 Performance Evaluation results for K-nearest neighbor

Naive Bayes

```
In [59]: from sklearn.model_selection import cross_val_score
from sklearn import metrics
accuracy = cross_val_score(clf_Naive, X_Df_test, Y_Df_test, cv=10, scoring='accuracy')
print("Accuracy: %0.5f (+/- %0.5f)" % (accuracy.mean(), accuracy.std() * 2))
precision = cross_val_score(clf_Naive, X_Df_test, Y_Df_test, cv=10, scoring='precision')
print("Precision: %0.5f (+/- %0.5f)" % (precision.mean(), precision.std() * 2))
recall = cross_val_score(clf_Naive, X_Df_test, Y_Df_test, cv=10, scoring='recall')
print("Recall: %0.5f (+/- %0.5f)" % (recall.mean(), recall.std() * 2))
f = cross_val_score(clf_Naive, X_Df_test, Y_Df_test, cv=10, scoring='f1')
print("F-measure: %0.5f (+/- %0.5f)" % (f.mean(), f.std() * 2))
print("train_time:%.3fs\n" %train1)
print("test_time:%.3fs\n" %test1)

Accuracy: 0.82971 (+/- 0.01664)
Precision: 0.92301 (+/- 0.01625)
Recall: 0.76467 (+/- 0.02410)
F-measure: 0.83636 (+/- 0.01724)
train_time:0.128s

test_time:0.019s
```

Figure 4. 31 Performance Evaluation results for naïve Bayes

SVM

```
In [63]: from sklearn.model_selection import cross_val_score
from sklearn import metrics
accuracy = cross_val_score(clf_SVM_Df, X_Df_test, Y_Df_test, cv=10, scoring='accuracy')
print("Accuracy: %0.5f (+/- %0.5f)" % (accuracy.mean(), accuracy.std() * 2))
precision = cross_val_score(clf_SVM_Df, X_Df_test, Y_Df_test, cv=10, scoring='precision')
print("Precision: %0.5f (+/- %0.5f)" % (precision.mean(), precision.std() * 2))
recall = cross_val_score(clf_SVM_Df, X_Df_test, Y_Df_test, cv=10, scoring='recall')
print("Recall: %0.5f (+/- %0.5f)" % (recall.mean(), recall.std() * 2))
f = cross_val_score(clf_SVM_Df, X_Df_test, Y_Df_test, cv=10, scoring='f1')
print("F-measure: %0.5f (+/- %0.5f)" % (f.mean(), f.std() * 2))
print("train_time:%.3fs\n" %train1)
print("test_time:%.3fs\n" %test1)

Accuracy: 0.90751 (+/- 0.01316)
Precision: 0.86203 (+/- 0.02059)
Recall: 0.99735 (+/- 0.00536)
F-measure: 0.92472 (+/- 0.00970)
train_time:841.018s

test_time:17.595s
```

Figure 4. 32 Performance Evaluation results for support vector machine

Some supervised learning algorithm comparison:

In this section, the experimental results on both scenarios (learning with default parameters and learning using best parameters by applying the hyperparameter technique) for the selected classification algorithms are interpreted. All the classification models are evaluated by using a 2x2 and 5x5 confusion matrix. This confusion matrix shows the number of correct and incorrect predictions made by the classification model compared to the target values in the data by calculating the number of test samples under four categories (TP, TN, FP, and FN).

These categories are already discussed in Chapters 2 and 3. In this section, the actual results calculated based on these categories are presented for each of the models. Using default parameters, the **MLP** has obtained **99.61** % accuracy. While the accuracy of 99.17%, 99.31%, 99.31%, 99.31%, 99.31, and 90.78% are found for the ANN, RF, SVM, J48, NB, MLP and RNN respectively as shown in Tables 4.1 and 4.2. The MLP classifier is the best performer compared to the other two classifiers in both scenarios while the NN is the second-best performer. The NN achieved an overall classification accuracy of 88% and 99.17% with default parameters and using

the hyperparameter technique respectively. RF obtained better classification accuracy than SVM, MLP, NB, and RNN in both scenarios next to NN. To improve the accuracy of the three classifiers further, a hyperparameter tuning method is performed.

Table 4.1 Accuracy Evaluation on U2R attack

Algorithms	Accuracy in a score (%)
MLP	99.61
J48	99.31
NB	99.31
RF	99.31
SVM	99.31
ANN	99.17
RNN	90.78

Table 4.2 Efficiency Evaluation

Algorithm	Accuracy %	Precision %	Recall %	F-measure %	Train time	Test time
DT	98.15	98.35	98.3	98.3	1.124s	0.006s
KNN	95.76	95.02	97.76	96.32	51.142	30.831s
NB	82.97	92.3	76.47	83.64	0.128s	0.019s
SVM	90.75	86.2	99.74	92.47	841.018s	17.595s

The Decision tree claim that they can obtain a good classification result while significantly reducing the training time. Our CNN works much faster than the DT and also gives a better detection ACC, and so it is feasible for real-time detection in SDNs.

4.6. Build Intrusion Detection Net Model -

```
In [12]: model = models.Sequential()

model.add(Conv2D(32, (3, 3), activation='relu', padding="same", input_shape=(256, 256, 3)))
model.add(BatchNormalization(name='norm_1'))
model.add(MaxPooling2D((2, 2)))

model.add(Conv2D(32, (3, 3), padding="same", activation='relu'))
model.add(BatchNormalization(name='norm_2'))
model.add(MaxPooling2D((2, 2)))

model.add(Conv2D(64, (5, 5), padding="same", activation='relu'))
model.add(BatchNormalization(name='norm_3'))
model.add(MaxPooling2D((2, 2)))

model.add(Conv2D(64, (3, 3), padding="same", activation='relu'))
model.add(BatchNormalization(name='norm_4'))
model.add(MaxPooling2D((2, 2)))

model.add(Conv2D(128, (3, 3), padding="same", activation='relu'))
model.add(BatchNormalization(name='norm_5'))
model.add(MaxPooling2D((2, 2)))

model.add(Conv2D(256, (1, 1), padding="same", activation='relu'))
model.add(BatchNormalization(name='norm_6'))
model.add(MaxPooling2D((2, 2)))

model.add(Conv2D(256, (3, 3), padding="same", activation='relu'))
model.add(BatchNormalization(name='norm_7'))
model.add(MaxPooling2D((2, 2)))

model.add(GlobalAveragePooling2D())
model.add(Dense(6, activation='softmax'))
model.summary()
```

Figure 4. 33 Intrusion Detection CNN Model

- ❖ Intrusion Detection Net or CNN (Convolutional neural network or ConvNet) is a class of deep neural networks, commonly applied to analyzing visual descriptions. Here is the Intrusion Detection Net model's example of CNN with few layers using Conv2D -2D convolutional layer, Batch Normalization and MaxPooling2D application of a moving window across 2D input space.
- ❖ Hyperparameters are set before training they represent the variables that determine the Neural network structure and how it is trained.
- ❖ Conv2D with hyperparameters mentioned above: Conv2D (kernel size, (filters, and filters), input shape = (256, 256, 3) with activation function for each layer as a softmax.
- ❖ MaxPooling2D layer to reduce the spatial size of the incoming features; 2D input space:

MaxPooling2D ((2, 2)).

- ❖ Do the same increasing the kernel size: 4-8-16-32-64-128-256.
- ❖ Provide the last fully connected layer which specifies the number of classes of intrusion detection system for software defined network and normal. softmax activation function outputs classified features: Dense (6 activations=' softmax')
- ❖ Print the summary of the model.

Accuracy: As shown in Figure 4.34 the accuracy of the model with both the validation dataset and test dataset is shown below in the Figure. Model accuracy for both is 99.61% which is close to 100%. The total number of correct identifications for attack and benign data 99.61% was right.

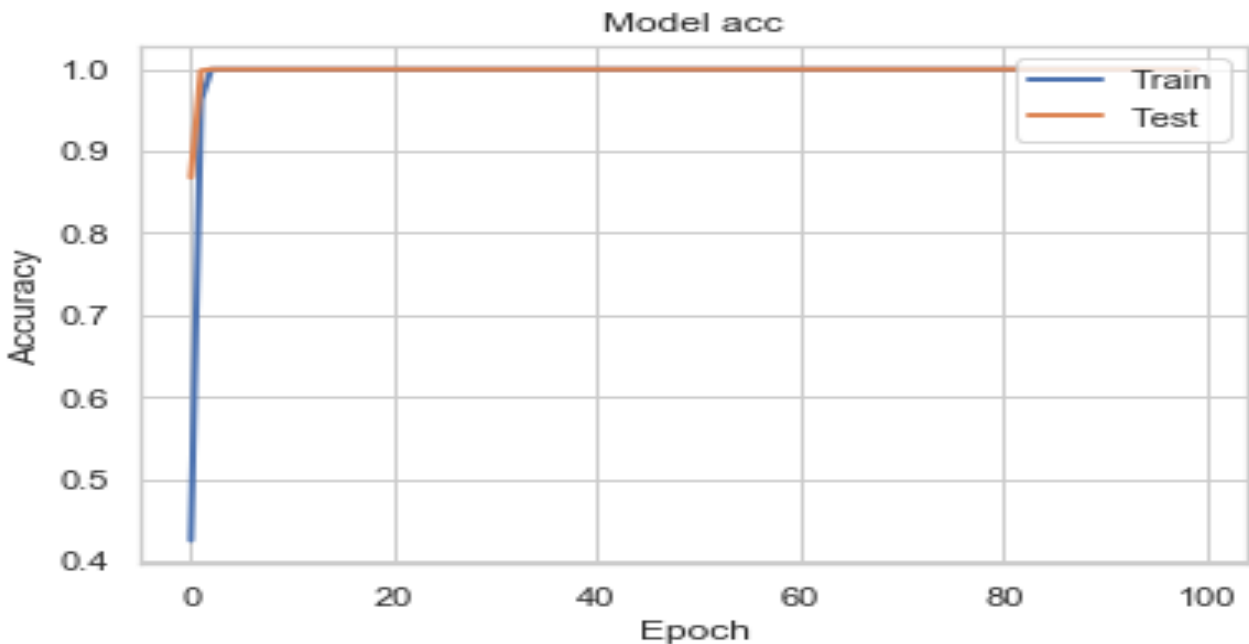


Figure 4. 34 Model Accuracy

Loss: Loos of the developed model is measured on both the validation dataset and test dataset. Loss of the model continuously decreases while the model trains repeatedly as the epoch increases. The loss of the model is shown below in Figure 4.35.

