FLOW BASED ANOMALY DETECTION IN SOFTWARE DEFINED NETWORKING: A DEEP LEARNING APPROACH WITH FEATURE SELECTION METHOD

SAMRAT KUMAR DEY
(*BSc Engg., PSTU*)

A THESIS SUBMITTED
FOR THE DEGREE OF MASTER OF SCIENCE IN
COMPUTER SCIENCE AND ENGINEERING
DEPARTMENT OF COMPUTER SCIENCE AND
ENGINEERING
MILITARY INSTITUTE OF SCIENCE AND TECHNOLOGY

2018

APPROVAL

The thesis titled **"FLOW BASED ANOMALY DETECTION IN SOFTWARE DEFINED NETWORKING: A DEEP LEARNING APPROACH WITH FEATURE SELECTION METHOD"** Submitted by Samrat Kumar Dey, Roll No: 1016140003 Session: 2015-2016 has been accepted as satisfactory in partial fulfillment of the requirement for the degree of Master of Science in Computer Science and Engineering on 15 December 2018

DECLARATION

I hereby declare that this thesis is my original work and it has been written by me in its entirety. I have duly acknowledged all the sources of information which have been used in the thesis.

This thesis has also not been submitted for any degree in any university previously.

Samrat Kumar Dey
15 December 2018

ACKNOWLEDGEMENTS

I am precisely thankful to Almighty for his unceasing and immense blessings without which my thesis completion would remain scattered and incomplete. I express my heartiest gratitude, profound indebtedness and deep respect to my supervisor, **Dr. Md. Mahbubur Rahman**, Professor, Department of CSE, Military Institute of Science and Technology, for his constant supervision, affectionate guidance and great encouragement and motivation. His keen interest on the topic and valuable advices throughout the study was of great help in completing thesis.

I am especially grateful to the Department of Computer Science and Engineering of Military Institute of Science and Technology (MIST) for providing their all out support during the thesis work.

Finally, I would like to thank my parents, family members and friends for their appreciable assistance, patience and suggestions during the course of my thesis.

Dhaka Samrat Kumar Dey

December 2018

TABLE OF CONTENTS

LIST OF TABLES .... **X**

LIST OF FIGURES .... **XI**

LIST OF ABBREVIATION .... **XIII**

LIST OF SYMBOLS .... **XIV**

CHAPTER 1 INTRODUCTION **1**

1.1 Introduction 1

1.2 The Software Defined Networking Paradigm 1

1.2.1 Planes in software defined networks 2

1.2.2 Abstractions in software defined networks 3

1.3 Motivations 4

1.3.1 Monitoring and measurements in SDN 6

1.3.2 OpenFlow 6

1.3.3 The OpenDaylight controller 8

1.4 Problem Statement 9

1.5 Purpose, Scope and Contribution 9

1.6 Organization of Thesis 10

CHAPTER 2 LITERATURE REVIEW **13**

2.1 Objectives with Specific Aims and Possible Outcome 17

CHAPTER 3 BACKGROUND **18**

3.1 Introduction 18

3.2 Intrusion Detection Method 18

3.2.1 Signature based detection 18

3.2.2 Anomaly based detection 19

3.2.3 Flow based detection 19

3.2.4 Packet based detection 20

3.3 What is Software Defined Networking? 20

3.3.1 SDN overview 21

3.3.2 OpenFlow 21

3.3.3 Flow based detection using SDN 22

3.4 Data Set for Intrusion Detection 23

3.4.1 Public labeled dataset for an IDS 23

3.4.2 NSL KDD dataset 24

3.5 Machine Learning and Deep Learning 32

3.5.1 Why is deep learning better than machine learning? 32

3.5.2 Recurrent neural network (RNN) 33

3.5.3 Long short term memory (LSTM) 33

3.5.4 Gated recurrent unit (GRU) 34

3.5.5 Multi-layer GRU RNN 35

3.6 Hyper Parameters 36

3.7 Evaluation Metrics 37

CHAPTER 4 METHODOLOGY AND IMPLEMENTATION 39

4.1 Introduction 39

4.2 Proposed Architecture 39

4.3 Recurrent Neural Network (RNN) 40

4.4 Long Short Term Memory (LSTM) 41

4.5 Gated Recurrent Unit (GRU) 42

4.6 Dataset 43

4.7 Description of Scikit-Learn 44

4.8 ANOVA F-test and RFE 44

4.9 SDN Based Flow Based Anomaly Detection Architecture 45

CHAPTER 5 EXPERIMENTAL PROCEDURES FOR METHODOLOGY ASSESSMENT 48

5.1 Introduction 48

5.2 Data Cleaning and Pre-processing 49

5.3 Feature Scaling 49

5.4 Features Selection 49

5.5 Building the Model 50

5.5.1 LSTM model 51

5.5.2 GRU model 51

5.6 Loss Function Definition 52

CHAPTER 6 EXPERIMENTAL RESULTS AND DISCUSSIONS 53

6.1 Introduction 53

6.2 Appropriate Features Selection 53

6.3 Hyper Parameter Tuning 54

6.4 Performance Results of IDS Classifier: 55

6.5 Comparison with Existing Work 58

CHAPTER 7 CONCLUSIONS AND FUTURE WORK 60

REFERENCES 62

APPENDIX A FEATURE SELECTION CODE SEGMENT 69

APPENDIX B ANOMALY DETECTION MODEL BUILD CODE SEGMENT.. 70

LIST OF PUBLICATIONS 72

ABSTRACT

Software Defined Networking (SDN) has come to prominence in recent years and demonstrates an enormous potential in shaping the future of networking by separating control plane from data plane. As a newly emerged technology, SDN has its inherent security threats that can be mitigated by securing the OpenFlow controller that manages flow control in SDN. On the other hand, Recurrent Neural Networks (RNN) show a remarkable result in sequence learning, particularly in architectures with gated unit structures such as Long Short-term Memory (LSTM). In recent years, several permutations of LSTM architecture have been proposed mainly to overcome the computational complexity of LSTM. Therefore, in this dissertation, a novel study is presented that will empirically investigate and evaluate flow-based anomaly detection method in OpenFlow controller using LSTM architecture variants such as Gated Recurrent Unit (GRU). Hence, in this exploration, we propose a combined Gated Recurrent Unit Long Short-Term Memory (GRU-LSTM) Network intrusion detection architecture. In order to improve the classifier performance, an appropriate ANOVA F-Test and Recursive feature Elimination (RFE) (ANOVA F-RFE) feature selection method also have been applied. The proposed approach is tested using the benchmark dataset NSL-KDD. A subset of complete dataset with convenient feature selection ensures the highest accuracy of 87% with GRU-LSTM Model.

LIST OF TABLES

Table 3.1: Structure of Match Information 23

Table 3.2: Structure of Flow Information 23

Table 3.3: List of NSL-KDD Dataset Files and their Description [46] 25

Table 3.4: Basic features of each network connection [46] 26

Table 3.5: Content related traffic features of each network connection [46] 27

Table 3.6: Time related traffic features of each network connection [46] 28

Table 3.7: Host related traffic features of each network connection [46] 29

Table 3.8: Host related traffic features of each network connection [46] (continued) ... 30

Table 3.9: Attribute value type of NSL KDD dataset [46] 31

Table 3.10: Different attacks types in NSL KDD dataset [46] 31

Table 5.1: Parameter Values 50

Table 6.1: Selected Features after Applying ANOVA F-Test and REF 54

Table 6.2: Evaluation Metrics for IDS Classifier (learning rate =0.01) 55

Table 6.3: Evaluation Metrics for IDS Classifier (learning rate =0.1) 56

Table 6.4: Evaluation Metrics for IDS Classifier (learning rate =0.001) 57

Table 6.5: Comparison of accuracy with other studies 59

LIST OF FIGURES

Figure 1.1: Software Defined Networking architecture [1] 2

Figure 1.2: Components of a flow entry in OpenFlow 7

Figure 1.3: Architecture of the OpenDaylight beryllium release [7] 9

Figure 3.1: Layer representation in SDN [39] 21

Figure 3.2: Structure of Feed Forward Neural Network 33

Figure 3.3: Structure of Recurrent Neural Network 33

Figure 3.4: Basic structure of Long-Short-Term Memory RNN [49] 34

Figure 3.5: Structure of a Single layer Gated Recurrent Unit [49] 35

Figure 3.6: Structural representation of Multi-layer GRU [50] 36

Figure 3.7: Tabular form of confusion matrix 37

Figure 4.1: Proposed research methodology for Network Intrusion Detection 40

Figure 4.2: Working procedure of Recurrent Neural Network 41

Figure 4.3: Working procedure of Long-Short-Term Memory 42

Figure 4.4: Architecture of single layer Gated Recurrent Unit 42

Figure 4.5: Description of NSL-KDD dataset with features and type 43

Figure 4.6: SDN controller based IDS architecture [19] 45

Figure 4.7: Working procedure of OpenFlow Protocol in SDN 46

Figure 4.8: Proposed flow based anomaly detection architecture in SDN 47

Figure 5.1: Code segment of GRU LSTM Model build 51

Figure 5.2: Code segment of Loss function definition of our Model 52

Figure 6.1: Hyper Parameter set initilization for proposed model 54

Figure 6.2: Evaluation metrics for IDS classifier (learning rate =0.01) 56

Figure 6.3: Evaluation metrics for IDS classifier (learning rate =0.1) 57

Figure 6.4: Evaluation metrics for IDS classifier (learning rate =0.001) 58

LIST OF ABBREVIATION

AC: Accuracy

ANOVA: Analysis of variance

BPTT: Backpropagation through time (BPTT)

DDoS: Distributed Denial of Services

DNN: Deep neural network

DoS: Denial of Service

FAR: False Alarm Rate

FNN: Feed Forward Neural Network

GRU: Gated Recurrent Unit

GSA: Gravitational Search Algorithm

IA: Idle–timeout Adjustment

ISP: Internet Service Provider

LSTM: Long Short-Term Memory

MAE: Mean Absolute Error

MCC: Mathews Correlation Coefficient

MLP: Multi-Layer Perceptron

NIDS: Network Intrusion Detection Systems

OF: Open Flow

R2L: Root to Local

RFE: Recursive Feature Elimination

RNN: Recurrent Neural Network

SAE: Stacked Auto Encoder

SDN: Software Defined Networking

SOHO: Small Office/ Home Office

SVM: Support Vector Machine

U2R: Use to Root

LIST OF SYMBOLS

$\boldsymbol{x_t}$:	input state
$\boldsymbol{o_t}$:	output state unit
$\boldsymbol{s_t}$:	hidden state
$\boldsymbol{\sigma}$:	sigmoid function
$\boldsymbol{tanh}$:	tanh function
$\boldsymbol{r_t}$:	reset gate
$\boldsymbol{h_t}$:	activation function
$\widetilde{\boldsymbol{h_t}}$:	candidate activation
$\boldsymbol{z_t}$**:**	gravitational search algorithm

CHAPTER 1

INTRODUCTION

1.1 Introduction

This chapter provides an introduction including the main topics and technologies related to the architecture developed for this master thesis. Firstly, we provide a section describing the scope of the topic. There, we briefly discuss the origin of the emerging Software-Defined Networking (SDN) paradigm and describe some details to better understand how these new networks operate. In particular, we focus on traffic monitoring and the new challenges to address in SDN.

Then, we provide a section motivating the main issues around the proposed model and some technologies we used in this thesis. This section is followed by another section that describes the main contributions of the proposal presented in this thesis. Finally, we outline the structure of this report including a description of the chapters and appendices included in this thesis.

1.2 The Software Defined Networking Paradigm

Current Internet protocol (IP) networks are increasingly more multifaceted and hard to manage, and this is a tendency that it is going to be more accused with new emerging paradigms of services such as virtualized cloud computing, big data applications, data center services or multimedia content delivery. With the aim of reverse this situation, the Software-Defined Networking (SDN) paradigm was proposed as a solution to build a more flexible network infrastructure with programmable devices, where new protocols and policies can be implemented via software without needing any hardware modification.

The SDN paradigm proposes to separate the control and data planes of “legacy” networks for the sake of flexibility. In this way, the data plane is located in the SDN-enabled forwarding devices (i.e., SDN switches), while the control plane is logically centralized in new entities called SDN controllers. Thus, the different planes in SDN allow to create different layers of abstraction and, thereby, providing an unprecedented level of flexibility in network management. Following Figure 1.1 depicts the architecture of an SDN-based network. There, we can see the different entities, planes and layers of abstraction in SDN. We briefly describe the main elements below.

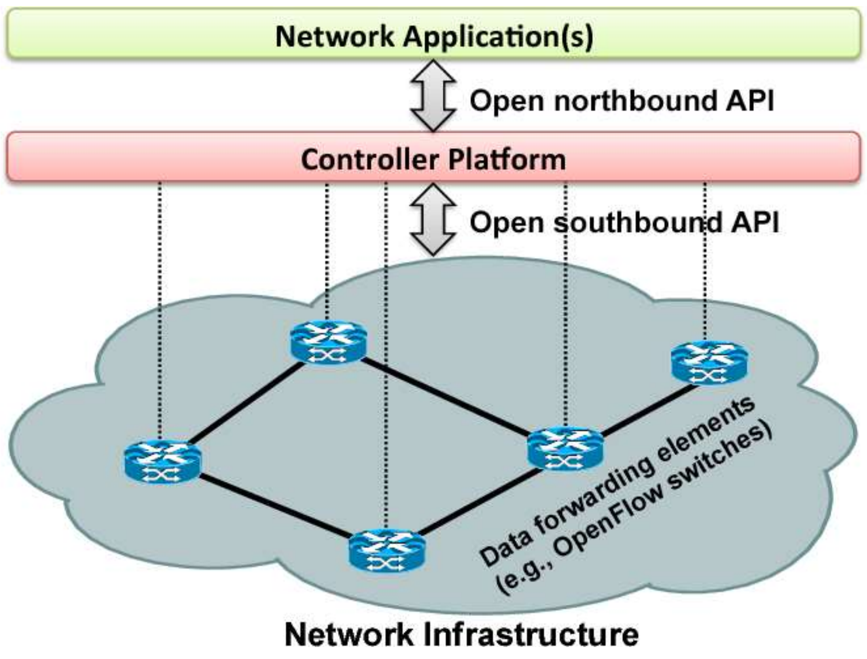


Figure 1.1: Software Defined Networking architecture [1]

1.2.1 Planes in software defined networks

- **Control plane:** This plane is in charge of calculating the local state of the forwarding devices in the network and enforcing the proper policies for the correct operation of the network. This includes all the network management tasks such as routing, traffic engineering or security policy enforcement. Unlike traditional networks, in SDN forwarding decisions are flow-based, instead of destination-based. All the packets matching a specific criterion are applied the same actions. This flow abstraction enables to unify the behavior of different types of network devices (e.g., switches, routers, firewalls). In a nutshell, the control plane makes decisions about how and where the traffic is sent and manages all the signaling of the network to properly configure the network devices.

- **Data plane:** This plane performs traffic forwarding in the network devices (i.e., the switches) according to the rules defined by the control plane. This also includes traffic filtering and the different actions that can be typically executed for new incoming packets in a switch. It is worth noting that switches in SDN are of general purpose, i.e., they execute the flow-level rules installed by the control plane and can combine actions that used to correspond to different types of network devices (e.g., routers, switches, middle boxes) in traditional networks. This plane is also known as the "forwarding plane".

1.2.2 Abstractions in software defined networks

Following the definition in [1], we distinguish the three abstractions described below:

- **Forwarding abstraction:** This abstraction allows the data plane to ideally perform any forwarding action determined by the control plane while hiding the underlying hardware in the network infrastructure. Currently, OpenFlow [2] is the dominant protocol of SDN that implements this abstraction. This protocol provides a standard API (Southbound API) for the communication between the control and the data planes, i.e., between the SDN controllers and the OpenFlow enabled switches.

- **Distribution abstraction:** This abstraction enables SDN applications to operate without the necessity of being aware of the distributed state of the network. In comparison to traditional networks, the status information in SDN is logically centralized and it enables to make much better decisions with a global view of the network in the controllers. This is possible only due to "Network Operating Systems" (NOS) of SDN, which collect the status information of the network and it is in charge of installing the desired rules in the switches through a standard communication protocol.

- **Specification abstraction:** This abstraction allows to describe the desired operation of the network by means of high-level policies defined by network management applications. Thus, these policies are then mapped into sets of physical configurations for the global network view exposed by the SDN controller. This abstraction provides an interface (Northbound API) which ideally allows network applications to operate over simplified abstract models of the network which are oblivious of the underlying network topology.

The design of the SDN paradigm enables to perform a fine-grained management of the network, taking advantage of the decision making from the global perspective of the network in the controller. However, to be successful in current dynamic environments, traffic monitoring becomes a cornerstone in SDN given that management applications often need to make use of accurate and timely traffic measurements. Scalability is considered one of the inherent issue of SDN. For an appropriate design of a monitoring system, it is required to consider the network and processing overhead to store and gather the flow statistics. On the one hand, note that controllers are critical points in the infrastructure since all the management decisions are made and communicated from there to each switch under its control. On the other hand, the most straightforward way of implementing per flow monitoring is by maintaining an entry for each flow in a table of the switch. Each of these entries has some counters which are updated every time a packet matches them. Thus, obtaining accurate measurements of all flows results in a great constraint, since nowadays OpenFlow commodity switches do not support a large number of flow entries due to their limited hardware resources available [3].

This thesis covers a study of flow based traffic monitoring in SDN-based networks. More specifically, it aims to identify all the issues around traffic measurement derived from the SDN paradigm and proposes a scalable and OpenFlow compliant monitoring system more suitable for those networks of the future by developing a deep learning model. The approach of this solution is to obtain flow-level measurement reports equivalent to those of NetFlow [4] in traditional networks and to design an architecture which can detect intrusion for flow based traffic. Likewise, we evaluated our system to analyze its feasibility and to quantify its accuracy, resource requirements and network overhead.

1.3 Motivations

The SDN paradigm was the outcome of some works of the research world that suggested the necessity of building programmable networks as a practical way to experiment with new protocols in production networks. From its inception, it has gained lots of attention from academia and industry. It is supported by giants of the Internet world like Google, Cisco, HP, Juniper or NEC and by standardization organizations like the Open Network Foundation (ONF) or the Internet Engineering Task Force (IETF), so one can state that this network paradigm has a lot of potential to succeed. The proposal of the OpenFlow protocol [2] in 2008 was its major driver. In that text, they talk about the commonly held belief that the network infrastructure was “ossified”. Thus, they proposed OpenFlow as a protocol for the communication between the forwarding and control planes in order to decouple logically and physically these two planes.

