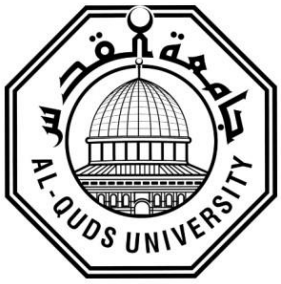
Deanship of Graduate Studies

Al-Quds University



Anomaly-Based Network Intrusion Detection System Using Deep Neural Networks (ANIDS-DNN)

Sharif Mohammed Ahmad Yasin

M.Sc. Thesis

Jerusalem – Palestine

1444 / 2023

Anomaly-Based Network Intrusion Detection System Using Deep Neural Networks (ANIDS-DNN)

Prepared By:
Sharif Mohammed Ahmad Yasin

B.Sc. in Electrical Engineering, 1997,
The University of Jordan, Amman, Jordan

Supervisor: Dr. Rushdi Hamamreh

This Thesis was Submitted in Partial Fulfillment of the Requirements for the Master's Degree of Science in Electronics and Computer Engineering from the Faculty of Graduate Studies at Al-Quds University.

1444 / 2023

Al-Quds University

Deanship of Graduate Studies

Master of Electronics & Computer Engineering

Thesis Approval

Anomaly-Based Network Intrusion Detection System Using Deep Neural Networks (ANIDS-DNN)

Prepared By: Sharif Mohammed Ahmad Yasin

Registration Number: 21920082

Master thesis submitted and accepted. Date: 08-01-2023
The names and signatures of the examining committee members are as follows:

Head of Committee:	**Dr. Rushdi Hamamreh**	**Signature:** .....
Internal Examiner:	**Prof. Ali Jamoos**	**Signature:** .....
External Examiner:	**Prof. Nabil Arman**	**Signature:** .....

Jerusalem – Palestine

1444 / 2023

Dedication

I dedicate this work to the soul of my parents,
My parents-in-law,
My wife,
My daughter Ruba,
My sons Abdulrahman, and Mohammad,
My sisters, brothers, and their families.

Declaration

I certify that this thesis submitted for the degree of Master, is the result of my own research, except where otherwise acknowledged, and that this study (or any part of the same) has not been submitted for a higher degree to any other university or institution.

Signed: 

Sharif Mohammed Ahmad Yasin

Date: 08/01/2023

Acknowledgment

I thank the almighty whose blessings have bestowed the willpower and confidence to complete my thesis.

I would like to express my deepest gratitude to my supervisor, Dr. Rushdi Hamamreh, who provided immense support, guidance, and feedback throughout this thesis.

I am also thankful to all staff members of the Department of Computer and Electronics Engineering at Al-Quds University for their help and support.

Abstract

Network intrusion detection systems (NIDS) monitor and analyze inbound and outbound network traffic and raise alarms when intrusions or malicious activities are detected. They are considered the second line of defense after the firewall.

In recent years, the increasing number of cyber-attacks motivated the need to develop automated and intelligent network intrusion detection systems that learn the normal behavior of network traffic, and thus the traffic that deviates from normal is considered anomalous or malicious. Although anomaly-based NIDS are better than signature-based NIDS in detecting zero-day or unknown attacks, they generate a high false alarm rate if the normal behavior of the network is changed.

To solve the aforementioned problems, this thesis proposes ANIDS model for detection of unknown or zero-day attacks in network flows using feed forward deep neural networks as a classification algorithm. The model takes the collected NetFlow features as input and after the data is prepared the deep learning algorithm classifies every network flow as either normal or anomaly. In order to improve detection accuracy of attack patterns that have time-dependent frequencies such as nerisbotnet attack, three new input features namely, sa_nsessions_T, da_nsessions_T, and sa_da_nsessions_T are devised by aggregating the flows based on source address, destination address, and timestamp attributes using a time window of one minute. Also, after preprocessing input features the most important 45 input features are selected. Moreover, the model's parameters are learned using large number of multiclass labeled flows from the public UGR'16 dataset [32]. The hyperparameters of the model are optimized for best performance in terms of accuracy, recall, precision, and training time of the model.

The experimental results confirmed the high performance of the proposed model when tested on unseen network flows from the UGR'16 dataset. In addition, the optimal

network architecture consists of one input layer, three hidden layers, and one output layer. The model achieved an accuracy of 99.773 %, a false positive rate of less than 1%, and an area under the receiver operating characteristic curve (ROC-AUC) of 0.999915. Also, the detection accuracy of the multiclass classifier is 99.58% and the detection rate for all individual classes is above 99% except for the NerisBotNet attack where the detection rate is 94.70%. When the proposed model is compared with other recent models in literature [33], [52], that were evaluated on the same UGR'16 dataset, the experimental results show that the proposed model outperforms other models in terms of precision and recall.

Index Terms: Deep learning, cyberattacks, NetFlow classification, network intrusion detection, UGR'16 dataset.

نظام كشف الاختراق الشبكي المبني على تدفق البيانات غير المألوف باستخدام الشبكات العصبية العميقة

إعداد: شريف محمد احمد ياسين

بإشراف: د. رشدي الحمامرة

الملخص

تقوم أنظمة الكشف عن اختراق الشبكات (NIDS) بمراقبة وتحليل حركة مرور الشبكة الواردة والصادرة وإصدار الإنذارات عند اكتشاف عمليات اقتحام أو أنشطة ضارة حيث تعتبر هذه الأنظمة خط الدفاع الثاني بعد جدار الحماية.

في السنوات الأخيرة، حفز العدد المتزايد من الهجمات السيبرانية الحاجة إلى تطوير أنظمة آلية وذكية للكشف عن اختراق الشبكة والتي تتعلم السلوك الطبيعي لحركة مرور الشبكة (ANIDS)، وبالتالي فإن حركة المرور التي تنحرف عن المعتاد تعتبر شاذة أو ضارة. على الرغم من أن أنظمة كشف التسلل المستندة إلى حركة المرور غير المعتادة أفضل من أنظمة كشف التسلل القائمة على التوقيع في اكتشاف هجمات اليوم الصفري أو غير المعروفة، إلا أنها تولد معدل إنذار خاطئ مرتفعًا إذا تم تغيير السلوك الطبيعي للشبكة.

لحل المشكلات المذكورة أعلاه، تقترح هذه الأطروحة نموذج ANIDS لاكتشاف الهجمات غير المعروفة او هجمات اليوم الصفري في تدفقات الشبكة باستخدام تغذية الشبكات العصبية العميقة الى الامام كخوارزمية تصنيف. يأخذ النموذج ميزات NetFlow المجمعة كمدخلات وبعد إعداد البيانات، تصنف خوارزمية التعلم العميق كل اتصال بالشبكة على أنه عادي أو شاذ. من أجل تحسين دقة الكشف عن أنماط الهجوم التي لها ترددات تعتمد على الوقت مثل هجوم nerisbotnet، تم تصميم ثلاث ميزات إدخال جديدة وهي sa_nsessions_T و da_nsessions_T و sa_da_nsessions_T عن

طريق تجميع التدفقات بناءً على عنوان المصدر وعنوان الوجهة وسمات الطابع الزمني باستخدام نافذة زمنية مدتها دقيقة واحدة. وبعد المعالجة المسبقة لميزات المدخلات، يتم تحديد أهم 45 ميزة إدخال. علاوة على ذلك، يتم تعلم معلمات النموذج باستخدام عدد كبير من التدفقات ذات التصنيف متعدد الفئات من مجموعة بيانات UGR'16 العامة [32]. تم تحسين المعلمات الفائقة للنموذج للحصول على أفضل أداء من حيث الدقة ومعدل الاكتشاف والوقت اللازم لتدريب النموذج المقترح.

أكدت النتائج التجريبية الأداء العالي للنموذج المقترح عند اختباره على تدفقات الشبكة غير المرئية من مجموعة بيانات UGR'16. تتكون بنية الشبكة المثلى من طبقة إدخال واحدة وثلاث طبقات مخفية وطبقة إخراج واحدة. حقق النموذج دقة 99.773٪، ومعدل موجب خاطئ أقل من 1٪، ومساحة تحت منحنى خاصية تشغيل المستقبل (ROC–AUC) تبلغ 0.999915. أيضًا، تبلغ دقة الكشف للمصنف متعدد الفئات 99.58٪ ومعدل الكشف لجميع الفئات الفردية أعلى من 99٪ باستثناء هجوم NerisBotNet حيث يبلغ معدل الكشف 94.70٪. عند مقارنة النموذج المقترح بالنماذج الحديثة الأخرى في الأدبيات [33]، [52]، والتي تم تقييمها على نفس مجموعة بيانات UGR'16، تظهر النتائج التجريبية أن النموذج المقترح يتفوق على النماذج الأخرى من حيث معدل الاكتشاف والدقة.

Table of Contents

List of Figures

List of Tables

List of Equations

Abbreviations

ANIDS	Anomaly-based Network Intrusion Detection System
AUC	Area under the curve
CIC-IDS2017	Canadian Institute for Cybersecurity Intrusion Detection Evaluation
CM	Confusion Matrix
CNN	Convolutional Neural Networks
DNN	Deep Neural Network
DoS	Denial of Service
FaaC	Feature as a Counter
FFDNN	Feedforward Deep Neural Networks
FN	False Negatives
FP	False Positives
FPR	False Positives Rate
FTP	File Transfer Protocol
HTTP	Hypertext Transfer Protocol
ICMP	Internet Control Message Protocol
KDD99	Knowledge Discovery and Data Mining 1999
KNN	K Nearest Neighbor
LAN	Local Area Network
LR	Logistic Regression
MDI	Mean Decrease in Impurity
MLP	Multilayer Perceptrons
MSNM	Multivariate Statical Network Monitoring
NSL-KDD	Network Security Laboratory–Knowledge Discovery and Data Mining
RF	Random Forrest

RNN	Recurrent Neural Networks
ROC	Receiver Operating characteristic curve
SNIDS	Signature-based Network Intrusion Detection System
SSH	Secure Shell
SVM	Support Vector Machines
TN	True Negatives
TP	True Positives
TPR	True Positives Rate
TCP	Transmission control protocol
UDP	User Datagram Protocol
UGR'16	University of Granada 2016

Chapter One: Introduction

Chapter Outline

Chapter One: Introduction

1.1 Background

Network Intrusion Detection System (NIDS) is a cybersecurity tool for automated monitoring and analysis of network traffic. If abnormal network activities or intrusions are detected, the system administrator will be notified, or a security event will be collected by the security information and event management system [1]. Although there are many options for placement of NIDS, it is usually placed at the perimeter of the network behind the external firewall to monitor outbound and inbound network traffic as illustrated in Figure 1. Hence, it could detect intrusions and security policy violations that could not be detected by a firewall [2].

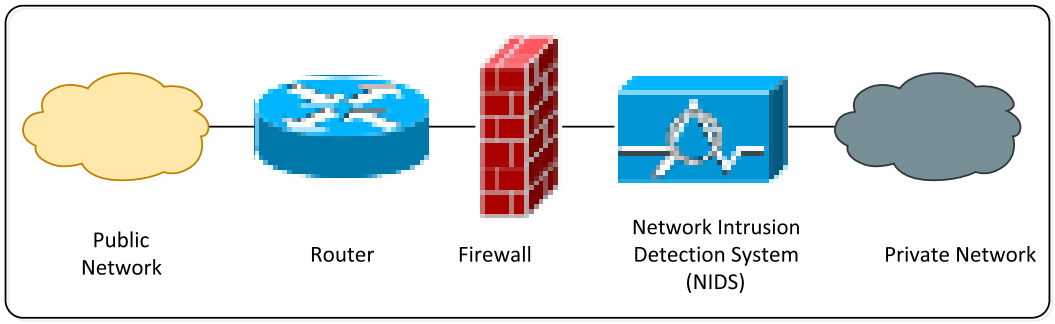


Figure 1: NIDS placement in the network

NIDS is usually classified according to the detection approach. Signature-based Intrusion Detection System (SNIDS) detects intrusion based on recognized patterns or known signatures of malware. Anomaly-based Intrusion Detection System (ANIDS) usually builds a profile of normal behavior, and an alarm is triggered when the traffic deviates from the baseline or normal behavior. Thus, ANIDS often relies on statistical analysis, soft computing, and machine learning [3].

In this research, a framework and methods are proposed to build binary ANIDS and multiclass ANIDS models based on feedforward deep neural networks. The

performance of the models will be evaluated on a UGR'16 dataset, which is built with real traffic and up-to-date attacks. The general architecture of ANIDS is depicted in Figure 2.

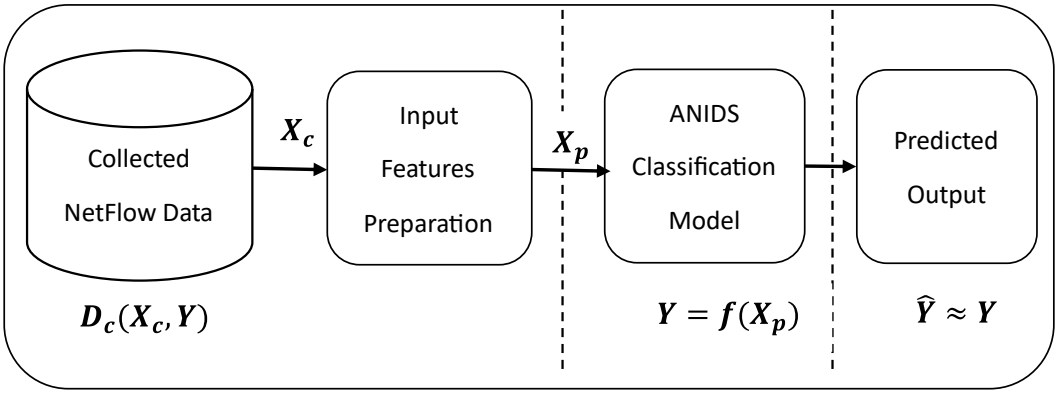


Figure 2: General architecture of ANIDS.

Where,

$\boldsymbol{D_c}$: is the dataset of collected NetFlow data.

$\boldsymbol{X_c}$: is the input vector for collected NetFlow features or attributes.

Y: is the true label for each network connection or flow.

$\boldsymbol{X_p}$: is the input vector for prepared NetFlow features or attributes.

$\widehat{\boldsymbol{Y}}$: is the predicted label for the network connection or flow.

1.2 Problem Statement

Network intrusion detection systems (NIDS) are being used to monitor and analyze inbound and outbound network traffic to identify intrusions and malicious activities in the network. While signature-based NIDS have high detection rate of known and predefined attack signatures, they underperform on detection novel or zero-day attacks[4]. As a result, anomaly-based NIDS are being used to build models of normal behavior of network traffic so that any traffic that deviates from normal behavior will be considered as anomalous or attack. However, ANIDS generate high false alarm rate when new patterns of normal

behavior are detected and also fail to detect attack patterns that are similar to normal patterns [5].

To solve the above problems, this thesis proposes a model called anomaly-based network intrusion detection system using feedforward deep neural networks (ANIDS-DNN). The model takes the collected NetFlow features as input and after the data is prepared the deep learning algorithm classifies each network connection as either normal or anomaly. The model has been trained on a multiclass labeled traffic from the public UGR'16 dataset to improve the model's detection accuracy and minimize false positive rate. Furthermore, new features were extracted to capture the temporal representation from the sequence of network flows using source and destination IP addresses and timestamps of the flows. In addition, collected features are transformed and encoded, and important features are selected.

1.3 Motivation

Network intrusion detection systems are vital tools to accurately detect intrusions and malicious attacks that jeopardize the confidentiality, integrity, and availability of computer networks. Besides, the rapid advancement in information technology, and the dramatic increase in quantity and complexity of cyberattacks, motivated the research in cybersecurity data science [6]. In this thesis a model is proposed to develop an automated and intelligent anomaly-based network intrusion detection system using deep neural networks.

1.4 Objectives

This research aims at building and training efficient and reliable binary and multiclass classifiers for anomaly-based network intrusion detection systems using

feedforward deep neural networks (ANIDS-DNN). An important objective of this research is to determine what methods and techniques provide the best discriminating power. The classifier should be able to discriminate between benign and malicious network flows with a high rate of detection and a low rate of false alarms. Furthermore, another objective is to tune the proposed models to increase the detection accuracy of unseen abnormal network flows.

1.5 Research Questions

To build a reliable and efficient deep learning based ANIDS, the following research questions must be answered:

(1) How to handle the class imbalance in the dataset to avoid bias toward the majority class?

(2) How to prepare the data to be suitable for a deep learning algorithm?

(3) How to extract relevant features to handle temporal representation and frequency of network connections or flows?

(4) Which features are more effective for certain types of attacks?

(5) What is the best architecture for deep neural networks?

(6) What are the most effective hyperparameters?

(7) What are the performance metrics to test the proposed models?

(8) How is the performance of the proposed models based on performance metrics?

1.6 Assumptions

This study was carried out based on the following assumptions:

- **Intrusion Detection System**: Anomaly-based network intrusion detection system.

- **Algorithm**: Supervised feed-forward deep neural networks.
- **Input variables:** features of network traffic flows or connections. The features are extracted from packet headers using the Cisco NetFlow version 9.
- **Output variables:** for binary classifiers, the output is a decision on the behavior of a network connection if it is background or anomaly. For a multiclass classifier, the output is a class label of the flow which is the background if the connection is normal and the type of attack if the connection is anomalous. The security attacks included are Anomaly-Spam, Anomaly-SSH-Scan, Anomaly-UDP-Scan, NerisBotNet, DoS and Scan.
- **Dataset**: UGR'16 dataset.

1.7 Thesis Contribution

The four main contributions of this thesis are:

1. Presents an ANIDS model for detection of unknown or zero-day attacks in network flows using feedforward deep neural networks as a classification algorithm. The architecture and hyperparameters of the model such as number of hidden layers, number of neurons in each layer, learning rate, and batch size are optimized for best performance in terms of accuracy, recall, precision, and training time of the model.
2. The parameters of the model (weights and biases) are learned and validated using large number of instances (3777662 flows) from selected and preprocessed balanced subset of UGR'16 dataset.
3. To improve detection of attack patterns that have time-dependent frequencies such as nerisbotnet attack, the following three new features are devised by aggregating

the flows based on source address, destination address, and timestamp attributes using a time window of one minute:

- **sa_nsessions_T** : total number of flows sent by same source IP address (sa) of designated flow within a time window of one minute.
- **da_nsessions_T**: total number of flows received by same destination IP address (da) of designated flow within a time window of one minute.
- **sa_da_nsessions_T**: total number of flows sent by same IP address (sa) of designated flow and received by same destination IP address (da) of designated flow within a time windows of one minute.

4. The proposed model achieved high performance when evaluated on unseen instances of the UGR'16 dataset using state-of-the-art performance metrics such as accuracy, recall, precision, F1-score, and Receiver Operating Characteristic – Area Under the Curve (ROC-AUC). The experimental results shows that the performance of the proposed model outperforms other recent models [33], [52], which are evaluated on same UGR'16 dataset in terms of precision and recall.

1.8 Thesis Structure

The rest of the thesis is organized into the following chapters:

- **Chapter 2**: provides background information about network intrusion detection systems, the TCP/IP communication protocol, Capturing and analyzing Netflow features, machine learning, and deep learning, intrusion detection datasets and a review of the state-of-the-art anomaly-based network intrusion detection models and a summary for the chapter.
- **Chapter 3**: the architecture of proposed ANIDS-DNN model is presented and the design and implementation of proposed model is explained in detail.

- **Chapter 4**: the experimental setup, performance evaluation metrics and the results are presented and analyzed.
- **Chapter 5:** first, the results are discussed then the study is concluded. Also, recommendations and future work are presented.

1.9 Summary

This chapter presents an overview about network intrusion detection systems and the two approaches to detect intrusions. Then the problem statement of this thesis is defined. Also, the motivation for using deep learning algorithms for building ANIDS is summarized. After that, the objectives of the research are introduced followed by the research questions that should be answered, and lastly the contributions of this study to the state-of-the-art are listed.