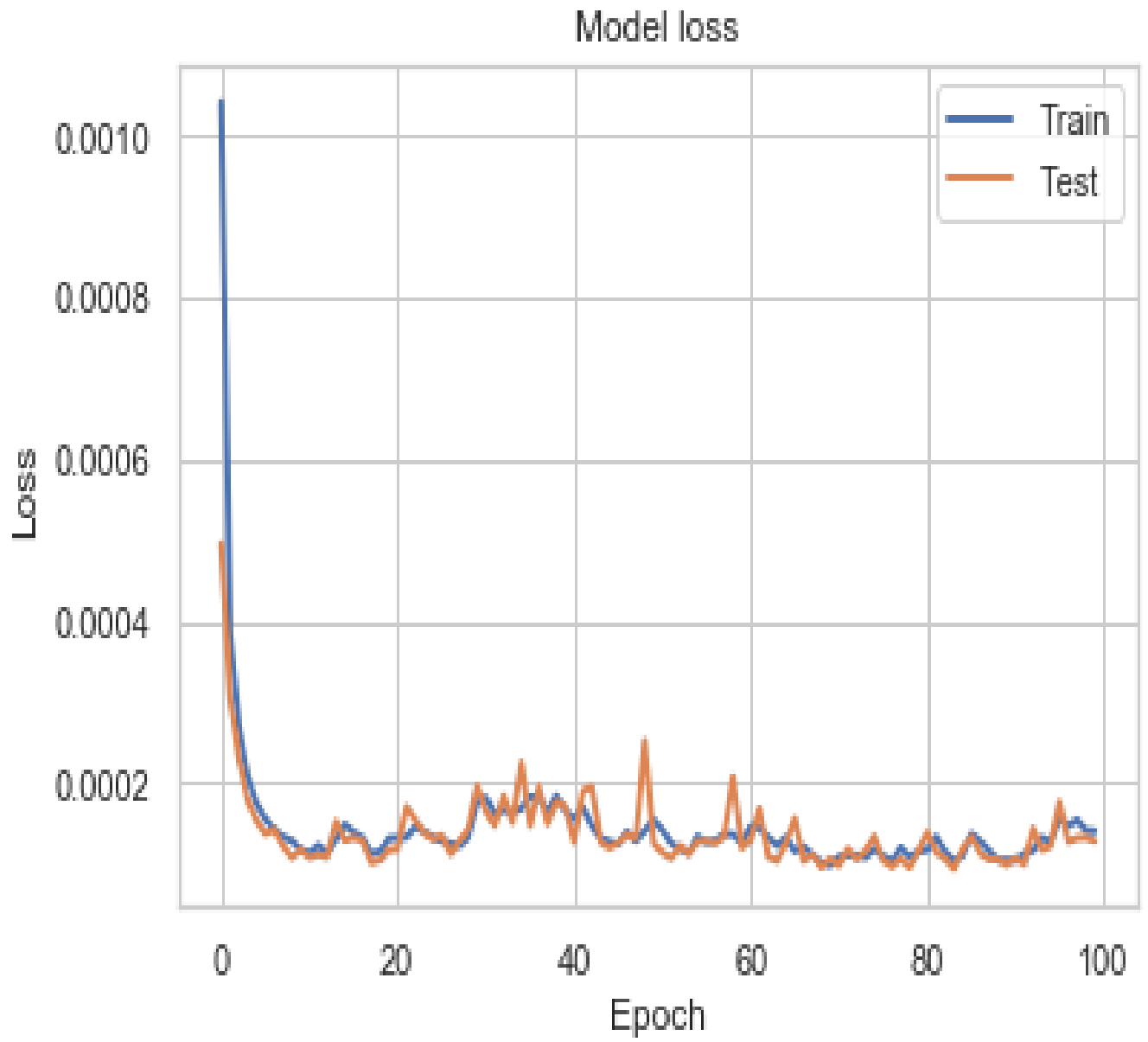


Figure 4. 35 Model Loss

CHAPTER FIVE

RESULTS AND DISCUSSION

5.1. RESULTS AND DISCUSSION

Numerous research methodologies have been proposed to solve cybersecurity threats. In this chapter finding of some researches is compared with the experiment conducted in this study. These research work which was done by other researcher uses the same dataset but apply different algorithm and methodologies. This study tried to provide a new approach toward a developing flow-based software-defined network anomaly intrusion detection model. There are several metrics used for comparison of this study such as Dataset, Algorithm used, Accuracy, Model Loss, Reconstruction Error, AUC-ROC, and training time of the model. Table 5.1 summarizes different research work findings and approaches being used with this study work.

To access our models, K-fold cross-validation is used, and based on the validation result a confusion matrix and classification report is generated. And also, the effect of normalizing the data is accessed on the top two performing models. We used a sklearn library for the model evaluation reports. Finally, hyperparameter tuning is done on Convolutional Neural Network, as it exhibits a better performance in detecting flow-based intrusion in software defined network using Machine learning/Deep learning ML/DL.

5.1.1 DeepIDS: Deep Learning Approach for Intrusion Detection in Software Defined Networking

An SDN-based NIDS is introduced with three core modules: The Collector, The Anomaly Detector, and The Counter Measure Deployment. Developing the Anomaly Detector module in which DL algorithms are implemented. We evaluate the potential of ML and DL in detecting network anomalies with different learning rates and feature sets. Experimental results show some machine learning algorithms such as DT, J48, RF, MLP, SVM, and ANN and DNN, RNN, GRU, CNN, and RNN can detect network anomalies effectively with just some basic network features. Network overheads are an important factor when designing the NIDS, so they are also evaluated in detail.

The SDN architecture decouples the network control and forwarding functions enabling the network control to become directly programmable and the underlying infrastructure to be abstracted for applications and network services [25, 26, 27]. The SDN controller is centralized and separated from its underlying switches, and so it simplifies network management and facilitates network evolution. Instead of deploying a hardware-based NIDS as in the traditional network, we can implement a software-based NIDS on the SDN controller and then take the advantages of SDNs. We can now collect network information, detect any sign of intrusions and then deploy a flow rule to mitigate attacks in a real-time manner.

Considering the success of DL in many fields [19], [20] [22] [82], a combination of SDN and DL can improve the NIDS performance and then secure the network better. The structural design and implementation of DeepIDS (a developing flow-based anomaly detection model using DL) in the SDN architecture. The attack detection mechanism is the main research contribution.

Our main contributions are summarized in the following points:

- ❖ We propose and implement an SDN-based NIDS with three core modules of this dataset. Collector, the Anomaly Detector, and Anomaly mitigator.
- ❖ We have implemented some DL and ML models in DeepIDS. We trained and evaluated the model with different sets of features of the NSL-KDD dataset [79], [86]. Through experiments, in a literature review, the result of detection ACC of 80.7% using just some network features from 41 features and by using all feature are able to improve detection Accuracy to 99.61%.
- ❖ This method is scalable, and the structure of the DL can be changed according to the characteristic of the data features, which makes our method applicable to detect other kinds of attack.
- ❖ We have also evaluated the network performance of DeepIDS in the SDN environment. We implemented our DeepIDS in a POX controller and stress-tested DeepIDS through extensive simulation. The test results demonstrate that our approach does not degrade the POX controller's performance.

5.1.2 Performance of a Classification

5.1.2.1 Confusion matrix of Deep Learning Algorithms

Different variants of the recurrent models ranging from the Vanilla RNN to Bidirectional Recurrent networks are evaluated.

SimpleRNN:

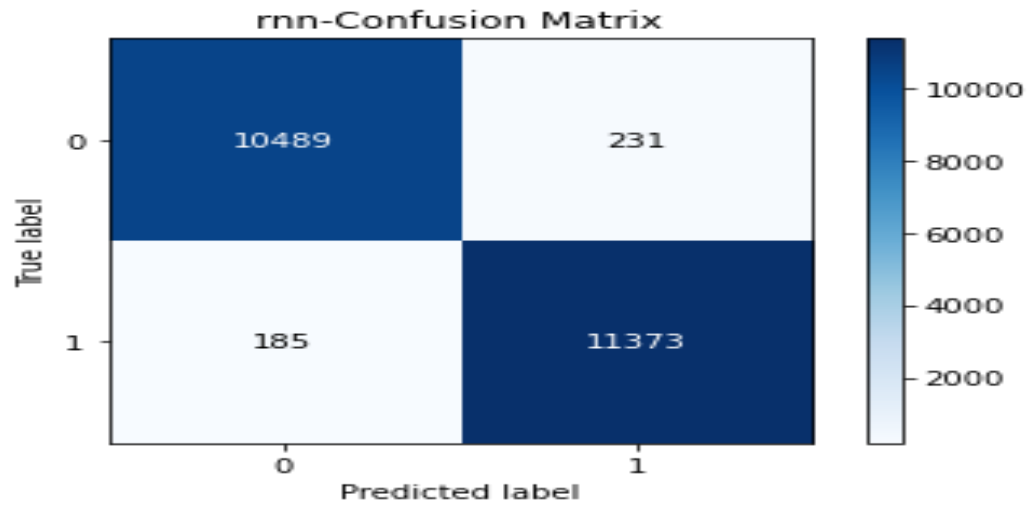


Figure 5. 1 Confusion matrix for SimpleRNN

LSTM-based Models:

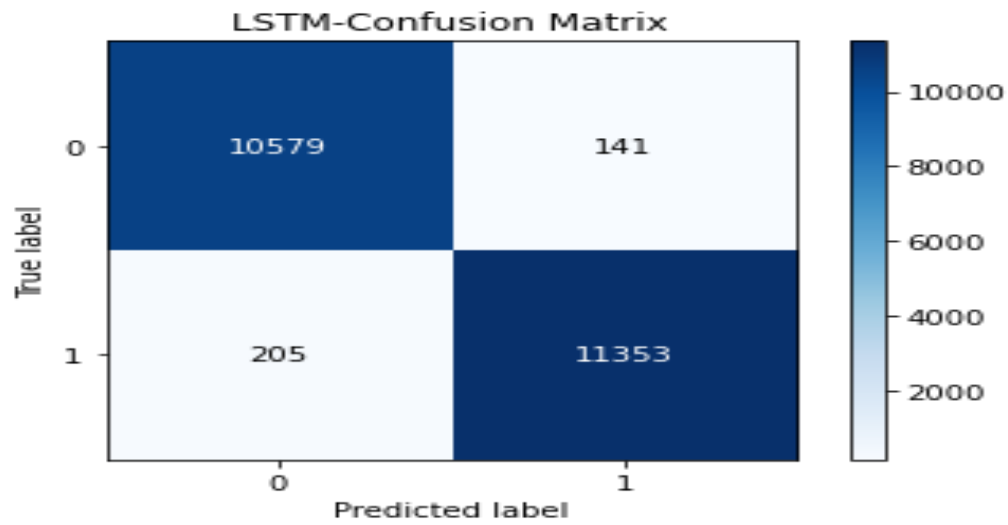


Figure 5. 2 Confusion matrix for LSTM

GRU-based Model:

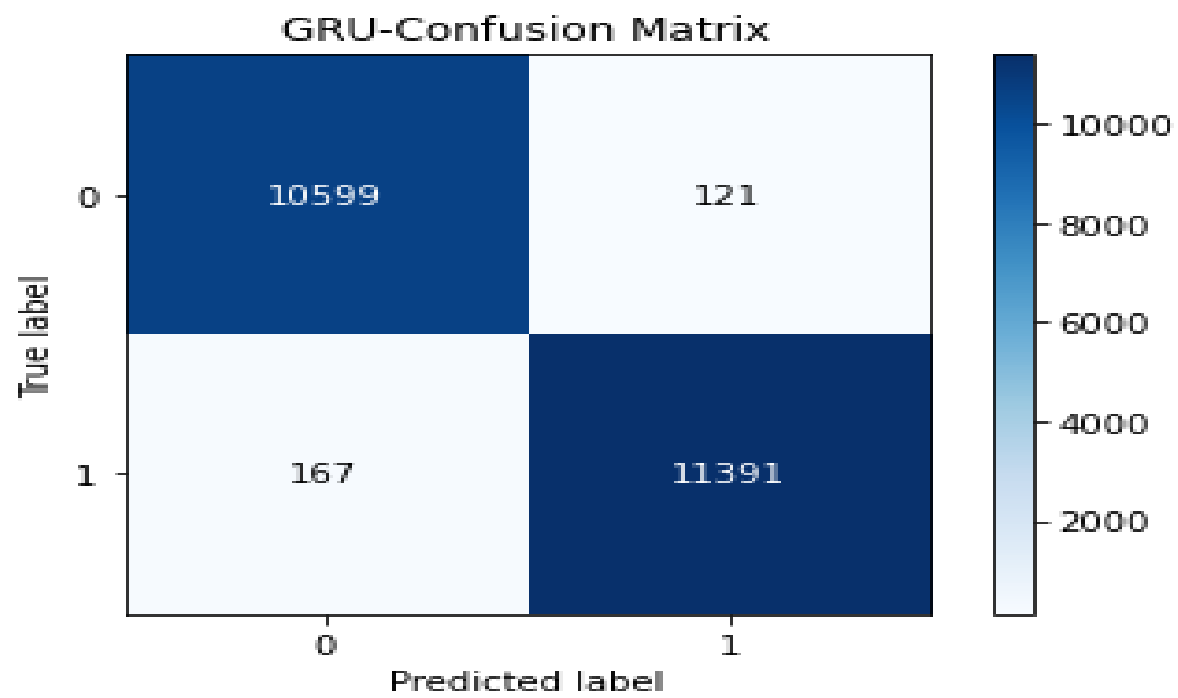


Figure 5. 3 Confusion matrix for GRU

DNN3-based Model:

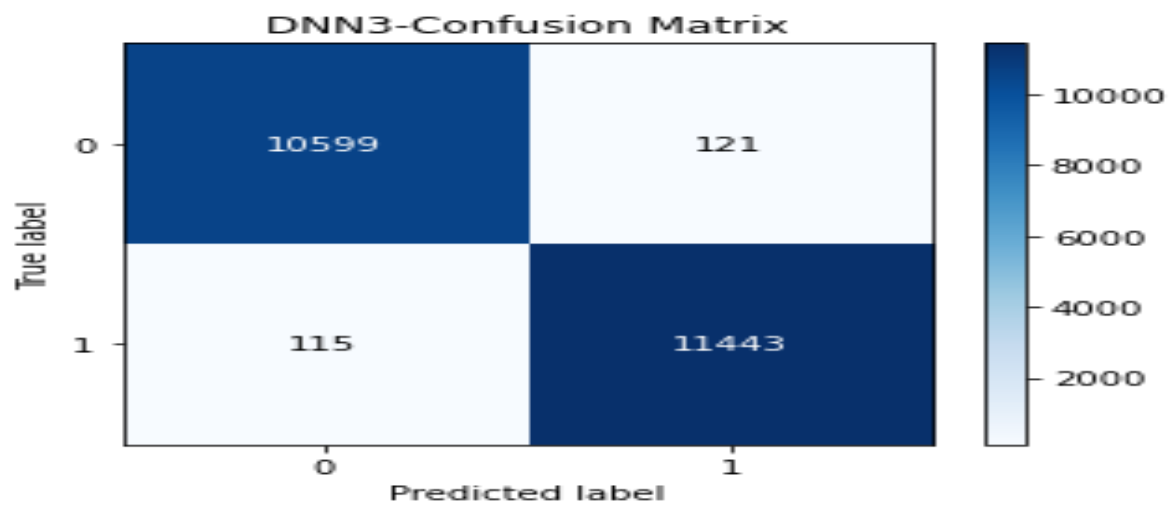


Figure 5. 4 Confusion matrix of DNN3

DNN4-based Model:

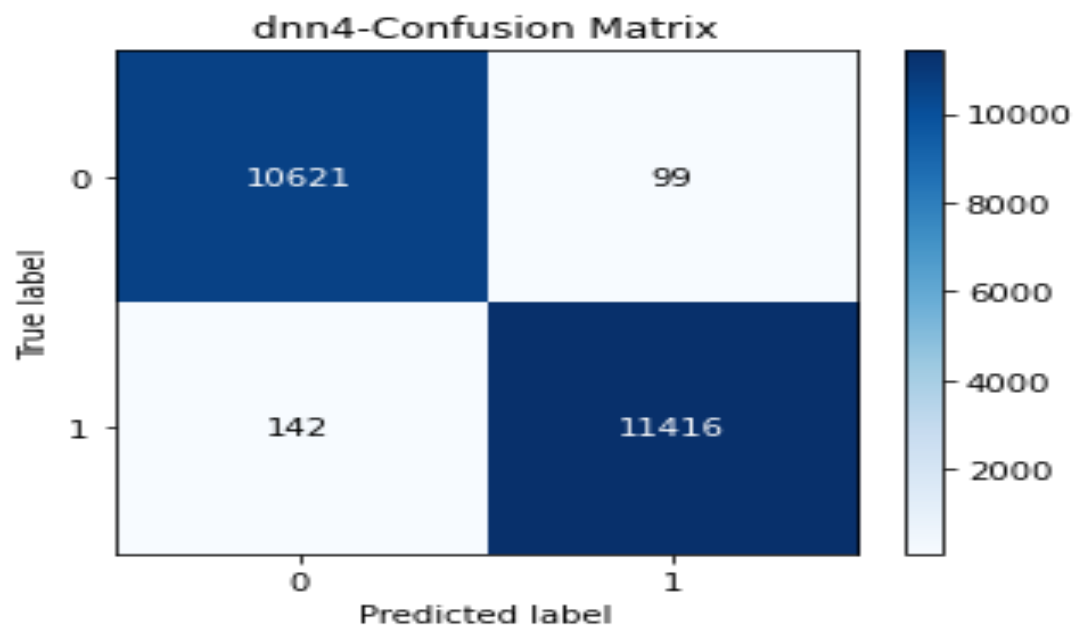


Figure 5. 5 Confusion matrix of DNN4

CNN-based Model:

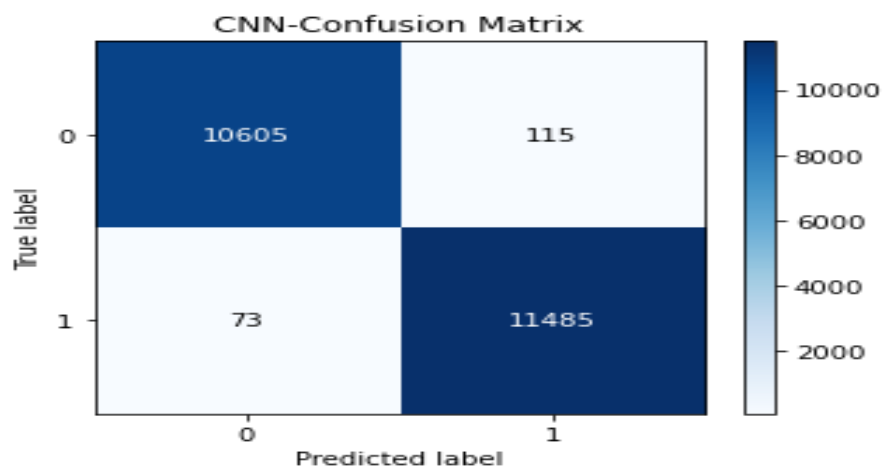


Figure 5. 6 Confusion matrix of CNN

Table 5. 1 Summary for Classification Performance of the Model

Models	Accuracy (%)	Loss (%)	Precision (%)	Recall (%)	F1-Score (%)
SimpleRNN	98.13	0.9813	98	98.4	92.2
LSTM	98.45	0.9813	98.77	98.2	98.5
GRU	98.7	0.485913	98.9	98.5	98.75
DNN3	98.9	0.396515	98.95	99	98.95
DNN4	98.9	0.851223	99	98.7	98.95
CNN	99.61	0.414159	99.61	99.4	99.2

The Receiver Operating Characteristic (ROC) curve is also presented to evaluate our proposed approach. The ROC curve is created by plotting the false positive rate versus the true positive rate. Area Under Curve-Receiver Operating Characteristic Curve: area under curve indicates the performance of the system which fails to distinguish between attack and normal network flows. The area under the curve is 99.61%. The receiver operating characteristic curve is 99.0% which shows the performance of the model at all classification thresholds. The area under the ROC curve (AUC) is used to determine which classifier predicts the classes best. The higher the AUC, then the better is the classifier. Figure 5.7 shows that the proposed CNN achieves the highest AUC amongst all the tested algorithms with a True Positive Rate of 99.61% and False positive rate of 0.39.