SDN introduces the benefits of a centralized approach to network configuration. That way, each time network administrators want to make a policy change do not have to configure individually by-hand each network device using its own vendor-specific code. Unlike traditional networks with distributed management, in SDN the administrator can apply some new high level policies from the controller and this is the responsible for translating the policies into rules and install them in the forwarding devices involved. This allows software developers to be oblivious of the underlying devices and develop their networking logic the same way they do in computer software.

Furthermore, SDN-based networks offer a level of flexibility never seen before. It allows to perform a fine-grained flow-level management ideal for services with strict QoS requirements. It successfully adapts to current environments with high fluctuate traffic to make an efficient use of the network resources. In SDN, the forwarding devices are of general purpose, i.e., they are not designed for a specific network function (router, firewall). This results in the possibility of re-configuring via software the topology and change the role of the different devices at run-time, taking advantage of the global knowledge of the network state in the controller. This is an arduous task in traditional networks, since forwarding devices are inflexible due to the underlying hardwired implementation of routing rules. Concerning the measurements and monitoring tasks, SDN allows to collect some traffic statistics that in traditional large networks in some cases it is unfeasible.

In [3], they envision that, for the moment, the majority of research efforts are focused on providing solutions and services over SDN-based networks, while network Security is not given much attention. Likewise, they remark the importance to consider the network security before the technology is widely deployed and, therefore, network security of SDN arises as a real need.

1.3.1 Monitoring and measurements in SDN

The huge scale and diversity of today's Internet traffic make it difficult for the operators to measure and maintain the status and dynamics of the network in short timescales. This, in turn, has become a great challenge, since there are more and more services and applications with guaranteed QoS requirements to be maintained along end-to-end network paths. It motivates the necessity of ubiquitous accurate traffic measurement and monitoring mechanisms.

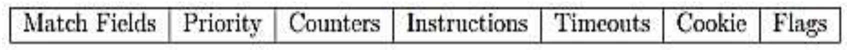
The SDN paradigm makes it easier to perform QoS measurements and enables to perform a fine-grained management as well as making an efficient use of the network resources. However, although the SDN paradigm solves some classical problems of the traditional networks, it brings new challenges. The decoupling of the control and forwarding planes has some new implications that need to be identified and considered in order to devise new smart solutions. Among these issues, the introduction of a centralized control plane makes necessary to consider now a latency between the forwarding and control planes that did not exist in legacy networks. This latency depends on the delay due to the network connection as well as the availability of the SDN controller. Thus, the controller becomes a critical point in the infrastructure and it is prone to become a network bottleneck. In this way, it is of vital importance to find a tradeoff between the tasks where the controller is involved and those that may be devolved to the forwarding devices.

As a conclusion, we see that nowadays there are a number of proposals around measurement and monitoring in SDN with their respective advantages and drawbacks, but there is still much work to be done.

1.3.2 OpenFlow

Since its inception in 2008, OpenFlow [2] has become the de facto protocol for the South bound interface (communication between control and data planes) in SDN. This makes this protocol the main enabler of the SDN paradigm, since it was the first proposal that allowed to completely decouple the control and data planes of a network, which is the basis of the SDN paradigm. This proposal also introduced the idea of performing flow-level management with the aim of creating a standard where the control plane could be oblivious of the underlying hardware of the forwarding devices in the network.

OpenFlow allows to dynamically define the forwarding state of the network from the SDN controller by installing in the switches sets of flow entries. These flow entries are stored in flow tables in the switches and determine their behavior. Figure 1.2 illustrates the main components of a flow entry.



Match Fields	Priority	Counters	Instructions	Timeouts	Cookie	Flags

Figure 1.2: Components of a flow entry in OpenFlow

We describe below these components:

- **Match fields:** This field defines a filter (packet headers, ingress port, metadata, and others) to specify the packets that will be processed by this flow entry.

- **Priority:** Defines a priority to determine the flow entry to be applied to a packet when some flow entries have an overlap.

- **Counters:** These are some records that maintain the number of packets and bytes processed by the flow entry. It also store the life time of the flow entry since it was installed in the switch.

- **Instructions:** For packets matching the flow entry a set of actions to be applied.

- **Timeouts:** Defines how the flow entry is expired by the switch according to time. There are two types of timeouts: 1. the hard timeout defines the maximum amount of time since the flow entry was installed by the SDN controller in the switch. 2. The idle timeout defines the maximum time interval between two consecutive packets matching the flow entry. Both timeouts can be installed simultaneously to decide when the flow entry will be evicted from the switch.

- **Cookie:** Unique opaque value selected by the SDN controller to identify the flow entry. This allows the controller to filter specific flow entries when modifying or deleting flow entries.

- **Flags:** These fields define how the flow entries are managed in the switch. For instance, it is possible to define if the switch sends a flow removed message to the controller including the data of the counters when the flow entry expires

1.3.3 The OpenDaylight controller

The OpenDaylight controller [5] is the result of an open-source project leaded by the Linux foundation which was announced in 2008. This project is strongly supported by both academia and industry and it was created with the aim of accelerating as much as possible the adoption of Software-Defined Networking (SDN) and Network functions virtualization (NFV) in future networks. This project has a vast support from big players of the current Internet world including companies such as Cisco, Ericsson, Red Hat, ZTE, NEC, AT&T, DELL, Fujitsu, Huawei, Intel, Juniper and many others [6].

OpenDaylight is a vendor-independent platform, which makes it much easier to be adopted in any SDN-based network using switches from different vendors with support for different protocols. Thus, it offers support for a wide range of protocols for the Southbound interface in SDN. It includes support for protocols such as OpenFlow, OF-Config, NETCONF, LISP, OVSDB, BGP or SNMP. This allows also to operate in network environments combining pure SDN devices (e.g., OpenFlow-based switches) with other network devices that support popular protocols already used in traditional networks (e.g., BGP, SNMP).

Probably, the key success of the rapid growth of OpenDaylight lies in its large support community. Since it is open-source, anyone can collaborate in the development of the product. Thus, the members of the community can either contribute in many different projects that are already in progress or propose new ones which are evaluated for their approval. Likewise, members in the community are very active answering questions, which eases even more the collaboration among developers. As a result, they are able to constantly release new versions including many novel features. Figure 1.3 depicts all the different plugins that offers OpenDaylight in its Beryllium version, which is the release we used for the implementation of the monitoring system presented in this report. Typically, each of these plugins belong to different projects that are independently developed by different groups of developers following some common programming guidelines.

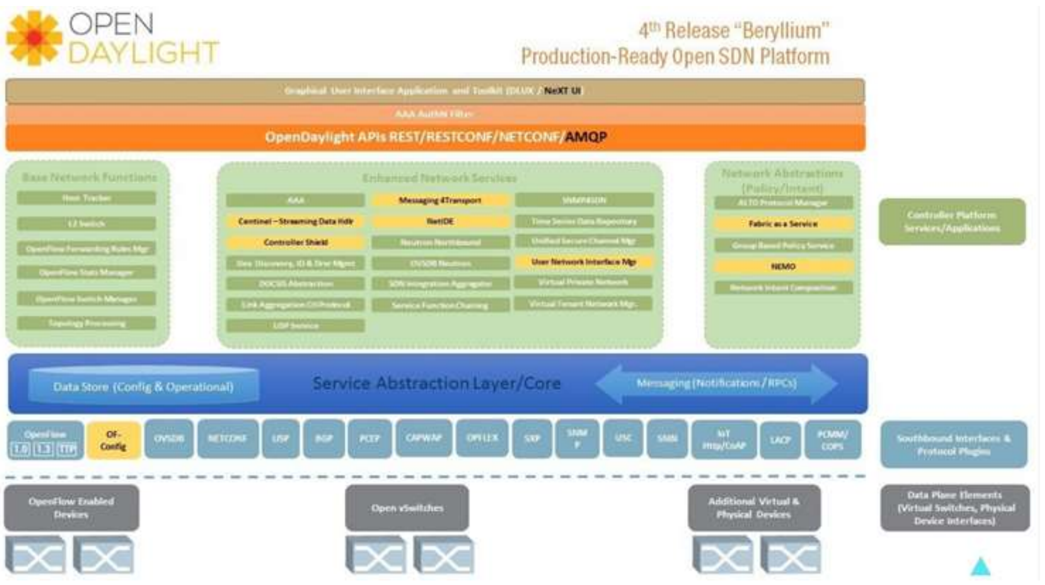

OPEN DAYLIGHT
4th Release "Beryllium"
Production-Ready Open SDN Platform
Graphical User Interface Application and Toolkit (DLUX) / NeXT UI
AAA AuthN Filter
OpenDaylight APIs REST/RESTCONF/NETCONF/AMQP
Base Network Functions
Host Tracker
L2 Switch
OpenFlow Forwarding Rules Mgr
OpenFlow Stats Manager
OpenFlow Switch Manager
Topology Processing
Enhanced Network Services
AAA
Centinel – Streaming Data Hdlr
Controller Shield
Dev Discovery, ID & Drvr Mgmt
DOCSIS Abstraction
Link Aggregation Ctl Protocol
LISP Service
Messaging 4Transport
NetIDE
Neutron Northbound
OVSDB Neutron
SDN Integration Aggregator
Service Function Chaining
VNMRSION
Time Series Data Repository
Unified Secure Channel Mgr
User Network Interface Mgr
Virtual Private Network
Virtual Tenant Network Mgr.
Network Abstractions (Policy/Intent)
ALTO Protocol Manager
Fabric as a Service
Group Based Policy Service
NEMO
Network Intent Composition
Controller Platform Services/Applications
Data Store (Config & Operational)
Service Abstraction Layer/Core
Messaging (Notifications / RPCs)
OpenFlow 1.0 1.3 TTP
OF-Config
OVSDB
NETCONF
LISP
BGP
PCEP
CAPWAP
OPFLEX
SXP
SNMP
USC
SNBI
IoT Http/CoAP
LACP
PCMM/COPS
Southbound Interfaces & Protocol Plugins
OpenFlow Enabled Devices
Open vSwitches
Additional Virtual & Physical Devices
Data Plane Elements (Virtual Switches, Physical Device Interfaces)

Figure 1.3: Architecture of the OpenDaylight beryllium release [7]

1.4 Problem Statement

The goal of this thesis is to analyze and answer the following research questions:

- What are the security and privacy issues relevant to the SDN environment?
- Does GRU-LSTM better than the other machine learning approaches for Intrusion Detection on the SDN?
- Does a feature selection mechanism for NSL KDD dataset perform better for a Network intrusion detection architecture?

1.5 Purpose, Scope and Contribution

The purpose of this thesis is to analyze the applications of deep learning to the SDN environment by evaluating recurrent neural network algorithms on the intrusion detection dataset. I believe that this research has an immense potential to open the doors for Deep Learning applications to both the cyber security domain and the SDN domain. The importance of security in today's connected world requires analyzing the humongous amount of heterogeneous data, and this cannot be possible without the help of artificial intelligence. This research can be extended by applying the algorithms on GPU environment on real-time SDN data.

Though there are various deep learning algorithms such as deep neural networks, auto encoders, convolutional neural networks and recurrent neural networks, the research problem requires an algorithm that can learn from historical data. Therefore, we have selected the family of recurrent neural networks for the research. Considering the need of building smart and lightweight solutions for the SDN Environment, we have performed the experiments with only the Long Short Term Memory (LSTM) and Gated Recurrent-Unit (GRU) algorithm. However we have evaluated various versions of LSTM and GRU to obtain optimized results on the dataset. Due to the unavailability of intrusion detection data for an SDN network, we have considered the NSL-KDD dataset, an improved version of KDDCup99, which is used as a benchmark by most other IDS-Machine Learning researchers.

As this is an inter disciplinary research work which involved cybersecurity, artificial intelligence and computer networks, a lot of time has been spent in understanding the depth of the concepts in each field. We started with understanding the attack types in an intrusion detection dataset. We followed up with learning the architecture of SDN and analyzed the possible malware attacks in OpenFlow controller. We then realized that the intrusion detection dataset must be classified with machine learning algorithms. However, we recognized that the application of deep learning algorithms is the most suitable approach for the research problem defined. We performed the experiments using a robust deep learning tool called Google’s TensorFlow. We have also applied the higher versions of recurrent neural networks. The performance results are compared for each designed IDS with existing IDS available in literature. This overall interdisciplinary practical approach made this research unique.

1.6 Organization of Thesis

With the aim of providing an overview of this thesis to the reader, we describe in this section the structure of the report. The present report is composed of the seven chapters and two appendices described below.

In chapter One, I describe all the main aspects related to the model developed for this thesis. It begins with an introduction to the SDN paradigm. Then, we provide a section with motivations around SDN and, particularly, traffic monitoring in SDN-based networks. Finally, there is a section that provides a description of all the contributions achieved in this thesis.

In chapter Two, I covered most relevant research efforts around the topics addressed in this thesis. It includes numerous proposals in the literature that were the part of SDN security with feature selection approach as well as the main solution around the Deep learning approach. In particular we showed most relevant flow monitoring solution using different deep learning approach and described their drawbacks and advantages.

Chapter Three provides information on the background involved in developing this research. It introduces the concepts and technologies of Deep learning methods followed by SDN security and privacy concerns. Further, this chapter addresses the characteristics to be considered while developing SDN based flow monitoring systems. The main intention of this chapter is to provide a complete overview of the concepts and algorithms for the reader with minimal knowledge in this field of research

Chapter Four provides information regarding in detail methodology and how these theory utilized in this thesis work. Apart from that it also addresses the key issues of methodologies for implementation. It also demonstrates each algorithm architecture and its equations, along with all related topics including intrusion detection, parameter definitions, and evaluation matrices used in this thesis.

Chapter Five showed the procedure of experiment conduction for the successful assessment of our methodology. It also describes the design and model of the flow monitoring mechanism for Software Defined Networking. Initially, it provides some theoretical background with algorithm and model about our methodology. Then, the procedure of experiment is thoroughly explained. Finally, with coding we showed and analyzed developed model for achieving our experimental results.

In chapter Six the result of the experiment is reported, as well as a discussion of the performance of each algorithm with an overall analysis. Furthermore, we also compared our experimental results with other state-of-the-art approaches and showed how our proposed model outperformed others results in terms of methodology and experiment conduction.

Chapter Seven summarizes the core aspects about the model developed for this master thesis. It also includes some guidelines for future work to extend the monitoring system and the experiments we describe in this thesis.

CHAPTER 2

LITERATURE REVIEW

This section focuses on highlighting the related work for network intrusion detection in SDN with deep learning algorithms. In particular, GRU based LSTM architectures with feature selection mechanism, as it is considered to be a special kind of RNN. According to [8-10] LSTM architecture is widely used in sequential data problems. RNN, GRU, and LSTM are being used in many studies in the field of intrusion detection.

Nowadays, Flow-based anomaly detection systems are widely studied. Flow-based anomaly detection system using a Multi-Layer Perceptron (MLP) neural network with one hidden layer and Gravitational Search Algorithm (GSA) proposed in [11] can classify benign and malicious flows with a high degree of accuracy. Authors of [12] introduce a novel concept for an inductive NIDS that uses One-Class Support Vector Machines for analysis and is trained with malicious network data in contrast to other systems gives a low false alarm rate. A lightweight method for DDoS attack detection based on traffic flow features is presented in [13], in which such information is extracted with a very low overhead by taking advantage of a programmatic interface provided by the NOX platform. This method produces a high rate of detection obtained by flow analysis using Self Organizing Maps.

Kokila RT *et al.* [14] analyze the SVM classifier for DDoS detection and their experiments show that SVM classifier gives less false positive rate and high classification accuracy as compared to other techniques. Trung *et al.* [15] propose an optimized protection mechanism (OpenFlowSIA) that uses SVM classifier along with the authors' proposed Idle−timeout Adjustment (IA) algorithm to secure and save the network resources under flooding attacks in SDN.

In [16] a lightweight solution based on the entropy variation of the destination IP address is provided to detect DDoS attacks within first 250 packets of the traffic that carries malicious packets. Authors of [17] design a Markov-based graph model that outperforms k-nearest neighbors classification. Niyaz *et al.* [18] use stacked auto encoder (SAE) based DL model to detect multi-vector DDoS attacks in SDN. Tang *et al.* [19] propose a Gated Recurrent Unit Recurrent Neural Network (GRU-RNN) enabled intrusion detection systems for SDNs which is tested using the NSL-KDD dataset and they achieve 89% accuracy.

Tuor *et al.* [20] presented an online deep learning method for intrusion detection due to its excellent ability to learn patterns, where they employed deep neural network auto encoders for unsupervised network anomaly detection using time aggregated statistics as features.

An intelligent attack detection method in social network works based on LSTM with NSL KDD dataset has proposed by Yunsheng Fu. *et al.* [21]. Their experiment consist of data preprocessing, feature abstraction, training and detection.

Yunsheng Fu. *et al.* [21] proposed an intelligent attack detection method in social network works based on LSTM, with the purpose of achieving a high detection rate. They used the NSL-KDD dataset to evaluate the performance of their proposed method. Experimental results demonstrated that their proposed intelligent attack detection method achieved state-of-the-art performance and is much faster than most post-processing algorithms. Their developed intelligent attack detection method scored more than 98%

Al-kasassbeh *et al.* [22] showed that Random Forest with higher accuracy at 94% and Meena *et al.* showed accuracy for J48 and Naïve Bayes at 99.49% and 92.72%, respectively. But none of them has followed any feature selection mechanism. Apart from that they analyzed their model with the KDD Cup 99 dataset, which is a predated version of current NSL KDD Dataset.

In another experiment Kim *et al.* [23] showed effective results using LSTM with different values for learning rate and hidden layer size with high detection rate and accuracy. They applied their experiment on 10% of the KDD Cup 99 training dataset and 10% of the KDD Cup 99 testing dataset. As we know the performance of the algorithm changed depending on the tuning of algorithms parameter values. Therefore appropriate learning rate and choice of number of hidden layers had a great impact on the classifier performance. Authors found that the detection rate and false alarm rate showed improvement precisely when the learning rate parameter was set to 0.01 and hidden layers size set to 80.

Miao *et al.* [24] have simplified LSTM based on their analysis of activation function and proposed two simplifications (1) deriving input gates from forget gates, and (2) removing recurrent inputs from output gates. Lyu *et al.* [25] also proposed another simplification for LSTM.