Chapter Two: Network Intrusion Detection

Chapter Outline

Chapter Two: Network Intrusion Detection

2.1 Background

2.1.1 Network Intrusion Detection

Intrusion Detection Systems (IDS) are security tools for detecting malicious activities in networks and systems. IDS can be classified based on the protected system as host-based IDS (HIDS) and network-based IDS (NIDS) [7]. HIDS inspect data that originates from the host system and audit sources, such as operating system logs, server logs, firewalls logs, application system audits, or database logs. While network-based IDS monitors the network traffic that is extracted from a network through packet capture, NetFlow, and other network data sources. NIDS classified according to the technique of detection as either signature-based NIDS or anomaly-based NIDS [1], [4], [8], [9].

Signature-based detection compares a hash of payload to a database of known malicious signatures. Whereas Anomaly-based detection detects abnormal network flows compared to a normal profile or baseline. Therefore, ANIDS is better in detecting unknown or zero-day attacks than SNIDS but generates more false alarms in case a change in normal network behavior occurs. ANIDS use statistics, machine learning or data mining techniques to identify intrusions while SNIDS usually use pattern recognition, implication rules or data mining [4], [10]. Figure 3 shows a taxonomy of Intrusion detection systems.

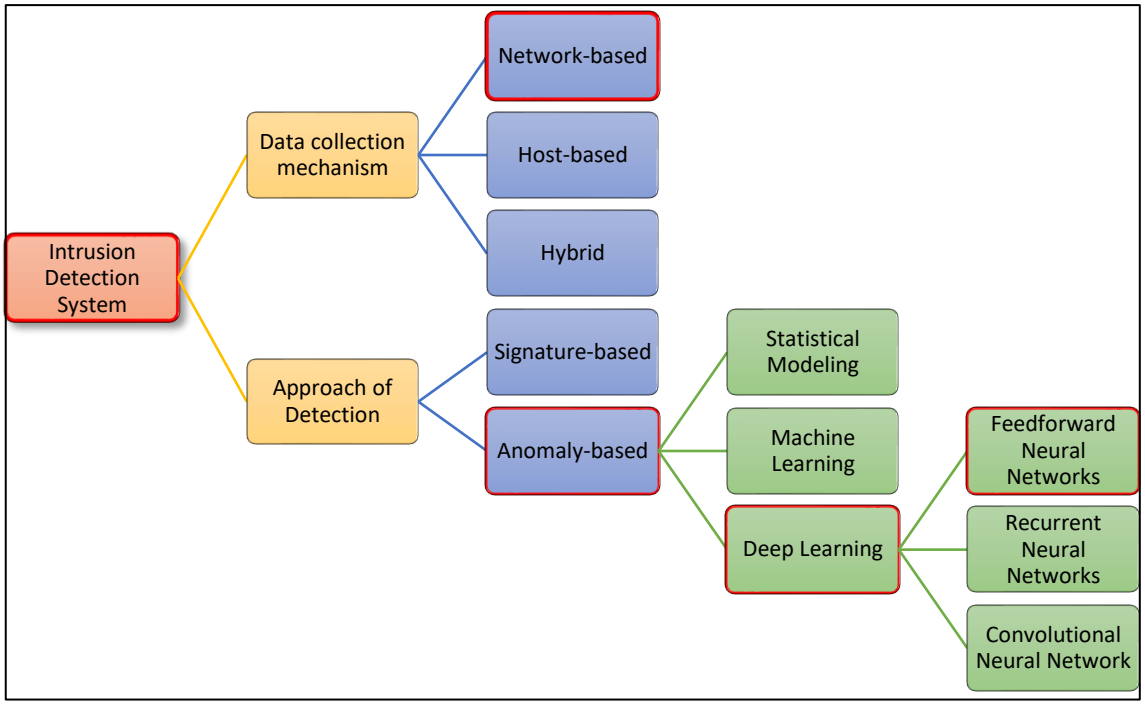


Figure 3: Intrusion detection systems taxonomy

2.1.2 Capturing Network Flows

Many tools and applications are used by network administrators and security analysts to capture and analyze network traffic. Some packet analyzers such as Wireshark and Tcpdump capture and analyze full packets payload. However, capturing and storing full packet payload to be analyzed for intrusions is challenging especially in large networks due to the massive amount of traffic. Another issue is privacy and confidentiality since the payload usually contains encrypted data.

The flow or a connection is a sequence of packets starting and ending at a time duration between a source IP address (**sa**) to a destination IP address (**da**) under certain protocol (**pr**).

NetFlow is a built-in feature in routers and switches. It is used to capture traffic statistics per flow and extract features from packet headers only. The flow collector aggregates IP packets belonging to a single connection and summarizes the information

about the sequence of packets into a network flow record. Figure 4 shows the general architecture for NetFlow collection [13].

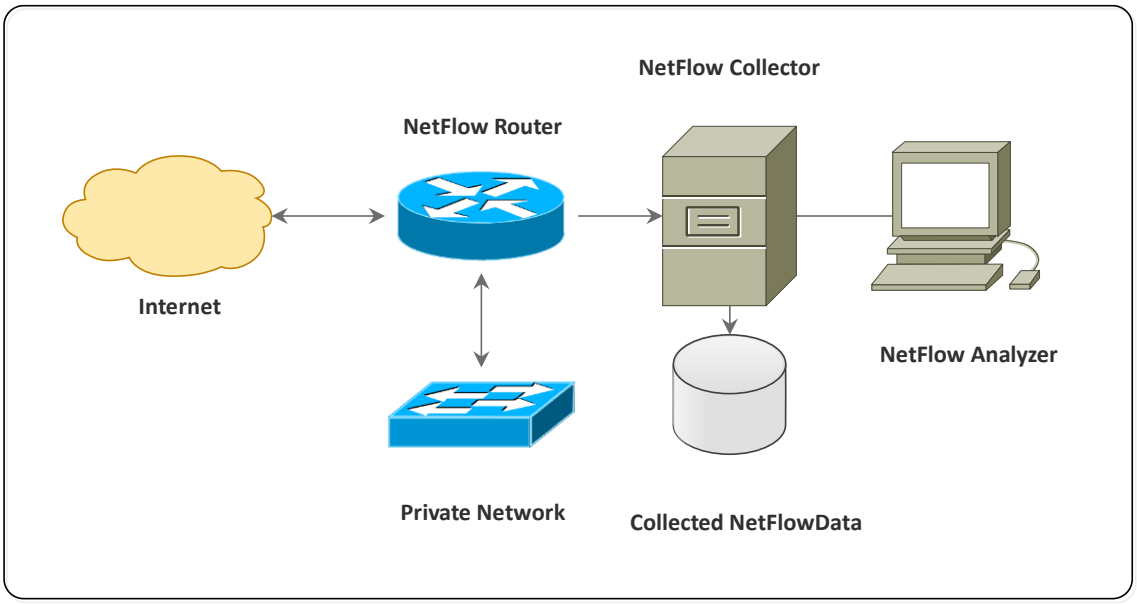


Figure 4: NetFlow architecture

For connectionless protocols, like UDP, a timeout value is used to determine the flow duration (**td**). The following five-tuple identify a network flow: the source IP address (**sa**), the destination IP address (**da**), the source port (**sp**), the destination port (**dp**), and the transport protocol (**pr**).

Moreover, there are two types of network flows: unidirectional and bidirectional. In the unidirectional case, the packets from network node A to network node B are aggregated into one flow record, and the response packets from B to A are aggregated into another flow record. In the bidirectional case, all the packets are aggregated into a single flow record [14]. Figure 5 illustrates the process diagram for capturing NetFlow data from the network and Figure 6 shows the creation of Netflow record.

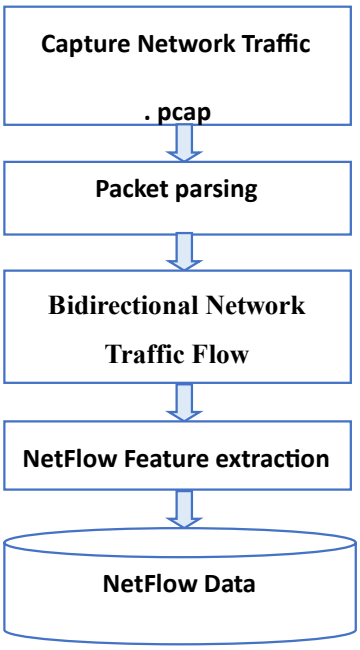


Figure 5: NetFlow process diagram

The following features are part of traffic statistics collected for network connections or flows:

- Source IP addresses (**sa**).
- destination IP addresses (**da**).
- TCP/UDP source ports (**sp**) and destination ports (**dp**).
- Number of bytes (**byt**) and packets (**pkt**) in the flow.
- IP type of service (**ToS**).

Depending on the configuration of the flow exporter, additional attributes may be added to flows, such as flow time duration (**td**), timestamp (**te**), and TCP flags (**flg**).

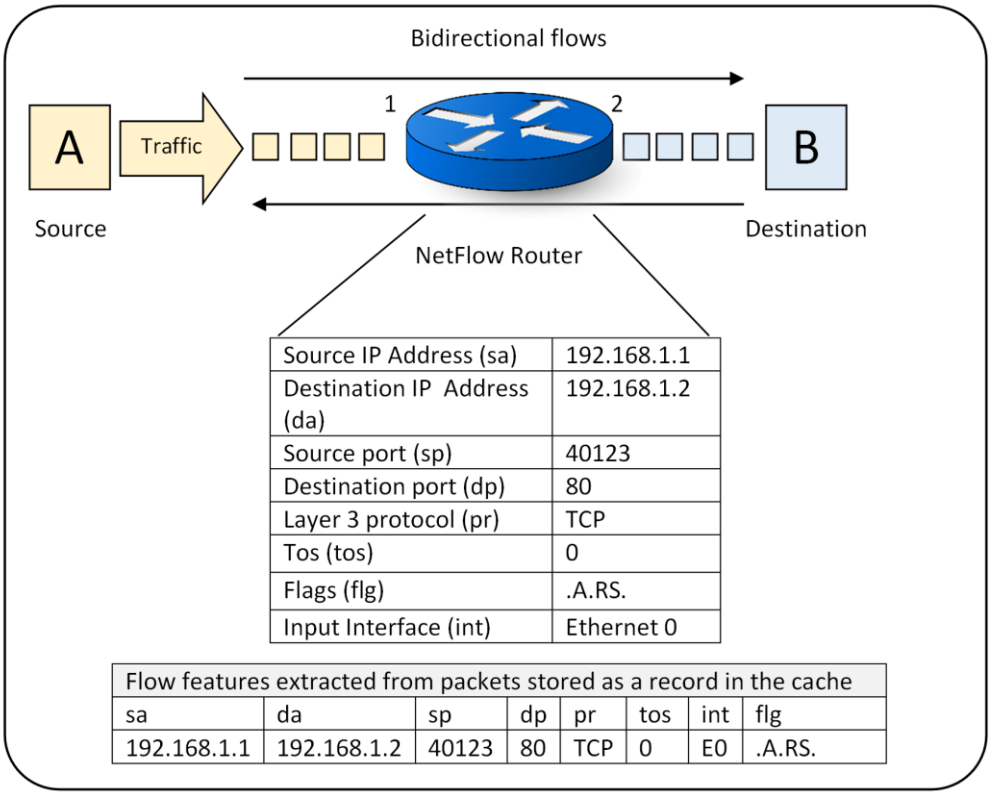


Figure 6: NetFlow record creation

2.1.3 TCP/IP Packet Metadata

TCP/IP is the standard communication protocol used when communicating between networks and the Internet. It divides the functionality of the network into four layers. The top-level layer is the application layer, followed by the transport layer, then the Internet layer and the lowest layer is the network interface layer [11]. Table 1 presents the layers of TCI/IP reference model and examples of protocols in each layer .

Table 1: TCP/TP reference model with examples of protocols in each layer

TCP/IP Model	Protocols
Application layer	DNS, DHCP, FTP, HTTP, HTTPS, NTP, SSH, SMTP, SNMP, Telnet, TFTP, POP3, IMAP
Transport layer	TCP, UDP
Internet layer	ICMP, IGMP, IP4, IP6
Network interface layer	Ethernet, IEEE 802.11

The application layer provides network services such as hypertext transfer protocol (**flag_HTTP**) and file transfer protocol (**flag_FTP**). Each service is represented by a port number ranging from **0** to **65536**. The transport layer maintains end-to-end communication across the network. Two protocols are in this layer. Transmission control protocol (**protocol_TCP**) and user datagram protocol (**protocol_UDP**). The Internet layer is responsible for exchanging packets across network boundaries. Protocols like Internet protocol, and Internet Control Message Protocol (**protocol_ICMP**) are in this layer. Finally, the Link layer or network access layer moves packets between networks.

Transmission Control Protocol (TCP) Session

TCP is a connection-oriented protocol that establishes a reliable logical connection between transport layers at two network nodes before transferring data. TCP provides flow control, error control, and congestion control. Each TCP connection is identified by a five-tuple consisting of the source address (**sa**), a destination address (**da**), source port (**sp**), destination port (**dp**), and the protocol (**pr**). Under normal conditions, a listening process must be running on the receiving host to accept and respond to TCP connection requests. If a TCP packet is received that is destined for a port with no listeners, a TCP RST (**flag_Rst**) packet will be sent back to the source by the receiving host[12].

To establish a new connection the source host sends SYN packet (**flag_Syn**), and the receiver replies with SYN/ACK packet, then the initiator of the connection replies with ACK packet (**flag_Ack**). The process is known as a three-way handshake as depicted in Figure 7.

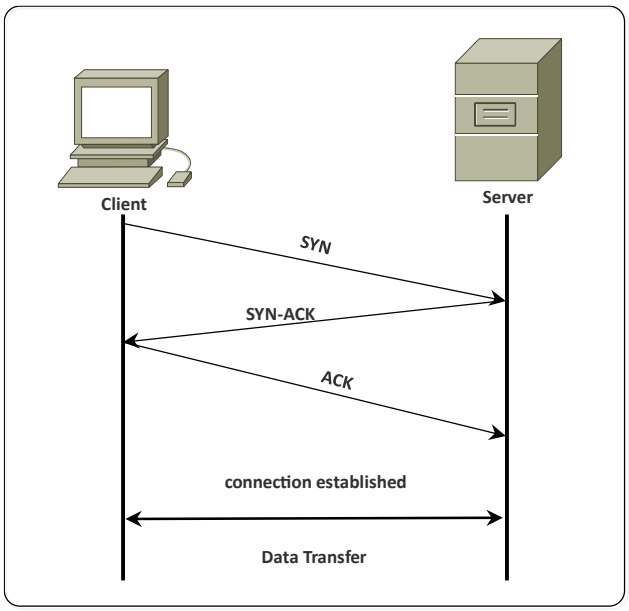


Figure 7: TCP three-way handshake

TCP Header:

TCP header holds information about the connection and the data that is being sent. The header of a TCP segment can range from 20-60 bytes. If there are no options, a header is 20 bytes else it can be the utmost of 60 bytes. TCP establishes and terminates a connection using control flags which are in the header of the packet. There are six flags:

- URG (**flag_Urg**): Urgent flag indicates a packet that should have priority.
- ACK (**flag_Ack**): Acknowledge flag indicates that the data was received.
- PSH (**flag_Psh**): Push flag to inform that data should be sent immediately.
- RST (**flag_Rst**): Reset flag to abort the TCP connection.
- SYN (**flag_Syn**): Synchronize the sequence numbers.
- FIN (**flag_Fin**): Finish flag to close the TCP connection [1].

Figure 8 illustrates the structure of the TCP header.

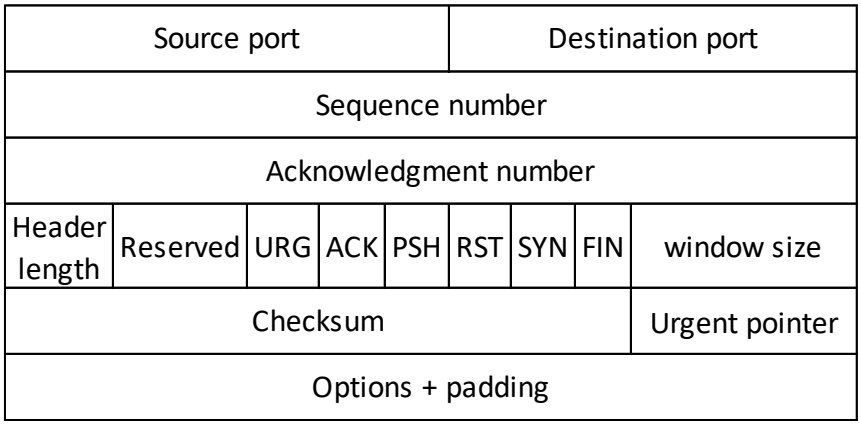


Figure 8: TCP header format

User Datagram Protocol (UDP) Header

UDP (**protocol_UDP**) is a connectionless protocol that cannot guarantee the delivery of transmitted data. UDP is useful in streaming applications. The header of UDP is shown in Figure 9.

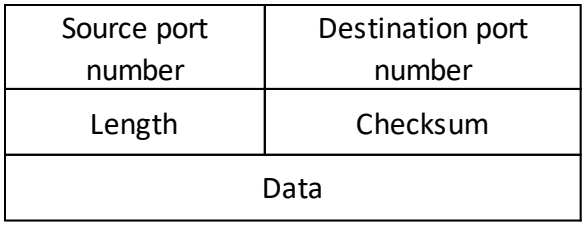


Figure 9: UDP header format

Internet Protocol (IP) Header

Internet protocol is responsible for logical addressing where each node is assigned an address. IPv4 (**protocol_IPv4**) address consists of 32 bits and the Ipv6 address consists of 128 bits. Routing packets in the internet layer are based on the source address, a destination address, and control data. IP header contains information about the IP version, source IP address (**sa**), destination IP address (**da**), and time-to-live as shown in Figure 10.

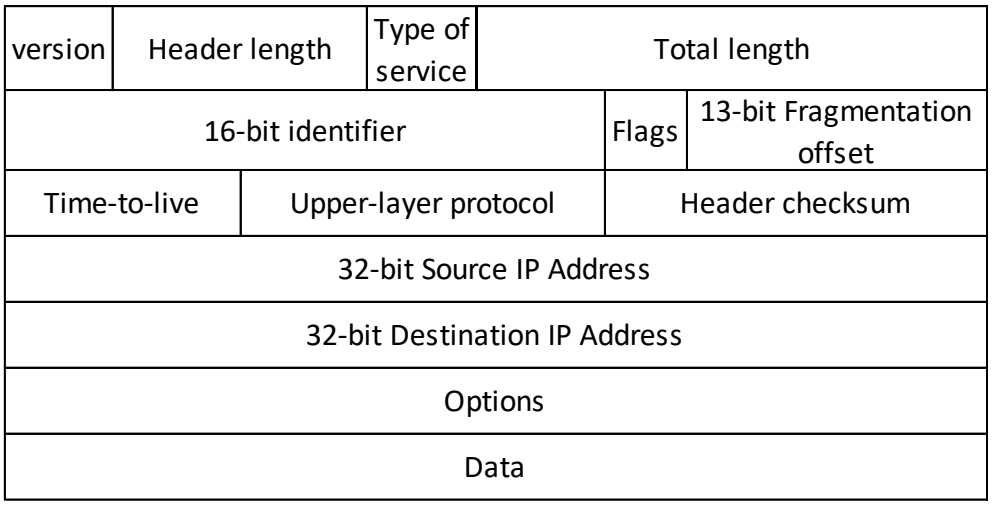


Figure 10: IPv4 datagram format

2.1.4 Analyzing Network Flows

Some common methodologies used to analyze network flows are:

a) **Statistics:** the flow features and data are used to build a statical model for analyzing network flows. For instance, Garcia et al. [15] proposed a framework for network security anomaly detection called multivariate statistical network monitoring (MSNM).

b) **Profiling:** where network flows are monitored and analyzed based on user profiling, application profiling, host profiling, and network profiling [14].

c) **Behavior-based:** in this approach the normal behavior of network flows is learned, then any network flows that deviate from the normal behavior are considered abnormal.

d) **Visualization:** in this approach network traffic is explored after visualization. Engle et al. introduced a visualization tool for network-wide intrusion detection [16].

e) **Machine learning:** both supervised and unsupervised machine learning algorithms are used for anomaly detection of network traffic data [7].

f) **Deep learning:** recently deep learning is used to automate the detection and classification of network attacks since deep learning algorithms are built to deal with a large amount of data efficiently.