As we can see, the CNN has a lowest False Positive Rate which is an important factor of the IDS. The Precision vs Recall curve shows the tradeoff between Precision and Recall for different thresholds. A high area under the curve represents both high recall and high precision. An ideal system with a high area under the curve will return many result, with all results labelled correctly. Our CNN is compared with other state-of-the-art algorithm like RNN, GRU, DNN and LSTM algorithms. Figure 5.7 shows the AUC ROC curve.

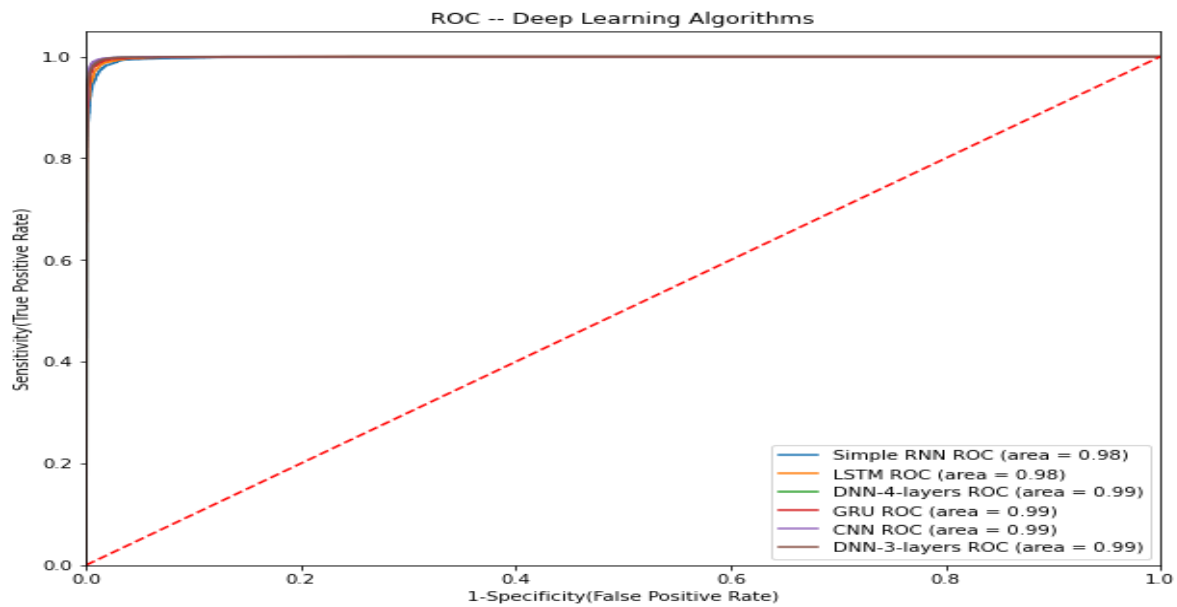


Figure 5. 7 AUC-ROC of Model

Multiple ROC in Single Plot:

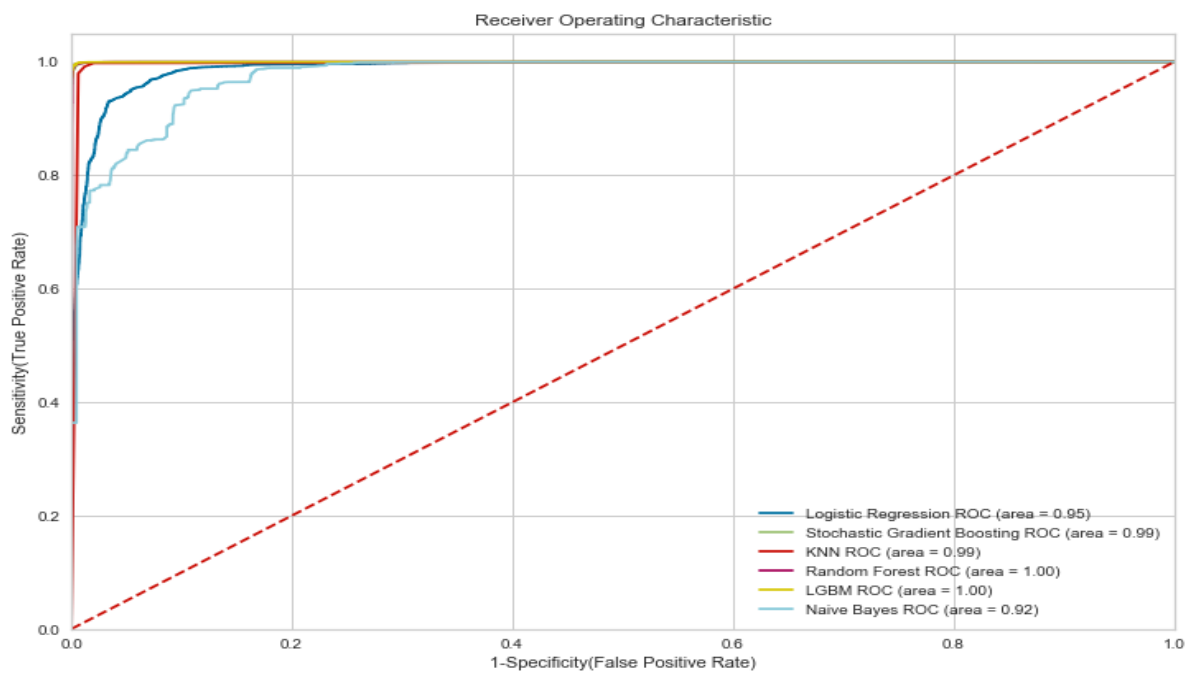


Figure 5. 8 Multiple ROC in Single Plot.

5.2. Effect of the Learning Rate

The DL model type is a very flexible architecture that consists of several layers with a different number of neurons. Different network architectures and hyperparameter settings will yield different results. Tuning the DL model and hyper-parameters is a challenging task. The learning rate is one of the most important hyper-parameters that we have to tune in each training process. In this section, we firstly analyze the effect of different learning rates on detection performance. Thus, we tried to optimize the model by varying the value of the learning rate in a range of $\{0.1, 0.01, 0.001, \text{ and } 0.0001\}$. When training using DL model types, we tried to minimize the loss and maximize the ACC.

Table 5. 2 Loss and Accuracy Evaluation for Different Learning Rates

Learning Rate	Train Set		Train Set	
	Loss	ACC (%)	Loss	ACC (%)
0.1	11.49	88.04	31.26	72.05
0.01	8.41	90.9	20.15	73.03
0.001	8.26	91.62	15.51	80.7
0.0001	7.45	91.7	20.3	74.67

By comparing the loss and ACC of the training phase (see Table 5.2), we can see that along with the decrease of the learning rate, the loss will decrease, and the ACC will increase. However, in the testing phase, when we lower the learning rate to 0.0001, the results are not as good as the learning rate of 0.001. This is because if the learning rate is too small, the NIDS model will be trained too accurately and the overfitting problem starts happening.

In the next experiment, we evaluate the sensitivity of each feature set with some existing algorithms: Naive Bayes (NB), Decision Tree (DT), and SVM. The NB and DT are quite simple and cost-effective. The SVM was proposed by [18] for DDoS detection in SDNs. They used their own dataset, and so we have to regenerate the results on the considered NSL-KDD dataset. The ROCs of each test case, each feature set have a different effect on the performance of each algorithm.

For the Basic Feature set, all the algorithms perform quite well and are comparable to each other. The MLP yields a slightly better result than the others. The SVM is a popular intrusion detection approach and has shown good results in detecting intrusion.

For the Traffic Feature set, following the results of previous evaluations, the AUC results are quite low for all the algorithms except the NB. This phenomenon is quite common in the field of ML/DL. One algorithm can work well with one dataset but it can be bad for another dataset. Therefore, we had to experiment with different algorithms to find the most suitable one for our problem.

In the case of the Mixed Feature set, the DNN and SVM have the same AUC result and are the best amongst all the algorithms. In this case, we can see that the overall FPR is quite low compared to those of other feature sets. This confirms our conclusion about the contribution of basic and traffic features in improving detection ACC. All in all, NB achieves a quite high AUC in the case of the Traffic Features Set. Despite the high AUC of the NB, its FPR is quite high and not feasible for the NIDS. In general, the CNN works quite well with all of the feature sets, with quite low FPR and high AUC.

SVM also achieves a quite high AUC, but it also has some drawbacks like high FPR and low computational efficiency. We will discuss the effectiveness of these algorithms in due course. The Mixed Feature Set gives us the best combination of AUC and the FPR. The Mixed Feature Set will be chosen for further evaluation. We also evaluated the P, R, and F1 of the model for more understanding.

This is because several attacks in the testing set are new and complex. These attacks cannot be fully generalized by our existing DNN model using just six basic network features. The R result of the testing phase shows that our DNN model can detect almost all the attacks mentioned in the KDDTest++ set, and our results are significantly promising with a high P of 85%.

Table 5. 3 Performance Metric Evaluation of the Mixed Feature Set

Dataset	P (%)	R (%)	F1 (%)
Training Set	96	96	96
Testing Set	85	81	81

In summary, we have demonstrated that our proposed ML model type approach can generalize and abstract the characteristics of the normal and anomaly traffic with a small number of features and gives us a promising ACC. The result of the DL algorithm facilitates the development of flow-based NIDS under the SDN context. All the basic network features that can be collected easily in the SDN architecture can be used directly by the DL algorithm without any feature engineering process.

5.3. Efficiency Evaluation

We also evaluate the efficiency of our model compared to other algorithms by comparing the training time and the testing time of these algorithms. Table 4.9 provides relevant parameters for these algorithms. The NB and DT algorithms have really low training and testing times in ms compared to the SVM and our CNN. However, they also yield low detection ACC. The SVM is one of the most popular algorithms in the field of intrusion detection, but it is not fast enough for real-time detection, especially under SDN architectures. The total processing time takes around 2.5 hours. Our CNN works much faster than the SVM and also gives a better detection ACC, and so it is feasible for real-time detection in SDNs.

Table 5. 1 Running Time Evaluation

Algorithm	Training Time	Testing Time
NB	19.8 ms	4 ms
SVM	2 h	30 min
DT	256.9 ms	2.74 ms
CNN	500 s	811.6 ms

5.4. Network Performance Evaluation

In this section, we provide a detailed network performance analysis of our DeepIDS in a POX [35] [39] controller. We also compare the network performance of the DeepIDS with the SVM algorithm. We also evaluate resource utilization and briefly discuss our observations in this section.