Moreover, Gers and Schmidhuber [26] introduced "peephole" connections which allows the gates to not only depend on the previously hidden state (S_t-1), but also on the previous internal state (C_t- 1).

One of the crucial observation provided by Greg *et al.* [27] as it introduced GRU as a light version of LSTM. The architecture addressed the complexity of LSTM by eliminating the "output gate", at each time step which writes the contents from its memory cell to the more substantial net.

A challenge has been addressed by Ahmed E. [28], which is to achieve a low false alarm rate with new unseen threats. Author tested the model with NSL-KDD Dataset. He built a model using different RNN models and used Bi-Directional RNN, LSTM, BLSTM to detect anomalies in sequence.

An auto encoder framework is proposed for both steady and variable length data Ali H. *et al.* [29] utilized LSTM for computer network intrusion detection. They used LSTM encoders such as GRU, and BLSTM through a comprehensive set of experiments. They developed an online sequential unsupervised dataset for network intrusion detection using LSTM auto encoders.

Thi-Thu-Huong Le *et al.* [30] built a classifier of IDS using LSTM with six optimizers: RMSprop, Adagrad, Adadelta, Adam, Adamax, and Nadam. They evaluated the performance of each optimizer using the KDD Cup'99 dataset on each attack type as follows: DoS, Probe, R2L, U2R and Normal. The main purpose of their experiment was to enhance the classification performance by increasing accuracy decreasing false alarm rate. Their experiment consists of two stages. The first stage determined hyper parameter values. The second stage applied LSTM with the six optimizers.

Bontemps *et al.* [31] present a collective anomaly detection model using LSTM in real-time network traffic. They demonstrate the efficient performance of the proposed model using the KDD Cup'99 dataset. The results showed the capability of the model to detect collective anomalies. However, they recommended that their model training data must be organized in a coherent manner to guarantee the stability of the system.

Current work by Benjamin *et al.* [32] has revealed that LSTM RNN can be applied to the problem of anomaly detection in computer network flow data. They applied a public dataset "ISCX IDS" for IDS taken from the University of New Brunswick's Canadian Institute for Cybersecurity (CIC) and the Information Security Centre of Excellence (ISCX). Their model can identify anomalous network traffic. Observed anomalies were the focus for training the models for prediction attacks. But no feature section method have employed by them.

Furthermore, SKIP-RNN is a newly proposed architecture by Campos *et al.* [33] in which they extend the existing LSTM model by learning to reduce the number of sequential operations and the effective size in the computational graph. Their experiment was based on developing SKIP-LSTM and SKIP-GRU on the MNIST dataset (a large database of handwritten digits developed by Modified National Institute of Standards and Technology database). All parameters were trained using backpropagation. Unfortunately the algorithm did not converge despite numerous attempts to train the model with different set values of its parameters.

In comparison to existing literature, this research offers insight into RNN architectures. Different feature selection techniques were compared, the best technique that fit intrusion detection was selected, and applied to a different deep learning model for intrusion detection. This resulted in presenting suitable parameter tuning to be changed while increasing the accuracy for this set of algorithm, which will be explained in detail in the results and analysis section.

2.1 Objectives with Specific Aims and Possible Outcome

This work focuses three general research fields' i.e. Machine Learning, Network Security, and Feature Selection. The objectives of this research are:

i. To detect flow-based traffic anomaly in an SDN Environment.
ii. To implement a deep learning algorithm for detecting network intrusion in the OpenFlow controller segment of SDN.
iii. To develop a deep learning model and train that model with NSL-KDD Dataset

As a result, this research will provide following expected outcomes

i. Both complete and reduced dataset (after using feature selection method) will be used for the performance analysis of our proposed model.
ii. Comparison will be made by using deep learning model for flow based anomaly detection in SDN.
iii. Through Experiments, the potential of deep learning approach will be observed in flow- based anomaly detection.

CHAPTER 3

BACKGROUND

3.1 Introduction

This chapter provides information on the background involved in developing this research. It introduces the concepts and technologies of Software Defined Networking (SDN) followed by its security and privacy concerns. Further, this chapter addresses the characteristics to be considered while developing security solutions for OpenFlow controller. This chapter continues to elucidate the concepts of network security and intrusion detection systems. Later, explanation of the network architecture is provided for the Long Short Term Memory (LSTM) and Gated Recurrent Unit (GRU) recurrent neural networks. The final sections of the chapter explain the importance of Machine Learning and Deep Learning for SDN environment by quoting their relevant applications. The main intention of this chapter is to provide a complete overview of the concepts and algorithms for the reader with minimal knowledge in this field of research.

3.2 Intrusion Detection Method

Intrusion detection is a key research area in network security. A common approach for intrusion detection is detecting anomalies in network traffic, however, network threats are evolving at an unprecedented rate. The difference between the evolution of threats and the current detection response time of a network leave the system vulnerable to attacks. Over the years, a number of intrusion detection techniques have been developed to detect network intrusions using various detection mechanism.

3.2.1 Signature based detection

Signature-based detection is the way to detect packets that have a signature in the network traffic corresponding to the rules established in the IDS [34]. Each rule has attributes and conditions about an attack. When the traffic from the network comes into the IDS, we must find some rules to match the provided data in the IDS. If matched rules were found in the traffic data, the IDS decides if the transmitted traffic contains an intrusion. In this respect, signature-based detection is similar to the operation of an anti-virus application. Because the actual IDS should be applied at the same time for at least several thousand rules for all traffic, we do not need to compare each rule, but use a decision tree or some kind of automated algorithm to speed up the process. Well-written signature rules can perform detection of known attacks with high probability, so misjudged results are very few. Because of these characteristics, signature based detection is used frequently in commercial usage, and an IPS in particular will use this detection to essentially perform the preventive reactions without mistakes.

3.2.2 Anomaly based detection

Anomaly-based detection will not find attacks using one-to-one correspondence, like signature based detection; this detection uses the tendency of the attack traffic to determine whether an attack is occurred or not. To operate this detection method, a prior learning process is required. First, security experts collect a great deal of general traffic and attack traffic, and generate an algorithm or heuristics based on statistics, artificial intelligence, or machine learning to judge each attack type [35]. When detecting attacks, the IDS tries to distinguish which pre-classified group is correct for the entered traffic. If a proper group for the traffic is found in pre-classified group, the IDS decides this traffic is a known attack. Otherwise the IDS decides the other traffic as an outlier, which means a new kind of attack. Anomaly-based detection can distinguish outlier traffic, so the IDS has the possibility of detecting new attacks if the detection algorithm is trained well.

However, because the learning outcomes of anomaly-based detection are represented by a sequence of numbers, most experts have difficulty seeing the working method. Consequently, even though a learned algorithm potentially has the critical problems, they cannot be found fast, or found when the IDS operates in a real environment in certain cases. Also, anomaly-based detection generally has a lower detection rate, compared to signature-based detection, and has a high false alarm rate. This result reduces reliability of the IDS, so flow-based detection is commonly used along with signature-based detection together.

3.2.3 Flow based detection

Flow-based detection is used to minimize network overhead when the IDS operates. In flow-based detection, flow is the basic unit between connections to be detected [36]. Even if connection time is long and the number of packets is large, they can be represented as one flow or a few flows. This is the reason flow-based detection requires far fewer network resources, compared to packet-based detection.

In flow-based detection, a network switch and router collect flow information from the network traffic, and send this information to the server during some intervals. Because switches and routers are installed throughout the entire network, not only on the boundary of the network, flow-based detection can detect insider attacks as well as outsider attacks. Therefore flow-based detection can be used in a university network, an industrial network and a city-wide network, in which all of the network members are not guaranteed harmless. A flow contains source internet protocol (IP) address, destination IP address, protocol, packets per flow, TCP flags (if possible), bytes per flow, and duration. A flow is not used with signature-based detection which requires many features, and is generally combined with anomaly-based detection.

3.2.4 Packet based detection

Packet-based detection (or payload-based detection) is a way to choose a data source from network traffic which requires the entire packet payload to detect an attack. Packet-based detection is mainly combined with signature-based detection, which requires a lot of features of the traffic, especially for operating an IPS. For example, Snort is a well-known IDS based on packet-based and signature-based detection. This detection method requires all packet payloads for detection. Therefore the IDS using packet-based detection is mainly installed near the gateway or root of a tree network structure, where almost all packets are transmitted.

In theory, packet-based detection can provide the highest detection rate due to detection target which implies all the information from the traffic. But detection hardware must be should have powerful devices to process several terabits of traffics. To address this limitation, NetFPGA [37] which processes packets at high speed, or a distributed IDS [38], can be used for a packet-based IDS.

3.3 What is Software Defined Networking?

Software Defined Networking (SDN) has come to prominence in recent years and demonstrates an enormous potential in shaping the future of networking by separating control plane from data plane. OpenFlow is the first and most widely used protocol that makes this separation possible in the first place. As a newly emerged technology, SDN has its inherent security threats that can be eliminated or at least mitigated by securing the OpenFlow controller that manages flow control in SDN.

3.3.1 SDN overview

Up to now, each network device has had limited settings for the network layer and for only the device itself. So if changing network policy is required, network operators must change settings of each devices. Because the equipment is not connected organically, the settings between devices can be confused within several modification of network configuration. A software-defined network [39] divides the existing network into the control layer, the infrastructure layer (data layer), and the application layer, as shown in Figure 3.1. Common routers and switches contain the link status and manage routing, forwarding the table itself. In an SDN, these devices have only link status and just transmit data, and devolve management functions to an SDN controller and network application, such as NAT, load balancer linked with an SDN controller. This separation allows the network administrator to easily change the entire network policy.

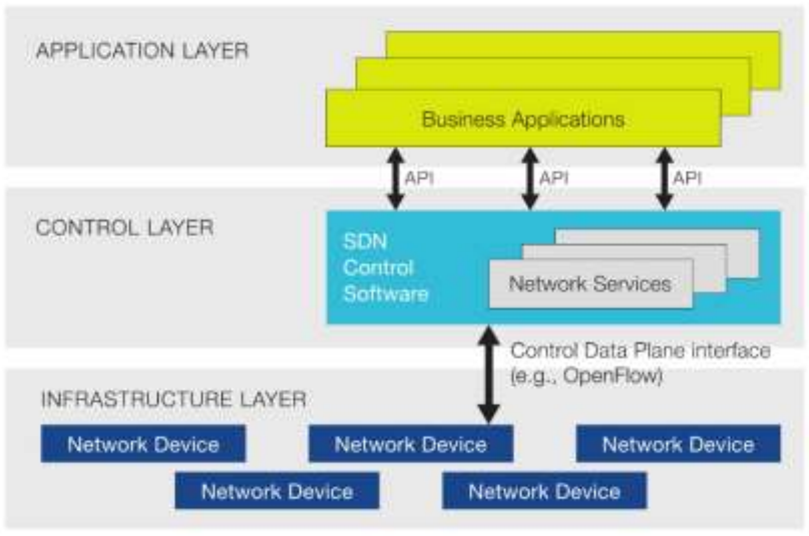


Figure 3.1: Layer representation in SDN [39]

3.3.2 OpenFlow

OpenFlow (OF) [39] is the most widely used protocol, which provides the functionality of an SDN. OpenFlow describes how the SDN controller and the OpenFlow switch (OF switch) communicate, and which message block is sent by them. In an OpenFlow network, OF devices (including OF controller and OF switches) have their own OF port number (DPID as SDN) and set destinations using the OF port number in the OpenFlow network. To communicate between controllers and not directly connect switches, they build a secure channel virtually using transport layer security (TLS) encryption.

In SDN and OpenFlow [40], forwarding and routing of packets are treated as follows. When a packet enters the OF switch, the switch will make sure a proper rule in the match table, and a rule contains match information as seen in Table 3.1. If a matched rule is found in the table, the OF switch does the action in the rule. If not, the OF switch requests the proper action by sending the packet to the controller. The OF controller forwards the packet to the OF application in the controller. If the controller receives the proper action for the packet from the application, then this provides a response rule and action to the OF switch to transmit the packet properly.

OpenFlow can do many things, not just receive packet information and send routing information, but also communicates with switches with a variety of information. For example, OpenFlow can receive port status, flow status, lookup table status by switch, and make new packets in the network. Therefore, various network application can be adopted in OpenFlow.

3.3.3 Flow based detection using SDN

OpenFlow uses the flow as a minimum unit of the routing table to reduce processing throughput. The flow status was created automatically without third-party tools to allow implementation of flow based IDS using Open Flow’s flow status. Information that is a response to a flow stats request message is shown in Table 3.1 and Table 3.2 respectively. Bold face attributes are the same as those provided in Labeled Data Set for Intrusion Detection [36]. Therefore, by using the flow stats request of the OpenFlow message, to perform flow-based detection without any limitations is feasible.

Table 3.1: Structure of Match Information

Attribute	Description
ingress port	Length of action entry
ether source	MAC address of source
ether dest	MAC address of the destination
VLAN id	VLAN id of flow
VLAN priority	VLAN priority of flow
IP src	IP address of source
IP dest	IP address of the destination
IP proto	IP protocol
IP ToS bits	Type of service of IPv4
source port	TCP, UDP source port and ICMP Type
dest port	TCP, UDP destination port and ICMP Code

Table 3.2: Structure of Flow Information

Attribute	Description
length	Length of action entry
table id	ID of match table flow came from
match	Match information of flow
duration_sec	Time flow has been alive in seconds duration in sec
duration_nsec	Time flow has been alive in seconds duration in nanosec
priority	Priority of the entry
idle timeout	Number of seconds idle before expiration hard timeout
hard timeout	Number of seconds before expiration
packet count	Number of packets in flow
byte count	Number of bytes in flow

3.4 Data Set for Intrusion Detection

As described above, selecting a data set is very important, because the data set is used to evaluate the performance of the IDS, or to make a pre-learned detection algorithm. However to make a useful data set, a well-designed data collection plan is required. Because of the difficulty in making a data set, using a public data set for intrusion detection is also a good idea in researching an intrusion detection system.

3.4.1 Public labeled dataset for an IDS

The Defense Advanced Research Projects Agency (DARPA) Data Set [41] is a data set for intrusion detection produced by the MIT Lincoln Laboratory at the request of DARPA. DARPA data sets were made in 1998, 1999, 2000, and each data set has a different purpose for detection. Regardless of the year, all DARPA data sets provide tcp dump for all the traffic, and expected attack types are in the data set. Among the data sets, the DARPA 1998 data set is most used for network intrusion detection systems because this dataset contains very huge network traffic and various attack types. The DARPA 1998 data set was made over eight weeks and has approximately 20GiB of network traffic. Also, this data set classified 27 attack types and contains about 15,000 IP addresses, including fake IPs for attacks.

The KDD99 Data Set [42] is a data set for the Knowledge Discovery and Data Mining Tools KDD Cup 99 competition which is based on the DARPA 1998 data set. This dataset extracted 41 features from the entire packet dump, and reclassified as 24 attack types from DARPA 1998 data set. By 2010, the DARPA 1998 and KDD99 data sets had been used most frequently for performance evaluation of IDS systems.

However, there are criticisms that the learned algorithms using the DARPA data set and the KDD99 data set are not proper for detecting attacks in real network environments [43]. When evaluating the DARPA 1998 data set using a commercial signature-based IDS, the IDS showed a lower detection rate, even if the IDS showed good performance on a real network. As a result of the analysis, the recorded TCP dump of attacks is quite different from general attack tools. To solve this problem, NSL-KDD [43], which removed and fixed the improper attack dump, was suggested. However, DARPA 1998 which is the basis of NSL-KDD is too old a data set, so NSL-KDD can not represent 'current' dataset.

3.4.2 NSL KDD dataset

Different statistical analyses discovered by researchers that prime drawbacks of KDD cup 99 dataset [44] has affected the detection accuracy of many network intrusion detection model. A refined version of KDD cup 99 dataset is NSL KDD [45] and it can be stated as predecessor of KDD cup 99. Following Table 3.3 presents a collection of downloadable files at the disposal for the researchers.

Table 3.3: List of NSL-KDD Dataset Files and their Description [46]

Serial No.	File Name	File Description
1	KDDTrain+.ARFF	The full NSL-KDD train set with binary labels in ARFF format
2	KDDTrain+.TXT	The full NSL-KDD train set including attack-type labels and difficulty level in CSV format
3	KDDTrain+_20Percent.ARFF	20% subset of the KDDTrain+.arff file
4	KDDTrain+_20Percent.TXT	20% subset of the KDDTrain+.txt file
5	KDDTest+.ARFF	The full NSL-KDD test set with binary labels in ARFF format
6	KDDTest+.TXT	The full NSL-KDD test set including attack-type labels and difficulty level in CSV format
7	KDDTest-21.ARFF	A subset of the KDDTest+.arff file which does not include records with difficulty level of 21 out of 21
8	KDDTest-21.TXT	A subset of the KDDTest+.txt file which does not include records with difficulty level of 21 out of 21

Following observation represents how NSL KDD dataset differs from traditional KDD cup 99 dataset

- In order to produce unbiased results from the classifier redundant records are removed
- In train and test dataset sufficient records is available, which is practically rational and enables us to perform the experiment on complete dataset.
- Comparing with the original KDD data set, NSL KDD dataset contains sufficient number of records which is inversely proportional for the selection of records from each difficulty level.

There are 41 attributes classifying different features of the flow in each record. A label is assigned to each either as an attack type or normal type. According to following Table 3.4, Table 3.5, Table 3.6, and Table 3.7 details of the attributes namely their attribute number, attribute name with description, and sample form of data. In following every table contain different types of features of each network connection. Table 3.4 contains all the basic features of the dataset whereas content, time and host based features of each network connection enlisted in Table 3.5, Table 3.6, and Table 3.7 respectively. The last attribute position as 42nd contains data about 5 classes in which they are categorized as one normal class and four attack class. In general this 5 classes are considered as network connection vector. Among the 4 attack classes theses can be further grouped as 4 attack pattern namely DoS, Probe, R2L and U2R. The description of the attack classes are described in following section.