2.1.5 Network Attacks

Computer Networks are vulnerable to many forms of cyberattacks. The most common attacks are:

Network Scanning: also called reconnaissance attack which is an attempt to discover open ports on the nodes of the network to be utilized in attacks.

Denial of Service Attacks: sending many connection requests to an application server to disrupt the normal activity of a service [10].

Botnets: a network of internet-connected compromised devices (bots) that are recruited and controlled by a botmaster and used to launch attacks on the network like DoS & spam[17], [18].

Spam: sending thousands of messages to users. These messages are sent by fake or hacked profiles, and often include unreal advertisements and phishing links.

Access attacks: trying to gain unauthorized access to network resources by obtaining the network credentials using brute force attacks.

2.1.6 Anomaly-based Detection

In the network intrusion detection context, security-related anomalies occur when the network traffic is different from the normal traffic. Malicious network activity can cause three types of anomalies:

- point anomaly if one instance of network data is anomalous.
- Contextual anomaly if one instance is anomalous but in a certain context such as anomalies in timeseries data.

- Collective anomaly is when a collection of instances is anomalous, but an individual instance is not anomalous. For instance, a DoS can generate a lot of SYN flags or a high number of flows to a certain IP address. In this case, an aggregation of the flows during a time window is a collective anomaly.

Many techniques were used to detect and classify anomalous traffic [19]. However, machine learning and deep learning techniques were lately the focus of the research.

2.1.7 Machine Learning

Due to the complexity of modeling network anomaly intrusion detection using statistical and traditional techniques, especially with a high volume of data and the need to detect unknown attacks, researchers used machine learning algorithms to build models that could classify normal and anomalous traffic or classify intrusions based on the type of attack[20].

To develop a machine-learning model, a dataset is required. The dataset is divided into a training set and a testing set, where the model is trained on the training set and evaluated on the test set. The design of a machine-learning model is usually managed by a framework such as scikit-learn[21] and TensorFlow [22].

Machine learning algorithms are categorized based on the availability of the data and labels and also to the tasks to be achieved by the algorithms as in Figure 11.

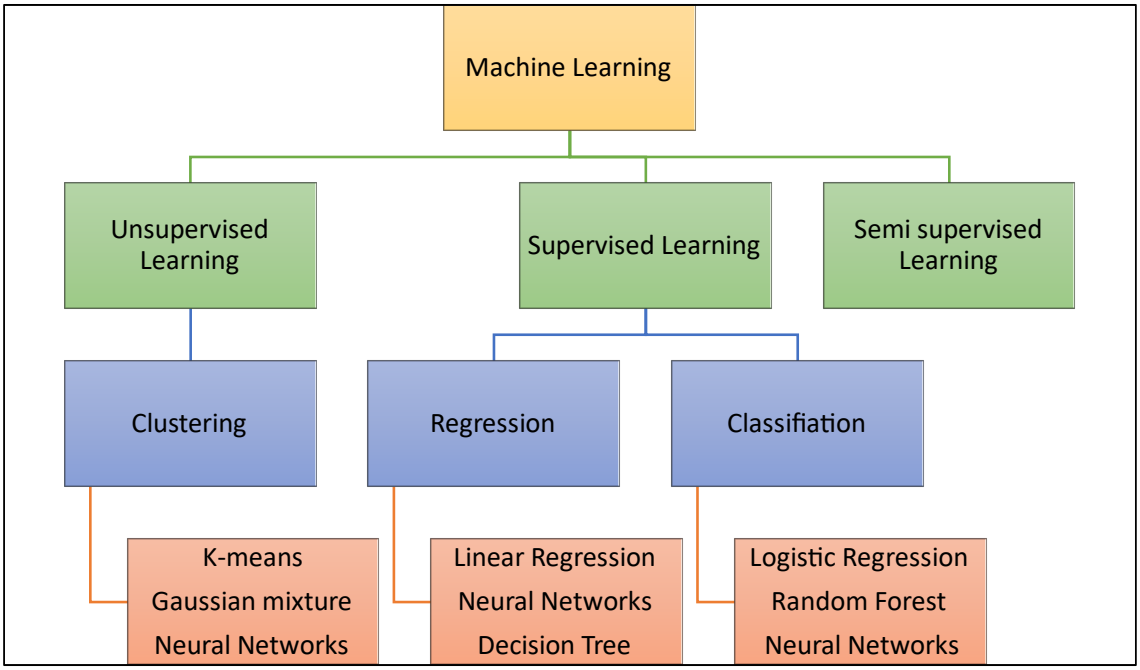


Figure 11: Taxonomy of machine learning algorithms

Machine learning taxonomy based on the dataset requirements

As depicted in Figure 12, machine learning algorithms are categorized according to the availability of labeled examples in the dataset as follows:

Supervised Learning: is a machine learning paradigm used to train a model or a function that relates the output variable (**Y**) to input variables (**X**) as formulated in equation 1.

$$\boldsymbol{Y} = \boldsymbol{f}(\mathbf{X}) \tag{1}$$

so that the output ($\widehat{\boldsymbol{Y}}$) could be predicted from the input with minimal error. To train the model a labeled dataset is required. The dataset is a set of examples from a specific domain where each example includes the input variables or features and output variable or a target. A supervised machine learning algorithm is used to learn model parameters from the dataset by minimizing the loss between the true labels and the predicted target. In intrusion detection the input variables are the Netflow features extracted from network traffic and the target is the class label.

Semi-supervised: is a machine learning paradigm that combines both supervised and unsupervised learning. It is used when we only have a limited amount of labeled data and a large amount of unlabeled data.

Unsupervised: this machine learning technique is used when only unlabeled data is available.

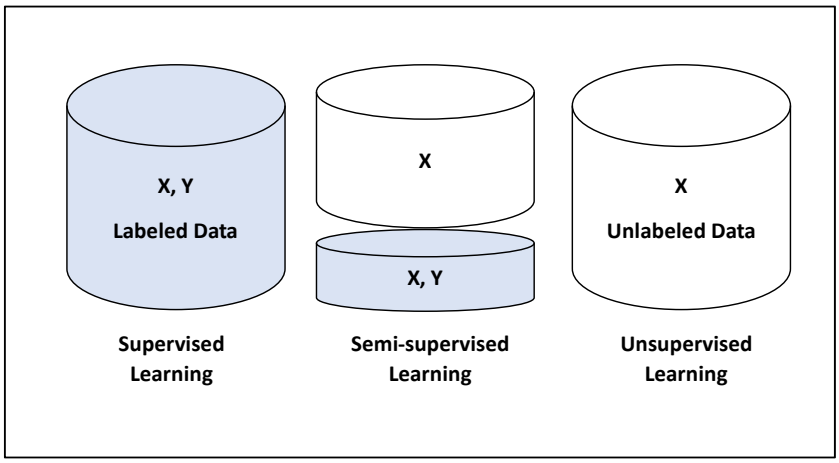


Figure 12: Taxonomy of machine learning based on dataset labels

Machine learning taxonomy based on the task to perform

Based on tasks they can perform machine learning algorithms are categorized as**:**

Regression**:** supervised learning algorithm is used to train a model to predict the continuous values of a target.

Classification: supervised learning algorithm is used to train a model to classify data. The classifier could be a binary classifier or a multiclass classifier. For example, in intrusion detection, a binary classifier model can classify network connections as benign or malicious. For multiclass classification, network traffic can be categorized according to the type of network attack in addition to normal class.

Clustering: data is partitioned into groups based on some measure of similarity or shared characteristic so that objects in the same cluster are similar and objects in other clusters are different. Clustering is used when we have an unlabeled dataset [23].

Machine Learning Classification algorithms:

a) **Logistic Regression (LR):** is a basic linear machine learning algorithm used for classification especially when the input values are categorical, and data can be separated with a single linear boundary. By learning the parameters of the model, it can predict the class based on given input features as formulated in equation 2.

$$\log\left(\frac{\hat{y}}{1-\hat{y}}\right) = \sum_{i=1}^{n} x_i w_i + b \qquad (2)$$

Where x_i is input vector, $\hat{y}$ is the predicted output, w_i is the weight vector, **b** is the bias, and g is the sigmoid activation function.

b) **Random Forrests (RF):** is a machine learning algorithm that can be used in classification problems. The algorithm builds a large number of random and uncorrelated decision tree classifiers. The class with high votes will be selected by random Forrest algorithm. Also, it can calculate the importance of the features by calculating the Gini score for each feature [24].

c) **K Nearest Neighbor (KNN):** Categorize a sample based on its nearest k neighbors' class. If they belong to same class, then the sample has high probability of belonging to the class [7].Thus the algorithm uses some distance metric such as Euclidean distance to classify samples.

d) **Support Vector Machine (SVM):** Classifies data by finding the linear decision boundary (hyperplane) that separates all data points of one class from those of the other class. The best hyperplane for an SVM is the one with the largest margin between the two classes when the data is linearly separable. If the data is not linearly separable, a loss function is used to penalize points on the wrong side of the hyperplane. SVMs sometimes use a kernel transform to transform nonlinearly separable data into higher dimensions where a linear decision boundary can be found [7].

Artificial Neural Networks (ANNs): Artificial neural networks are a subset of machine learning algorithms that resemble a biological neural network. A feedforward network consists of an input layer, one or more hidden layers, and an output layer. Each of the hidden layers has several nodes called neurons as depicted in Figure 13. Neurons are connected with the previous and next layers via links where each link has a weight. Each neuron aggregates the inputs from the previous layer and then applies an activation function and delivers the output to the next layer [23].

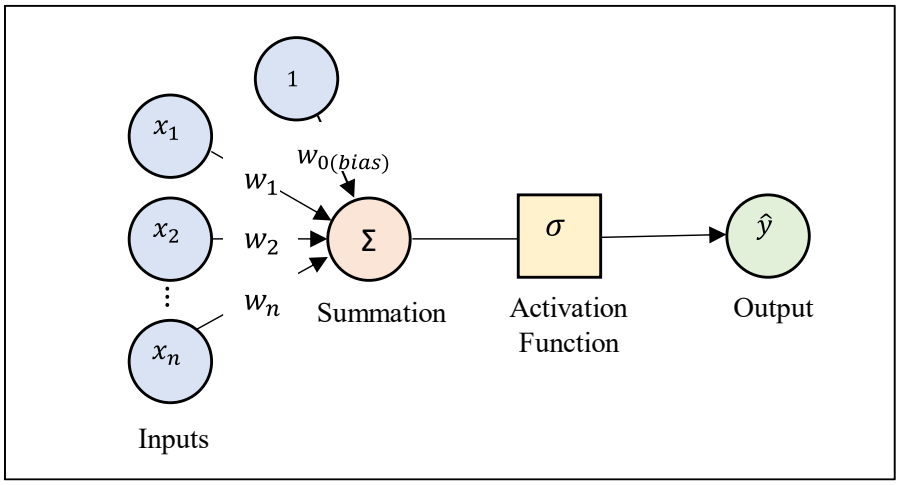


Figure 13: Architecture of artificial neuron

2.1.8 Deep Learning

Deep learning is a subset of machine learning algorithms based on artificial neural networks. DNN models use a neural network with two or more hidden layers to apply a nonlinear transformation on input data. A higher-level feature representation is extracted with each hidden layer. In supervised learning, the DNN model maps the input feature vector x to a label y through intermediate transformation [25]–[27].

However, the most common types of DNN are:

Feedforward Deep Neural Network (DNN): is a deep learning algorithm where the information flows from the input to hidden layers and then to the output without feedback

connections as illustrated in Figure 14. It is used for the deterministic representation of input data especially when the input features are fixed in length.

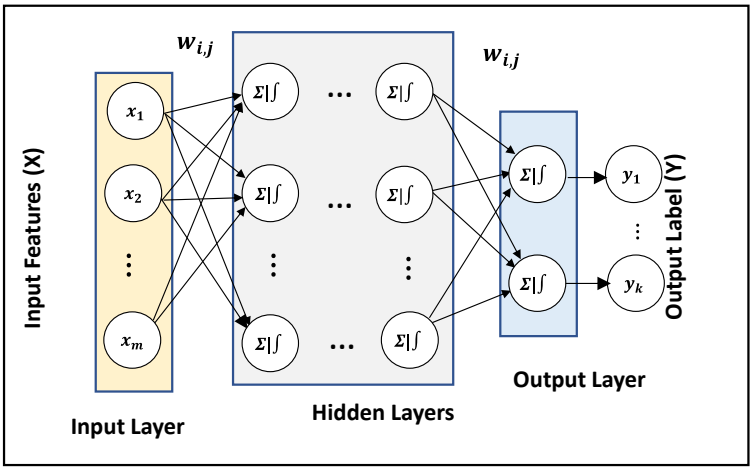


Figure 14: Feedforward deep neural network architecture

Recurrent Neural Network (RNN): is a deep neural network for processing sequential data. It can process two-dimensional inputs with variable size such as images or videos. It can model the stochastic representation of data since there are feedback connections between hidden layers as depicted in Figure 15.

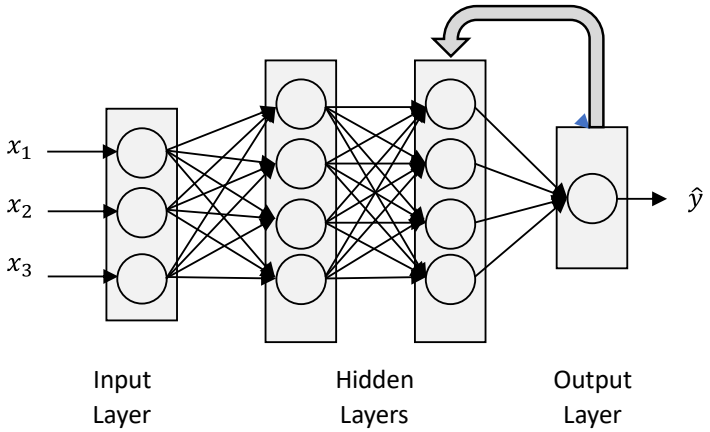


Figure 15 : Recurrent neural network architecture

Convolutional Neural Network (CNN): is a deep learning algorithm used in classification problems to capture patterns in data when the input is two-dimensional or

of variable size such as an image or a video. The spatial and temporal associations in input features can be learned by applying relevant filters. In convolutional layer the convolution operation is a matrix multiplication between a filter or kernel and the input. CNN is more efficient when applied to problems with very large number of input variables because the number of parameters to be learned are reduced compared with feedforward deep neural networks [25], [28] . Figure 16 depicts the architecture of a convolutional neural network.

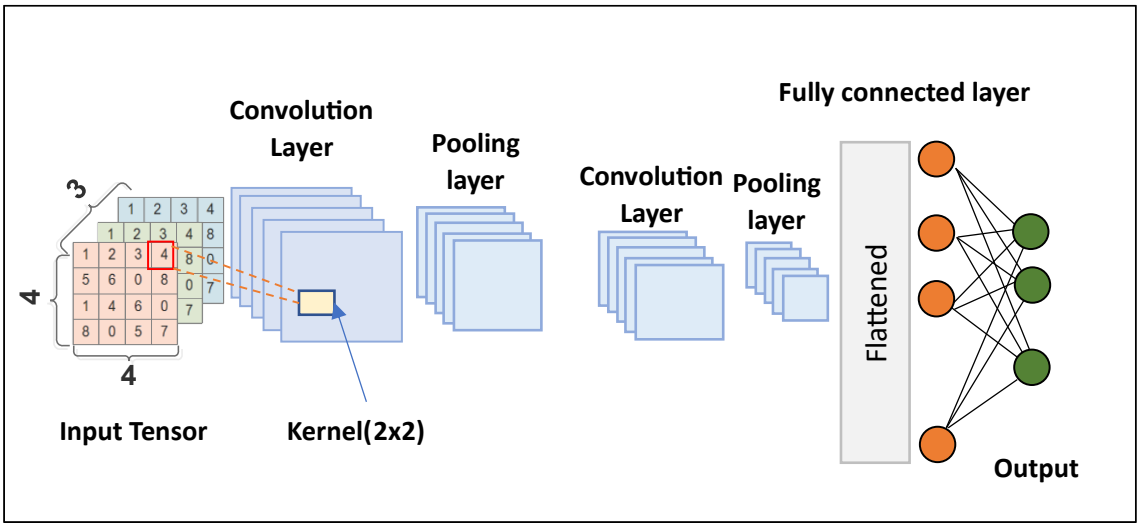


Figure 16: Architecture of convolutional neural network (CNN)

2.1.9 Intrusion Detection Datasets

When machine learning and deep learning algorithms are used to develop ANIDS models a large amount of network data is required to train and test the capability of the models to detect intrusions. Besides, a publicly available benchmark dataset is also essential to compare the efficacy and performance of different models and approaches. However, due to the privacy issues and the time and resources required to create a labeled network intrusion dataset, only few datasets are available in public for researchers to assess their models.

Some of the known benchmark datasets are introduced in this subsection as follows:

KDD Cup99: the dataset was created by Defense Advanced Research Project Agency in 1998 and known as the Knowledge Discovery and Data Mining (KDD99) [29]. The dataset was collected using multiple computers connected to the Internet to model a small US Air Force base of restricted personnel. Network packets and host log files were collected. Lincoln Labs built an experimental testbed to obtain 2 months of TCP packets dump for a Local Area Network (LAN), modelling a usual US Air Force LAN. They modelled the LAN as if it were a true Air Force environment but interlaced it with several simulated intrusions. The collected network packets were around four gigabytes containing about 4,900,000 records. The test data of 2 weeks had around 2 million connection records, each of which had 41 features and was categorized as normal or abnormal. Since the dataset doesn't contain recent attacks, it is considered outdated.

NSL-KDD: The Network Security Laboratory–Knowledge Discovery and Data Mining (NSL-KDD) is an improved version of KDD99 dataset. The duplicate records and errors were fixed, and the dataset was rebuilt in 2009. NSL-KDD consists of 125,973 records for training and 22,544 records for testing. The size of the NSL-KDD dataset is sufficient to make it practical to use the whole NSL-KDD dataset without the necessity to sample randomly. This has produced consistent and comparable results from various research works. The NSL_KDD dataset comprises 22 training intrusion attacks and 41 features. In this dataset, 21 attributes refer to the connection itself and 19 attributes describe the nature of connections within the same host[30].

UNSW-NB15: UNSW-NB15 is a network intrusion dataset. It contains nine different attacks, including DoS, worms, Backdoors, and Fuzzers. The dataset contains raw

network packets. The number of records in the training set is 175,341 records and the testing set is 82,332 records from the different types, attack and normal [31].

UGR'16: The dataset is built with real traffic and up-to-date attacks. These data come from several NetFlow v9 collectors strategically located in the network of a Spanish ISP. It is composed of two differentiated sets of data that are previously split in weeks. The main advantage of this dataset over previous ones is its usefulness for evaluating IDSs that consider long-term evolution and traffic periodicity. Models that consider differences in daytime/night or labor weekdays/weekends can also be trained and evaluated with it [32], [33].

CIC-IDS2017: the Canadian Institute for Cybersecurity Intrusion Detection Evaluation (CIC-IDS2017) dataset comprises both benign behavior and details of new malware attacks: such as Brute Force FTP, Brute Force SSH, DoS, Heartbleed, Web Attack, Infiltration, Botnet and DDoS. This dataset is labelled based on the timestamp, source and destination IPs, source and destination ports, protocols, and attacks. A complete network topology was configured to collect this dataset which contains Modem, Firewall, Switches, Routers, and nodes with different operating systems such as Microsoft Windows, Apple's macOS iOS, and open-source operating system Linux. This dataset contains 80 network flow features from the captured network traffic [4], [34].