The DeepIDS is implemented in the POX controller as an application written in Python language. Currently, POX supports OpenFlow 1.0 protocol and includes special support for the Open

vSwitch/Nicira extensions. To test the performance of DeepIDS in the POX controller, we used a Virtual Machine having Intel Core i5-4460 3.2 GHz processor with three cores available and 8 GB of RAM. The evaluation setup for this experiment has been described in chapter 4.

5.4.1 Throughput Evaluation

For throughput mode tests, we evaluate how many packets a controller can process in a second. This metric indicates the performance of the controller under heavy traffic conditions. Figure 5.9 below depicts the average response rate of the network.

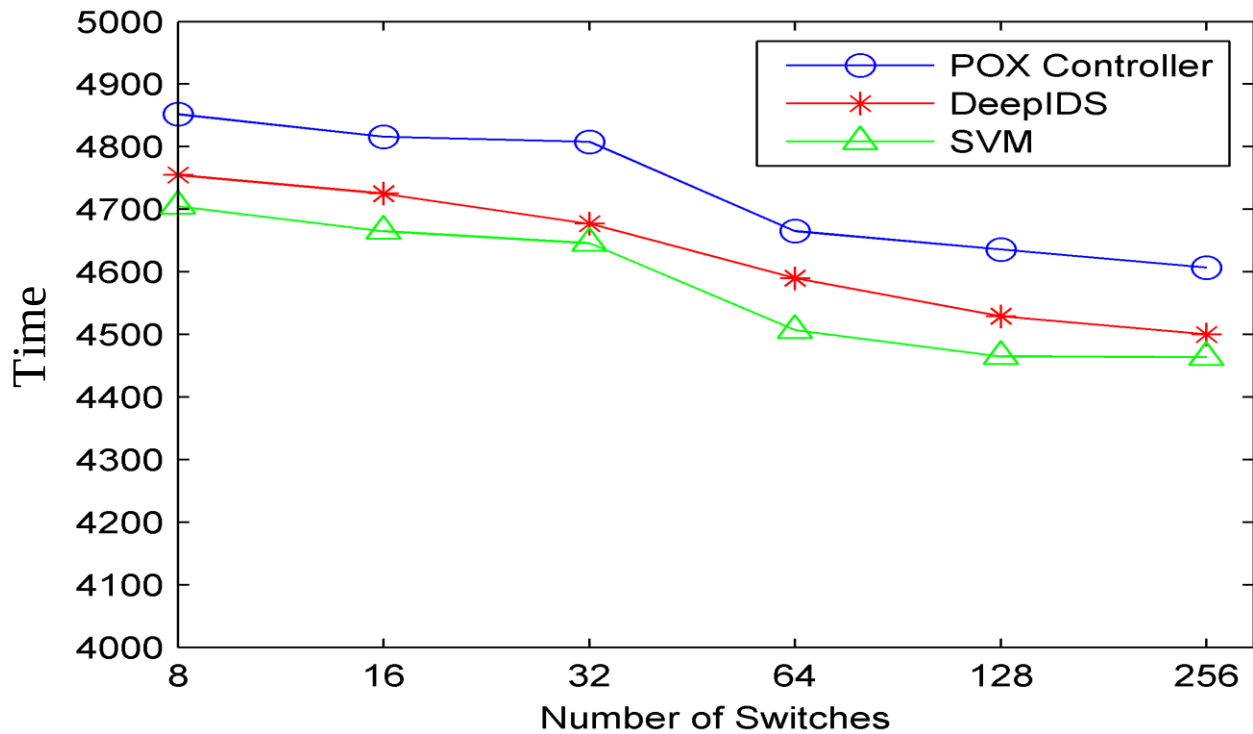


Figure 5. 9 Throughput Evaluation (log scale on the x-axis)

As we can see, the performance of the POX controller itself is quite poor with a maximum throughput of around 4900 responses/s. This is one of the limitations of the POX controller compared to other SDN controllers. However, the running stand-alone POX controller still performs best and is taken as a baseline for performance evaluation in our experiment. Of overall assessment, the bigger the size of the network then the higher the performance overhead. The throughput slightly decreases with the increasing network size according to the results in Figure 5.9. The network performance drops when the number of forwarding devices increases up to 64, showing that the controller throughput degrades by roughly 3% from 32 to 64 switches. The

baseline results for the smallest network scale and the largest network scale are 4852 responses/s and 4607 responses/s respectively.

5.4.2 Latency Evaluation

For latency mode tests, the controller is evaluated for the length of time it takes to process a single packet. This metric reflects the performance of the controller in light traffic conditions. The testing scenarios are the same as the throughput test. As depicted in Figure 5.10, the latency increases with the increasing network size. This is expected behavior: increasing the number of devices will increase the load at the controllers, causing a latency increase. In general, we can see in Figure 5.10 that the SVM always has higher latency than the DeepIDS.

The NIDS takes time to process the PACKET IN message, the OFP FLOW STATS REPLY message, and detect anomaly flows, and so the performance overhead is unavoidable. The SVM algorithm takes a longer time to process the information, and thus it results in a higher latency under high traffic rates. The DeepIDS and SVM overhead are about 2.6% and 5.8% respectively in the case of 256 switches. The DeepIDS's latency degradation is quite low and can be improved in the future.

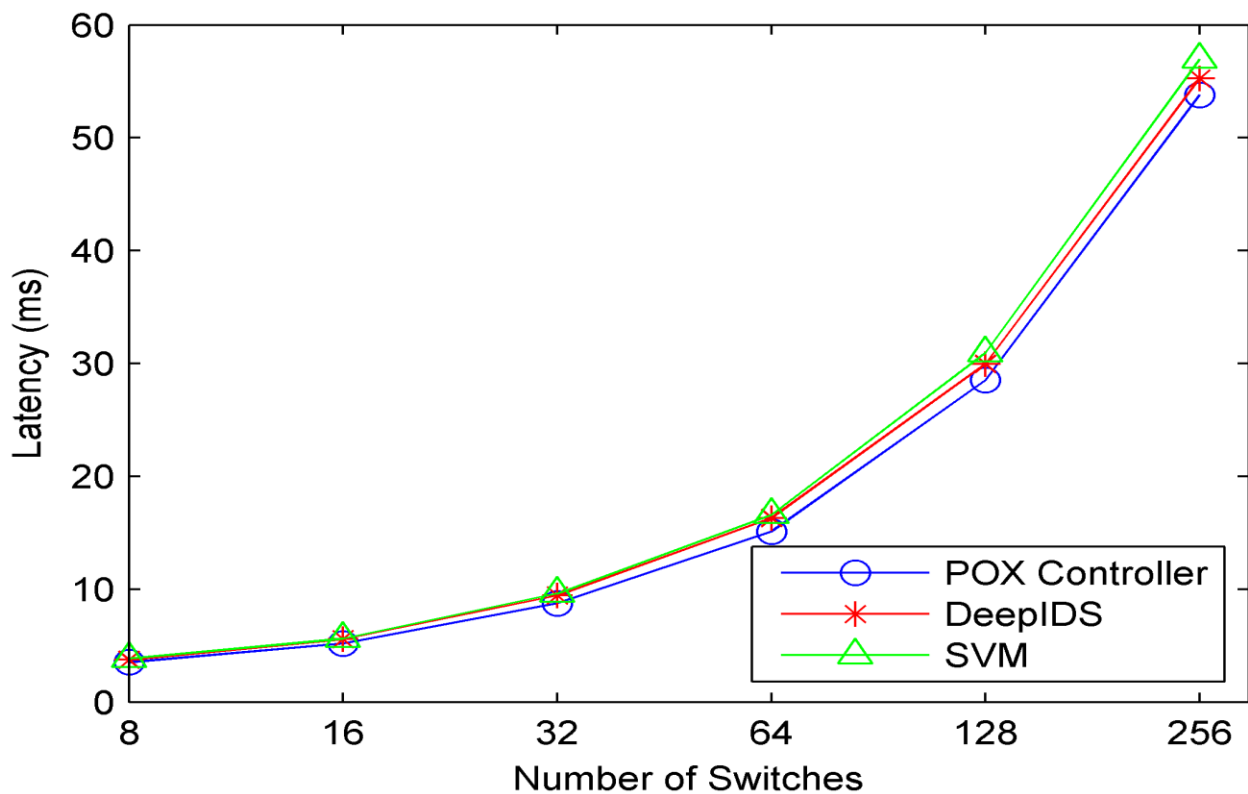


Figure 5. 10 Latency Evaluation (log scale on x-axis)

5.5. Experiment Testbed Setup on Mininet Emulators

The experiment testbed for SDN networks is implemented in an Intel PC using Mininet. Because of the lack of resources and the ease of experiment, a network with several switches and hosts in a star topology is emulated for simulation purposes. The testbed has three main components as seen below.

- ❖ **SDN Controller:** the POX [35] controller is used for SDN networks controlling purpose.
- ❖ **NIDS Module:** The DeepIDS module is written in Python and runs on top of the POX controller. This module is in charge of monitoring network traffic, detecting and mitigating anomalies.
- ❖ **Switches:** OpenFlow switches are emulated by Mininet and connected to the POX controller via OpenFlow protocol.
- ❖ **Hosts:** Several hosts are emulated by Mininet and connected to the switch in a star topology. These hosts act as attackers and victims in the simulation.