Table 3.4: Basic features of each network connection [46]

Attribute No.	**Attribute Name**	**Description**	**Sample Data**
1	Duration	Length of time duration of the connection	0
2	Protocol_type	Protocol used in the connection	Tcp
3	Service	Destination network service used	ftp_data
4	Flag	Status of the connection: Normal or Error	SF
5	Src_bytes	Number of data bytes transferred from source to destination in single connection	491
6	Dst_bytes	Number of data bytes Transferred from destination to source in single connection	0
7	Land	if source and destination IP addresses and port numbers are equal then, this variable takes value 1 else 0	0
8	Wrong_fragment	Total number of wrong fragments in this connection	0
9	Urgent	Number of urgent packets in this connection. Urgent packets are packets with the urgent bit activated	0

Table 3.5: Content related traffic features of each network connection [46]

Attribute No.	Attribute Name	Description	Sample Data
10	Hot	Number of hot indicators in the content such as: entering a system directory ,creating and executing program	0
11	Num_failed _logins	Count of failed login attempts	0
12	Logged_in	Login Status : 1 if successfully logged in; 0 otherwise	0
13	Num_compromised	Number of ``compromised' ' conditions	0
14	Root_shell	1 if root shell is obtained; 0 otherwise	0
15	Su_attempted	1 if ``su root" command attempted or used; 0 otherwise	0
16	Num_root	Number of ``root" accesses or number of operations performed as a root in the connection	0
17	Num_file_creations	Number of file creation operations in the connection	0
18	Num_shells	Number of shell prompts	0
19	Num_access_files	Number of operations on access control files	0
20	Num_outbound_cmds	Number of outbound commands in an ftp session	0
21	Is_hot_login	1 if the login belongs to the ``hot" list i.e., root or admin; else 0	0
22	Is_guest_login	1 if the login is a `guest" login; otherwise 0	0

Table 3.6: Time related traffic features of each network connection [46]

Attribute No.	Attribute Name	Description	Sample Data
23	Count	Number of connections to the same destination host as the current connection in the past two	2
24	Srv_count	Number of connections to the same service (port number) as the current connection in the past two seconds	2
25	Serror_rate	The percentage of connections that have activated the flag (4) s0, s1, s2 or s3, among the connections aggregated in count (23)	0
26	Srv_serror_rate	The percentage of connections that have activated the flag (4) s0, s1, s2 or s3, among the connections aggregated in srv_count (24)	0
27	Rerror_rate	The percentage of connections that have activated the flag (4) REJ, among the connections aggregated in count (23)	0
28	Srv_rerror_rate	The percentage of connections that have activated the flag (4) REJ, among the connections aggregated in srv_count (24)	0
29	Same_srv_rate	The percentage of connections that were to the same service, among the connections aggregated in count (23)	1
30	Diff_srv_rate	The percentage of connections that were to different services, among the connections aggregated in count (23)	0

Table 3.7: Host related traffic features of each network connection [46]

Attribute No.	Attribute Number	Description	Sample Data
32	Dst_host_count	Number of connections having the same destination host IP address	150
33	Dst_host_srv_count	Number of connections having the same port number	25
34	Dst_host_same_srv_rate	The percentage of connections that were to the same service, among the connections aggregated in dst_host_count (32)	0.17
35	Dst_host_diff_srv_rate	The percentage of connections that were to different services, among the connections aggregated in dst_host_count (32)	0.03
36	Dst_host_same_src_port_rate	The percentage of connections that were to the same source port, among the connections aggregated in dst_host_srv_count (33)	0.17
37	Dst_host_srv_diff_host_rate	The percentage of connections that were to different destination machines, among the connections aggregated in dst_host_srv_c	0
38	Dst_host_serror_rate	The percentage of connections that have activated the flag (4) s0, s1, s2 or s3, among the connections aggregated in dst_host_count (32)	0

Table 3.8: Host related traffic features of each network connection [46] (continued)

39	Dst_host_srv_serror_rate	The percent of connections that have activated the flag (4) s0, s1, s2 or s3, among the connections aggregated in dst_host_srv_count (33)	0
40	Dst_host_rerror_rate	The percentage of connections that have activated the flag (4) REJ, among the connections aggregated in dst_host_count (32)	0.05
41	Dst_host_srv_rerror_rate	The percentage of connections that have activated the flag (4) REJ, among the connections aggregated in dst_host_srv_count (33)	0

In the NSL-KDD dataset the following attack classes are present and they can be grouped as four categories [42], [44]:

DOS: DOS stands for Denial of service, is an attack category, which depletes the victim's resources. By depleting victims resources it stops the flow of handling legitimate requests- e.g. syn flooding. "source bytes" and "percentage of packets with errors" are treated as relevant features.

Probing: The prime objective of this attack type is to gain information about the remote victim- e.g. port scanning. Relevant features: "duration of connection" and "source bytes"

U2R: In user to root attack type an attacker uses a normal account to login into a victim system and tries to gain administrator privileges by exploiting some vulnerability in the victim- e.g. buffer overflow attacks. Relevant features: "number of file creations" and "number of shell prompts invoked,"

R2L: In remote to local attack type attacker gets unauthorized access from a remote machine as well as intrudes into a remote machine and gains local access of the victim machine- e.g. password guessing. Relevant features: Network level features – “duration of connection” and “service requested” and host level features - “number of failed login attempts”

Table 3.9: Attribute value type of NSL KDD dataset [46]

Type	**Features**
Nominal	Protocol_type(2), Service(3), Flag(4)
Binary	Land(7), logged_in(12), root_shell(14), su_attempted(15), is_host_login(21),is_guest_login(22)
Numeric	Duration(1), src_bytes(5), dst_bytes(6), wrong_fragment(8), urgent(9), hot(10), num_failed_logins(11), num_compromised(13), num_root(16), num_file_creations(17), num_shells(18), num_access_files(19), num_outbound_cmds(20), count(23) srv_count(24), serror_rate(25), srv_serror_rate(26), rerror_rate(27), srv_rerror_rate(28), same_srv_rate(29) diff_srv_rate(30), srv_diff_host_rate(31), dst_host_count(32), dst_host_srv_count(33), dst_host_same_srv_rate(34), dst_host_diff_srv_rate(35), dst_host_same_src_port_rate(36), dst_host_srv_diff_host_rate(37), dst_host_serror_rate(38), dst_host_srv_serror_rate(39), dst_host_rerror_rate(40), dst_host_srv_rerror_rate(41)

Table 3.10: Different attacks types in NSL KDD dataset [46]

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

3.5 Machine Learning and Deep Learning

Machine Learning is a branch of artificial intelligence that deals with the analysis and construction of systems from the knowledge gained from the data [47]. The wide range of applications of machine learning includes regression, classification, and prediction and so on. It is categorized mainly into three types based on the use of labeled data– a) Supervised learning, b) Unsupervised learning, and c) Semi-supervised learning. The commonly used algorithms in machine learning include linear regression, Naive-Bayes classifier, logistic regression, support vector machines, artificial neural networks and so on.

Deep Learning is a complex version of machine learning with multiple levels of abstraction of data at multiple processing layers [48]. Deep Learning can learn the intricate structures in the dataset through backpropagation and indicates how machine changes the internal parameters at each layer. The frequently used deep learning algorithms include deep belief networks, auto encoders, convolutional neural networks and recurrent neural networks.

3.5.1 Why is deep learning better than machine learning?

Deep learning, which is also known as hierarchical learning or deep structural learning, is a broader version of machine learning in terms of complexity in the structure and learning data representations. The key difference between machine learning and deep learning is the change in the performance as the scale of the data increases. Deep Learning algorithms require a larger amount of data to find the patterns in the network where machine learning requires the less data.

Artificial Neural Networks (ANN) that contain one or more hidden layers will make the structure deep and the data is processed at each layer, thus, making the learning task deeper. The commonly used deep learning architectures include deep belief networks (DBN), deep neural networks, (DNN) and recurrent neural networks (RNN), which are applied to research fields such as natural language processing, speech recognition, computer vision, audio recognition, machine translation and social network filtering. Deep Learning can also be applied to network intrusion data research where the data is heterogeneous and multi-modal. Traditional machine learning algorithms fail to deliver long term results for SDN based devices which are usually connected for longer time-periods.

3.5.2 Recurrent neural network (RNN)

Recurrent neural networks (RNN) in deep learning have the capability to learn from the previous time-steps and can be used with less human intervention. In RNN, the output of each node in the hidden layer is given as input to the same node at each time-step. The useful information is stored in the memory and can be used for learning purposes in future time steps. The structural difference between an RNN and Feed forward neural network (FNN) can be observed from Figure 3.2 and Figure 3.3 respectively.

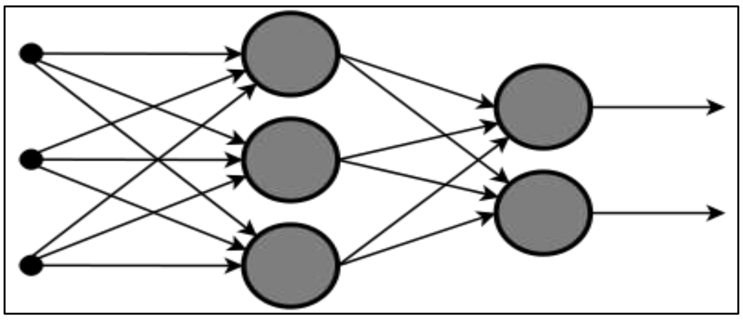


Figure 3.2: Structure of Feed Forward Neural Network

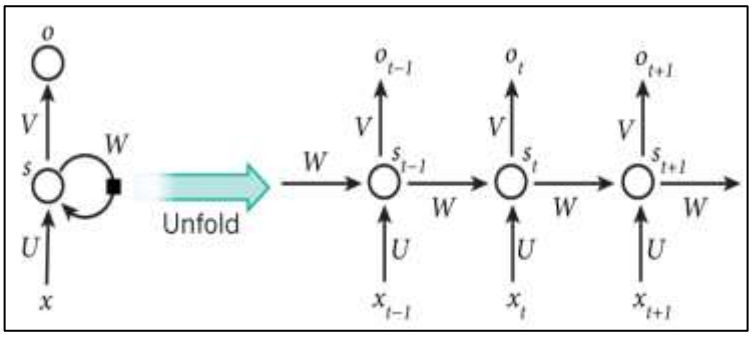


Figure 3.3: Structure of Recurrent Neural Network

3.5.3 Long short term memory (LSTM)

Recurrent Neural Networks (RNN), when trained in real-time learn from previous time steps by backpropagation through time (BPTT). A deep neural network is unfolded in time and constructs an FNN for every time-step. Then, the gradient rule updates the weights and biases for each hidden layer, thus, minimizing the loss between the expected and actual outputs. However, standard RNNs cannot perform better when the time-steps are more than 5-10. The prolonged back-propagation leads to vanishing or blow up of error signals, leading to oscillating weights, which makes the network performance poor. To overcome this vanishing gradient problem, researchers came up with the Long-Short-Term-Memory (LSTM) network which bridges the minimal time gaps. LSTM makes use of a gating mechanism to handle long-term dependencies. The LSTM structure can be seen in Figure 3.4 LSTM has a cell state which is passed to every-time step. A gating mechanism is used to optimize the information that is passing through. It contains a sigmoid function layer which outputs between one and zero. A value of one means “pass all the information through”, whereas the value of zero means “do not pass any information through”. The "forget gate" decides the information that needs to be let through between the current input and previous cell state output using the sigmoid function. The "input gate" decides what information is required to store in the cell state. This gate contains two functions - "sigmoid" to decide what values need to be updated, and the “tanh” function to create a new vector of values that are to be added to the cell state. The “output gate” decides on what information from the cell state is required to output with the help of a sigmoid function. The output information is passed through the tanh function before passing through sigmoid to make sure that the values are between -1 and +1.

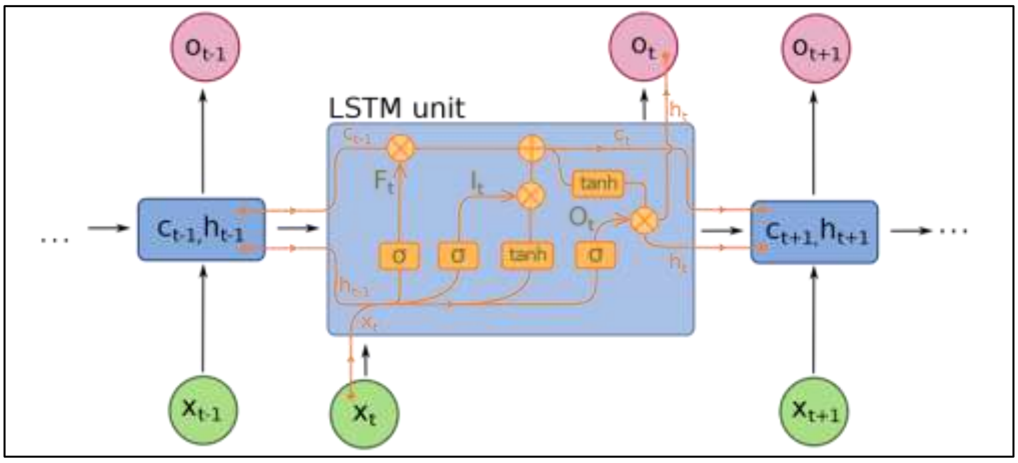


Figure 3.4: Basic structure of Long-Short-Term Memory RNN [49]

3.5.4 Gated recurrent unit (GRU)

A Gated Recurrent Unit (GRU) is a lighter version of an LSTM where the complexity in the structure is reduced by decreasing the gates in the architecture. The GRU merges both the “forget gate” and “input gate” in an LSTM to an “update gate” and combines the hidden state and cell state, resulting in a simpler architecture of the network as shown below in Figure 3.5.

GRU is a variant of LSTM which was introduced by K.Cho [22], [13]. GRU is basically an LSTM without an output gate, which therefore fully writes the contents from its memory cell to the larger net at each time-step. Its internal structure is simpler and therefore considered faster to train as there are fewer computations needed to make updates to its hidden state. GRU has two gates: reset gate *r*, and update gate *z*. Intuitively, the reset gate determines how to combine the new input with the previous memory cell, and the update gate defines how much of the previous memory cell to keep.

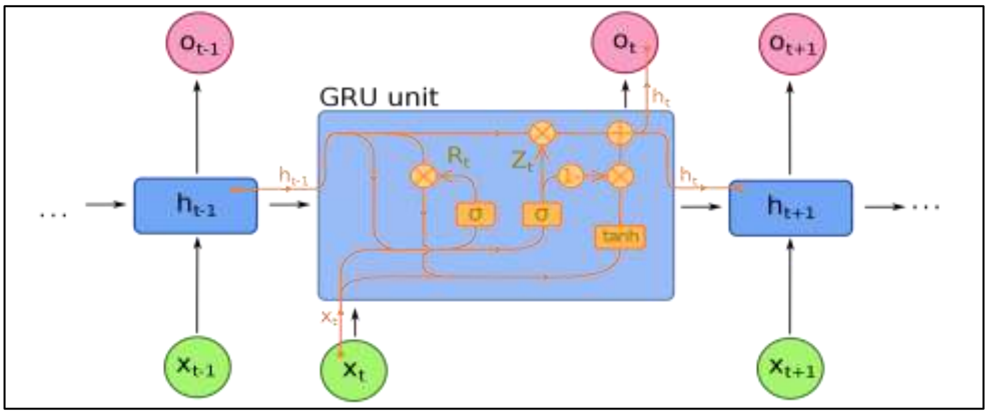


Figure 3.5: Structure of a Single layer Gated Recurrent Unit [49]

3.5.5 Multi-layer GRU RNN

Deep recurrent neural networks can have various architectures which influence greatly the performance of the algorithm. One can add many layers of RNN (plain RNN, LSTM or GRU) cells and stack the network into a deep structure called a Multi-Layer RNN as shown in Figure 3.6. This technique has a wide range of applications in speech recognition systems and weather forecast systems with high dimensional data. When GRU cells are used in each hidden layer of a recurrent neural network it is said to be a Multi-Layer GRU. In the multilayer structure, the input of the network is passed through multiple GRU layers apart from back propagation through the time. It has been proven that multi-layered RNNs learn from the different time lengths of input sequences [48]. Another key important feature of multi-layered RNNs is that they share the hyper parameters, weights, and biases across the layers, thus achieving optimized performance.

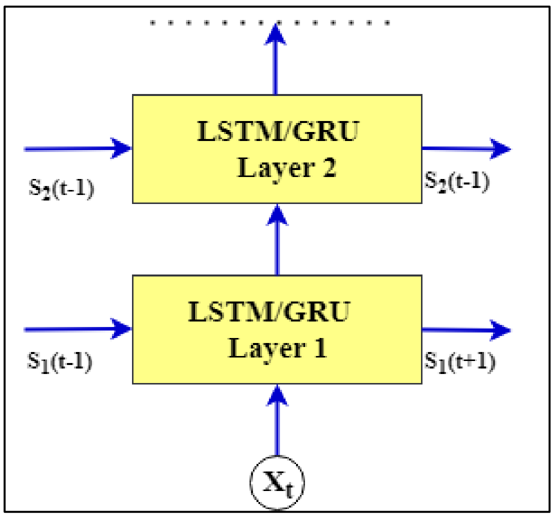


Figure 3.6: Structural representation of Multi-layer GRU [50]

3.6 Hyper Parameters

The hyper parameters used in the design of the recurrent neural network have a great impact on the performance of the network [51]. Although there are many hyper-parameters involved in the design of a recurrent neural network, the parameters having the largest impact on the performance of the network are learning rate, number of hidden layers, number of cells in the hidden layer and the number of time-steps.

Learning rate: It is a measure of the rate at which the network optimizes the minimization of the loss function in a neural network. Mathematically, if the loss function is L (X; W, b), then the goal of the network is to minimize the loss (cost) function L. The weights are constantly updated to achieve the best possible output reducing the loss value. The learning rate determines how fast the parameters are updated. One must vary the learning rate during the training of the neural network to obtain the best results.

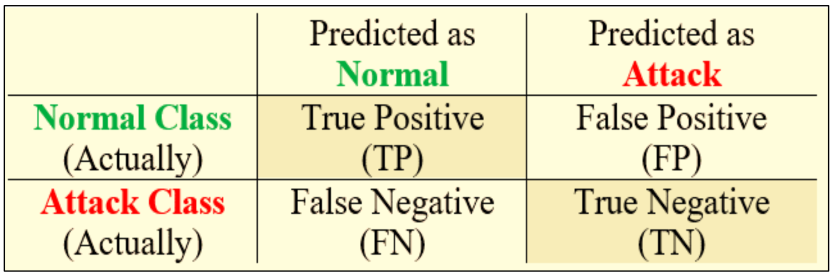
Time steps: Selecting the number of time-steps also plays a crucial role in the Performance of the system. The information required to find the correct patterns depends on the number of time-steps that are required to back propagate. Tuning the number of time-steps improves the output of the network. When more time-steps are selected, the network takes longer to time to train and vice-versa.