2.2 State-of-the-art ANIDS Models

Recently, many approaches have been proposed to accurately identify intrusions in network traffic. The main characteristics of these approaches are:

1) Anomaly-based NIDS are better in detection of zero-day attacks than signature-based NIDS. Besides, ANIDS are better in detection attacks that span several packets than SNIDS. As a result, ANIDS models have been proposed in literature is much more than SNIDS. Furthermore, machine learning algorithms were extensively used in building ANIDS models than statistics-based or knowledge-based models because they can learn normal and attack patterns from an intrusion dataset and require less human effort and knowledge to recognize intrusion patterns [4], [5].
2) Different machine learning classification algorithms were used in building ANIDS models such as logistic regression (LR), random forest (RF), support vector machines (SVM), artificial neural networks (ANN), and deep learning algorithms such as feedforward deep neural network (DNN), recurrent neural network (RNN), convolutional neural network (CNN). However, deep neural networks are more efficient than classical machine learning techniques in identifying representations from massive amount of data due to the deep structure of the network. The hidden layers can transform features to a high dimensional space to capture representations and recognize patterns [28], [35], [36].
3) The most frequently used dataset in research is KDD Cup99, then NSL-KDD that has been developed from KDD Cup99 in the year 2009. Other datasets were released later such as UNSW-NB15, UGR'16, CSE-CIC-IDS2017, and CSE-CIC-IDS2018. Intrusion detection datasets are different from each other in size, date of release, type of attacks included, real or simulated collection environment, unidirectional or bidirectional NetFlow data, timestamps availability, and number of features. Since

machine and deep learning depend on dataset for the learning the parameters of the model and evaluation of performance choosing the dataset is important for the generalization of the model to be integrated in real network environment [4], [32], [37].

4) Proposed approaches used some or all of the following processes to develop ANIDS models: data preprocessing, feature transformation, feature selection, model training, cross validation, hyperparameter tuning and performance evaluation [33].

Table 2 presents some of the related works about machine and deep learning ANIDS models.

In [36] Jia et al. presented the NIDS model based on a deep neural network with four hidden layers. The performance of the model is tested on KDD Cup 99 and NSL-KDD datasets. The authors found that the detection rate using deep neural networks outperforms other machine learning algorithms used in similar studies in the literature.

In another work, Vinayakumar et al. [38] proposed Scale-Hybrid-IDS-AlertNet (SHIA) framework to identify host-based and network-based intrusions. They presented binary and multiclass classification DNN models for host-based and network-based intrusion detection, the models were assessed on different benchmark datasets like KDD Cup 99, NSL-KDD, UNSW-NB15, WSN-DS, and CICIDS2017. The authors concluded that deep learning algorithms outperform classical machine learning techniques in terms of accuracy. Also, hyperparameter optimization is performed to find the most effective model hyper-parameters like the number of layers in the DNN, the learning rate, the number of epochs, and the number of nodes in each layer.

Researchers also used other deep learning algorithms to model NIDS. For instance, Sstla et al. [39] presented one model using a deep convolutional network and another model using SVM to detect network intrusions. They evaluated the performance using

different types of kernels and different types of activation functions. The performance of the proposed method is evaluated on the NSL-KDD dataset. From the experimentation, higher accuracy was achieved with DCNN compared to SVM.

In another study, Ahmad et al. [5] proposed a mechanism for anomaly detection in IoT networks based on deep neural networks and assessed its performance on IoT-Botnet 2020 dataset. They found that the detection results when using a feedforward deep neural network model outperform other deep learning models such as RNN, LSTM, and convolutional DNN. Also, the model performs well with a minimal number of features which increases the model's efficiency, especially for IoT networks.

Magan-Carrion et al. [33] presented a framework for reliable machine learning based NIDS evaluation. The proposed framework suitability was tested with classical machine learning algorithms on the UGR'16 dataset. A nonlinear model using random forest classifiers achieved higher accuracy than the linear model using logistic regression classifier. Although there is no standard framework to evaluate and compare different NIDSs, the authors conclude that the proposed methodology can help researchers in building better NIDS.

Sarhan et al. [37] presented four new NIDS datasets using NetFlow features to be used in ML-based NIDS training and evaluation stages. The NetFlow datasets are named NF-UNSW-NB15, NF-BoT-IoT, NF-ToN-IoT, NF-CSE-CIC-IDS2018, and NF-UQ-NIDS. They are derived based on NetFlow features from four benchmark NIDS datasets known as UNSW-NB15, BoT-IoT, ToN-IoT, and CSE-CIC-IDS2018 using their publicly available packet capture files. Using a common ground feature set for multiple datasets will enable researchers to evaluate and test their machine learning based NIDS models on detection accuracy and ability to generalize and detect unknown attacks. However, they

found that the new datasets achieve similar binary-class detection compared with the performance of original datasets but are less efficient on multiclass classification.

In [40] Altwaijry et al. proposed binary and multiclass classification IDS models based on deep learning algorithms. The models were evaluated on the NSL-KDD dataset. The performance of both models was tested using several metrics and compared with the performance of other classical machine learning models and with similar models in the literature. However, the binary classifier BDNN achieved an accuracy of 99.5% on the training and validation set and 84.7% on the test set. This indicates the model overfitting problem that could be improved by tuning the architecture of deep neural networks. Also, their proposed multiclass model MDNN achieved 99.5% accuracy on the training set and 77.55% accuracy on the test set. However, the results for individual classes are not provided.

In another work, Al-Turaiki et al. [28] proposed binary and multiclass ANIDS models using feedforward deep neural networks named BDNN and MDNN respectively. In addition, they proposed binary and multiclass ANIDS models using convolutional deep neural networks named BCDN and MCDN respectively. The performance of the models was evaluated on both NSL-KDD [41] and UNSW-NB15 [31] datasets. The accuracy of each model on both datasets is summarized in the last row of Table 2. They conclude that the models that use convolutional deep learning outperform feedforward deep learning. Moreover, the accuracy of binary classifiers is much higher than the accuracy of multiclass classifiers. If the performance of multiclass classifiers for individual classes is shown it could help in inferring that class imbalance could be the issue. So, increasing the number of samples of minority classes would improve the performance of the model.

Table 2: Summary of state-of-the-art ANIDS models based on machine learning and deep learning

Work	Algorithms	Dataset	Accuracy
Jia et al.[36] (2018)	DNN	KDD Cup 99 NSL-KDD	99.9%
Vinayakumar et al.[38] (2019)	DNN	KDD Cup 99 NSL-KDD UNSW-NB15 WSN-DS	95%~99% 65%~75%
Sstla et al.[39] (2020)	CNN, SVM	NSL-KDD	97%
Ahmad et al.[5] (2021)	DNN	IoT-Botnet 2020	99.01%
Magan-Carrion et al.[33] (2021)	LR, RF	UGR'16 UNSW-NB15 NSL-KDD	96.4% 97.7% 75%
Sarhan et al.[37] (2020)	Extra Trees Ensemble classifier	UNSW-NB15 CSE-CIC-IDS2018 BoT-IoT ToN-IoT	99.25% 98.31% 100.00% 97.06%
Altwaijry et al.[40] (2020)	Binary-DNN Multiclass-DNN	NSL-KDD	84.7% 77.55%
Al-Turaiki et al. [28] (2021)	BDNN, BCNN MDNN, MCNN BDNN, BCNN MDNN, MCNN	NSL-KDD UNSW-NB15	84.70%, 88.81% 77.55%, 81.10% 80.63%, 90.25% 62.87%, 69.46%

According to the previous studies, anomaly-based NIDS is better in detection of novel attacks because it builds a baseline for normal behavior and the traffic that is different from baseline will be considered as an attack. To improve detection accuracy and minimize the false alarm rate machine learning and deep learning algorithms are used to learn model parameters from public intrusion detection datasets. The quality and quantity of training examples and types of attacks included in the dataset have an effect on the model performance. Deep learning algorithms are more capable of learning the representations of data by transforming data into higher dimensional space using hidden

layers of deep learning model. Also the methodology used in developing the ANIDS model has important effect on model performance.

Based on the above literature review, a feed forward deep neural network will be used to build the proposed model. This research will be carried out over the UGR'16 dataset. The UGR'16 dataset consists of real and synthetic Netflow v9 data captured from sensors within a tier-3 ISP for 4 months. It consists of 16,900 million flows. It includes relatively new types of attacks. The UGR'16 dataset will be used to train and evaluate the performance of the proposed ANIDS model.

2.3 Summary

The first section in this chapter presents a background about the network intrusion detection systems and approaches of detection. In addition, a brief review for TCP/IP communication protocol and the process of capturing network flows from network traffic is introduced. Furthermore, machine learning and deep learning algorithms and some intrusion detection datasets are explained.

Second section presented the state-of-the-art NIDS models. One of main aspects of recent works is that machine learning and deep learning are most frequently used to develop an automated and efficient ANIDS with high detection rate and low false positives. However, due to privacy and legal issues the real network traffic cannot be disclosed so researchers use public intrusion detection datasets to train and evaluate the performance of their models. KDD99 and NSL-KDD were the most used datasets to evaluate proposed NIDS despite the fact they are relatively old and don't include recent attacks. Many recent surveys [4], [5] and published works recommend using deep learning since they are capable of modeling complex problems, and they are able to identify representations in massive amounts of raw data due to their deep structure. Moreover,

techniques used to prepare data, design, train, validate and evaluate the performance of ANIDS deep learning models are of high impact to the performance of proposed models.

Furthermore, deep learning based ANIDS models outperform machine learning ANIDS models since deep learning has the ability to identify better representations from raw data, compared with traditional machine learning approaches. Therefor feedforward deep neural networks will be used as ANIDS classification algorithm in this work.

UGR'16 dataset will be used to train and evaluate the proposed models. It consists of real and synthetic Netflow v9 data captured from sensors within a tier-3 ISP for 4 months. It consists of 16,900 million flows. It includes a relatively new attacks.

Chapter Three: Proposed Model: ANIDS-DNN

Chapter Outline

Chapter Three: Proposed Model: ANIDS-DNN

The first section in this chapter presents the architecture of ANIDS-DNN proposed model. The second section presents the design and implementation of the proposed model.

3.1 ANIDS-DNN Model Architecture

The proposed ANIDS-DNN system consists of three main components: NetFlow data collection, NetFlow data preparation, and deep neural network classifier as depicted in Figure 17.

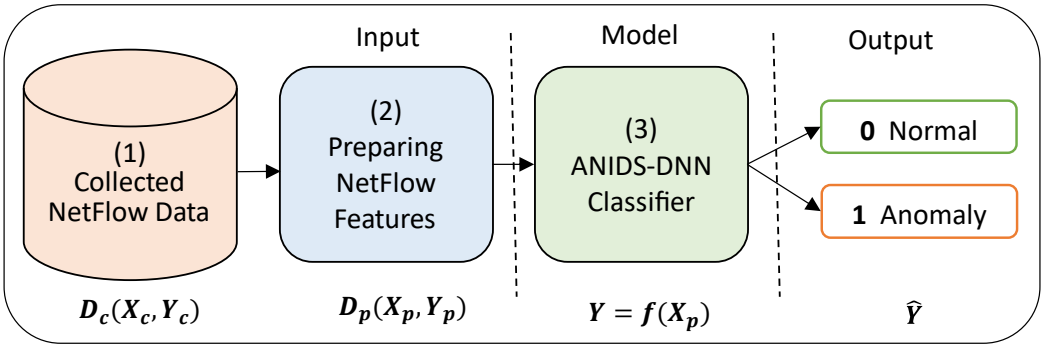


Figure 17: Proposed ANIDS-DNN model layout

3.1.1 NetFlow Data Collection

NetFlow data are captured from the network by enabled NetFlow routers and switches and exported to NetFlow collectors. Then the features or attributes of Network flows are extracted and stored in a database. During the training stage of ANIDS model, labeled Netflow records are required to learn the model parameters from data. However, due to privacy issues only a few of labeled benchmark intrusion detection datasets are available for researchers to train and evaluate the performance of their models. In this work, a selected subset from UGR'16 time series dataset has been used to train and evaluate the performance of proposed model. Each ***j***th record in the dataset represents the

features or attributes of the flow extracted and aggregated from packet headers for each bidirectional connection or flow. These features are considered as the input $\boldsymbol{X_{cj}}$ of the model. But deep learning models can process only numeric data so these features should be prepared first. Also, for each record in the dataset there is a label or target $\boldsymbol{y_j}$**.** The normal traffic is labeled as background. Other classes represent different types of network attacks such as DoS, Scan and Nerisbotnet attack. Thus, in the network anomaly detection context, each TCP or UDP connection in a flow-based dataset represents a record to be processed by the deep learning model as formulated in equation 3.

$$\mathbf{Dc_j} = [\boldsymbol{x_{1j}}, \boldsymbol{x_{2j}}, \boldsymbol{x_{3j}}, \ldots\ldots, \boldsymbol{x_{bj}}, \mathbf{y_j}] \quad (3)$$

Where $\mathbf{Dc_j}$ is an example or an instance from the dataset, $\boldsymbol{x_{bj}}$ is the feature $\boldsymbol{b}$ in the $\boldsymbol{jth}$ instance of dataset $\mathbf{Dc}$ and $\mathbf{Y_j}$ is the true class label or the target of the flow.

Figure 18 shows some records from the UGR'16 dataset. The UGR'16 is public and can be downloaded from the Internet. The files are available in both binary format and in comma separated value (CSV) format.

te	td	sa	da	sp	dp	pr	flg	fwd	stos	pkt	byt	class
7/27/2016 15:10	0.312	69.14.53.184	42.219.155.56	27066	80	TCP	.APRSF	0	40	9	868	background
7/27/2016 15:10	0	42.127.116.160	42.219.153.21	61321	80	TCP	.A.R..	0	0	1	40	background
7/27/2016 14:28	2.988	42.219.159.90	85.33.68.137	3638	445	TCP	....S.	0	0	2	96	background
7/27/2016 14:35	1.78	42.219.159.92	54.23.187.81	4052	80	TCP	.A...F	0	0	2	80	background
7/27/2016 14:24	103.512	42.219.156.243	158.34.106.175	22002	60481	TCP	.AP...	0	0	23	1432	background
7/28/2016 0:10	0	42.219.152.20	42.219.150.242	80	2924	TCP	.A..S.	0	0	1	40	dos
7/27/2016 23:34	1.704	54.143.54.105	42.219.153.74	14180	80	TCP	.AP..F	0	0	14	892	background
7/28/2016 0:01	0	42.219.158.16	42.219.150.246	80	8143	TCP	.A..S.	0	0	1	40	dos
7/28/2016 0:51	0	42.219.150.246	42.219.158.16	47828	1111	TCP	....S.	0	0	1	44	scan44
7/28/2016 1:09	0.52	54.143.48.135	42.219.156.215	25	53532	TCP	.AP.SF	0	0	5	958	anomaly-spam
7/28/2016 1:59	0.113	167.246.0.200	42.219.152.18	53	2077	UDP	.....	0	0	1	140	nerisbotnet
7/28/2016 3:45	0.696	42.219.156.213	193.26.243.145	41453	25	TCP	.APRS.	0	0	6	442	anomaly-spam

Figure 18: Example of records in UGR'16 dataset

3.1.2 NetFlow Data Preparation

Deep learning models can only process numerical data so categorical features should be encoded to numeric features. Although deep learning algorithms are powerful at extracting patterns from raw data and feature representation yet crafting new relevant

features will improve the performance of the model. Moreover, to increase the performance and decrease the complexity of the model only important features should be selected. In order to prepare data, the following processes can be used:

Feature Engineering

Feature engineering is the process of creating new features from provided features. The new crafted features can be added to the existing features or replace some of them [42]. Also, some existing features need to be transformed by changing the scale of a variable or changing the distribution to make the machine learning algorithm able to recognize patterns [26]. For example, in Support Vector Classifier (SVC) a kernel function may be applied to each data instance to map the original non-linear observations into a higher-dimensional space in which they become separable [43]. Feature engineering is important because it improves the discriminating power of a machine learning classifier and thus increases the prediction accuracy of the model. In this sense, in a time series dataset like UGR'16, we need to craft new features that represent the frequency of network flows from a source address or to a destination address within a time window to capture the temporal representation of network flows.

Data Preprocessing

Most machine learning algorithms can only learn from numeric data. Hence features extracted from raw data or transformed from existing features should be preprocessed before they could be fed to a machine learning model.

Data preparation includes data cleaning, encoding, discretization, and scaling.

a) Data cleaning:

Data cleaning is searching for missing values or duplicate instances and fixing them.

b) Data Encoding:

Data may have categorical features of nominal type. Since most machine learning algorithms can only process numerical data, nominal features must be encoded to either ordinal or boolean. Nominal categories with n values can be encoded to (n_categories -1) dummy 0-1 variables [44] which is also called One-Hot encoding. If number of values is high it can be encoded to ordinal integers which is also called Ordinal encoding

c) Data Discretization:

Discretization or binning is converting continuous data to discrete. There are many techniques for discretization such as uniform, and quantile.

d) Data Scaling or Normalization:

To deal with outliers each input variable in the dataset should be scaled separately. There are two techniques for scaling:

MinMaxScaler: scaling a variable to a range between 0 and 1 as in equation 4.

$$\boldsymbol{Xscaled} = \frac{(\boldsymbol{X} - \boldsymbol{minX})}{(\boldsymbol{maxX} - \boldsymbol{minX})} \quad (4)$$

Standard Scaler: if the input variable has a gaussian distribution it could be transformed to a zero mean and unit standard deviation.

After preprocessing features for each sample, the new number of preprocessed features will increase from $\boldsymbol{b}$ to $\boldsymbol{p}$ where $\boldsymbol{p} > \boldsymbol{b}$ and where $\boldsymbol{p}$ is the size of the new feature vector for the ith flow in the dataset. For binary classification, the class labels are also encoded to one boolean output variable $\boldsymbol{y_j}$ of value **0** when the class label is benign and **1** when the class label is malicious. For multiclass classification the class labels are encoded to $\boldsymbol{k}$ boolean output variables where $\boldsymbol{k}$ is the number of unique labels or classes in the dataset.

Feature Selection

Feature selection is the process of reducing the number of input variables. Building a machine learning model based on only the best set of features will increase the

generalization capability and improve the overall accuracy of the model and decrease the computational cost and complexity.

In this work, a Random Forest algorithm is used for feature selection. The random forest is a supervised learning classification algorithm consisting of many decision trees. The random forest creates decision trees on randomly selected data samples, gets predictions from each tree, and selects the best solution through voting [24]. A random forest classifier algorithm also can be used to compute the importance of features. Each feature is assigned a score calculated based on the Gini impurity of the node of that feature in each decision tree in the forest and averaged and normalized among all the trees so that the result is a value between **0** and **1** and it is called mean decrease in impurity (MDI). The higher the score the more relevant is the feature to the target variable [44].

This method of feature selection is a filter method which is independent from the deep learning classification model. Moreover, the new number of selected features $\boldsymbol{m}$ is less than preprocessed features $\boldsymbol{p}$ so the new input feature vector that will be ready to be processed by the deep learning model can be represented as in equation 5.

$$\boldsymbol{X_{ij} = [x_{1j}, x_{2j}, x_{3j}, \ldots\ldots, x_{mj}]^T} \tag{5}$$

Where $\boldsymbol{j = 1, 2, \ldots, n}$ number of observations in the dataset and $\boldsymbol{m}$ is number of selected input features.