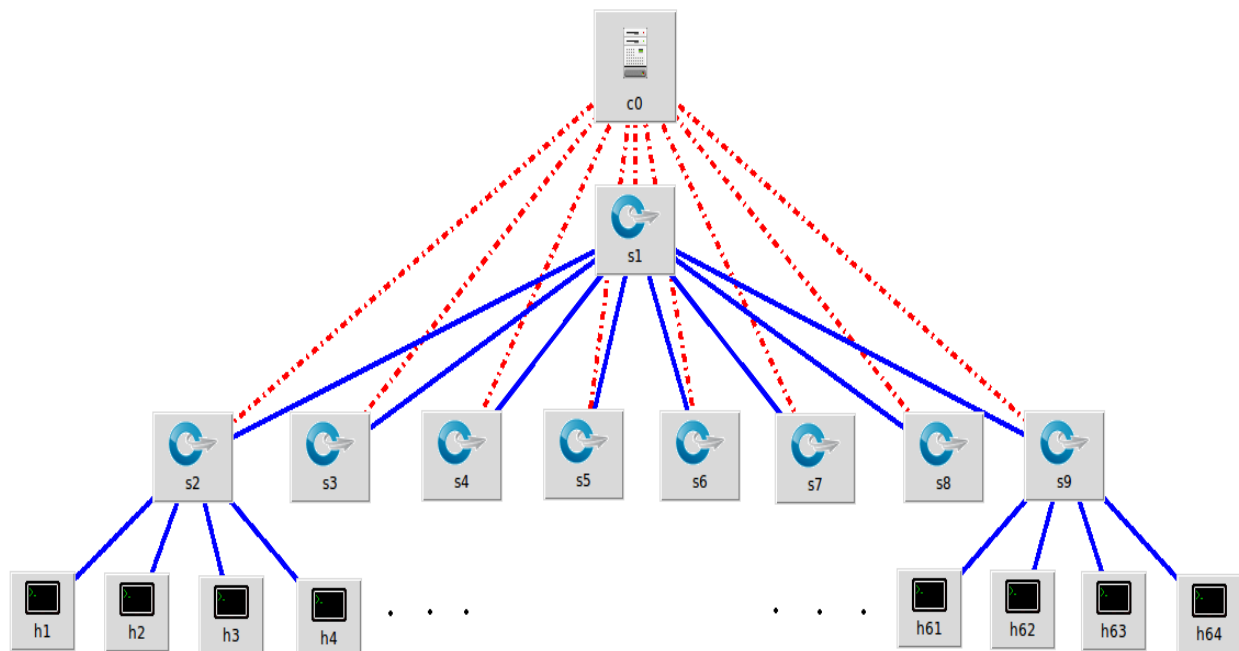


Figure 5. 11 The Emulated Network Topology

CHAPTER SIX

CONCLUSION AND RECOMMENDATION

6.1 Conclusion

In this study, a Convolutional Neural Network -based developing flow-based anomaly intrusion detection model in the software defined network is proposed. The study aims to detect an anomaly in software defined network flows by using the Machine learning and deep learning model. Deep learning models' type's algorithm to develop the model. The proposed system used the NSL-KDD dataset to learn complex features and their relationships reduced the high dimension dataset. NSL-KDD dataset contains information about the behavior of connection rather than the payload of the packet. NSL-KDD contains over Forty-one features, up-to-date divers attack network flow, and normal network flow traffic.

NSL-KDD network flows dataset feed to Deep learning model types. Deep learning model types learn the normal network behavior and build model, uses this model to identify attack network traffic flow. To know how well the proposed model, performs different performance measure metrics are used and finally, the performance of the proposed model is compared to other research work that uses the same dataset.

The dataset is preprocessed, important features are selected and selected features are used for the model creation. During the classification, four different ML models (MLP, DT, RF, LR, and NN) and DL models (Simpler, LSTM, GRU, DNN3, 4, and CNN) are created, trained, tested, and compared based on their evaluation result. To accomplish this study, the NSL-KDD dataset is used to evaluate the performance of the proposed models which is taken from the University of New Brunswick.

The models are implemented in Python programming language using this dataset. The proposed developing flow-based anomaly detection model has high detection accuracy of for ML an DL algorithm such as (MLP,CNN) 99.61% and 99.61% , very small model loss and reconstruction error which is almost zero including both normal network flow and attack network flow, the performance of the model in distinguishing between normal and attack network traffic was

measured using Area under Curve (AUC) 99% and Receiver Operating Characteristic Curve (ROC) 99%, which applies false positive rate was decreased. Our scheme uses a minimum number of features compared to other state-of-the-art approaches so computational costs can be reduced significantly. In addition, the network performance evaluation showed that our proposed approach does not significantly affect the controller performance. This makes our model a strong candidate for real-time detection.

Latent space representation was encouraging even though there is no clear cluster most of the attack and normal network flow appears to be neatly grouped. Training time spent by the model was very small, ranges from 178 microseconds per sample up to 231 microseconds per sample which helps the proposed model to build a model quickly and identify the attack before damage happens

The model performance was compared against other research work and better detection performance capability, high accuracy, ROC, model loss, and reconstruction error, with good latent space representation, low training time, and minimum false positive rates with medium AUC. Based on the performance of a developing flow-based anomaly intrusion detection model it can be applied for online software defined network security purposes to identify anomalies or attacks.

Finally, the most important factors that have impacts on the efficiency and accuracy of the model are several training epochs and learning rate the number of epochs is the number of iterations the the model learns the whole data set. It is not fixed that is how many epochs a model should be trained, but it obvious that the model learns more as it is trained more and more having a limit to stop somewhere and depends on the model architecture, type, and size of the data set.

6.2. Summary

In this chapter explain about developing flow-based intrusion detection model for software defined network using Machine learning/Deep learning. The methodology for the training and the testing of intrusion detection in software defined network is explained. The detection rates and results were also given and explained. This study used a real-life and public features dataset. The testing results show us this study can be used in a deep learning approach that detects intrusion detection in the software defined network. This thesis illustrates that the characteristics of the selected

parameters for the neural network are feasible and practical in the detection of intrusion in the SDN.

6.3. Recommendation

It is recommended that the proposed model can be further enhanced if the following upgrading can be achieved. First, if a hybrid multi-level deep learning model is applied to software defined network controllers in order to decrease false detection rate and increase accuracy. Second, more focus paid into decreasing false alarm rate and increase detection rate by how much finally identifying and optimizing different issues that affect its performance, and use a variety of performance measures metrics.

6.4. Future Work

This research work explores different points that can be further improved for better functioning of the system.

- ❖ Applying the system on two SDN controllers.
- ❖ Implementing the system by using more than one dataset.
- ❖ Exploring different feature selection methods by combining our approach with Other techniques.

Reference

- [1] D. Kreutz, F. Ramos, and P. Verissimo, "Towards secure and dependable software-defined networks," in *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*. ACM, 2013, pp.
- [2] D. Kreutz, F. M. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp.
- [3] S. Scott-Hayward, S. Natarajan, and S. Sezer, "A survey of security in software defined networks," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 1, pp. 623–654, 2016
- [4] S. A. Mehdi, J. Khalid, and S. A. Khayam, "Revisiting traffic anomaly detection using software defined networking," in *International Workshop on Recent Advances in Intrusion Detection*. Springer,
- [5] S. M. Mousavi and M. St-Hilaire, "Early detection of ddos attacks against sdn controllers," in *Computing, Networking and Communications (ICNC), 2015 International Conference on*. IEEE, 2015
- [6] S. Mukherjee and N. Sharma, "Intrusion detection using naive bayes classifier with feature reduction," *Procedia Technology*, vol. 4, pp. 119–128, 2012.
- [7] R. Kokila, S. T. Selvi, and K. Govindarajan, "Ddos detection and analysis in sdn-based environment using support vector machine classifier," in *Advanced Computing (ICoAC), 2014 Sixth International Conference on*. IEEE, 2014,
- [8] T. V. Phan, T. Van Toan, D. Van Tuyen, T. T. Huong, and N. H. Thanh, "Openflowsia: An optimized protection scheme for software-defined networks from flooding attacks," in *Communications and Electronics (ICCE), 2016 IEEE Sixth International Conference on*. IEEE, 2016, pp.
- [9] R. Sommer and V. Paxson, "Outside the closed world: On using machine learning for network intrusion detection," in *2010 IEEE symposium on security and privacy*. IEEE, 2010,
- [10] S. A. Mehdi, J. Khalid, and S. A. Khayam, "Revisiting traffic anomaly detection using software defined networking," in *International Workshop on Recent Advances in Intrusion Detection*. Springer, 2011, pp. 161–180.
- [11] Z. Jadidi, V. Muthukkumarasamy, E. Sithirasenan, and M. Sheikhan, "Flow-based anomaly detection using neural network optimized with gsa algorithm," in *2013 IEEE 33rd International Conference on Distributed Computing Systems Workshops*, 2013, pp. 76–81.
- [12] P. Winter, E. Hermann, and M. Zeilinger, "Inductive intrusion detection in flow-based network data using one-class support vector machines," in *New Technologies, Mobility and Security (NTMS), 2011 4th IFIP International Conference on*. IEEE, 2011, pp. 1–5.
- [13] S. A. Mehdi, J. Khalid, and S. A. Khayam, "Revisiting traffic anomaly detection using software defined networking," in *International Workshop on Recent Advances in Intrusion Detection*. Springer,

2011, pp. 161–180.

[14] “Q1 2016 State of the Internet / Security Report,” Available: <https://content.akamai.com/PG6301-SOTI-Security.html>, [Accessed 07 Jul. 2016].

[15] R. Braga, E. Mota, and A. Passito, “Lightweight ddos flooding attack detection using nox/openflow,” in *Local Computer Networks (LCN), 2010 IEEE 35th Conference on*. IEEE, 2010, pp. 408–415.

[16] K. Giotis, C. Argyropoulos, G. Androulidakis, D. Kalogeras, and V. Maglaris, “Combining openflow and sflow for an effective and scalable anomaly detection and mitigation mechanism on sdn environments,” *Computer Networks*, vol. 62, pp. 122–136, 2014.

[17] P. Van Trung, T. T. Huong, D. Van Tuyen, D. M. Duc, N. H. Thanh, and A. Marshall, “A multi-criteria-based ddos-attack prevention solution using software defined networking,” in *Advanced Technologies for Communications (ATC), 2015 International Conference on*. IEEE, 2015, pp. 308–313

[18] J. Li, Z. Zhao, and R. Li, “Machine learning-based IDS for softwaredefined 5G network,” *IET Networks*, Vol 7, pp. 53–60, 2018.

[19] J. Li, Z. Zhao, R. Li, and H. Zang, “AI-based two-stage intrusion detection for software defined IoT networks,” *IEEE Internet of Things*, Vol. 6, pp. 2093–2102, 2018.

[20] A. AlErroud, I. Alsmadi, “Identifying cyber-attacks on software defined networks: an inference-based intrusion detection approach,” *Journal of Network and Computer Application*, pp. 152–164, 2017.

[21] S. Hong, L. Xu, H. Wang, and G. Gu, “Poisoning network visibility in software-defined networks: New Attacks and Countermeasures,” *NDSS Symposium*, 2015.