Hidden units: The number of cells in a hidden layer determines the amount of computation performed on the input data [52]. The more hidden units in the network, the longer it takes to train. The neural network should be trained for a various numbers of hidden units to verify the performance of the system.

Hidden layers: The stacking of GRU layers as discussed in the above multilayer GRU section, has a great impact on higher dimensional datasets. However, most deep neural networks obtain optimized performance with a single hidden layer [52]. One must decide on the number of hidden layers to be used with respect to their data-set size and the dimensions.

3.7 Evaluation Metrics

A good intrusion detection scheme entails high rate of accuracy and high detection rate with a very low false alarm rate. The relation between false alarm rates with misclassification rate that they are directly proportional to each other. The following metrics are used for evaluating a model. A brief discussion and calculating formula are showing below for each metric. In general, the confusion matrix visualizes the performance of the algorithm in a tabular form as shown in the Figure 3.7 below.



	Predicted as **Normal**	Predicted as **Attack**
Normal Class (Actually)	True Positive (TP)	False Positive (FP)
Attack Class (Actually)	False Negative (FN)	True Negative (TN)

Figure 3.7: Tabular form of confusion matrix

Where,

- True Positive (TP) is the total number of samples predicted as "normal" while they were "normal".
- False Negative (FN) is the total number of samples predicted "normal" while they were "attack".

- False Positive (FP) is the total number of samples predicted “attack” while they were “normal”.
- True Negative (TN) is the total number of samples predicted “attack” while they were “attack”.

All other important metrics such as Precision, Accuracy, Recall, and False Alarm Rate (FAR) with F1 score can be calculated using these 4 measures taken from the confusion matrix and their formula can be established as follows.

Accuracy (AC): shows the proportion of the classification from overall N Examples that were correct.

$$\boldsymbol{AC = \frac{TP + TN}{TP + FP + TN + FN}}$$

Precision (P): shows the proportion of intrusion anticipated by a Network intrusion detection systems are a real intrusion. As the value of P is higher, then the probability of lower false alarm rate is:

$$\boldsymbol{P = \frac{TP}{TP + FP}}$$

Recall (R): shows the proportion of positive examples that were correctly classified. We are in search of a high value of R

$$\boldsymbol{R = \frac{TP}{TP + FN}}$$

F-measure (F): By conveying balance between accuracy and recall, it gives a better measure of accuracy. We are in quest of a high F-measure value.

$$\mathrm{F} = \frac{2}{\frac{1}{\mathrm{P}}+\frac{1}{\mathrm{R}}}$$

CHAPTER 4

METHODOLOGY AND IMPLEMENTATION

4.1 Introduction

This chapter provides information on the methodology and our proposed architecture involved in developing this research. It introduces the concepts and technologies of various deep learning methods and feature section algorithm with the code segment of our developed intrusion detection model. Finally, we conclude this chapter by providing our SDN based flow-based anomaly detection architecture.

4.2 Proposed Architecture

Domain issues and challenges were identified with regards to intrusion detection through literature reviews. The most often used algorithms and experiments conducted were identified as well as the accuracy rate for each. As a result, it became evident that RNN architectures are new techniques in the intrusion detection domain. Proof of the concept of using LSTM and GRU algorithms in the field of intrusion detection is scarce due to lack of intensive experiments. None of the literature consulted demonstrated the best architecture or were compared among the algorithms in terms of their parameters to achieve the best performance with high accuracy. There are some challenges facing IDS which make it difficult to achieve that goal. Classification of data and labeling of unlabeled data seems to be a challenging task, as the current high volume of network traffic increases the number of attacks.

A module for each selected architecture was therefore developed. Feature selection was then implemented to ensure the best representation of all the data and better represent the underlying problem to each prediction model, resulting in improved model accuracy on unseen data. In this research, two phases were demonstrated for the experiment. One can be described as the features selection phase, using the two different selection mechanisms, Analysis of variance test (ANOVA F-test) and Recursive Feature Elimination (RFE). The second phase is to attempt to evaluate the selected algorithms on a full NSL KDD dataset which is a benchmark intrusion detection dataset. A baseline was created with initial values for each parameter, based on literature review. Different values were used with each run to fine tune the parameters and to identify the suitable ones that showed best prediction accuracy. Research methodology illustrated in following Figure 4.1.

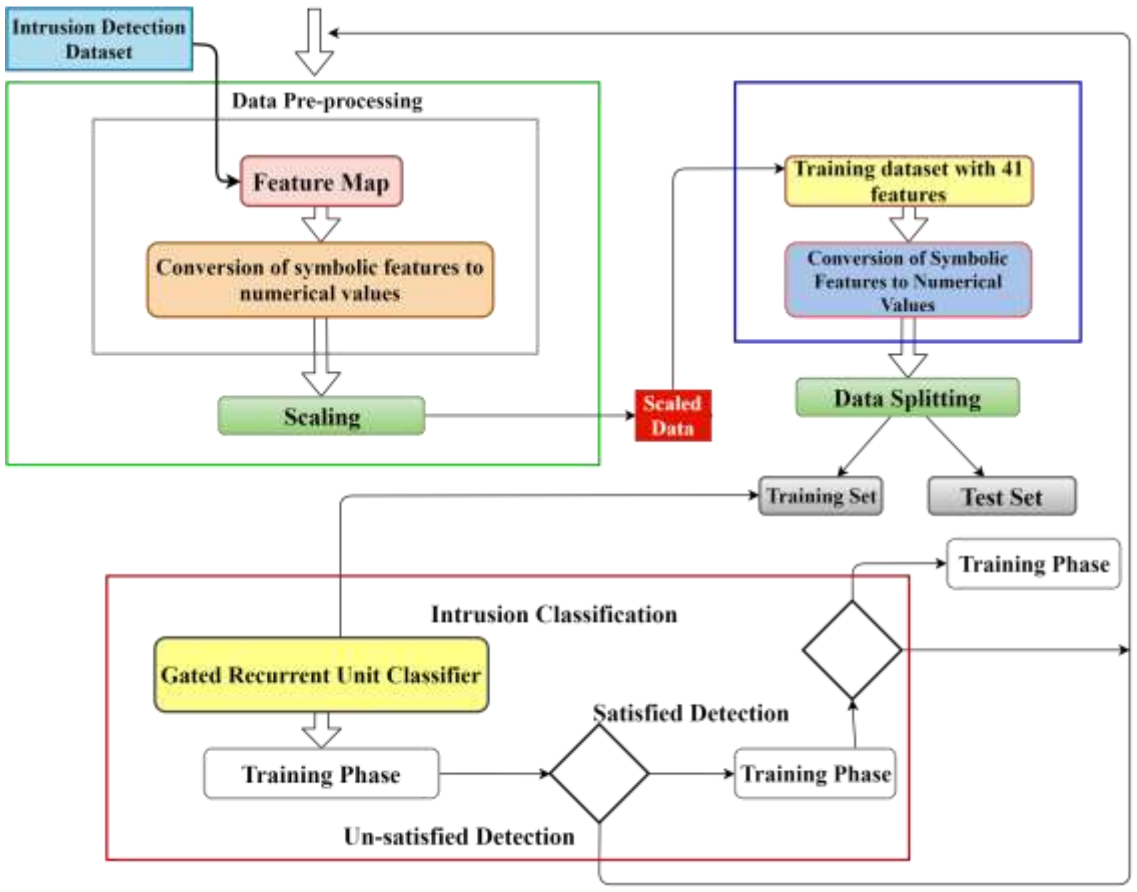

Intrusion Detection Dataset
Data Pre-processing
Feature Map
Conversion of symbolic features to numerical values
Scaling
Scaled Data
Training dataset with 41 features
Conversion of Symbolic Features to Numerical Values
Data Splitting
Training Set
Test Set
Training Phase
Intrusion Classification
Gated Recurrent Unit Classifier
Training Phase
Satisfied Detection
Training Phase
Un-satisfied Detection

Figure 4.1: Proposed research methodology for Network Intrusion Detection

4.3 Recurrent Neural Network (RNN)

In deep learning, recurrent neural network has the capability to learn from previous time-steps. Basically, a Recurrent Neural Network (RNN) is an extension of traditional Feed-forward neural network (FNN). A Simple structure of RNN can be seen in Figure 4.2. The RNNs are called recurrent since the output of each node in the hidden layer is given as input of the same node at each time-step.

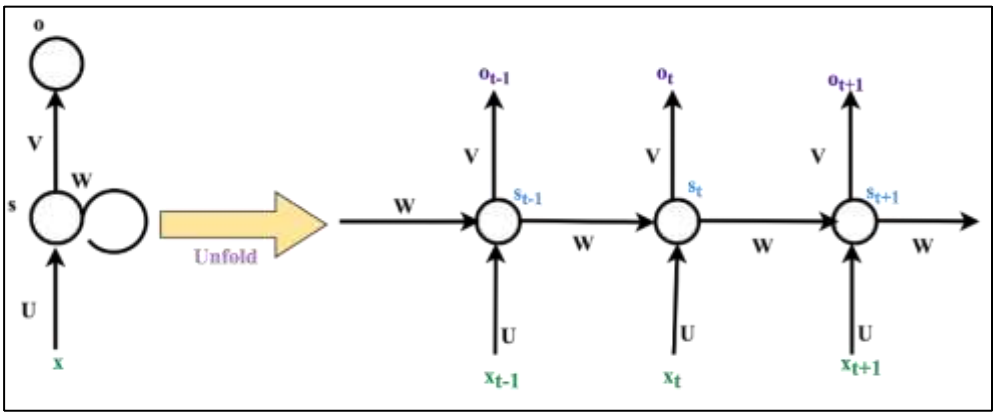


Figure 4.2: Working procedure of Recurrent Neural Network

From the above figure, x_t and o_t are input and output respectively. s_t Considered as hidden state. Nonlinear function like $tanh$ or $ReLU$ are indicated by f. Based previous hidden state and the input at the current step s_t is calculated: $s_t = f(Ux_t + Ws_{t-1})$.. s_{-1}, which is required to calculate the initial hidden state, by default which is initialized to zeroes. Computational formula of RNN hidden sates is

$$s_t = f(Ux_t + Ws_{t-1}), \quad \text{for } t = T, \dots, 1, \tag{1}$$

Backpropagation through Time (BPTT) algorithm is usually required to train the RNN. However, the simplest RNN Model has a major drawback, called vanishing gradient problems as it prevents it from being accurate. That is why more powerful combined models like Long short-term memory (LSTM) and Gated Recurrent Units (GRUs) were suggested to evolve above mentioned issue.

4.4 Long Short Term Memory (LSTM)

A deep neural network is unfolded in time and constructs an FNN for every time-step. Then, the gradient rule updates the weights and biases for each hidden layer, thus, minimizing the loss between the expected and actual outputs. However, standard RNNs cannot perform better when the time-steps are more than 5-10. The prolonged back-propagation leads to vanishing or blow-up of error signals, leading to oscillating weights, which makes the network performance poor. To overcome this vanishing gradient problem, researchers came up with the Long-Short-Term-Memory (LSTM) network, which bridges the minimal time gaps. LSTM makes use of a gating mechanism to handle long-term dependencies. The LSTM structure can be seen in Figure 4.3.

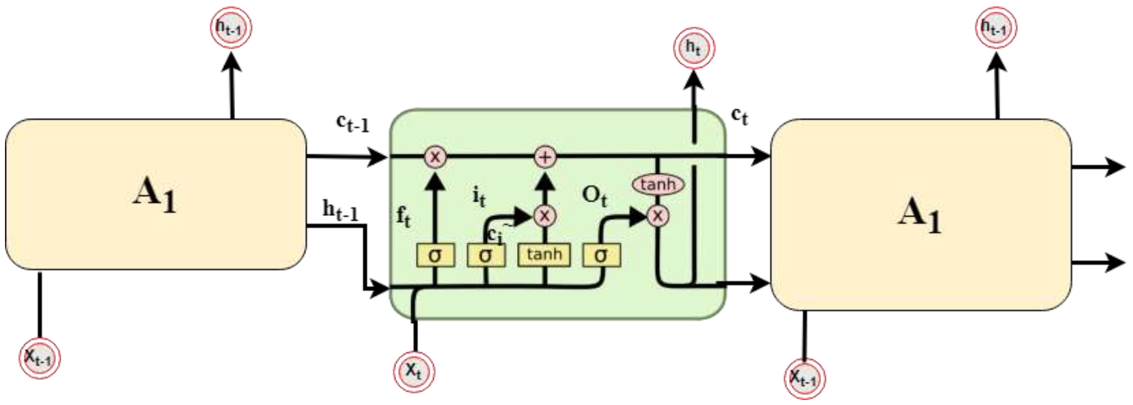


Figure 4.3: Working procedure of Long-Short-Term Memory

4.5 Gated Recurrent Unit (GRU)

A Gated Recurrent Unit (GRU) is considered to be a lighter version of an LSTM where the complexity in the structure is reduced by decreasing the gates in the architecture. To resolve the vanishing gradient problem of a standard RNN, both update gate and reset gate are used by GRU. Basically, these are two vectors which decides what kind of information should be passed to the output. Because of their smooth and faster training phase in comparison with LSTM, GRU has been selected for our model development. The GRU merges both the “forget gate” and “input gate” in an LSTM to an “update gate” and combines the hidden state and cell state, resulting in a simpler architecture as shown in Fig 4.4.

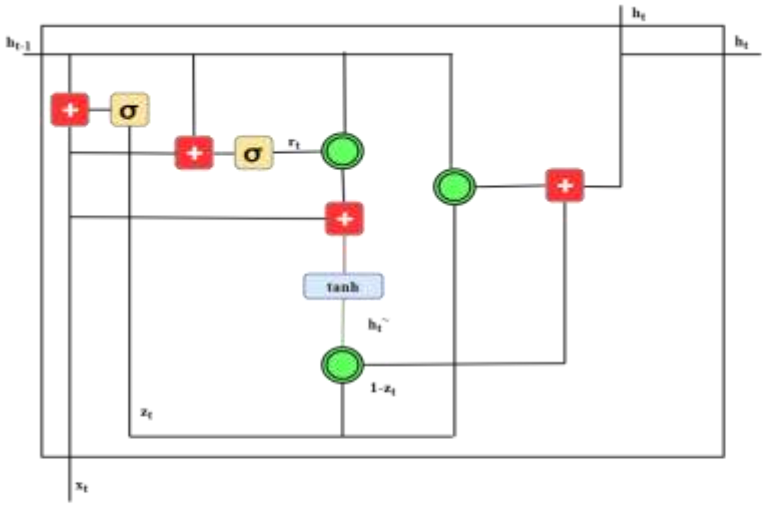


Figure 4.4: Architecture of single layer Gated Recurrent Unit

The following relationship can be obtained from Figure 4.4.

Update gate $z_t = \sigma\left(W^{(z)}x_t + U^{(z)}h_{t-1}\right)$, (2)

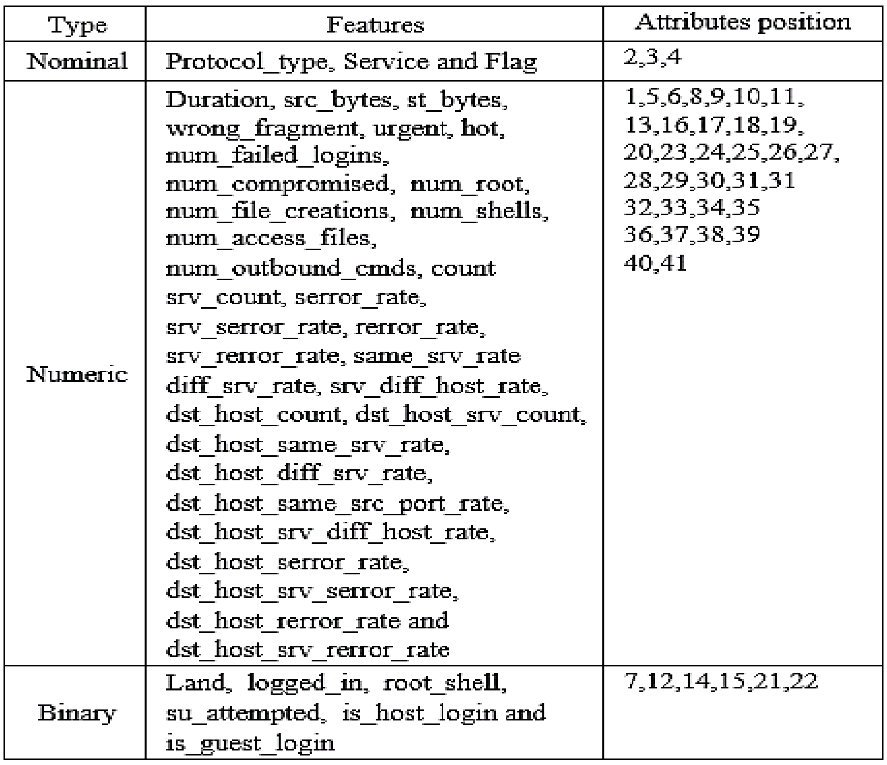
Candidate activation $\widetilde{h_t} = tanh(Wx_t + r_t \odot Uh_{t-1})$, (3)

Reset gate $r_t = \sigma(W^{(r)}x_t + U^{(r)}h_{t-1}$, (4)

Activation function $h_t = z_t \odot h_{t-1} + (1 - z_t) \odot h_t'$, (5)

4.6 Dataset

For network intrusion detection systems evaluation NSL-KDD is one of the cutting-edge benchmark datasets [42]. A total of 41 features contained by NSL-KDD which are categorized according to different pattern of features. Four main files of two-train (KDDTrain+.TXT, KDDTrain+_20percent.TXT) dataset and two test dataset (KDDTest+.TXT, KDDTest-21.TXT) are contained by NSL-KDD. In this research, full dataset of train and test are used for training and testing our Intrusion Detection System. Number of record contained by KDDTrain+ dataset and KDDTest+ dataset 126,620 and 22,850 respectively. Figure 4.5. Shows complete description of NSL-KDD dataset with different data pattern and attribute position.