The true class labels can be formulated as in equation 6 for binary classification.

$$\boldsymbol{Y_{pj} = y_j} \tag{6}$$

The true class labels can be formulated as in equation 7 for multiclass classification.

$$\boldsymbol{Y_{ij} = [y_{1j}, y_{2j}, y_{3j}, \ldots\ldots, y_{kj}]} \tag{7}$$

3.1.3 Deep Neural Network Classifier

Feedforward neural networks or multilayer perceptrons (MLPs) are deep learning algorithms with one input layer, one output layer, and two or more hidden layers. In classification problems a feedforward network defines a deterministic mapping between input features vector $\boldsymbol{X}$ and the true label $\boldsymbol{Y}$ that can be represented as an approximation function as in equation 8.

$$\boldsymbol{Y} \approx \boldsymbol{f}(\boldsymbol{X}, \boldsymbol{W}, \boldsymbol{b}) \tag{8}$$

where $\boldsymbol{X}$ is the input vector of features, $\boldsymbol{Y}$ is the true class label, $\boldsymbol{W}$ is the weight matrix, and $\boldsymbol{b}$ is the bias vector. $\boldsymbol{W}$ and $\boldsymbol{b}$ are the model parameters that should be learned from the labeled observations. In a feedforward model information flows from the input through the intermediate computations in the hidden layers, and finally to the output $\boldsymbol{Y}$ without feedback connections [25].

The architecture of the proposed binary ANIDS-DNN model is depicted in Figure 19. with one input layer, multiple hidden layers, and one output layer with sigmoid activation function.

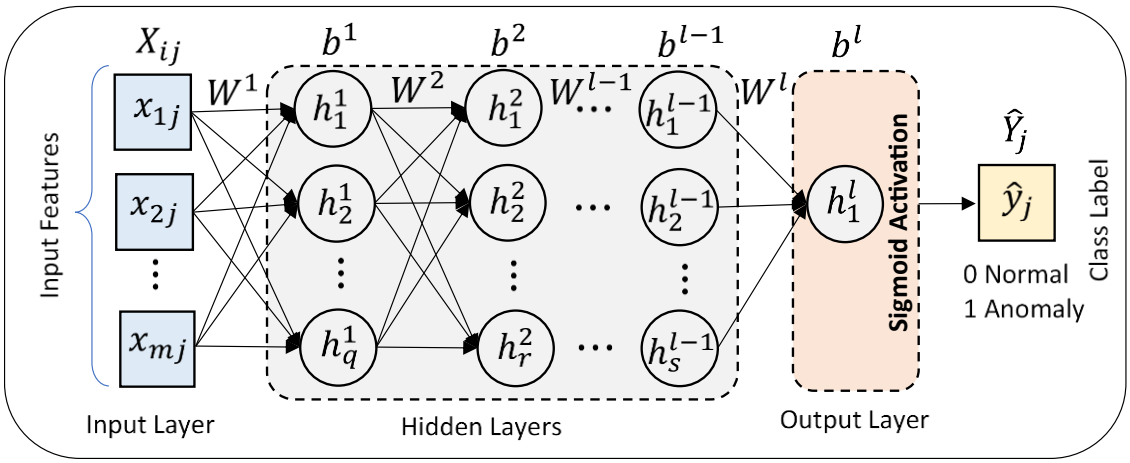


Figure 19: The proposed binary ANIDS-DNN classifier architecture

Input Layer:

The input layer of a deep neural network accepts input variables $\boldsymbol{X_p}$ after features are preprocessed and normalized. The size of input layer equals to the number of preprocessed and selected features. For instance, if m features are selected after data preprocessing then the input feature vector for the *j*th observation can be formulated as in equation 9.

$$\boldsymbol{X_{ij} = [x_{1j}, x_{2j}, x_{3j}, \ldots\ldots, x_{mj}]^T} \tag{9}$$

Hidden Layers:

In a fully connected deep neural network, each hidden layer contains hidden units called neurons. Each neuron is connected to all the nodes of the previous layer and all the nodes of the next layer via links. Each link is associated with a weight $\boldsymbol{w}$ and each neuron is associated with a bias $\boldsymbol{b}$. Equation 10 defines the output of the first neuron in the first hidden layer of $\boldsymbol{jth}$ observation.

$$\mathbf{h_{1j}^1 = \varphi(\sum_{i=1}^{m} x_{ij} w_{i1} + b_1)} \tag{10}$$

where $\boldsymbol{\{x_{1j}, \ldots, x_{mj}\}}$ is the input features vector of the $\boldsymbol{jth}$ observation. and $\boldsymbol{\{w_{11}, \ldots\ldots, w_{m1}\}}$ are the weights for the links between input variables and first hidden unit, $\boldsymbol{b_1}$ is the bias of the first hidden unit, $\boldsymbol{\varphi}$ is the activation function and $\mathbf{h_{1j}^1}$ is the output of the first hidden unit.

The number of hidden layers (the depth of the model), and the number of neurons in each hidden layer (the width of the model), are part of the model's hyperparameters that decided based on the number of input features, the complexity of the problem to be solved and the size of available training examples.

To represent the architecture of deep neural network in a mathematical model, weights in each layer is represented as a matrix $\boldsymbol{W^l}$ and biases for each layer is represented as a vector $\boldsymbol{b^l}$. Weights and biases are the parameters of the model that need to be learned from labeled observations or instances. Furthermore, an activation function is used with each layer to transform the linear output of each neuron to a nonlinear function. In hidden layers rectified linear unit (ReLU) activation function is used. Rectified linear unit activation function in equation 11 activates the neurons based on the output, i.e., if the output falls below zero, the neurons will be disconnected.

$$\boldsymbol{ReLU}(\boldsymbol{x}) = \mathbf{max}\,(\mathbf{0}, \boldsymbol{x}) \tag{11}$$

The output vector of the first hidden layer consists of all the outputs of hidden units of the first hidden layer as in equation 12.

$$\boldsymbol{H}^{(\mathbf{1})} = \mathbf{ReLU}(\boldsymbol{W}^{(\mathbf{1})\boldsymbol{T}}\mathbf{X} + \boldsymbol{b}^{(\mathbf{1})}) \tag{12}$$

Where: $\boldsymbol{q}$ is number of hidden units in first layer, $\boldsymbol{m}$ is number of input features,

$$\boldsymbol{H}^{(1)} = \begin{bmatrix} \boldsymbol{h}_1^{(1)} \\ \boldsymbol{h}_2^{(1)} \\ \vdots \\ \boldsymbol{h}_q^{(1)} \end{bmatrix},\ \boldsymbol{W}^{(1)T} = \begin{bmatrix} \boldsymbol{w}_{11}^{(1)} & \boldsymbol{w}_{21}^{(1)} & \dots & \boldsymbol{w}_{m1}^{(1)} \\ \boldsymbol{w}_{12}^{(1)} & \boldsymbol{w}_{22}^{(1)} & \dots & \boldsymbol{w}_{m2}^{(1)} \\ \vdots & \ddots & \ddots & \vdots \\ \boldsymbol{w}_{1q}^{(1)} & \boldsymbol{w}_{2q}^{(1)} & \dots & \boldsymbol{w}_{mq}^{(1)} \end{bmatrix},\ \boldsymbol{X} = \begin{bmatrix} \boldsymbol{x}_{1j} \\ \boldsymbol{x}_{2j} \\ \vdots \\ \boldsymbol{x}_{mj} \end{bmatrix}, \boldsymbol{b}^{(1)} = \begin{bmatrix} \boldsymbol{b}_1^{(1)} \\ \boldsymbol{b}_2^{(1)} \\ \vdots \\ \boldsymbol{b}_q^{(1)} \end{bmatrix}$$

In general, the output calculation for each hidden layer $\boldsymbol{H}^{(l)}$ is mathematically given as in equation 13.

$$\boldsymbol{H}^{(l)} = \boldsymbol{ReLU}(\boldsymbol{W}^{(l)T}\,\boldsymbol{H}^{(l-1)} + \boldsymbol{b}^{(l)}) \tag{13}$$

Where:

$(\boldsymbol{l})$ is the hidden layer number $\mathbf{1}, \dots, \boldsymbol{l}$.

$\boldsymbol{W}^{(l)T}$ is the weights matrix of the hidden layer.

$\boldsymbol{b}^{(l)}$ is the bias vector of the hidden layer.

$\boldsymbol{h}^{\mathbf{0}} = \boldsymbol{X}_{\boldsymbol{p}}$ the input features vector.

So, the architecture of deep neural network is a chain of composite functions that could be formulated as in equation 14.

$$\boldsymbol{H^{(l)} = H^{(l-1)}(H^{(l-2)}(\cdots H^{(1)}(X)))} \quad (14)$$

Output Layer:

In binary classification the output layer is the last fully connected layer in a deep learning model where only one neuron is needed in the output layer and sigmoid activation function is applied to transforms the output value to a probability in the range between 0 and 1 as formulated in equation 15.

$$\boldsymbol{sigmoid(x) = \frac{1}{1 + e^{-x}}} \quad (15)$$

After sigmoid activation function is applied to the output of last hidden layer neuron, the predicted score is a value between **0** and **1**. The predicted class label is then decided based on the threshold and the predicted score as in equation 16.

$$\boldsymbol{\hat{y} = \begin{cases} 0, & \sigma(h_1^l) \leq \text{threshold} \\ 1, & \sigma(h_1^l) > \text{threshold} \end{cases}} \quad (16)$$

Where $\boldsymbol{\sigma(h_1^l)}$ is the estimated score, **0** represents the normal class, and **1** represents the anomaly class.

In a multi-classification model, the number of hidden units in the output layer equals to the number of classes as illustrated in Figure 20. SoftMax activation function is used to transform the output of each neuron to a probability between **0** and **1** where the neuron with largest probability will be fired as in equation 17.

$$\boldsymbol{softmax(x) = \frac{e^{x_i}}{\sum_{i=1}^{K} e^{x_i}}} \quad (17)$$

Where $\boldsymbol{x_i}$ is the output of the neuron of ***i***th class, and **K** is the number of classes.

The output of the proposed multiclass ANIDS-DNN is a vector of dimension equals to number of classes as in equation 18.

$$\hat{\boldsymbol{y}} = [\hat{\boldsymbol{y}}_{\boldsymbol{1}}, \hat{\boldsymbol{y}}_{\boldsymbol{2}}, \ldots, \hat{\boldsymbol{y}}_{\boldsymbol{k}}] \tag{18}$$

Where **k** is the number of classes (normal and attack classes), and each output is an estimated score between **0** and **1**.

The neuron of the largest probability will be **1** and others will be zeros. For example, the background class in our model is represented as **[1 0 0 0 0 0 0]**.

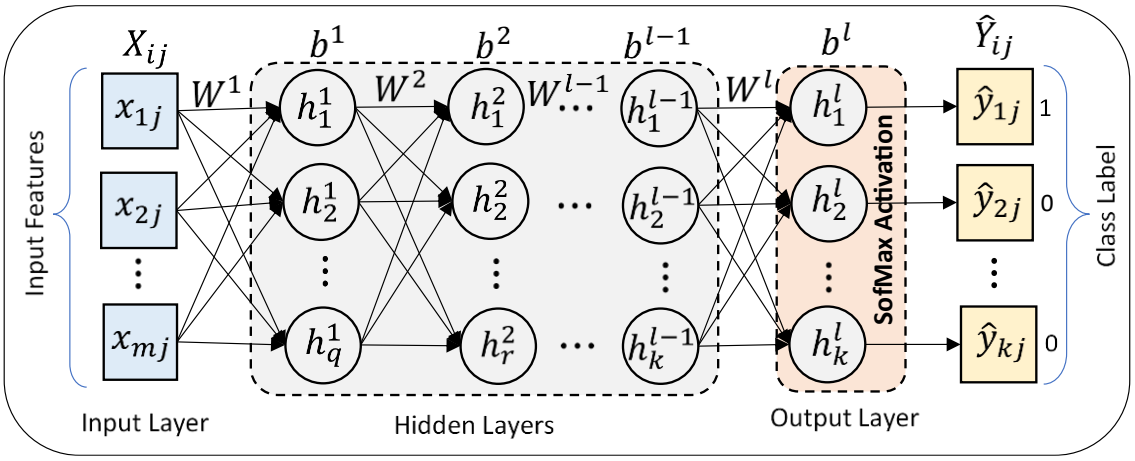


Figure 20: Architecture of multiclass ANIDS-DNN classifier

3.2 Design and Implementation of Proposed Model

In this section the dataset selection and sampling, data preprocessing, and the design, training and evaluation of proposed models will be presented. Figure 21 illustrates the proposed framework for implementing ANIDS-DNN model.

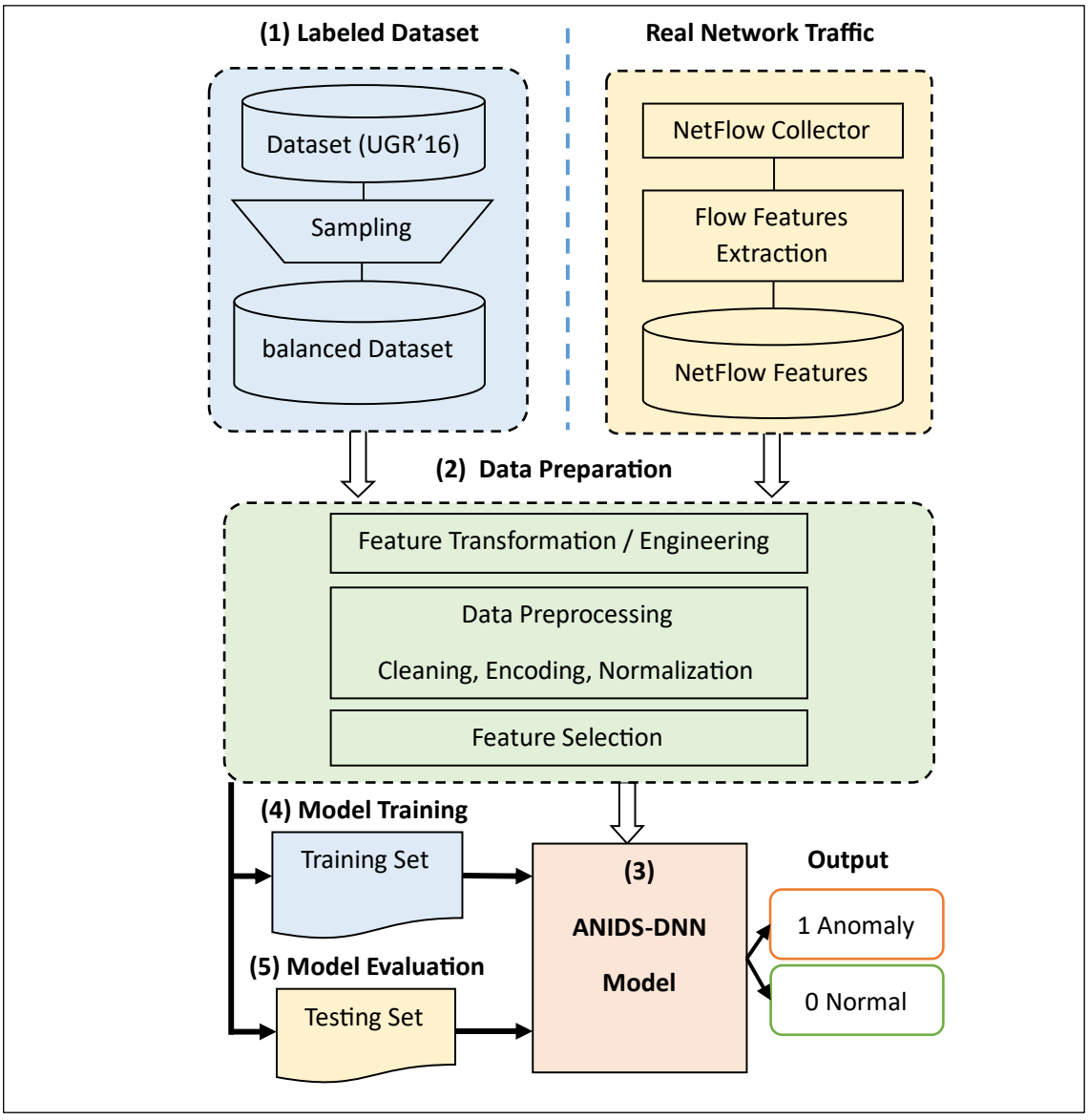


Figure 21: Proposed framework for ANIDS-DNN model

Algorithm 1 illustrates the steps to design and implement the proposed model.

Algorithm 1 Strategy for implementing Proposed ANIDS-DNN Model

Input: Collected NetFlow data
Output: Prediction of the label for each NetFlow record (0 normal, 1 anomaly)

Step 1. collect NetFlow data
Step 2. Prepare NetFlow Data
Step 3. Partition the dataset into the training subset, the validation subset, and the testing subset
Step 4. Design and setup the ANIDS-DNN model architecture
Step 5. Train and validate the ANIDS-DNN model on training and validation subsets
Step 6. Evaluate the performance of the model using the testing subset
Step 7. Tune the performance of the model using different hyperparameters
Step 8. Save the trained model and deploy it to detect intrusions in unseen NetFlow traffic

3.2.1 UGR'16 Dataset

Dataset description:

In this research, UGR'16 is used to run experiments and test the performance of DNN models. The complete dataset contains two different captures: a calibration set and a test set. The calibration set purpose is to assist in building and calibrating normality models, mainly because no attacks were synthetically generated. While the test set purpose is to evaluate the performance of detection algorithms.

Table 3 shows the characteristics of the UGR'16 dataset.

Table 3: Features of the calibration and test sets [32].

Feature	Calibration	Test
Capture start	10:47h 03/18/2016	13:38h 07/27/2016
Capture end	18:27h 06/26/2016	09:27h 08/29/2016
Attacks start	N/A	00:00h 07/28/2016
Attacks end	N/A	12:00h 08/09/2016
Number of files	17	6
Size	181 GB	55 GB
connections	≈ 13,000 M	≈ 3900 M

Table 4 presents the anomalous and synthetic attacks included in the UGR'16 dataset and their corresponding labels.

Table 4: Types of attacks and corresponding labels

Attack	Class	Label
DoS11	signature (synthetic)	DoS
DoS53s	signature (synthetic)	DoS
DoS53a	signature (synthetic)	DoS
Scan11	signature (synthetic)	Scan11
Scan44	signature (synthetic)	Scan44
Botnet	signature (synthetic)	Nerisbotnet
IP in blacklist	signature	Blacklist
UDP Scan	anomaly	Anomaly-udpscan
SSH Scan	anomaly	Anomaly-sshscan
SPAM	anomaly	Anomaly-spam

UGR'16 dataset includes network flow features captured from real network traffic as listed in Table 5.

Table 5: UGR'16 dataset features

#	Feature Name	Abbreviation	Type	Example
1	Timestamp	te	date-time	4/6/2016 0:32
2	Flow duration	td	continuous numeric	107.974 (seconds)
3	Source IP address	sa	categorical	173.192.170.88
4	Destination IP address	da	categorical	42.219.152.18
5	Source port number	sp	discrete numeric	80
6	Destination port number	dp	discrete numeric	1600
7	Protocol	pr	categorical	TCP
8	Flag	flg	categorical	(.APRS.)
9	Forwarding status	fwd	numeric	0
10	Source type of service	stos	Discrete numeric	192
11	Total number of packets	pkt	Continuous numeric	27
12	Total number of bytes	byt	Continuous numeric	31980
13	Class (Label)	class	categorical	nerisbotnet

Selected subset of UGR'16 dataset:

Deep learning algorithms require high hardware resources such as memory, CPU, and GPU for data preprocessing and model training. To overcome hardware limitations a representative subset of data points could be sampled for each class if we have a large dataset.

Since UGR'16 is a large dataset of 16.9 billion records, only selected intervals of the dataset that cover all types of normal and anomalous traffic were considered to train and test our proposed models. The selected subset was then cleaned from missing values and filtered from records with labels of blacklist. The details of selected subsets are shown in Table 6.