[22] S. Shalini, N. Shafreen, P. L. Sathiya, V. Vetriselvi, “Intrusion detection system for software-defined networks using fuzzy system,” *International Conference on Computing and Communication System*. Springer, Singapore, pp.603-620, 2018.

[23] J. Ibrahim, S. Gajin, “SDN based intrusion detection system literature review,” *Symposium Infoteh-Jahorina*, Vol. 16, 2017.

[24] B. W. Masduki, K. Ramli, F. A. Saputra, D. Sugiarto, “Study on implementation of machine learning methods combination for improving attacks detection accuracy on intrusion detection system (IDS),” In *2015 International Conference on Quality in Research (QiR)*,Indonesia, 2016.

[25] Q. Niyaz, W. Sun, A. Y. Javaid, “A deep learning based DDoS detection system in software defined networking (SDN),” 2016, arXiv preprint: <https://arxiv.org/abs/1611.07400>

[26] T. Tang, L. Mhamdi, D. McLernon, S. Zaidi, M. Ghogho, “Deep learning approach for network intrusion detection in software defined networking,” *International Conference on Wireless Networks and Mobile Communications (WINCOM)*, Morocco, 2016.

[27] T. Tang, L. Mhamdi, D. McLernon, S. Zaidi, M. Ghogho, “Deep recurrent neural network for intrusion detection in SDN based networks,” *4th IEEE Conference on Network Softwarization and*

Workshop (netsoft), Canada, 2018.

[28] E. Hodo, X. Bellekens, A. Hamilton, C. Tachtatzis, R. Atkinson, "Shallow and deep networks intrusion detection system: A Taxonomy and Survey," 2017, arXiv preprint arXiv:1701.02145.

[29]. Atkinson RC, Bellekens XJ, Hodo E, Hamilton A, Tachtatzis C (2017) Shallow and deep networks intrusion detection system: a taxonomy and survey. CoRR, arXiv preprint arXiv: 1701.02145. 2017 Jan 9

[30]. Wang L, Jones R (2017) Big data analytics for network intrusion detection: a survey. Int J Netw Commun. <https://doi.org/10.5923/j.ijnc.20170701.03>

[31] Tavallaee, Mahbod and Bagheri, Ebrahim and Lu, Wei and Ghorbani, Ali A, "A Detailed Analysis of the KDD CUP 99 Dataset," in *Symposium on Computational Intelligence for Security and Defense Applications*, IEEE, 2009, PP. 1--6.

[32] Potluri, Sasanka and Diedrich, Christian, "Accelerated Deep Neural Networks for Enhanced Intrusion Detection System," *IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA)*, PP. 1--8, 2016.

[33] Alex Galea, Luis Capelo, *Applied Deep Learning with Python*, Packt Publishing, 2018.

[34] Andropov, Sergey and Guirik, Alexei and Budko, Mikhail and Budko, Marina, "Network Anomaly Detection Using Artificial Neural Networks," *IEEE*, 2017.

[35] Monowar H. Bhuyan, Dhruva K. Bhattacharyya, Jugal K. Kalita, *Network Traffic Anomaly Detection and Prevention*, Springer Nature, 2017.

[36] Veeramreddy Jyothsna and Koneti Munivara Prasad, "Anomaly-Based Intrusion Detection System," Available at: <https://www/Anomaly-Based Intrusion Detection System/IntechOpen.html>. Last Accessed on 16 January 2020.

[37] A.Jamadar, Riyazahmed, "Network Intrusion Detection System Using Machine Learning," *Indian Journal of Science and Technology*, Vol. 11(48), 2018.

[38] Dhruva Kumar, Bhattacharyya Jugal, and Kumar Kalita, *Network Anomaly Detection a Machine Learning Perspective*, 2014.

[39] Maleh, Yassine and Ezzati, Abdellah and Qasmaoui, Youssef and Mbida, Mohamed, "A Global Hybrid Intrusion Detection System for Wireless Sensor Networks," *Procedia Computer Science*, Vol. 52, PP. 1047--1052, 2015.

[40] Vajiheh Hajisalem, Shahram Babaie, "A Hybrid Intrusion Detection System Based on ABC-AFS Algorithm for Misuse and Anomaly Detection," *Computer Networks*, PP. 37--50, 2018.

[41] Mehrnaz Mazini, Babak Shirazi, and Iraj Mahdavi, "Anomaly Network-Based Intrusion Detection System Using a Reliable Hybrid Artificial Bee Colony and Adaboost Algorithms," *Journal of King Saud University – Computer and Information Sciences*, 2018.

[42] Karimi, Ahmad Maroof, "Distributed Machine Learning Based Intrusion Detection System, "

Master's Thesis, 2016.

- [43] Liu Hua Yeo, Xiangdong Che,, "Understanding Modern Intrusion Detection Systems: A Survey," 2016.
- [44] Randy Weaver, Dawn Weaver and Dean Farwood, Guide to Network Defence and Countermeasures, US: Information Security Professionals, 2014.
- [45] Pathan, Al-Sakib Khan, The State of the Art in Intrusion Prevention and Detection, CRC Press, 2016.
- [46] Alazab, Ammar and Hobbs, Michael and Abawajy, Jemal and Khraisat, Ansam and Alazab, Mamoun, "Using Response Action With Intelligent Intrusion Detection and Prevention System Against Web Application Malware," *Information Management and Computer Security*, Vol. 22, PP. 431--449, 2014.
- [47] Modi, Chirag and Patel, Dhiren and Borisaniya, Bhavesh and Patel, Hiren and Patel, Avi and Rajarajan, Muttukrishnan, "A Survey of Intrusion Detection Techniques in Cloud," *Journal of Network and Computer Applications*, Vol. 36, PP. 42--57, 2013.
- [48] "Active and passive IDS," Available at: <https://www.dummies.com/computers/operating-systems/windows-xpvista/examining-different-types-of-intrusion-detection-systems/>, Last Accessed on 27 Feburary 2020.
- [49] Singh, Simranjeet, "A Hybrid Intrusion Detection System Design for Computer Network Security," *International Journal of Engineering Sciences and Research and Technology*, 2018.
- [50] Khan, Ruzaina and Hasan, Mohammad, "Network Threats, Attacks and Security Measures: A Review," *International Journal of Advanced Research in Computer Scienc*, Vol. 8, 2017.
- [51] Sibanjan Das, Umit Mert Cakmak, Hands-on Authomated Machine Learning: A Beginners Guide to Building Automated Machine Learning Systems Using AutoML and Python, Packt Publishing Ltd, 2018.
- [52] Garrido, a. P. J. Gabriel, OpenCV 3.x with Python by Example, Birmingham:, Packt Publishing Ltd, 2018.
- [53] Singh, Pramod, Learn PySpark, India: Apress, 2019.
- [54] Brownlee, Jason, Master Machine Learning Algorithms: Discover How They Work and Implement them from Scrach, 2017.
- [55] Torfi, Amirsina, "Cross validation," Available at: <https://machinelearningcourse.readthedocs.io/en/latest/content/overview/overfitting.html>, Last Accessed on 30 October 2019.
- [56] Dipanjan Sarkar, Raghav Bali , and Tamoghna Ghosh, Implement Advanced Deep Learning and Neural Network Models Using Tensorflow and Keras, Packt Publishing Ltd, 2018.
- [57] Andreas C. Mueller and Sarah Guido, Introduction to Machine Learning with Python a Guide for

Data Scientists, USA: O'Reilly Media, 2016.

[58] Brownlee, Jason, Machine Learning Mastery with Python, 2016.

[59] Krupa Joel Chabathula, Jaidhar C.D and Ajay Kumara M.A, "Comparative Study of Principal Component Analysis Based Intrusion Detection Approach Using Machine Learning Algorithms," in *International Conference on Signal Processing, Communication and Networking (ICSCN)*, 2015.

[60] Prashant Kushwaha, Himanshu Buckchash, and Balasubramanian Raman, "Anomaly Based Intrusion Detection Using Filter Based Feature Selection on KDD CUP 99," *IEEE Region 10 Conference (TENCON)*, 2017.

[61] Ammar Alazab, Michael Hobbs, Jemal Abawajy, Moutaz Alazab, "Using Feature Selection for Intrusion Detection System," *International Symposium on Communications and Information Technologies (ISCIT)*, 2012.

[62] Ahmed Fawzy Gad, Practical Computer Vision Applications Using Deep Learning With CNNs, Egypt: Apress, 2018.

[63] Taylor & Francis Group, Feature Engineering for Machine Learning and Data Analytics, London (New York): CRC Press, 2018.

[64] "Evaluating Models: Classification Metrics," Available at: <https://apple.github.io/turicreate/docs/userguide/evaluation/classification.html>, Last Accessed on 9 December 2019.

[65] Hamed Habibi Aghdam, Elnaz Jahani Heravi, Guide to Convolutional Neural Networks, Springer, 2017.

[66] Pramod Singh, Machine Learning with PySpark: With Natural Language Processing and Recommender Systems, Apress, 2019.

[67] Danny Plammer, "Cybercrime drains \$600 billion a year from the global economy", February 21-2018, July 2 2019, <https://www.zdnet.com/article/cybercrime-drains-600-billion-a-year-from-the-global-economy-says-report>. .

[68] Abdulla Amin Aburomman and Mamun Bin Ibne Reaz, "Survey of learning methods in intrusion detection Systems", *International Conference on Advances in Electrical, Electronic and System Engineering*, pp. 362-365, 2016.