Type	Features	Attributes position
Nominal	Protocol_type, Service and Flag	2,3,4
Numeric	Duration, src_bytes, st_bytes, wrong_fragment, urgent, hot, num_failed_logins, num_compromised, num_root, num_file_creations, num_shells, num_access_files, num_outbound_cmds, count srv_count, serror_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 and dst_host_srv_rerror_rate	1,5,6,8,9,10,11, 13,16,17,18,19, 20,23,24,25,26,27, 28,29,30,31,31 32,33,34,35 36,37,38,39 40,41
Binary	Land, logged_in, root_shell, su_attempted, is_host_login and is_guest_login	7,12,14,15,21,22

Figure 4.5: Description of NSL-KDD dataset with features and type

4.7 Description of Scikit-Learn

During experiment a python based machine learning library namely scikit-learn was used [53]. As, Data of most machine learning algorithm needs to be stored in 2D array or in a matrix shape, scikit-learn is capable of storing these 2D shaped data effectively. Following matrix shows the scikit-learn data representation. Here, there are N samples and D features.

$$feature\ matrix: x = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1D} \\ x_{21} & x_{22} & \cdots & x_{2D} \\ x_{31} & x_{32} & \cdots & x_{3D} \\ \vdots & \vdots & \ddots & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ x_{N1} & x_{N2} & \cdots & x_{ND} \end{bmatrix}$$

$$label\ vector: y = [y_1, y_2, y_3, \dots, y_N]$$

4.8 ANOVA F-test and RFE

To eliminate the irrelevant and redundant data form the dataset it is necessary to perform a feature selection mechanism. According to Reference [53] feature selection is a method of selecting a subset of relevant feature without small decay of presentation. The existence of extraneous attributes in the intrusion dataset often deterred the detection of accuracy. Number of causes were analyzed why it would be obligatory to restrict the features. Irrelevant features increased the computation time without classifier improvement and sometimes suggest correlation between feature and desire class, which arises by chance. In our experiment, we have used a Univariate feature selection with ANOVA F-test. This analyzes each feature individually to determine the strength of the relationship between the feature and labels. *SelectPercentile* method (sklearn.feature_selection) is used to select features based on percentile of the highest scores. When this subset is found: Recursive Feature Elimination (RFE) is applied. RFE is a method which frequently build a model where features are kept aside and reiterating the process until all features in dataset are removed. The basic idea is, to develop a feature ranking it uses the weight of a classifier. Following table represents the selected features after applying both ANOVA F-test and RFE.

4.9 SDN Based Flow Based Anomaly Detection Architecture

The SDN based flow based anomaly detection model is described in Figure 4.6 with three prime components: Flow Collector, Anomaly Detector, and Anomaly Mitigator. This segment focuses on the use of SDN model as a network infrastructure for the Intrusion Detection System (IDS). Finally IDS is implemented as an application on the SDN controller.

Flow collector: Flow collector is considered as one of the prime module of this architecture where this module is triggered by a timer method to aggregate all the flow statistics such as protocol, source and destination IP and source and destination port. This all features then sent to Anomaly Detector Module.

Anomaly detector: According to our proposed methodology Anomaly Detector module loads a GRU-LSTM trained model, receives the network statistics and take decision if a flow is anomaly or normal.

Anomaly mitigator: Based on the results of anomaly detector Anomaly Mitigator can take decision on the packet flow by dropping or forwarding the flow.

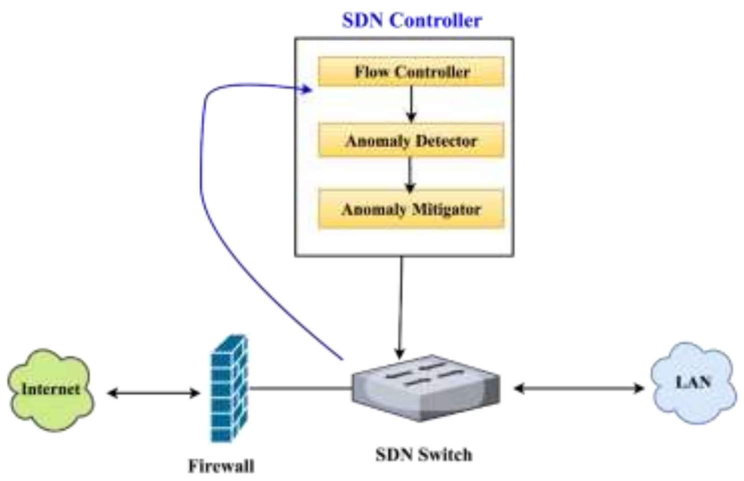


Figure 4.6: SDN controller based IDS architecture [19]

For requesting network data, on OpenFlow stats request message will be sent from the controller to all OpenFlow switches. As controller request for all the available statistics, an OpenFlow stats reply message with all available data send back to the controller by OpenFlow switch. Figure 4.7 describes the architecture of how OpenFlow switch handles the incoming packet and responds according to the availability of data in Flow table by using OF protocol.

The centralized controller of SDN can take Opportunities of the complete network to evaluate and associate feedback from the network. For analyzing and detecting any real time network intrusion will then be sent to Intrusion detection segment according to Figure 4.7 OF protocol can effectually alleviate an intrusion via flow table adjustment if once a network anomaly is discovered and recognized.

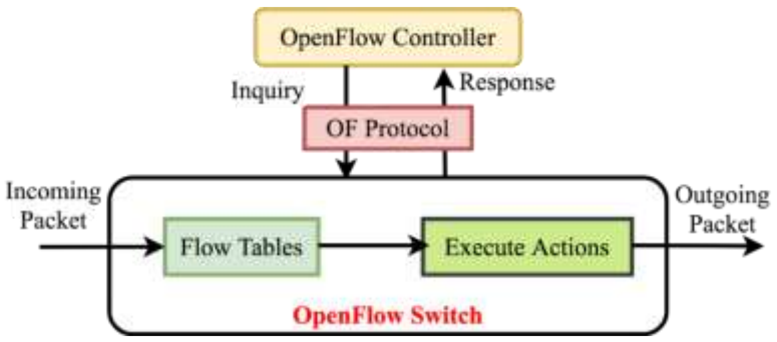


Figure 4.7: Working procedure of OpenFlow Protocol in SDN

SDN controller is capable of monitoring all the OpenFlow switches and send the request to all network data whenever it is necessary. Therefore, our proposed and designed intrusion detection segment is implemented in SDN controller, which is depicted in Figure 4.8. Following Algorithm 1 encapsulates our suggested approach.

Algorithm 1 Deep Learning based anomaly class detector for SDN attacks

```
1: procedure FLOW BASED ANOMALY DETECTOR BASED ON DEEP MODEL
2:  Selection of appropriate Deep learning algorithm
3: Train the Deep Neural Network (DNN)-based model using benchmark dataset NSL-KDD
4: Nomination of appropriate feature after using different feature selection methods
5: if The trained model predicts an anomaly class on a OpenFlow Controller by the DNN based Intrusion Detection Model
    then
6: Update the SDN OpenFlow controller rules to block that class attack type
7: Else, Allow the normal class to pass through SDN controller and access the available resources.
```

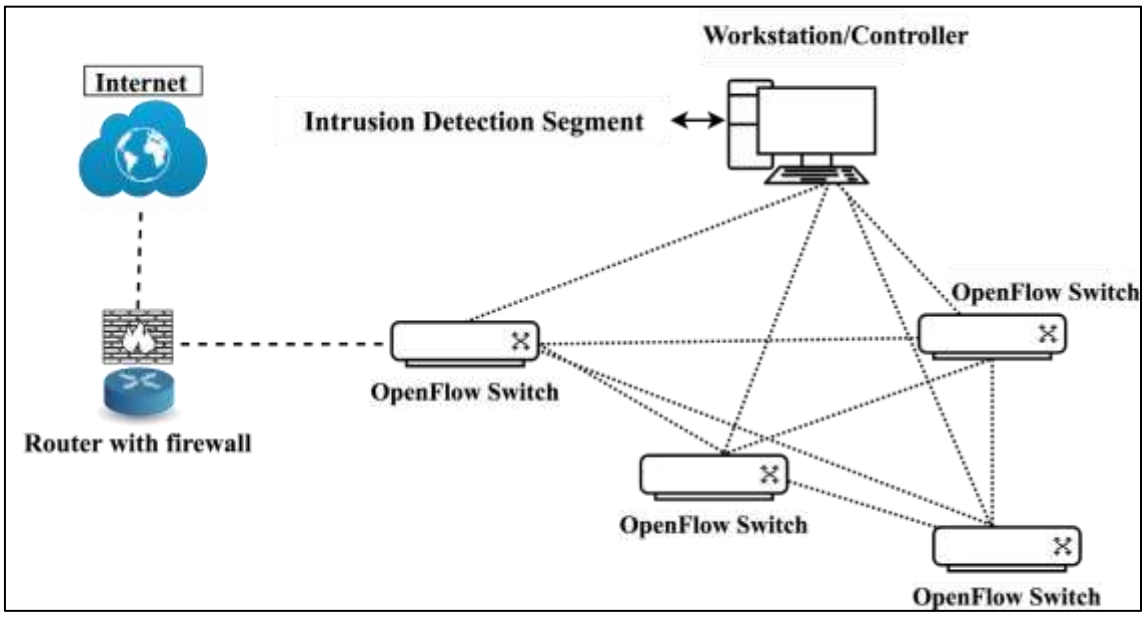


Figure 4.8: Proposed flow based anomaly detection architecture in SDN

In this chapter, we have presented a deep learning based GRU-LSTM model for detecting network intrusion in SDN with ANOVA F-Test and RFE feature selection mechanism. In SDN environment, Deep learning approach has enormous potential to detect malicious activity. SDN supports the nature of centralized controller and a very flexible structure. Our proposed intrusion detection module which is depicted in Figure 4.8 capable of easily extract the information about network traffic due to its centralized controller and flexible nature. In near future, I plan to implement this proposed model in a real environment of SDN with real traffic of network.

CHAPTER 5

EXPERIMENTAL PROCEDURES FOR METHODOLOGY ASSESSMENT

5.1 Introduction

This chapter gives examples of all the experimental procedures and tools that we have used for our experiment. In this segment we will discuss how different methodology and tools are combined together to assess the limits of Methodology scientifically and impartially.

In this research the most current framework is used, Tensorflow [54], in implementing a model for each architecture. The experiments were performed in an environment of Intel i5 3.2 GHz, 16 GB RAM, and NVIDIA GTX 1070 with Linux based Ubuntu 16.10 Distribution Operating System. The experiments were designed to evaluate the performance of each model (GRU, LSTM and GRU-LSTM) on the full NSL-KDD dataset in terms of accuracy and training time required for each model.

The experiment was executed using the developed prediction model (LSTM, GRU) on the NSL-KDD Dataset. First, by preprocessing the dataset by scaling the features and converting non-numerical features to numerical values. Second, by implementing feature selection using two different algorithms for feature selection: ANOVA-F and RFE, for the purpose of evaluating the best technique with the NSL-KDD dataset. Two models, LSTM and GRU, were evaluated to determine which algorithm to move forward with evaluating the rest of the experiment models. Third, by splitting the dataset into two sets: 80% for training and 20% for testing. The prediction model was run for both training and testing classifiers about 10 times, recording the best values of all readings.

Finally the accuracy of all prediction models and the time required to train the models was logged. All the matrices were calculated including True False Alarm Rate (Recall), False Alarm Rate (FAR), Efficiency and Precision.

5.2 Data Cleaning and Pre-processing

In order to remove the duplicate records, the NSL KDD dataset has to go through a cleaning process. It's one of the primary need for preprocessing the dataset. As NSL KDD is the updated version of KDD Cup 99 therefore this step is not anymore required. As the dataset contains both numerical and non-numerical instances therefore a mandatory pre-processing operation has to be taken in place. The classifier we defines in scikit-learn work significantly well with numerical inputs, therefore one-hot-encoding method is used to make that transformation. This method will transform each categorical featured with ***m*** possible inputs to ***n*** binary features. This binary features will then go through for feature scaling which described in next section.

5.3 Feature Scaling

Scaling of features is a mandatory requirement for deep learning or machine learning methods to evade that large values features may weights too much on the final results. For any single features we need to calculate the average, subtract the mean value from the feature value, and divide their results by their Standard Deviation (SD). After successful scaling of feature data, each feature will have a zero average, with a standard deviation of one.

5.4 Features Selection

In order to remove the redundant and irrelevant data selection of appropriate features is entirely necessary. It is a method of selecting a subset of pertinent features that completely represents full problem with a minimum worsening of presentation [26]. Generally two possible reasons were investigated that why it would be needed to restrict number of features: Firstly, it is possible that irrelevant features could suggest correlations between features and target classes that arise just by chance and do not correctly model the problem. This aspect is also related to over-fitting, usually in a decision tree classifier. Secondly, a large number of features could greatly increase the computation time without a corresponding classifier improvement. In our experiment we starts our feature selection process with a univariate feature selection Analysis of Variance (ANOVA F-test) for feature scoring. The reason behind using a univariate technique as this methods of feature selection analyzes each feature individually to determine the strength of the relationship of the feature with labels. The ***SelectPercentile*** method in the ***sklearn.feature_selection*** module were used as ***SelectPercentile*** select features based on a percentile of the highest scores. Once, the best subset of features were found, a Recursive Feature Elimination (RFE) was applied which repeatedly build a model, placing the feature aside and then repeating the process with the remained features until all features in the dataset are exhausted. As such, it is a good optimization for finding the best performing subset of features. The idea is to use the weights of a classifier to produce a feature ranking.

5.5 Building the Model

In phase two, RNN architectures: LSTM-GRU are used to be trained to detect anomalies. The implemented models were developed using Python and TensorFlow platforms **[54]** to show the capability of each model to learn the definition of being normal and anomalous from labeled datasets. Each model on the NSL-KDD dataset was evaluated as previously mentioned. The first model was a vanilla LSTM, which was trained and evaluated, as all RNN models selected for this experiment. For each model the best defaulted value parameter setup was identified to begin, tuning each model to achieve the best performance and the highest accuracy. Several runs were conducted with different values as shown in Table 5.1, such as learning rate, training cycle, time-step and hidden layers. Batch size limits the number of samples to be shown to the network before a weight update can be performed. For building our model we need to specify a loss function. Loss indicates how bad the model's prediction was on a single example; we try to minimize that while training across all the examples. Here, our loss function is the cross-entropy between the target and the softmax activation function applied to the model's prediction. We have used tf.train.AdamOptimizer that uses Kingma and Ba's Adam algorithm to control the learning rate. Adam offers several advantages over the simple ***tf.train.GradientDescentOptimizer***. Foremost is that it uses moving averages of the parameters (momentum).

Table 5.1: Parameter Values

Parameter Name	Value	Note
Learning Rate	0.01	-
Training Cycle	100 / 500 / 1000	-
Hidden Layers	25/50	-
Time-step	5/10	-

5.5.1 LSTM model

First, a vanilla LSTM model was developed, and then determined through the experiment what the suitable learning rate was for this model. The learning rate is one of the most important parameters to be tuned due to its impact on the training model for faster and effective training. It is important to not over fit the training model. The experiment was run with three different learning rates: 0.0001, 0.001, and 0.01. After running the experiment several times at value 100, the training cycle's results show that the learning rate 0.001 gives the best loss value, which then decreases during training to allow more weight updates. LSTM model was trained at three different cycles; 100, 500, and 1000. The accuracy, precision, recall, FAR and time required for the model to be trained for each training cycle was calculated.

5.5.2 GRU model

The same steps were followed as the LSTM model to train the GRU model, with a learning rate once more at 0.01. The GRU model was trained at three different training cycles; 100, 500, and 1000, calculating the accuracy, precision, recall, FAR and the time required for the model to be trained for each of the training cycles. However, it is important to keep in mind that the training time was less than LSTM due to the fact that GRU architecture consists of two gates only.

```
#Unstacking the inputs with time steps to provide the inputs in seq
uence
# Unstack to get a list of 'time_steps' tensors of shape (batch_siz
e, input_features)
x_ = tf.unstack(x,time_steps,axis =1)

#Defining a single GRU cell
gru_cell = tf.contrib.rnn.GRUCell(hidden_units)

#GRU Output
with tf.variable_scope('MyGRUCel36'):
    gruoutputs,grustates = tf.contrib.rnn.static_rnn(gru_cell,x_,dt
ype=tf.float64)

#Linear Activation , using gru inner loop last output
output =  tf.add(tf.matmul(gruoutputs[-1],tf.cast(W,tf.float64)),tf
.cast(b,tf.float64))
```

Figure 5.1: Code segment of GRU LSTM Model build

5.6 Loss Function Definition

We can specify a loss function just as easily. Loss function indicates how bad the model's prediction was on a single example; we try to minimize that while training across all the examples. Here, our loss function is the cross-entropy between the target and the softmax activation function applied to the model's prediction. Note that ***tf.nn.softmax_cross_entropy_with_logits*** internally applies the softmax on the model's unnormalized model prediction and sums across all classes, and ***tf.reduce_mean*** takes the average over these sums. The tf.train.AdamOptimizer uses Kingma and Ba's Adam algorithm to control the learning rate. Adam offers several advantages over the simple tf.train.GradientDescentOptimizer. Foremost is that it uses moving averages of the parameters (momentum).

```
#Defining the loss function
cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(label
s=y,logits = output))
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate).min
imize(cost)
```

Figure 5.2: Code segment of Loss function definition of our Model

CHAPTER 6

EXPERIMENTAL RESULTS AND DISCUSSIONS

6.1 Introduction

In this chapter, we provide the detailed results for each IDS classifier obtained using a Gated-Recurrent-Unit (GRU) neural network and their evaluation measures. We started the experiments with a light-weight GRU where we used one hidden layer and one hidden unit. We performed 10 sets of experiments for each hyper parameter set (learning rate, time-steps, hidden layers) and tuned them to obtain the optimized results. This turned out to be a binary classification problem where the classifier classifies each sample as "normal" or "attack". Hence, we used the evaluation metrics for classification like accuracy, precision, recall, false positive rate and F1 Score.