Table 6: A selected subset of the UGR'16 dataset

From	To	Class label	Counts
2016-04-11 02:48:00	2016-04-11 03:00:00	background anomaly-sshscan	1004012 41797
2016-04-11 19:42:13	2016-04-11 20:15:16	background anomaly-spam anomaly-sshscan	3476190 1834814 45593
2016-07-27 23:14:08	2016-07-28 01:25:32	background dos scan44 anomaly-spam nerisbotnet scan11	10342170 391184 188560 81123 69248 36140
2016-07-28 01:25:32	2016-07-28 03:55:41	background anomaly-spam nerisbotnet	10911589 85909 82180
2016-08-01 03:25:02	2016-08-01 04:18:47	background anomaly-udpscan	3404655 867097

Balanced subset of UGR'16 dataset:

In network security context, malicious traffic is usually less frequent than benign traffic. This will result in skewed class proportions and an imbalanced dataset. Learning from imbalanced data is a machine-learning problem. To solve this problem the majority

class can be under-sampled, or the minority class can be oversampled. In this work, dataset records with class label of background are under-sampled to obtain a balanced subset of the UGR'16 dataset. The number of records and classes of the derived dataset is shown in Table 7.

Table 7: Number of records and classes of the derived dataset

Class Label	Number of Samples	Percentage
background	1888831	50.00%
DoS	391184	10.36%
Scan	224700	5.95%
nerisbotnet	151428	4.01%
anomaly_spam	167032	4.42%
anomaly_sshscan	87390	2.31%
anomaly_udpscan	867097	22.95%
Total	3777662	100%

3.2.2 Data Preparation

Algorithm 2 Preprocessing UGR'16 dataset

Input: Selected subsets of UGR'16 dataset $Dc \subset D_{UGR'16} = [Xc_{ij}, Yc_j]$
$with\ features\ representation\ Xc_{ij} \rightarrow (b \times j)$
$Yc_j\ is\ the\ class\ label \rightarrow (1 \times j)$
$j: number\ of\ NetFlow\ records, b: number\ of\ basic\ NetFlow\ features$
Output: Preprocessed balanced subset of UGR'16 dataset
$Dp = D_{train} \cup D_{test} = [Xp_{ij}, Yp_{ij}]$
$with\ preprocessed\ and\ selected features\ representation\ Xp_{ij} \rightarrow (m \times j)$
$Yp_{ij}\ is\ the\ class\ label \rightarrow (k \times j)$
$where\ k\ is\ number\ of\ classes$
Step 1. Load selected subset of UGR'16 dataset Dc
Step 2. Clean Data and remove records with blacklist class and save result in *Dc*
Step 3. Extract three new features from source address, destination address and timestamp and add them to *Dc*.
Step 4. Remove source address, destination address features
Step 5. Undersample records with class label is background.
Step 5. For every record in Dc do
    If feature is categorical
        Do One-Hot encoding
    Elseif feature is continuous
        Discretize
Step 6. Select important features
Step 7. Scale feature values between 0 and 1 using MinMaxScaler
Step 8. **Return** preprocessed features in Dp
Step 9. Train-Test split Dataset Dp to 70% in D_{train} and 30% in D_{test}

Feature Engineering

Many types of cyberattacks have temporal patterns than can be detected based on the frequency of network flows [45], [46]. In [47] Johan Knapskog et al. presented a method to visualize the temporal dynamics of IRC-botnet traffic. In this sense, to capture the temporal representation of data in UGR'16 time series dataset, three new features are extracted from source address, destination address and timestamp features.

The first new feature (**sa_nsessions_T)** is the number of flows sent by the same source IP address of the evaluated flow within a time window of one minute. The second feature **(da_nsessions_T)** is the number of flows received by the same destination address of evaluated flow within a time window of one minute. The third feature **(sa_da_nsessions_T)** is the number of flows sent by the same IP address of evaluated flow and received by same destination IP address of evaluated flow within a time window of one minute.

Based on their experimental results Garcia et al. stated that a one-minute aggregation window width is enough to capture botnet synchronization patterns and short enough not to capture too much traffic [48]. Therefore, a one-minute time window was selected to find the frequency of the flows and evaluated for each NetFlow.

Figure 22 is a visualization of the relation between the **sa_nsessions_T** attribute versus the **da_nsessions_T** attribute colored and marked based on the class of NetFlows. It shows that the UDP scan class is separable because the values of **sa_nsessions_T** for UDP scan are too high compared to other classes of network flows. In addition, DoS, Nerisbotnet, and TCP Scan classes have high values of **da_nsessions_T** compared with other network flows.

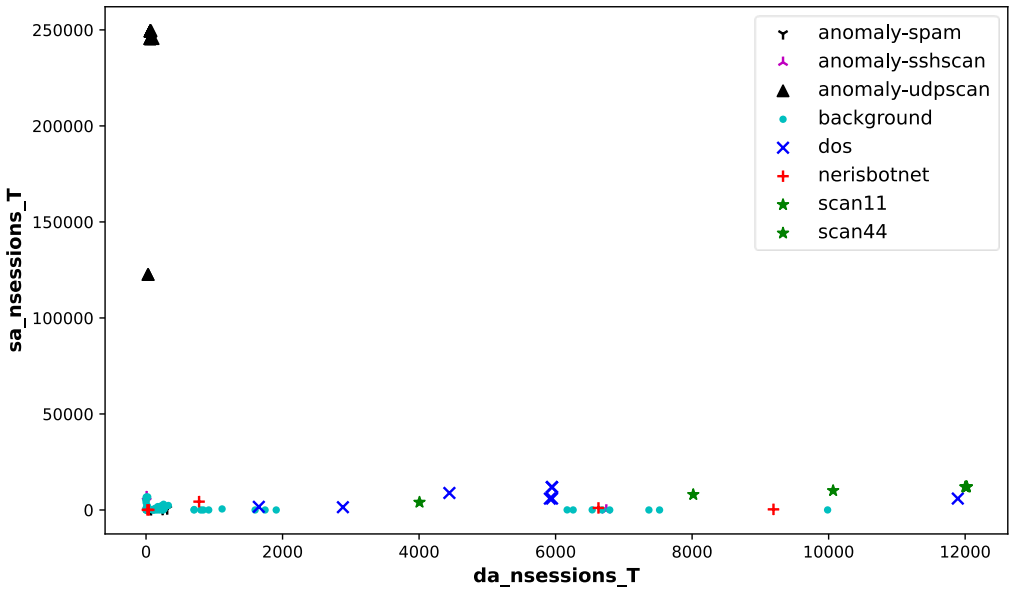


Figure 22: Scatter plot for an extracted feature **sa_nsessions_T** versus **da_nsessions_T** with a time window width of one minute

Figure 23 is a visualization of the relation between the **sa_nsessions_T** attribute versus the **da_nsessions_T** attribute when the frequency of the flows calculated based on a one second time window. It shows that one second is not enough to capture some of the cyclostationary network attacks such as nerisbotnet and DoS attacks.

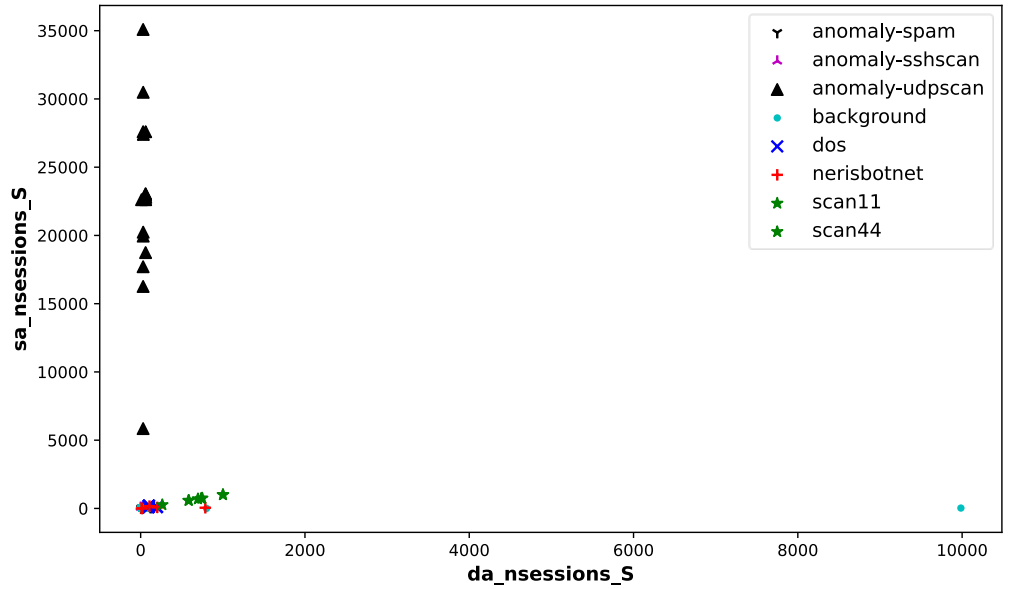


Figure 23: Scatter plot for an extracted feature **sa_nsessions_S** versus **da_nsessions_S** with a time window width of one second

Data preprocessing:

Data Cleaning:

Data cleaning is searching for missing values or duplicate instances and fixing them.

Data Encoding:

Since deep learning algorithms can only process numerical data, categorical features should be encoded. TCP flag values (UAPRSF) are parsed and then one-hot encoded into six Boolean features. The Protocol (**pr**) input feature is also encoded into eight boolean features. Source ports and destination ports are encoded into 51 boolean features based on the value of the port number. If the port value is between 0 and 1023 then a new boolean feature named **reserved** is set to True. If the port value is between 1024 and 49151 then a new boolean feature named **registered** is set to True. For other values of the port number, a new feature named **private** is set to True. Moreover, a new feature is introduced for each application layer service such as HTTP and SMTP. The stos feature is encoded into three Boolean features.

The source address and destination address features are excluded and replaced with the new features **sa_nsessions_T**, **da_nsessions_T**, and **sa_da_nsessions_T**. Finally, the class label is encoded into seven boolean output variables. The San11 and Scan44 class labels are replaced by the **Scan** as an output variable. Table 8 shows all the features after encoding.

Data Discretization:

Discretization or binning is converting continuous data to discrete. There are many techniques for discretization such as uniform, and quantile. The continuous features (**td**, **pkt**, **byt**) are discretized.

Table 8: Input features of the dataset after one-hot encoding

Feature (Input variable)	Encoded Features	Description
td (flow duration)	1	Time duration in seconds for a flow
sp (source port)	51	reserved, registered, private, HTTP, SMTP, etc.
dp (destination port)	51	reserved, registered, private, HTTP, SMTP, etc.
sa_nsessions_T	1	total number of network flows initiated by each unique source IP address within a timestamp of one minute.
da_nsessions_T	1	total number of network flows received by each unique destination IP address within a timestamp of one minute.
sa_da_nsessions_T	1	total number of network flows initiated from each unique source IP address to the same destination IP address within a timestamp of one minute.
Pr (protocol)	8	TCP, UDP, ICMP, GRE, ESP, IPIP, IPv6, and other
Flg (flag)	6	U, A, P, R, S, F
Stos (type of service)	3	0, 192, other
pkt (packets)	1	Total number of packets transferred during a flow
byt (bytes)	1	Total number of bytes transferred during a flow
Total Inputs	125	
Outputs for binary classifier	2	Background, Anomaly
Outputs for multiclass classifier	7	Background, Anomaly-spam, Anomaly- udpscan, Anomaly-sshscan, Scan, Dos, Nerisbotnet.

Scaling Data:

In this work, from Scikit-learn library a preprocessing utility MinMaxScaler is used for scaling features to scale values between a minimum of zero and a maximum of one so that the maximum absolute value for each feature is scaled to unit size. This type of scaling will keep the Boolean values unchanged for one-hot-encoded features in the dataset.

Feature Selection:

After feature engineering, the input features were transformed from 11 to 125 numerical and boolean features. Using the Random Forest classifier, the importance of all features is calculated based on Gini Importance or Mean Decrease in Impurity (MDI)

where each feature's importance is calculated as the sum over number of splits across all trees that include the feature, proportionally to the number of samples it splits. Figure 24 shows the highest 32 numerical features of the UGR'16 dataset based on the Mean Decrease in Impurity (MDI). For training the proposed deep learning models, the most 45 important features were selected.

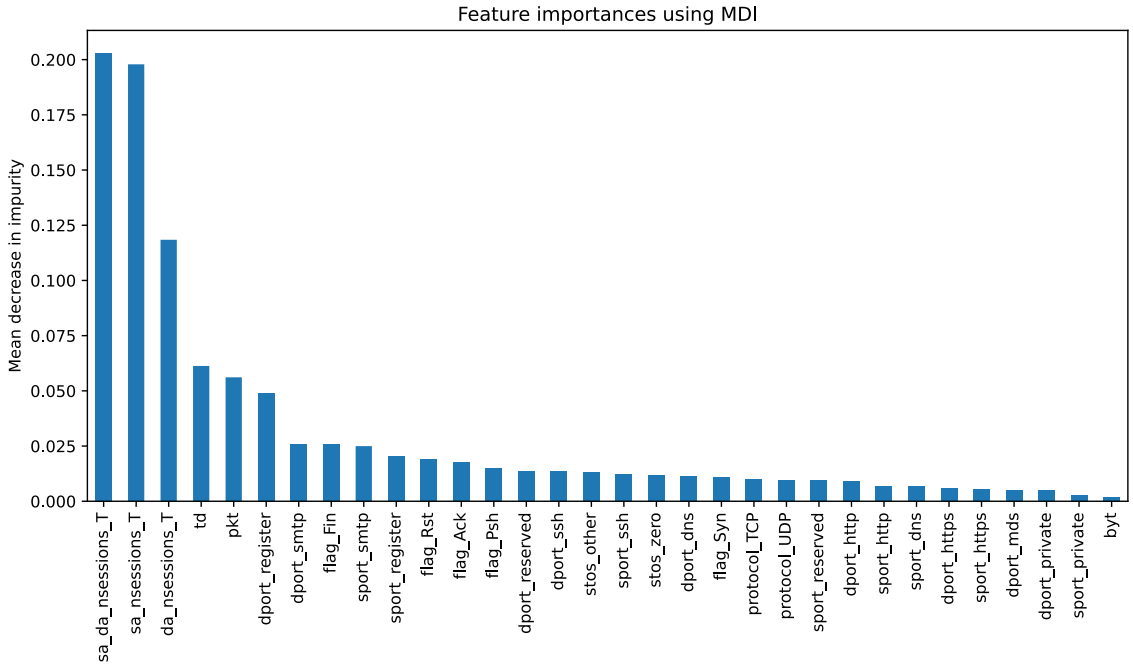


Figure 24: Input features of UGR'16 dataset with highest mean decrease in impurity (MDI)

3.2.3 Setup of the Proposed Model

a) Optimal Network Architecture

Before developing the architecture of ANDIS models some design decisions should be considered as depicted in Figure 25.

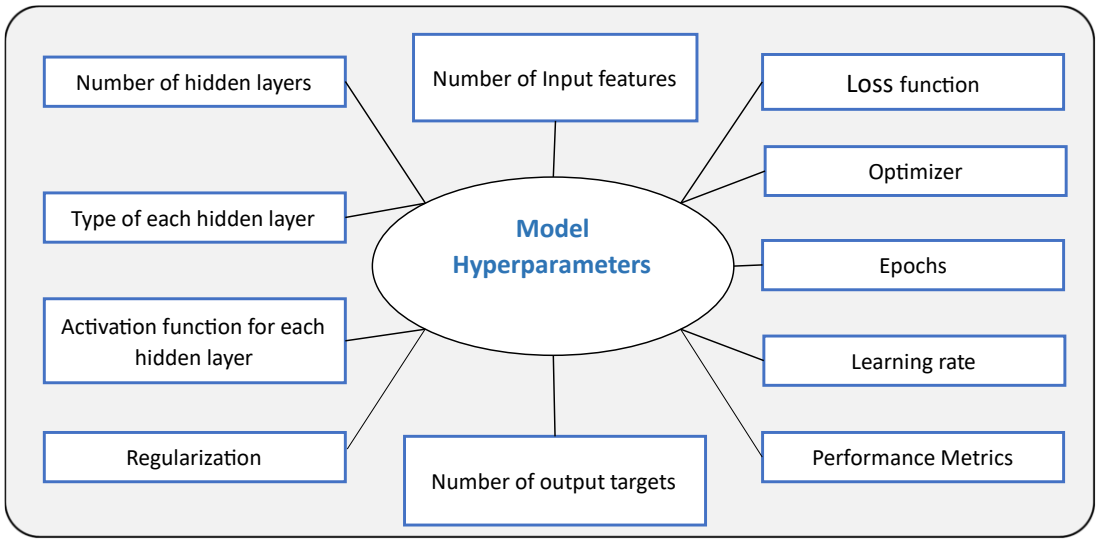


Figure 25: Design decisions for ANIDS model's architecture and hyperparameters

To find the best architecture of the deep neural network, the following experiments are done:

- **Experiment 1:** One Input layer, four fully connected hidden layers with [360, 180, 90, 45] hidden units respectively and ReLU activation function for each layer , and one output layer with one neuron and sigmoid activation function).
- **Experiment 2:** One Input layer, three fully connected hidden layers with [90, 60, 45] hidden units respectively and ReLU activation function for each layer , and one output layer with one neuron and sigmoid activation function).
- **Experiment 3:** One Input layer, two fully connected hidden layers with [90, 45] hidden units respectively and ReLU activation function for each layer , and one output layer with one neuron and sigmoid activation function).
- **Experiment 4:** One Input layer, one fully connected hidden layers with [45] hidden and ReLU activation function , and one output layer with one neuron and sigmoid activation function).

The best accuracy was for experiment 2 when DNN consists of three hidden layers with output shapes (90, 60, 45) respectively and one output layer with one neuron and

sigmoid activation function. As a result, our proposed model architecture will be selected based on experiment 2 as shown in Figure 26.

b) Selecting Model Hyperparameters

Hyperparameters are the parameters of the model that need to be set before the training process. In this work, some hyperparameters are defined based on heuristics and literature recommendations. Table 9 lists the selected hyperparameters of the proposed model.

Table 9. Selected hyperparameters of the model

Hyperparameter	Value
Optimizer	Adam (learning rate = 0.001)
Loss	Cross Entropy
Weight initialization	uniform
Regularization	Dropout (20%) in 2nd and 3rd hidden layer
Epochs	100 with early stopping and patience of 20
Batch size	100

For example, the ReLU activation function is used in hidden layers and, Adam is used as an optimizer with a default learning rate of 0.001. The sigmoid activation function is used for binary classification. The SoftMax activation function is used for multiclass classifiers. The early stopping regularization technique is used to stop training when parameter updates no longer begin to yield improvements on a validation set. As a result, the optimal number of epochs will be decided by the early stopping technique.

c) Binary ANIDS Model Setup

For binary classification, we are interested in classifying data into one of two binary classes. The background flows are represented as 0's and the anomaly flows are represented as 1's in our data. The proposed binary classifier is a deep Multi-Layer Perceptron (MLP) composed of an input layer, three hidden layers, and an output layer. The input shape is 45 since the most 45 effective features are selected to build the model.

The first hidden layer is composed of 90 units with a Rectified Linear Unit (ReLU) activation function. The second dense layer is composed of 60 units and the activation function is ReLU and a regularization dropout of 20% is added to avoid overfitting. The third hidden layer is composed of 45 units and the activation function is ReLU. The last layer is the output layer which contains one hidden unit and the sigmoid as an activation function. The output of the binary classifier is the predicted class of the network $\hat{\boldsymbol{y}}$ that is **1** if anomaly is detected and **0** if the connection is normal traffic. Figure 26 shows the design of ANIDS-DNN model. For binary classification, sigmoid activation function is used to transform the output between **0** and **1**.

d) Multi-Class ANIDS Model Setup

For multiclass classification, the proposed model is designed to classify the network flows into seven categories. The normal connections will be classified as background and the unique labels for malicious network flows are: anomaly spam, anomaly ssh scan, anomaly UDP scan, DoS, NerisBotNet, and scan attacks.