[69] A. Midzic, Z. Avdagic and S. Omanovic, "Intrusion Detection System Modeling Based on Neural Networks and Fuzzy Logic", *INES 2016 • 20th Jubilee IEEE International Conference on Intelligent Engineering Systems*: pp 198-194, 2016

[70] J. François, "Deep Learning with Python," *New York: Manning Publications*, 2017

- [71] M. Tavallaee, E. Bagheri, W. Lu, and A.-A. Ghorbani, "A detailed analysis of the kdd cup 99 data set," in Proceedings of the Second IEEE Symposium on Computational Intelligence for Security and Defence Applications, 2009.
- [72] Julian Luengo, Salvador Garcia, Francisco Herrera, "On the choice of the best imputation methods for missing values considering three groups of classification methods", Knowl Inf Syst in Springer-Verlag, London , 2011.
- [73] D. Raumer, L. Schwaighofer, and G. Carle, "Monsamp: A distributed sdn application for qos monitoring," in Computer Science and Information Systems (FedCSIS), 2014 Federated Conference on. IEEE, 2014, pp. 961-968.
- [74] T. Ha, S. Kim, N. An, J. Narantuya, C. Jeong, J. Kim, and H. Lim, "Suspicious traffic sampling for intrusion detection in software-defined networks," Computer Networks, vol. 109, pp. 172-182, 2016.
- [75] F. Chollet *et al.*, "Keras," <https://keras.io>, 2015. 75, 99
- [76] Cbench, <https://github.com/mininet/oflops/tree/master/cbench>. Accessed 04 Jul 2018, n.d.
- [77] C.Y. J. Peng, M. Harwell, S.M. Liou, L. H. Ehman, Advances in missing data methods and implications for educational research, In S. Sawilowsky (Ed.), Real data analysis, pp 31-78, Greenwich, 2006.
- [78] Shichao Zhang , Xindong Wu , Manlong Zhu, "Efficient Missing Data Imputation for Supervised Learning, " In proceeding of: Proceedings of the 9th IEEE International Conference on Cognitive Informatics, ICCI 2010, Beijing China, 2010
- [79] Julian Luengo, Salvador Garcia, Francisco Herrera, "On the choice of the best imputation methods for missing values considering three groups of classification methods", Knowl Inf Syst in Springer-Verlag, London , 2011.
- [80] R.B Kline, Principles and Practice of Structural Equation Modelling, Guilford Press, New York, 1998.
- [81] Amreen Sultana, M.A. Jabbar, "Intelligent Network Intrusion Detection System using Data Mining Techniques", pp. 328-333, 2016.
- [82] Kurniabudi, Deris Stiawan, Darmawijoyo, Mohd Yazid Bin Idris,, Alwi M. Bamhdi ,and Rahmat Budiarto, "CICIDS-2017 Dataset Feature Analysis With Information Gain for Anomaly Detection", IEEE, Volume 8, July 30, 2020,
- [83] J. Gabela *et al.*, "The Effect of Linear Approximation and Gaussian Noise Assumption in Multi-Sensor Positioning Through Experimental Evaluation," in IEEE Sensors Journal, vol. 19, no. 22, pp. 10719-10727, 15 Nov. 15, 2019.
- [84] B. Ding, H. Qian and J. Zhou, "Activation functions and their characteristics in deep neural networks," 2018 Chinese Control and Decision Conference (CCDC), Shenyang, 2018, pp. 1836-1841.

- [85] Raghavendra Chalapathy, Sanjay Chawla “Deep Learning for Anomaly Detection: a survey,”ResearchGate, pp. 1–47, 2019.
- [86] Ding, Bin & Qian, Huimin & Zhou, Jun. (2018). Activation functions and their characteristics in deep neural networks. 1836-1841.65
- [87] Cisco.com,”Cisco IOS NetFlow Configuration Guide”, Release 12.4 available at <http://www.cisco.com>, Sept. 2010.
- [88] Crothers M., Implementing Intrusion Detection Systems, a Hands-On Guide for Securing the Network, USA, 2002.
- [89] T. Tang, L. Mhamdi, D. McLernon, S. A. R. Zaidi, and M. Ghogho, “Deep recurrent neural network for intrusion detection in sdn-based networks,” in 2018 4th IEEE Conference on Network Softwarization and Workshops (NetSoft). IEEE, 2018, pp. 202–206
- [90] T. A. Tang, L. Mhamdi, D. McLernon, S. A. R. Zaidi, and M. Ghogho, “Deep learning approach for network intrusion detection in software defined networking,” in 2016 International Conference on Wireless Networks and Mobile Communications (WINCOM). IEEE, 2016, pp. 258–263
- [91] Iman Sharafaldin, Arash Habibi Lashkari, and Ali A. Ghorbani, “Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization”, 4th International Conference on Information Systems Security and Privacy (ICISSP), Portugal, January 2018.

Appendix A

Python Code

```
In [27]: from sklearn.tree import DecisionTreeClassifier
        from sklearn.ensemble import RandomForestClassifier
        from sklearn.neural_network import MLPClassifier
        from sklearn.svm import SVC
        from sklearn.naive_bayes import GaussianNB
        import matplotlib.pyplot as plt
        import seaborn as sns

        mlp = MLPClassifier(solver='adam', learning_rate_init = 0.005, learning_rate = 'adaptive')
        dec = DecisionTreeClassifier(criterion="entropy", max_depth=3)
        ran = RandomForestClassifier(n_estimators=50)
        svm = SVC(random_state=1)
        naive = GaussianNB()

        models = {"J48" : dec,
                  "NB" : naive,
                  "RF" : ran,
                  "MLP" : mlp,
                  "SVM" : svm
                 }

        scores = { }

        #Test the models
        for key, value in models.items():
```

```
            model = value
            model.fit(x,y)
            scores[key] = model.score(xtest, ytest)

        # results
        scores_frame = pd.DataFrame(scores, index=["Accuracy Score"]).T
        scores_frame.sort_values(by=["Accuracy Score"], axis=0 ,ascending=False, inplace=True)
        print(scores_frame)
        print('\n')
        # Plot the results
        plt.figure(figsize=(8,5))
        sns.barplot(x=scores_frame.index,y=scores_frame["Accuracy Score"])
        plt.ylim(0, 1)
        plt.ylabel("Accuracy%")
        plt.xlabel("C:/Users/muhe/Desktop/Deep-learning-for-intrusion-detection-using-Recurrent-Neural-network-RNN-main/Deep-learning-for
```

```
In [530]: y_pred_rnn = model_rnn.predict_classes(X_test)
        y_probs_rnn=model_rnn.predict_proba(X_test)
        np.savetxt('rnn_predictions.txt', np.transpose(np.concatenate((y_test.reshape((y_test.size, 1)),y_pred_rnn), axis=1)), fmt='%01d')
        np.savetxt('rnn_prob_predictions.txt', np.around(np.transpose(y_probs_rnn),decimals=5), fmt='%.5f')

        #####Plot confusion matrix
```

```
In [604]: skplt.metrics.plot_confusion_matrix(y_test, y_pred_rnn)
        plt.title("RNN-Confusion Matrix")
        plt.rcParams['figure.figsize']=(5,4)
        plt.show()
```

```

In [531]: accuracy = accuracy_score(y_test, y_pred_rnn)
          print("accuracy:", accuracy)
          f1score=f1_score(y_test, y_pred_rnn)
          print("f1-score:", f1score)
          cm=confusion_matrix(y_test, y_pred_rnn)
          print("confusion matrix:\n", cm)
          pr=precision_score(y_test, y_pred_rnn)
          print("Precision:", pr)
          rs=recall_score(y_test, y_pred_rnn)
          print("Recall_score:", rs)

```

```

In [533]: id=dsp_tr[2].shape[1]
          batch_size=32

```

```

In [534]: model_lstm = Sequential()
          #model_dnn3.add(Flatten())
          model_lstm.add(LSTM(8, input_dim=id, return_sequences=True))
          model_lstm.add(Dropout(0.1))
          model_lstm.add(LSTM(8, return_sequences=False))
          model_lstm.add(Dropout(0.1))
          model_lstm.add(Dense(1))
          model_lstm.add(Activation('sigmoid'))

```

```

In [536]: y_pred_lstm = model_lstm.predict_classes(X_test)
          y_probs_lstm=model_lstm.predict_proba(X_test)
          np.savetxt('lstm_predictions.txt', np.transpose(np.concatenate((y_test.reshape((y_test.size, 1)), y_pred_lstm), axis=1)), fmt='%01d')
          np.savetxt('lstm_prob_predictions.txt', np.around(np.transpose(y_probs_lstm), decimals=5), fmt='%.5f')

          #####Plot confusion matrix

          skplt.metrics.plot_confusion_matrix(y_test, y_pred_lstm)
          plt.title("LSTM-Confusion Matrix")
          plt.show()

```

```
In [540]: batch_size = 64
         id=dsp_tr[2].shape[1]
```

```
In [541]: model_gru = Sequential()
         #model_gru.add(Flatten())
         model_gru.add(GRU(32,input_dim=id, return_sequences=True))
         model_gru.add(Dropout(0.1))
         model_gru.add(GRU(32, return_sequences=True))
         model_gru.add(Dropout(0.1))
         model_gru.add(GRU(32, return_sequences=True))
         model_gru.add(Dropout(0.1))
         model_gru.add(GRU(32, return_sequences=False))
         model_gru.add(Dropout(0.1))
         #model_gru.add(Flatten())
         model_gru.add(Dense(1))
         model_gru.add(Activation('sigmoid'))
```

```
In [543]: y_pred_gru = model_gru.predict_classes(X_test)
         y_probs_gru=model_gru.predict_proba(X_test)
         np.savetxt('gru_predictions.txt', np.transpose(np.concatenate((y_test.reshape((y_test.size, 1)),y_pred_gru), axis=1)), fmt='%01d')
         np.savetxt('gru_prob_predictions.txt', np.around(np.transpose(y_probs_gru),decimals=5), fmt='%.5f')

         #####Plot confusion matrix

         skplt.metrics.plot_confusion_matrix(y_test, y_pred_gru)
         plt.title("GRU-Confusion Matrix")
         plt.show()
```

```
In [559]: batch_size = 64
         id=dsp_tr[2].shape[1]
         # define network
         model_dnn3 = Sequential()
         model_dnn3.add(Flatten())
         model_dnn3.add(Dense(1024,input_dim=id,activation='relu'))
         model_dnn3.add(Dropout(0.01))
         model_dnn3.add(Dense(768,activation='relu'))
         model_dnn3.add(Dropout(0.01))

         model_dnn3.add(Dense(1))
         model_dnn3.add(Activation('sigmoid'))
```

```
In [578]: #lstm_output_size = 128

         model_cnn = Sequential()
         model_cnn.add(Conv1D(64, 3, padding="same",activation="relu",input_shape=(id, 1)))
         model_cnn.add(Conv1D(64, 3, padding="same", activation="relu"))
         model_cnn.add(MaxPool1D(pool_size=(2)))
         model_cnn.add(Conv1D(128, 3, padding="same", activation="relu"))
         model_cnn.add(Conv1D(128, 3, padding="same", activation="relu"))
         model_cnn.add(MaxPool1D(pool_size=(2)))
         model_cnn.add(Flatten())
         model_cnn.add(Dense(128, activation="relu"))
         model_cnn.add(Dropout(0.5))
         model_cnn.add(Dense(1, activation="sigmoid"))
```