6.2 Appropriate Features Selection

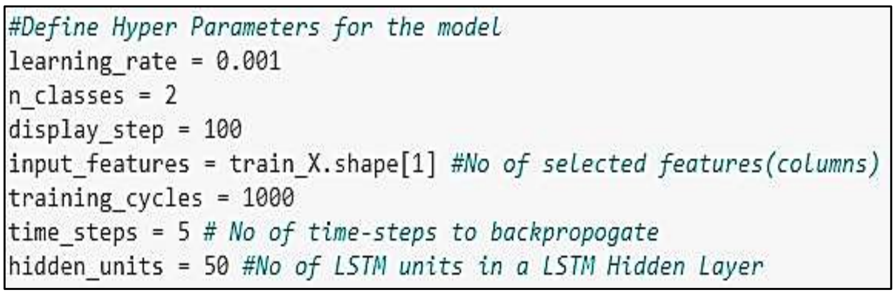
Feature selection mechanism is a required process to get rid of the irrelevant and extraneous data form the dataset. According to [34] feature selection is a process of deriving a subset of relevant features from the complete feature set without decaying presentation. Intrusion dataset containing superfluous attributes often prevents detection from being accurate. Numerous reasons were analyzed to show why restricting the features is obligatory. Irrelevant features increase computation time without contributing to classifier improvement and sometimes incorrectly indicate correlation between feature and desired class. In our experiment, we have used a univariate feature selection with Analysis of Variance (ANOVA) F-test. ANOVA is used to determine whether the means of some groups are different using F-test which statistically checks the equality of means. Each feature is individually analyzed which calculates the strength of feature-labels relationship. Percentile of the highest scores based feature selection is performed by SelectPercentile method (***sklearn.feature_selection***). Upon finding a subset Recursive Feature Elimination (RFE) is applied. REF frequently builds a model where features are kept aside and reiterates the process until all features in dataset are removed. Feature ranking is developed by using the weight of a classifier. Following table represents the selected features after applying both ANOVA F-test and REF.

Table 6.1: Selected Features after Applying ANOVA F-Test and REF

Attack Category	Selected Features
DoS	(1, 'flag_SF'), (2, 'dst_host_serror_rate'), (3, 'same_srv_rate'), (4, 'c ount'), (5, 'dst_host_srv_count'), (6, 'dst_host_same_srv_rate'), (7, ' logged_in'), (8, 'dst_host_count'), (9, 'serror_rate'), (10, 'dst_host_s rv_serror_rate'), (11, 'srv_serror_rate'), (12, 'service_http'), (13, 'fla g_S0')
Probe	(1, 'service_private'), (2, 'service_eco_i'), (3,dst_host_srv_count'), (4, 'dst_host_same_src_port_rate'), (5, 'dst_host_srv_rerror_rate'), (6, 'dst_host_diff_srv_rate'), (7, 'dst_host_srv_diff_host_rate'), (8, 'dst_host_rerror_rate'), (9, 'logged_in'), (10, 'srv_rerror_rate'), (11,'Protocol_type_icmp'), (12, 'rerror_rate'), (13, 'flag_SF')
R2L	(1, 'src_bytes'), (2, 'hot'), (3, 'dst_host_same_src_port_rate'), (4, 'dst_host_srv_count'), (5, 'dst_host_srv_diff_host_rate'), (6, 'dst_bytes'), (7, 'service_ftp_data'), (8, 'num_failed_logins'), (9, 'is_guest_login'), (10, 'service_imap4'), (11, 'service_ftp'), (12, 'flag_RSTO'), (13, 'service_http')
U2R	(1, 'hot'), (2, 'dst_host_srv_count'), (3, 'dst_host_count'), (4, 'num_file_creations'), (5,'root_shell'), (6,'dst_host_same_src_port_rate'), (7,'dst_host_srv_diff_host_rate'), (8, 'service_ftp_data'), (9, 'service_telnet'), (10, 'num_shells'), (11, 'urgent'), (12, 'service_http'), (13, 'srv_diff_host_rate')

6.3 Hyper Parameter Tuning

In this section, we have evaluated the performance of the IDS classifiers by tuning the hyper-parameters of the GRU algorithm. We have performed a similar type of experiments on each IDS classifier for different learning rate. We compared the values of training accuracy, recall and false alarm rate with learning rate and time-steps to understand the behavior of model with change in hyper-parameters.



```
#Define Hyper Parameters for the model
learning_rate = 0.001
n_classes = 2
display_step = 100
input_features = train_X.shape[1] #No of selected features(columns)
training_cycles = 1000
time_steps = 5 # No of time-steps to backpropogate
hidden_units = 50 #No of LSTM units in a LSTM Hidden Layer
```

Figure 6.1: Hyper Parameter set initilization for proposed model

6.4 Performance Results of IDS Classifier:

In this experiment section, first, we started with a time-step range [10,20,30,40,50,60,70,80,100] and selected the time-steps which has best training accuracy. After selecting the time-step we searched for the learning-rate which produces best training accuracy. We used one hidden layer and one hidden unit in the network. The detailed results are shown in Table 6.2, 6.3 and 6.4. The detailed Evaluation Matrix for IDS classifier with the best hyper-parameter combination (time-steps = 70, learning_rate = 0.01) is presented in Table 6.2 and Figure 6.2.

Table 6.2: Evaluation Metrics for IDS Classifier (learning rate =0.01)

Time-Steps	Train Accuracy	Precision	Recall	F-1 Score	FAR
10	86.632	0.829	0.747	0.785	0.43
20	85.534	0.186	0.431	0.259	0.525
30	84.51	0.817	0.711	0.76	0.505
40	86.613	0.794	0.606	0.687	0.691
50	85.434	0.858	0.819	0.838	0.293
60	72.89	0.86	0.829	0.858	0.297
70	**87.911**	**0.835**	**0.779**	**0.806**	**0.362**
80	83.243	0.832	0.762	0.795	0.398
90	83.323	0.814	0.726	0.767	0.461
100	82.167	0.828	0.752	0.788	0.419

From the Table above, it can be inferred that the model performance is optimized when the input is given with ‘70’ time-steps and thus, this value is selected for further experiments for the IDS in this research.

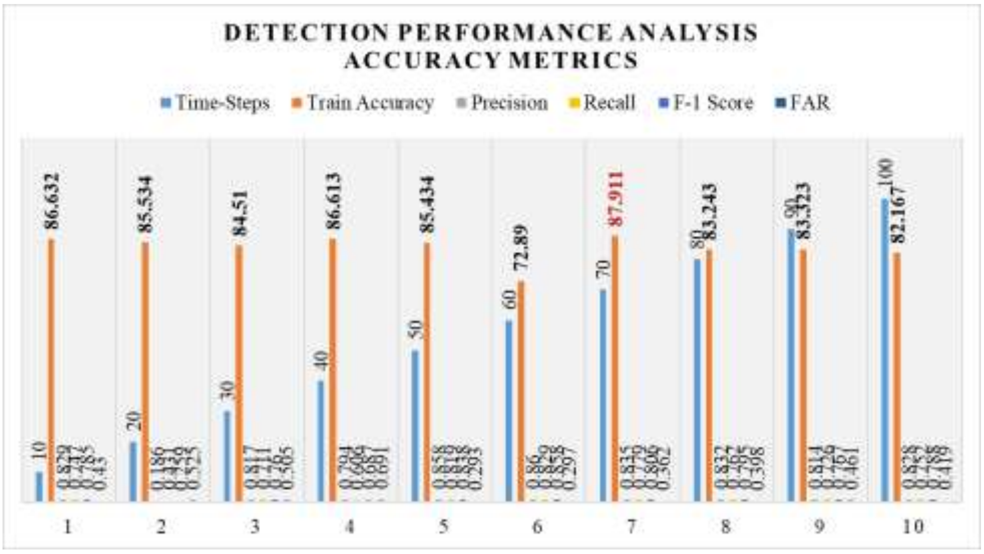


Figure 6.2: Evaluation metrics for IDS classifier (learning rate =0.01)

In this experiments section, a similar series of experiments is performed with a learning rate of 0.1 and we have achieved the best training accuracy with ‘100’ time steps. The complete results this IDS can be interpreted in Table 6.3 and Figure 6.3.

Table 6.3: Evaluation Metrics for IDS Classifier (learning rate =0.1)

Time-Steps	Train Accuracy	Precision	Recall	F1 score	FAR
10	78.006	0.84	0.781	0.809	0.364
20	79.36	0.846	0.794	0.819	0.341
30	77.102	0.835	0.771	0.802	0.382
40	72.068	0.789	0.721	0.753	0.442
50	72.595	0.814	0.726	0.767	0.461
60	71.965	0.787	0.72	0.752	0.441
70	73.203	0.816	0.732	0.772	0.45
80	73.984	0.82	0.74	0.778	0.436
90	74.84	0.823	0.748	0.784	0.42
100	79.249	0.839	0.792	0.815	0.333

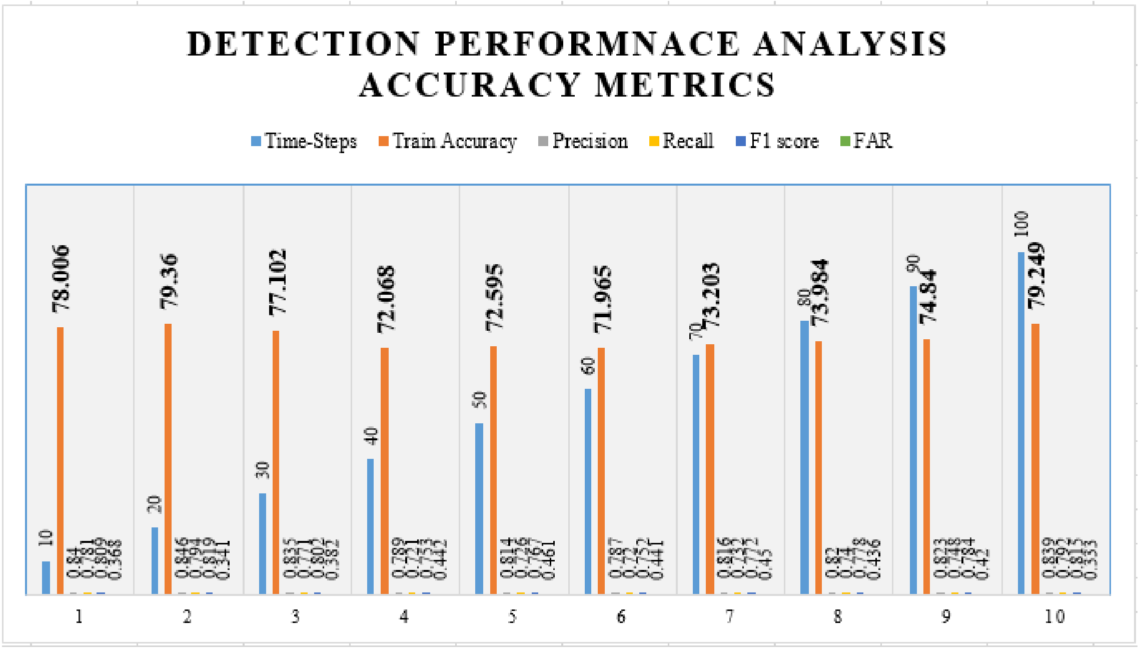


Figure 6.3: Evaluation metrics for IDS classifier (learning rate =0.1)

Again in following section, a similar series of experiments is performed with a learning rate of 0.001 and we have achieved the best training accuracy with '20' time steps. The complete results this IDS can be interpreted in Table 6.4 and Figure 6.4.

Table 6.4: Evaluation Metrics for IDS Classifier (learning rate =0.001)

Time-Steps	Train Accuracy	Precision	Recall	F1 score	FAR
10	78.927	0.853	0.804	0.828	0.348
20	80.708	0.848	0.733	0.786	0.317
30	80.371	0.813	0.726	0.767	0.318
40	73.292	0.814	0.735	0.772	0.444
50	72.595	0.812	0.716	0.761	0.461
60	73.522	0.808	0.781	0.794	0.438
70	71.562	0.838	0.801	0.819	0.478
80	78.051	0.85	0.78	0.813	0.363
90	80.132	0.84	0.726	0.779	0.328
100	77.989	0.79	0.726	0.757	0.367

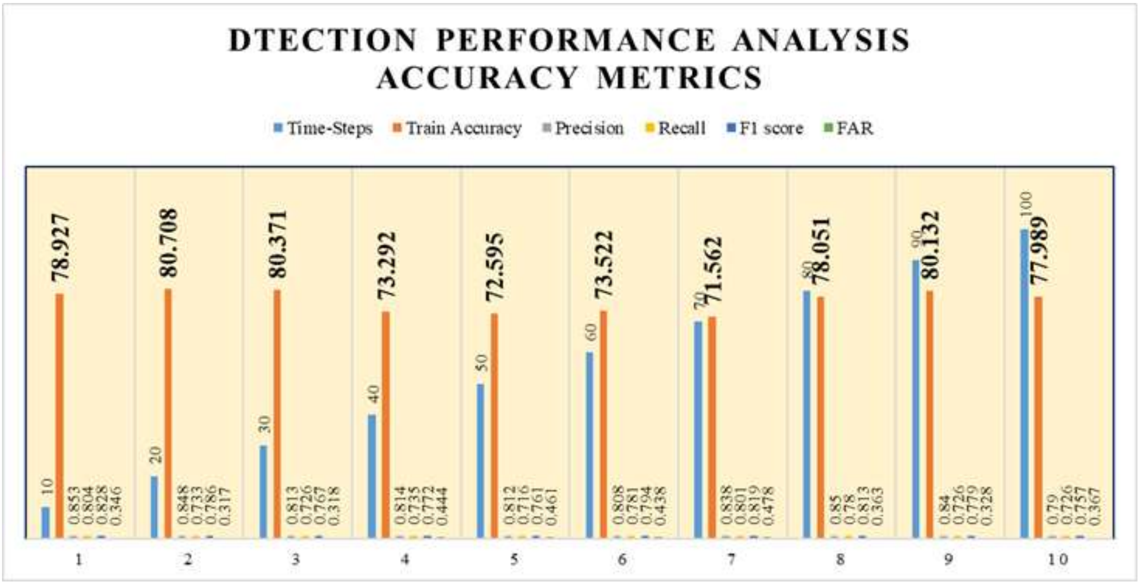


Figure 6.4: Evaluation metrics for IDS classifier (learning rate =0.001)

6.5 Comparison with Existing Work

Initially, we implemented the network intrusion detection for 2-class based classification namely normal and anomaly. In addition, we assessed our work by assimilate the results and proposed a model security architecture for detecting flow based anomaly in OpenFlow based Controller. From the above discussion and also from Figure 6.2 its clearly shows that with the ANOVA F-Test and RFE feature selection methods GRU-LSTM classifier provides highest Accuracy of 87% with a very low false alarm rate of 0.76%. Moreover, our results is generated based on the selection of features from complete dataset of NSL-KDD. Very few approaches has been presented from different authors in order to show the accuracy of Deep learning algorithm for NSL-KDD Dataset. However, there also exists some lacking of preprocessing of dataset and appropriate feature selection for testing and training. For comparing with others work we will only consider that approaches which is only based on NSL-KDD dataset for the detection of flow based anomaly in Software Defined Networking. Following Table 6.5 shows the accuracy of the anomaly detection scheme of the state-of-the-art results against our proposed model. The optimized results of our GRU-LSTM classifier are compared and it was found that the performance of our model are comparatively lower than others. However, all the compared methods are not aware of using an appropriate feature selection with updated dataset whereas we have showed a strong potential in terms of accuracy detection by applying ANOVA F-Test and RFE Methods with NSL-KDD dataset. Though, our results are not high enough but for anomaly detection in SDN with proper feature selection it will produce a great significance in terms of accuracy calculation.

Table 6.5: Comparison of accuracy with other studies

IDS Methods	**Dataset**	**Feature Selection Method**	**Accuracy**	**False Alarm Rate**
GRU-RNN [23]	KDD Cup 99	--------	97.65%	10.01%
FNN [55]	KDD Cup 99	---------	97.35%	2.65%
GRNN [55]	KDD Cup 99		93.05%	12.46%
RBNN [55]	KDD Cup 99	-----	93.05%	6.95%
SVM [42]	KDD Cup 99	-----	69.52%	-----
LSTM-RNN [23]	KDD Cup 99	-----	97.54%	------
DNN [56]	NSL-KDD	-----	75.75%	-----
GRU-RNN [19]	NSL-KDD	-----	89%	-----
NB Tree [42]	KDD Cup 99	-----	82.02%	-----
GRU-LSTM (proposed)	**NSL KDD**	**ANOVA F-Test and RFE**	**87%**	**0.76%**

CHAPTER 7

CONCLUSIONS AND FUTURE WORK

In this thesis, we have presented a Deep learning based GRU-LSTM model for detecting network intrusion in SDN and shows the best classifier in terms of different evaluation metrics with ANOVA F-Test and REF feature selection mechanism. Although our experimental results are not yet high enough comparing with others but it still has significant contribution in the field of appropriate feature selection from a dataset. In SDN environment, Deep learning approach has enormous potential to detect malicious activity. SDN supports the nature of centralized controller and a very flexible structure.

The novelty of this research stems from the fact that it is the first experiment that implements and compares RNN's architecture and offers more insight into each architecture, particularly LSTM and GRU, on a benchmark intrusion detection dataset NSL-KDD. Most literature in the domain demonstrates the concept of using LSTM as one of the RNN architectures to improve the accuracy in predicting attacks, as well its different variants, however they only focused on one architecture for one application, comparing it with other deep learning approaches. This research took the path further in understanding the architecture of each RNN algorithm, then applying it in an intrusion detection dataset. It evaluates the performance of each architecture in terms of prediction accuracy and the time required for each architecture to be trained.

Moreover, this experiment is unique as it runs these architectures on the full NSL KDD dataset rather than the commonly used NSL-KDD 10% dataset. Feature selection was performed using two different mechanisms, ANOVA F-Test and Recursive Feature Elimination, which are suitable for intrusion detection. This has offered a clean dataset that carries all the important features. Feature selection reduced the dataset features to improve the performance of accuracy, recall, training time and false alarm rate. As part of this research, the experiment was limited in tuning the set of parameters with the goal of finding the optimal parameters such as learning rate, hidden layers, and training cycle to improve the model's prediction accuracy and the amount of time required to be trained.

The results of this evaluation revealed that GRU-LSTM still stands up and outperforms other architectures. GRU has fewer parameters resulting in a faster-trained model compared to LSTM. In a large-scale implementation, however, LSTM may yield better results. Our proposed intrusion detection module is capable of easily extract the information about network traffic due to its centralized controller and flexible nature. From the experiment, it’s clear that GRU-LSTM shows a high test accuracy comparing with all other algorithms therefore for flow based anomaly detection use of our model is very essential in order to achieve high accuracy and speeding up the process of intrusion detection in SDN.

For future work, the aim is to evaluate further architectures on the intrusion detection dataset. Moreover, the aim is to investigate the application of deep learning by having multiple layers and hybrid layers of different architectures in one framework, as well as deploying these techniques in SDN applications to develop robust security solutions. In near future, we plan to implement this proposed model in a real Environment of SDN with real traffic of Network.