The deep feed-forward neural network multiclass classifier is composed of an input layer, three hidden layers, and an output layer. The input shape is 45 since the most 45 effective features are selected to build the model. The first hidden layer is composed of 90 units with a Rectified Linear Unit (ReLU) activation function. The second dense layer is composed of 60 units and the activation function is ReLU and a regularization dropout of 20% is added to avoid overfitting. The last hidden layer is composed of 45 units and the activation function is ReLU. The output layer contains seven neurons and uses SoftMax as an activation function as depicted in Figure 26.

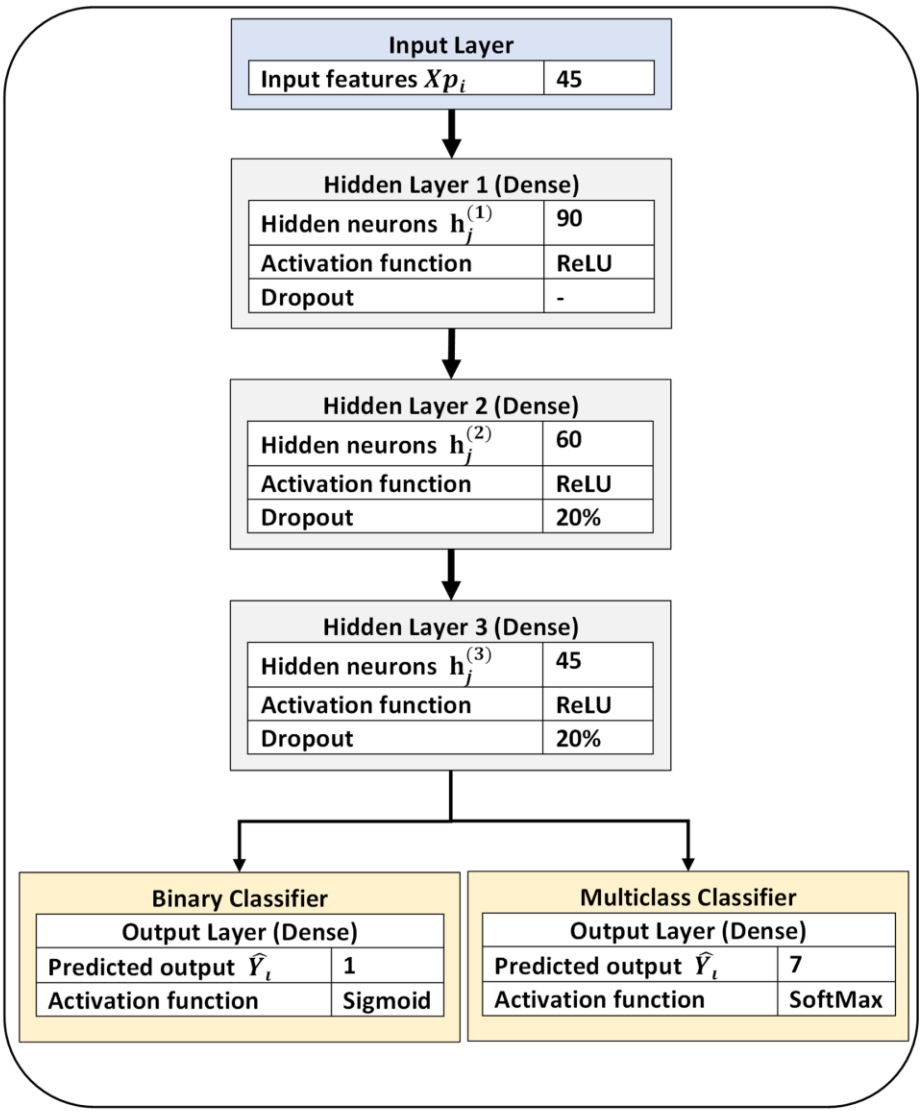


Figure 26: Design and setup of ANIDS-DNN model

3.2.4 Model Training and Validation

To learn model parameters (weights and biases), the model is fitted to labeled observations from training dataset.

Train-Test Split:

A machine learning model aims to make good predictions on new unseen data. So, to accurately evaluate the performance of our model, it is common to split data into training and test sets. The training set is used to fit the model, and the test set is to evaluate the

predictive accuracy of the trained model. The derived dataset was split into 70% for training and validation and 30% for testing.

The split simulates the problem of detecting zero-day attacks. Because the model has never seen the observations in the test set, then this evaluation is a way to predict how well the model will do against new anomaly network flows.

The distribution of samples is shown in Table 10.

Table 10: Number of training samples and testing samples for each class label

Class Label	Training samples	Testing samples
Background	1322182	566649
Anomaly_spam	116922	50109
Anomaly_sshscan	61173	26217
Anomaly_udpscan	606968	260129
DoS	273829	117356
Nerisbotnet	106000	45428
Scan	157290	67410
Total Samples	2644364	1133299

Cross Validation

To improve the generalization of a binary classification model the training set is partitioned to 80% for fitting the model and 20% for validation. The classifier is trained on training set and validated on validation set. After each epoch, if the validation loss didn't improve, a patience counter is incremented till a preset value is reached then the training is early stopped to prevent overfitting.

In multiclass classification a stratified k-fold cross-validation is used, the dataset is partitioned such that the ratio of the observations in each class remains the same in each fold. The k-fold method gives a model with less bias compared to other methods. **K** is the number of folds the dataset is divided by and usually **k=5** or **10**. This sampling method will decrease overfitting and improve generalization when the model is tested on unseen data. The model is trained on **k-1** folds and validated on one-fold each iteration for **k**

number of iterations so that the model is exposed and tested to all minor classes, especially in imbalanced datasets with the unproportioned class distribution. In this study, a stratified 10-fold cross-validation is employed in multiclass classifier to ensure that each class is represented with equal proportions in both the training and test datasets. Figure 27 depicts the 5-fold cross validation process.

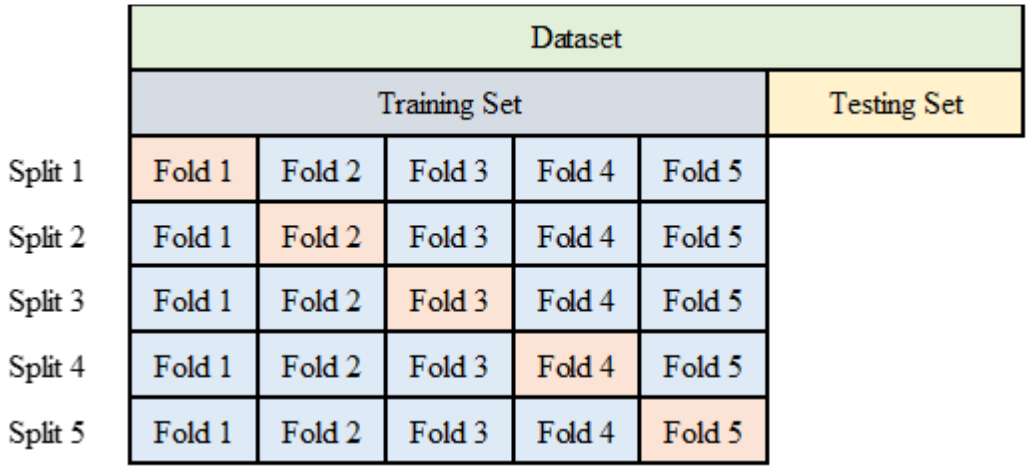


Figure 27: 5-fold cross-validation

Loss Function

After building the architecture of DNN, model parameters should be learned from the labeled dataset so that the predicted output of the model is as close as possible to the true label. During training a batch of data samples is fed into the model at a time in the forward pass and output is calculated. Then the difference between the true label and the estimated output is calculated using a loss function.

For binary classifiers the cross-entropy loss function is used as in equation 19.

$$\boldsymbol{Loss_{CE}} = -\frac{\mathbf{1}}{\boldsymbol{N}}\sum_{\mathbf{j=1}}^{\boldsymbol{N}} \boldsymbol{y_j} \cdot \mathbf{log}\left(\boldsymbol{p}(\boldsymbol{y_j})\right) + \left(\mathbf{1} - \boldsymbol{y_j}\right) \cdot \mathbf{log}\left(\mathbf{1} - \boldsymbol{p}(\boldsymbol{y_j})\right) \quad (19)$$

where $\boldsymbol{y_j}$ is the true label (**1** for malicious traffic and **0** for benign traffic) and $\boldsymbol{p}(\boldsymbol{y_j})$ is the predicted probability of the observation ($\boldsymbol{j}$) calculated by the sigmoid activation function or $\hat{\boldsymbol{y}}_{\boldsymbol{j}}$. **N** is the number of observations in the batch.

For multiclass classifier Categorical cross-entropy loss function is used as formulated in equation 20.

$$\mathbf{Loss}_{CCE} = -\frac{1}{N}\sum_{j=1}^{N}\sum_{i=1}^{k} y_{ji} \cdot \log\left(p(y_{ji})\right) \tag{20}$$

where y_{ji} is the true class label for class (i) and observation (j).

N is number of observations in the batch, and k is number of categories or class labels.

$p(y_{ji})$ is the predicted probability of class (i) and observation (j) calculated by SoftMax.

Optimizer

The purpose of an optimizer is to adjust the model parameters (weights and biases) to optimal values that minimize the loss function. Since the loss function in the deep neural networks is a nonconvex function an optimization technique is required to control the learning rate so that the cost function can converge to the global minimum. The classical optimization algorithm is the stochastic gradient descent, and a more improved version is the Adam optimizer.

Regularization

To make the deep learning algorithm performs well on new unseen inputs and avoid overfitting training data, a regularization strategy is required. Some regularization techniques add a penalty term to the cost function to reduce the weights as in L1 and L2 regularization. In deep learning, dropout is the most used regularization technique. Dropout removes a random selection of a fixed percentage of the neurons in a network layer for a single gradient step. The dropout rate is the hyperparameter that should be tuned between 10% to 50% to improve generalization. In this study, a 20 % dropout rate is used in the second and third dense layers.

Training of the Proposed Models

The binary DNN model is trained for 100 epochs with early stopping and batch size of 100. The loss function is cross entropy. The optimizer is Adam with a learning rate of 0.001. The training and validation are carried out on 70% of the dataset, and the testing is on the remaining 30% of the dataset.

The multiclass DNN model is trained on 15 epochs with a batch size of 100. The loss function is categorical cross-entropy. The optimizer is Adam with a learning rate of 0.001. The training and 10-fold cross-validation are carried out on 70% of the dataset, and the testing is on the remaining 30% of the dataset. A python code example for fitting a multiclass model using stratified 10-fold cross validation is shown in Figure 28.

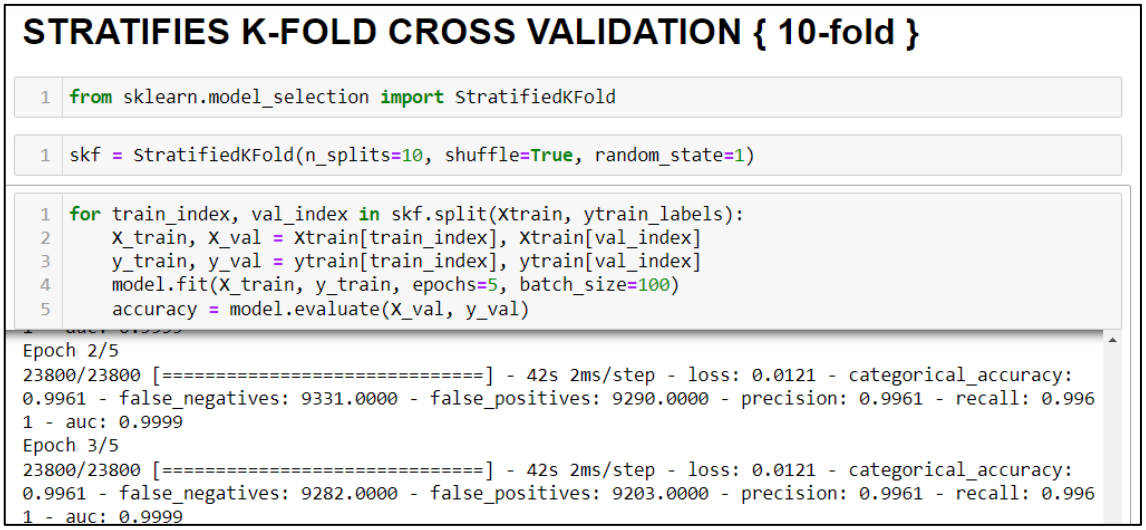


Figure 28: Code example of training and 10-fold cross validation of proposed model

Flow chart for the detailed steps of fitting the ANDIS-DNN model is depicted in Figure 29.

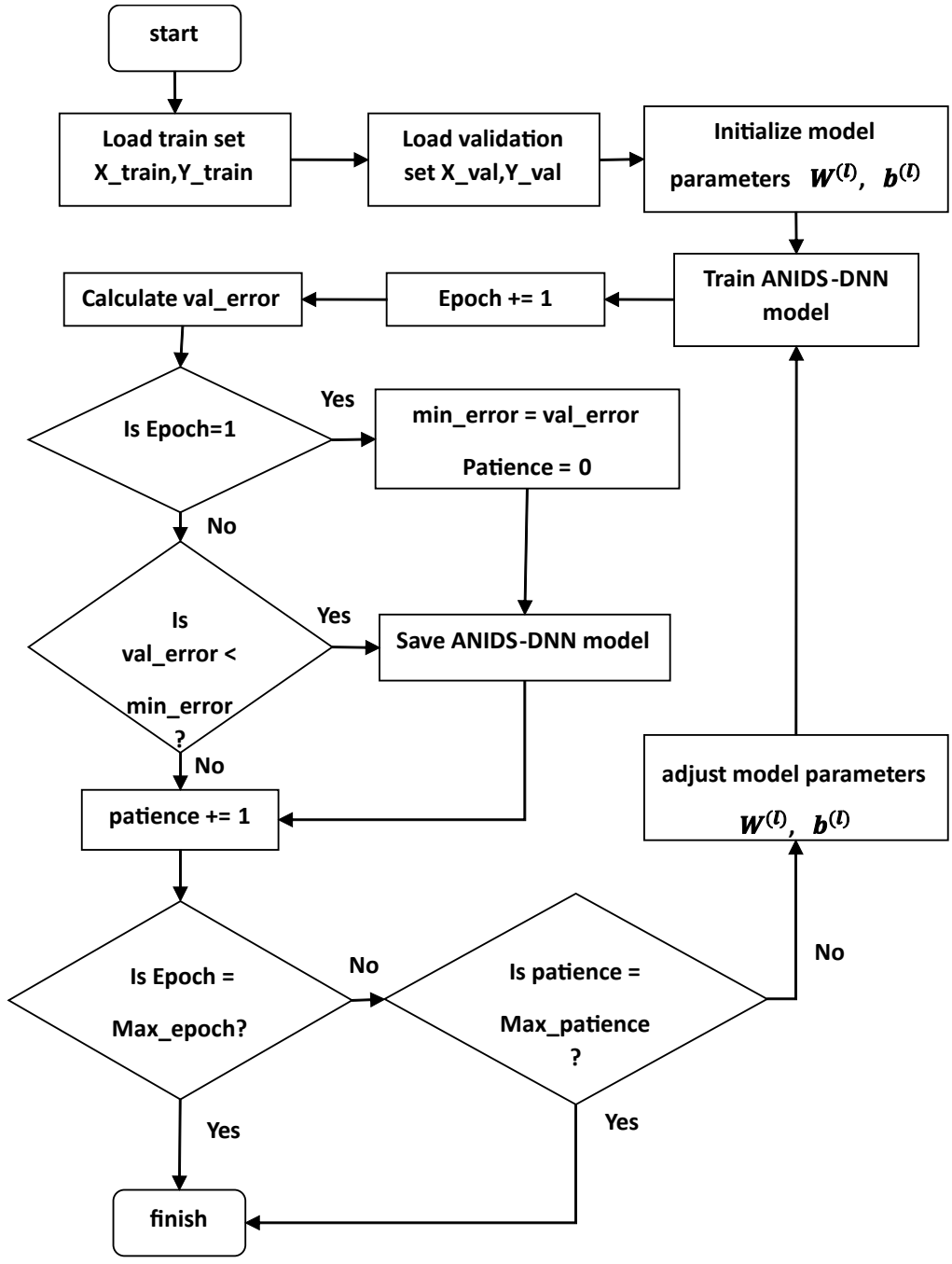


Figure 29: Flow chart for training ANIDS-DNN model

3.2.5 Model Testing

After the model trained, we test the model on the unseen samples from the test set. The predicted values then compared with the true labels and as a result the performance metrics can be calculated to assess the model performance. If test error (generalization error) is greater than training error then the model is overfitting training data. Figure 30 depicts the process flow of training and testing the proposed ANIDS-DNN models.

3.2.6 Hyperparameters Tuning

The parameters of the deep learning model are learned from data. Hyperparameters are the parameters of the model that need to be set before the training process. These parameters should be tuned because they have an important effect on the model complexity, accuracy, and speed of the training. Some of the important hyperparameters are learning rate, batch size, number of epochs, loss function to be used, optimizer, dropout rate if regularization is used, number of neurons in each hidden layer, and number of hidden layers in the model.

Some of the strategies to tune hyperparameters are:

- **Randomized search**: this approach reduces the computation cost by searching a fixed number of hyperparameters.
- **Grid Search**: this approach searches for the best set of hyperparameters from a grid of hyperparameters values.
- **Manual search**: this approach searches a fixed number of hyperparameters manually and repeatedly.

In this work, some hyperparameters are defined based on heuristics and literature recommendations. For example, the ReLU activation function is used in hidden layers and, Adam is used as an optimizer. The sigmoid activation function is used for binary classification. The SoftMax activation function is used for multiclass classifiers. The early

stopping regularization technique is used to stop training when parameter updates no longer begin to yield improvements on a validation set. As a result, the optimal number of epochs will be decided by the early stopping technique.

The learning rate is a parameter that could be tuned to improve performance of the model. To find the optimal learning rate, six learning rate values [0.0005, 0.001, 0.005, 0.01, 0.05, 0.1] are selected one per trial to train the ANIDS-DNN model with 70000 samples and test performance with 30000 samples. The best detection accuracy was when a learning rate of **0.001** was selected for Adam optimizer. As a result, the 0.001 learning rate is selected for training the model. However, the training time for a smaller learning rate is slightly higher than the time elapsed when selecting larger learning rate.

Another hyperparameter to be tuned is the batch size, using the same training samples the following batch sizes are experimented [32,64,128,256,512,1024,2048]. The best performance was when a batch size of **512** was selected. Also, the training time is slightly lower for larger batch size than smaller batch size.