REFERENCES

[1] D. Kreutz, F. M. V. Ramos, P. E. Veríssimo, C. E. Rothenberg, S. Azodolmolky and S. Uhlig, “Software-Defined Networking: A Comprehensive Survey,” in *Proceedings of the IEEE*, vol. 103, no.1, pp. 14-76, Jan. 2015. doi: 10.1109/JPROC.2014.2371999

[2] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker and J. Turner, “OpenFlow: Enabling Innovation in Campus Networks,” *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69-74, Mar. 2008. doi:10.1145/1355734.1355746

[3] M. Malboubi, L. Wang, C. Chuah and P. Sharma, “Intelligent SDN based traffic (de)Aggregation and Measurement Paradigm (iSTAMP),” *IEEE INFOCOM 2014 - IEEE Conference on Computer Communications*, Toronto, ON, 2014, pp. 934-942.doi: 10.1109/INFOCOM.2014.6848022

[4] “Introduction to Cisco IOS NetFlow-A Technical Overview”. Available: http://www.cisco.com/c/en/us/products/collateral/ios-nx-os-software/ios-netflow/prodwhite paper0900aecd80406232.html. [Accessed on: Mar. 16, 2018].

[5] “OpenDaylight Project”. Available: https://www.opendaylight.org. [Accessed on: Apr. 16, 2018].

[6] “OpenDaylight-Project Members”. Available: https://www.opendaylight.org/support/members. [Accessed on: May. 5, 2018].

[7] “OpenDaylight-Project”. Available: https://www.opendaylight.org/what-we-do/current-release/beryllium. [Accessed on: Mar. 18, 2018].

[8] A. Graves, A. Mohamed and G. Hinton, “Speech recognition with deep recurrent neural networks,” *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, Vancouver, BC, 2013, pp. 6645-6649. doi: 10.1109/ICASSP.2013.6638947

[9] K. Cho, B. van Merrienboer, C. Gulcehre, F. Bougares, H. Schwenk, and Y. Bengio, “Learning Phrase Representations Using RNN Encoder-Decoder for Statistical Machine Translation”, 2014, CoRR, vol. abs/1406.1078. [Online]. Available: http://arxiv.org/abs/1406.1078.

[10] P. Shah, V. Bakrola and S. Pati, “Image Captioning Using Deep Neural Architectures,” *2017 International Conference on Innovations in Information, Embedded and Communication Systems (ICIIECS),* Coimbatore, 2017, pp. 1-4. doi: 10.1109/ICIIECS.2017.8276124

[11] Z. Jadidi, V. Muthukkumarasamy, E. Sithirasenan and M. Sheikhan, “Flow-Based Anomaly Detection Using Neural Network Optimized with GSA Algorithm,” *2013 IEEE 33rd International Conference on Distributed Computing Systems Workshops*, Philadelphia, PA, 2013, pp. 76-81. doi: 10.1109/ICDCSW.2013.40

[12] P. Winter, E. Hermann and M. Zeilinger, “Inductive Intrusion Detection in Flow-Based Network Data Using One-Class Support Vector Machines,” *2011 4th IFIP International Conference on New Technologies, Mobility and Security*, Paris, 2011, pp.1-5. doi: 10.1109/NTMS.2011.5720582

[13] R. Braga, E. Mota and A. Passito, “Lightweight DDoS Flooding Attack Detection Using NOX/OpenFlow,” *IEEE Local Computer Network Conference*, Denver, CO, 2010, pp. 408-415. doi: 10.1109/LCN.2010.5735752

[14] Kokila RT, S. Thamarai Selvi and K. Govindarajan, “DDoS detection and analysis in SDN-based environment using support vector machine classifier,” *2014 Sixth International Conference on Advanced Computing (ICoAC)*, Chennai, 2014, pp. 205-210.doi: 10.1109/ICoAC.2014.7229711

[15] 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,” *2016 IEEE Sixth International Conference on Communications and Electronics (ICCE),* Ha Long, 2016, pp. 13-18. doi: 10.1109/CCE.2016.7562606

[16] S. M. Mousavi and M. St-Hilaire, “Early detection of DDoS attacks against SDN controllers,” *2015 International Conference on Computing, Networking and*

Communications (ICNC), Garden Grove, CA, 2015, pp. 77-81. doi: 10.1109/ICCNC.2015.7069319

[17] A. AlEroud and I. Alsmadi, “Identifying cyber-attacks on software defined networks: An inference-based intrusion detection approach,” *Journal of Network and Computer Applications*, vol. 80, pp. 152–164, Feb. 2017. doi:10.1016/j.jnca.2016.12.024

[18] Q. Niyaz, W. Sun and A. Y. Javaid, “A Deep Learning Based DDos Detection System in Software-Defined Networking (SDN)”, *ICST Trans. Security Safety, 4,* e2, Nov. 2016. doi: 10.4108/eai.28-12-2017.153515

[19] T. A. Tang, L. Mhamdi, D. McLernon, S. A. R. Zaidi and M. Ghogho, “Deep Recurrent Neural Network for Intrusion Detection in SDN-based Networks,” *2018 4th IEEE Conference on Network Softwarization and Workshops (NetSoft)*, Montreal, QC, 2018, pp. 202-206. doi: 10.1109/NETSOFT.2018.8460090

[20] A. Tuor, S. Kaplan, B. Hutchinson, N. Nichols and S. Robinson, “Deep learning for unsupervised insider threat detection in structured cybersecurity data streams”, 2017, CoRR, vol. abs/1710.00811.[Online]. Available: http://arxiv.org/abs/1710.00811

[21] Y. Fu, F. Lou, F. Meng, Z. Tian, H. Zhang and F. Jiang, “An Intelligent Network Attack Detection Method Based on RNN,” *2018 IEEE Third International Conference on Data Science in Cyberspace (DSC)*, Guangzhou, 2018, pp. 483-489. doi: 10.1109/DSC.2018.00078

[22] M. Alkasassbeh, G. Al-Naymat, N. Hamadneh, I. Obeidat and M. Almseidin, “Intensive preprocessing of KDD cup 99 for network intrusion classification using machine learning techniques”, 2018, CoRR, vol. abs/1805.10458. [Online]. Available: http://arxiv.org/abs/1805.10458

[23] J. Kim, J. Kim, H. L. T. Thu and H. Kim, “Long Short Term Memory Recurrent Neural Network Classifier for Intrusion Detection,” *2016 International Conference on Platform Technology and Service (PlatCon)*, Jeju, 2016, pp. 1-5. doi: 10.1109/PlatCon.2016.7456805

[24] Y. Miao, J. Li, Y. Wang, S. Zhang and Y. Gong, “Simplifying long short-term memory acoustic models for fast training and decoding,” *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Shanghai, 2016, pp. 2284-2288. doi: 10.1109/ICASSP.2016.7472084

[25] Q. Lyu and J. Zhu, “Revisit Long Short-Term Memory: An Optimization Perspective”, *in Advances in neural information processing systems workshop on deep Learning and representation Learning*, 2014, pp. 1–9.

[26] F. A. Gers, N. N. Schraudolph and J. Schmidhuber, “Learning Precise Timing with LSTM Recurrent Networks”, *Journal of machine learning research*, vol. 3, pp. 115–143, Aug. 2002.

[27] J. Chung, C. Gulcehre, K. Cho and Y. Bengio, “Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling”, 2014, CoRR, vol. abs/1412.3555. [Online]. Available: http://arxiv.org/abs/1412.3555

[28] A. Elsherif, “Automatic Intrusion Detection System Using Deep Recurrent Neural Network Paradigm,” *Journal of Information Security and Cybercrimes Research (JISCR)*, vol. 1, no. 1, 2018.

[29] A. H. Mirza and S. Cosan, “Computer Network Intrusion Detection Using Sequential LSTM Neural Networks Auto Encoders”, *in 2018 26th Signal Processing and Communications Applications Conference (SIU)*, pp. 1–4, May. 2018.

[30] T. Le, J. Kim and H. Kim, "An Effective Intrusion Detection Classifier Using Long Short-Term Memory with Gradient Descent Optimization," *2017 International Conference on Platform Technology and Service (PlatCon)*, Busan, 2017, pp. 1-6. doi: 10.1109/PlatCon.2017.7883684

[31] L. Bontemps, V. L. Cao, J. McDermott and N. Le-Khac, “Collective Anomaly Detection Based on Long Short Term Memory Recurrent Neural Network”, 2017, CoRR, vol. abs/1703.09752. [Online]. Available: http://arxiv.org/abs/1703.09752

[32] B. J. Radford, L. M. Apolonio, A. J. Trias and J. A. Simpson, “Network Traffic Anomaly Detection Using Recurrent Neural Networks”, 2018, CoRR, vol. abs/1803.10769. [Online]. Available: http://arxiv.org/abs/1803.10769

[33] V. Campos, B. Jou, X. I. Nieto Giro´, J. Torres and S. Chang, “Skip RNN: Learning to Skip State Updates in Recurrent Neural Networks”, 2017, CoRR, vol. abs/1708.06834. [Online]. Available: http://arxiv.org/abs/1708.06834

[34] S. Axelsson, “Intrusion detection systems: A survey and taxonomy”, 2000, Vol. 99. Technical report.

[35] V. Chandola, A. Banerjee and V. Kumar, “Anomaly detection: A survey”, 2009, *ACM Computing Surveys (CSUR),* 41(3), 15.

[36] A. Sperotto, G. Schaffrath, R. Sadre, C. Morariu, A. Pras and B. Stiller, “An Overview of IP Flow-Based Intrusion Detection,” in *IEEE Communications Surveys & Tutorials*, vol. 12, no. 3, pp. 343-356, Third Quarter 2010. doi: 10.1109/SURV.2010.032210.00054

[37] J. W. Lockwood *et al.*, “NetFPGA-An Open Platform for Gigabit-Rate Network Switching and Routing,” *2007 IEEE International Conference on Microelectronic Systems Education (MSE'07),* San Diego, CA, 2007, pp. 160-161. doi: 10.1109/MSE.2007.69

[38] A. Abraham, R. Jain, J. Thomas and S. Y. Han, “D-SCIDS: Distributed Soft Computing Intrusion Detection System”, 2007, *Journal of Network and Computer Applications*, 30(1), pp. 81-98.

[39] “Software-Defined Networking: The New Norm for Networks”, ONF White Paper, 2012, [online]. Available: https://www.opennetworking.org. [Accessed on: Jun. 18, 2018].

[40] “Open Networking Foundation, OpenFlow Switch Specification,” 2009, [online]. Available: https://www.sdxcentral.com/sdn/definitions/who-is-open-networking-foundation-onf. [Accessed on: May. 17, 2018].

[41] Lincoln laboratory MIT. “DARPA Intrusion Detection Evaluation,” 1998, [online].Available:*http://www.ll.mit.edu/mission/communications/cyber/CSTcorpora/ideval/data/*. [Accessed on: Nov. 07, 2017].

[42] M. Tavallaee, E. Bagheri, W. Lu and A. A. Ghorbani, “A detailed analysis of the KDD CUP 99 data set,” *2009 IEEE Symposium on Computational Intelligence for Security and Defense Applications*, Ottawa, ON, 2009, pp. 1-6. doi: 10.1109/CISDA.2009.5356528

[43] C. Thomas, V. Sharma and N. Balakrishnan, “Usefulness of DARPA Dataset for Intrusion Detection System Evaluation”, 2008, *in SPIE Defense and Security Symposium*. International Society for Optics and Photonics, pp. 69730G-69730G.

[44] S. Kaushik Sapna, P. R. Deshmukh, “Detection of Attacks in an Intrusion Detection System”, 2011, *International Journal of Computer Science and Information Technologies*, Vol. 2 (3), 982-986.

[45] “NSL KDD Dataset”. [Online]. Available: http://nsl.cs.unb.ca/NSL-KDD/. [Accessed on: May 6, 2017].

[46] L. Dhanabal and S. P. Shantharajah, “A Study on NSL-KDD Dataset for Intrusion Detection System Based on Classification Algorithms”, 2015, *International Journal of Advanced Research in Computer and Communication Engineering* 4.6: 446-452.

[47] E. Alpaydin, “Introduction to Machine Learning”, 2014, MIT press, Second Edition.

[48] Y. LeCun, Y. Bengio & G. Hinton, “Deep learning. Nature”, 2015, 521(7553), 436-444. Available: https://www.nature.com/articles.

[49] W. Anani, “Recurrent Neural Network Architectures Toward Intrusion Detection”, 2018, (Doctoral dissertation, The University of Western Ontario).

[50] M. K. Putchala, “Deep Learning Approach for Intrusion Detection System (IDS) in the Internet of Things (IoT) Network using Gated Recurrent Neural Networks (GRU)”, 2017, (Doctoral dissertation, Wright State University).

[51] G K. Greff, R. K. Srivastava, J. Koutník, B. R. Steunebrink and J. Schmidhuber, “LSTM: A Search Space Odyssey,” in *IEEE Transactions on Neural Networks and Learning Systems,* vol. 28, no. 10, pp. 2222-2232, Oct. 2017. doi: 10.1109/TNNLS.2016.2582924

[52] N. Murata, S. Yoshizawa and S. Amari, “Network Information Criterion-Determining the Number of Hidden Units for an Artificial Neural Network Model,” in *IEEE Transactions on Neural Networks*, vol. 5, no. 6, pp. 865-872, Nov. 1994. doi: 10.1109/72.329683

[53] H. Nkiama, S. Zainudeen, M. Said and M. Saidu, "A Subset Feature Elimination Mechanism for Intrusion Detection System", 2016, *International Journal of Advanced Computer Science and Applications (IJACSA)*, 7(4).

[54] "Tensorflow Framework". [Online]. Available: https://www.tensorflow.org. [Accessed on: April 16, 2017].

[55] S. Devaraju and S. Ramakrishnan, "Performance Comparison For Intrusion Detection System Using Neural Network with KDD Dataset", 2014, *in ICTACT Journal on Soft Computing*, pp. 743-752.

[56] 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," *2016 International Conference on Wireless Networks and Mobile Communications (WINCOM)*, Fez, 2016, pp. 258-263. doi: 10.1109/WINCOM.2016.7777224

APPENDIX A

Feature Selection Code Segment

One-Hot-Encoding

```
enc = OneHotEncoder()
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=te
stdumcols)

df_cat_data.head()
```

Feature Scaling

```
# Split dataframes into X & Y
# assign X as a dataframe of feautures and Y as a series of outcome variables
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
```

Univariate Feature Selection

```
np.seterr(divide='ignore', invalid='ignore');
selector=SelectPercentile(f_classif, percentile=10)
X_newDoS = selector.fit_transform(X_DoS,Y_DoS)
X_newDoS.shape
```

APPENDIX B

Anomaly Detection Model Build Code Segment

Building the Model

```
#Unstacking the inputs with time steps to provide the inputs in sequence
# Unstack to get a list of 'time_steps' tensors of shape (batch_size, input_features)
x_ = tf.unstack(x,time_steps,axis =1)

#Defining a single GRU cell
gru_cell = tf.contrib.rnn.GRUCell(hidden_units)

#GRU Output
with tf.variable_scope('MyGRUCel36'):
    gruoutputs,grustates = tf.contrib.rnn.static_rnn(gru_cell,x_,dtype=tf.float64)

#Linear Activation , using gru inner loop last output
output =  tf.add(tf.matmul(gruoutputs[-1],tf.cast(W,tf.float64)),tf.cast(b,tf.float64))
```

Model Training

```
#Training the Model
sess = tf.InteractiveSession()
sess.run(tf.global_variables_initializer())
for i in range (training_cycles):
    _,c = sess.run([optimizer,cost], feed_dict = {x:newtrain_X, y:newtrain_Y})

    if (i) % display_step == 0:
        print ("Cost for the training cycle : ",i," : is : ",sess.run(cost, feed_dict ={x :newtrai
n_X,y:newtrain_Y}))
correct = tf.equal(tf.argmax(output, 1), tf.argmax(y,1))
accuracy = tf.reduce_mean(tf.cast(correct, 'float'))
print('Accuracy on the overall test set is :',accuracy.eval({x:newtest_X, y:newtest_Y}))
```

Evaluation Matrix

```
pred_class = sess.run(tf.argmax(output,1),feed_dict = {x:newtest_X,y:newtest_Y})
labels_class = sess.run(tf.argmax(y,1),feed_dict = {x:newtest_X,y:newtest_Y})
conf = tf.contrib.metrics.confusion_matrix(labels_class,pred_class,dtype = tf.int32)
ConfM = sess.run(conf, feed_dict={x:newtest_X,y:newtest_Y})
print ("confusion matrix \n",ConfM)


#Plotting the Confusion Matrix
```

```
labels = ['Normal', 'Attack']
fig = plt.figure()
ax = fig.add_subplot(111)
cax = ax.matshow(ConfM)
plt.title('Confusion matrix of the classifier')
fig.colorbar(cax)
ax.set_xticklabels([''] + labels)
ax.set_yticklabels([''] + labels)
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()
```

LIST OF PUBLICATIONS

[1] **Samrat Kumar Dey**, and Md. Mahbubur Rahman *"Performance Analysis of SDN Based Intrusion Detection Model with Feature Selection Approach"* International Joint Conference on Computational Intelligence (IJCCI 2018), 14th -15th December, Daffodil International University (DIU), Dhaka, Bangladesh.

[2] **Samrat Kumar Dey**, and Md. Mahbubur Rahman *"Flow Based Anomaly Detection in Software Defined Networking: A Deep Learning Approach With Feature Selection Method"* 4th International Conference on Electrical Engineering and Information and Communication Technology (iCEEiCT 2018), 13th-15th September, Military Institute of Science and Technology (MIST), Mirpur, Dhaka, Bangladesh.

[3] **Samrat Kumar Dey**, and Md. Mahbubur Rahman *"Detection of Flow Based Anomaly in OpenFlow Controller: A Machine Learning Approach in Software Defined Networking"* 4th International Conference on Electrical Engineering and Information and Communication Technology (iCEEiCT 2018), 13th-15th September, Military Institute of Science and Technology (MIST), Mirpur, Dhaka, Bangladesh.

[4] **Samrat Kumar Dey**, and Md. Mahbubur Rahman *"Analyzing the Performance of Anomaly Detection Model in Software Defined Networking: Feature Selection Approach with Machine Learning Algorithm"* International Conference on Electrical, Electronics, Computers, Communication, Mechanical and Computing (EECCMC 2018), 28th -29th January, Vellore District, Tamil Nadu, India.