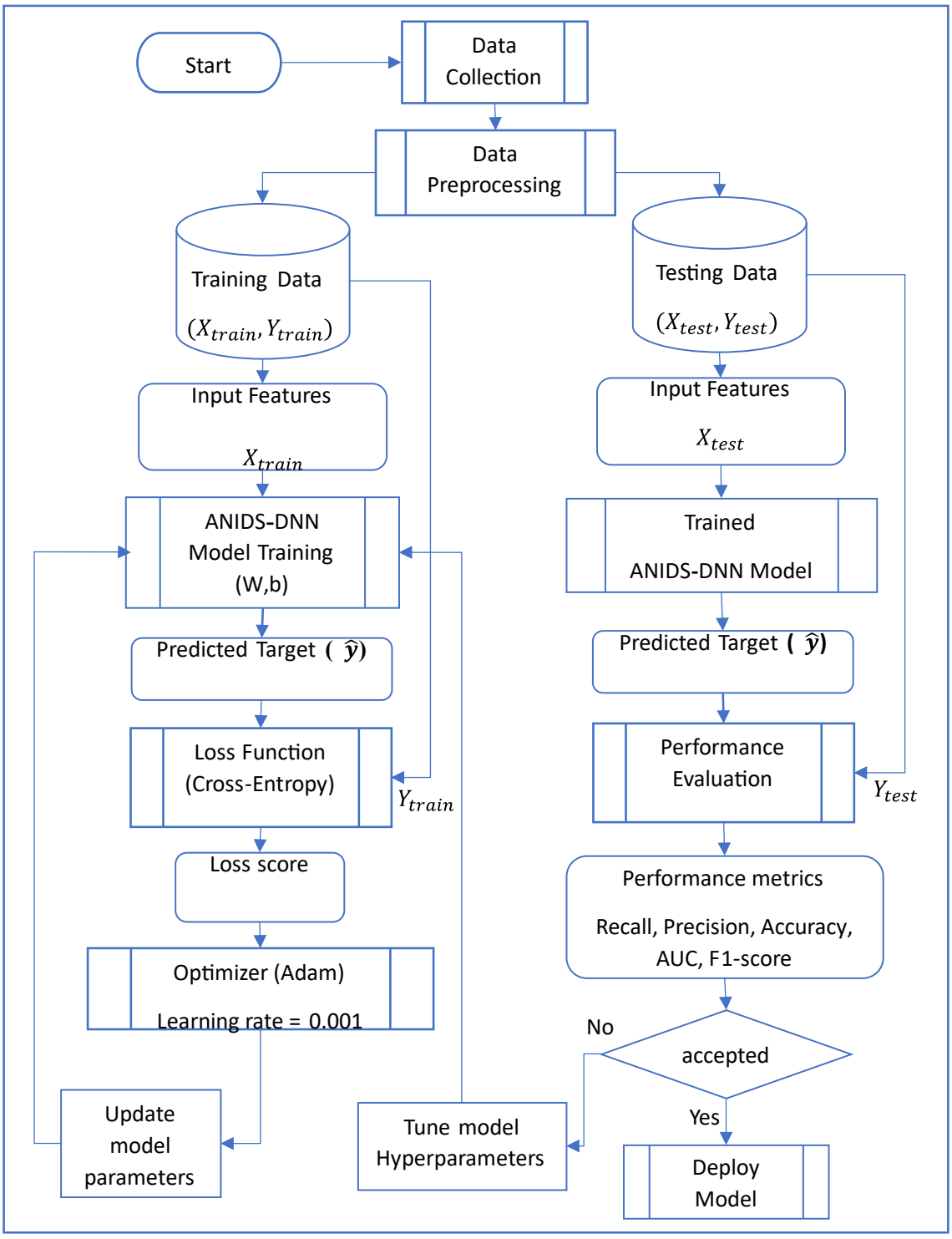


Figure 30: ANIDS-DNN model training and testing process flow diagram

3.3 Summary:

This chapter presented the main blocks for building models for ANIDS using deep learning. The labeled dataset is used to train a deep learning model. First, the features of

the dataset must be extracted, transformed, selected, preprocessed, and scaled. Then, the deep neural network learns the parameters of the model from preprocessed data. After the training, the performance of the model is evaluated and hyperparameters are tuned.

A detailed methodology was presented in this chapter to build and train the ANIDS models. UGR'16 benchmark dataset is used to train and test the performance of the models. A feedforward deep learning algorithm was used to model both binary and multiclass ANIDS. Due to memory and time constrain, a subset was selected from the big dataset. The models are trained and validated on 70% of the selected set and tested on unseen 30%. Data is prepared before training and important features are selected. The hyperparameters of the model are tuned to improve the performance the of proposed models.

Chapter four: Experimental Results

Chapter Outline

Chapter four: Experimental Results

This section presents the experimental setup, the performance metrics used to evaluate the proposed models, and the experimental results of both Binary and multiclass ANIDS models.

4.1 Experimental Setup:

To develop the proposed models the following resources are used:

- The open-source development environment Anaconda with Python 3.8.11.
- The deep learning library Keras.
- TensorFlow 2.3.0 [22],
- Scikit Learn [21].

The experiments were run on a Dell Inspiron 15 laptop computer with 1.99 GHz Intel(R) Core (TM) i7-8550U CPU and 16 GB of RAM.

Google Colaboratory which is a free Jupyter notebook environment also has been used to run some experiments. The implementation was done on 12 GB RAM with GPU support.

4.2 Performance Metrics:

To evaluate the performance of a classifier several metrics are used for binary classification models and multiclass classification models.

Recall or detection rate is the proportion of attacks that are correctly detected. It is also known as True Positive Rate (TPR), Sensitivity, and Probability of Detection.

$$\boldsymbol{Detection\ rate(recall)} = \frac{\boldsymbol{TP}}{\boldsymbol{TP + FN}} \tag{21}$$

Where,

True Positive (TP) refers to anomalous flow that is predicted as anomalous.

True Negative (TN) refers to normal flow that is predicted as normal.

False Positive (FP**)** refers to normal flow that is misclassified as an anomaly.

False Negative (FN) refers to an anomalous flow that is undetected.

The false positive rate is the proportion of normal traffic incorrectly flagged as an attack

$$\boldsymbol{False\ positive\ rate} = \frac{\boldsymbol{FP}}{\boldsymbol{TN + FP}} \tag{22}$$

Accuracy is the fraction of correctly identified results (attack and normal traffic)

$$\boldsymbol{Accuracy} = \frac{(\boldsymbol{TP + TN})}{(\boldsymbol{TP + TN + FP + FN})} \tag{23}$$

Precision is the proportion of identified attacks that are indeed attacks

$$\boldsymbol{Precision} = \frac{\boldsymbol{TP}}{\boldsymbol{TP + FP}} \tag{24}$$

F1-score is the harmonic mean of precision and recall. The F1-score is often used to measure performance for problems with class Imbalance, i.e. when one class has significantly more examples than other classes.

$$\boldsymbol{F1 - score} = \frac{\boldsymbol{2 * precision * recall}}{\boldsymbol{precision + recall}} \tag{25}$$

AUC (Area Under the Curve) and **ROC** (Receiver Operating Characteristics) is a graphical plot that measures the performance of a classification model. The curve plots the

true positive rate versus the false positive rate. The area under the curve AUC is the probability that a randomly chosen positive is ranked before a randomly chosen negative.

The confusion matrix is a visualization tool to evaluate the accuracy of the classification as illustrated in Table 11 [49] .

Table 11: Confusion matrix

		True class	
		Attack	Normal
predicted class	Attack	TP	FP
	Normal	FN	TN

4.3 Experimental Results and Analysis:

In the following two subsections, the experimental results of both Binary and multiclass ANIDS models will be presented and analyzed.

4.3.1 Binary ANIDS Classifier (B-ANIDS-DNN) Results:

After the B-ANIDS-DNN model was trained on 80% of the train set and cross-validated on 20% of the train set, the performance of the model is evaluated on the unseen test set. The state-of-the-art metrics were used to evaluate and compare the results.

Table 12 presents the test results of the B-ANIDS-DNN model when different DNN architectures have been experimented. The best performance for proposed model consists of five layers including one input layer, three hidden layers, and one output layer with hidden units of (90,60,45) respectively. The model detection accuracy is 99.773%. Furthermore, the time elapsed in training and validating the model on the training set using three hidden layers was 2601.19 sec.

Table 12: Performance evaluation metrics score [%] for B-ANIDS-DNN using different DNN architectures and 45 input features

Architecture	Accuracy	Precision	Recall	F1 Score	Training Time (sec)
Four Hidden Layers	99.709	99.709	99.708	99.708	2618.42
Three Hidden Layers	**99.773**	**99.773**	**99.773**	**99.773**	**2601.19**
Two Hidden layers	99.719	99.719	99.719	99.719	2751.02
One Hidden layers	99.572	99.572	99.572	99.572	2490.38

Figure 31 shows the training and validation loss while training the model for 100 epochs, The loss converges to approximately 0.01.

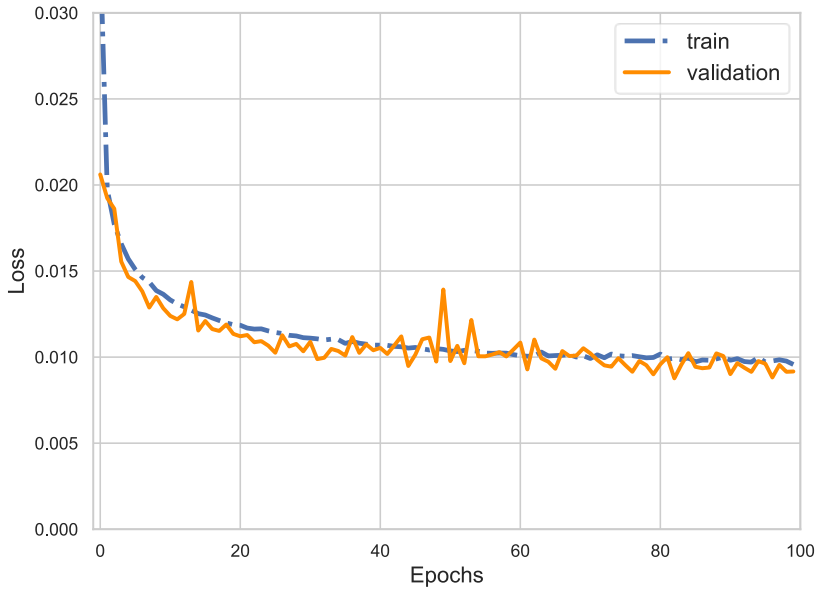


Figure 31: Training and validation cross-entropy loss curve for B-ANIDS-DNN (15 features)

Figure 32 shows the accuracy of the model during training and validation of the model for 100 epochs. The model accuracy converges to around 99.64%.

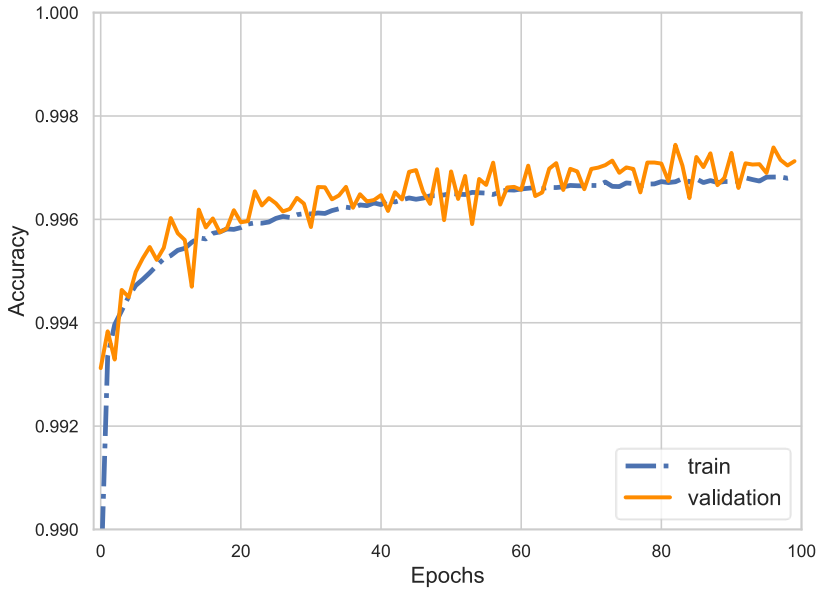


Figure 32: Training and validation accuracy curve for B-ANIDS-DNN (15 features)

Table 13 summarizes the results in a percentage of the performance evaluation metrics for the binary deep learning classifier when trained on basic 15 features, 28 features, and 45 features for each class. The highest F1-score is 99.713% was for the model when trained on 45 input features.

Table 13: Performance evaluation metrics score [%] for normal and anomalous classes

	Background			Anomaly		
	Precision	Recall	F1-score	Precision	Recall	F1-score
15 features	95.117	98.773	96.910	98.725	94.934	96.792
28 features	99.178	99.781	99.478	99.779	99.172	99.474
45 features	99.603	99.823	**99.713**	99.823	99.602	**99.712**

Table 14 compares the performance evaluation metric scores for the B-ANIDS-DNN model. The highest performance on all metrics is when 45 input features are used to train the model. However, only around 0.237% drop in performance if 28 input features are selected over 45 input features.

Table 14: Performance evaluation metrics score [%] for different number of input features

B-ANIDS-DNN	Accuracy	Precision	Recall	F1 Score	Support
15 features	96.853	96.921	96.853	96.851	1133299
28 features	99.476	99.478	99.476	99.476	1133299
45 features	**99.713**	**99.713**	**99.713**	**99.713**	1133299

Figure 33 shows that B-NIDS-DNN achieves between 96%~97% in all performance metrics when trained on only 15 basic features. Furthermore, an improvement of about 2%~2.5% in all performance metrics when trained on 28 uncorrelated features. However, only about 0.4%~0.5% increase in performance when trained on 45 selected features.

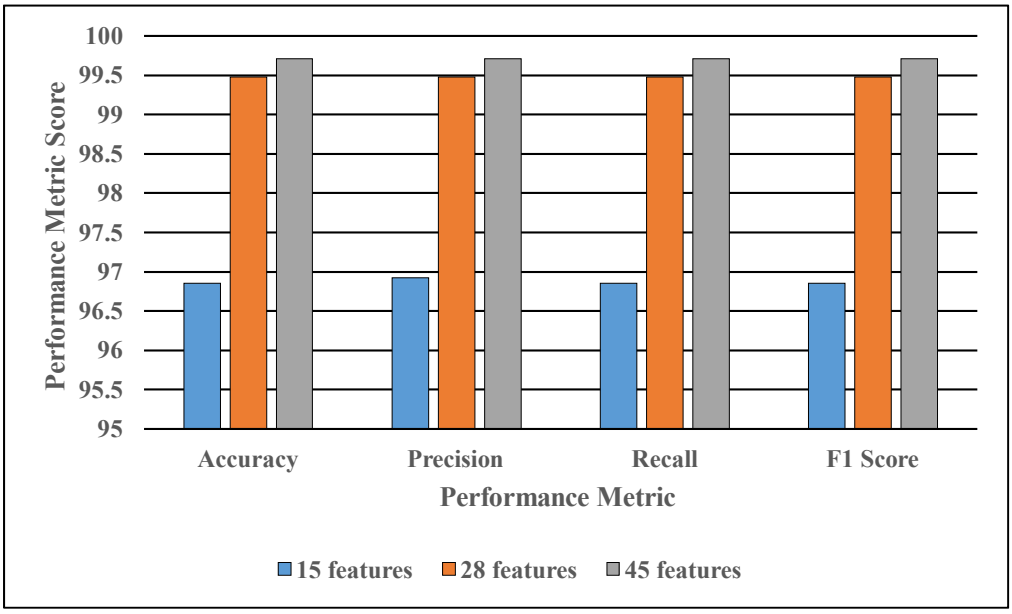


Figure 33: Performance evaluation metrics for B-ANIDS-DNN Classifier for different number of features

Figures 34, 35, 36 show the confusion matrices for the proposed B-ANIDS-DNN models when trained on 15, 28, 45 of input features respectively and tested on unseen data of the derived dataset.

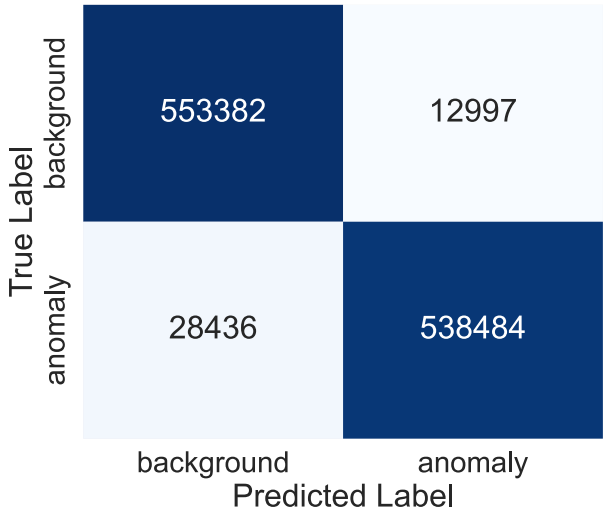


Figure 34: Confusion matrices for B-ANIDS-DNN with 15 input features

The number of incorrect detections (false negatives) was the highest when the minimal number of features are used whereas, the detection accuracy was highly improved when the most effective 45 features were considered for training the classifier. The same is for false positives where the normal connection is misclassified as malicious. However, a slight improvement in both precision and recall when 45 features are used instead of 28 features.

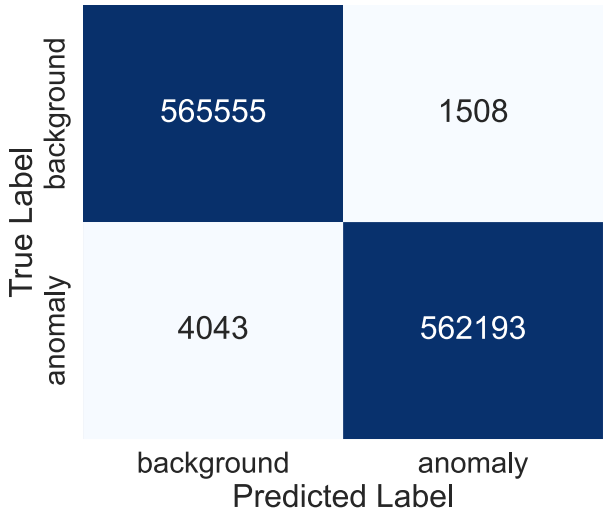


Figure 35: Confusion matrices for B-ANIDS-DNN with 28 input features

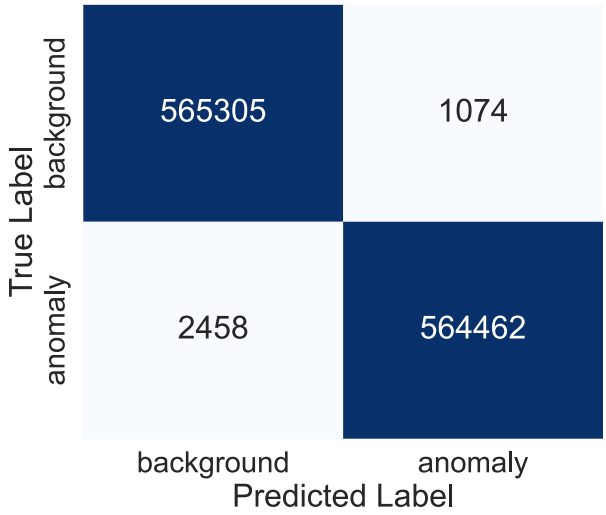


Figure 36: Confusion matrices for B-ANIDS-DNN with 45 input features

An excellent model has an Area Under the Curve (**AUC**) near **1** which means it has a good measure of separability. A poor model has an **AUC** near **0** which means it has the worst measure of separability and when the **AUC** is **0.5**, it means the model has no class separation capacity [50]. Figures 37, 38, 39 show that **AUC** is (**0.999915**) when 45 effective features are used to train the model and **AUC** is (**0.999814**) when 28 features are used. Also, **AUC** is (**0.993189**) when 15 features are used. So, the model has optimal separability when trained on 45 features.

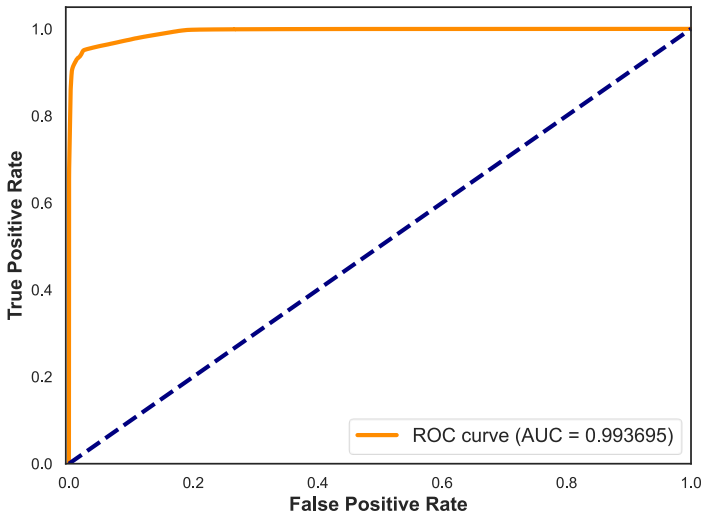


Figure 37: ROC curve for B-ANIDS-DNN with 15 input features

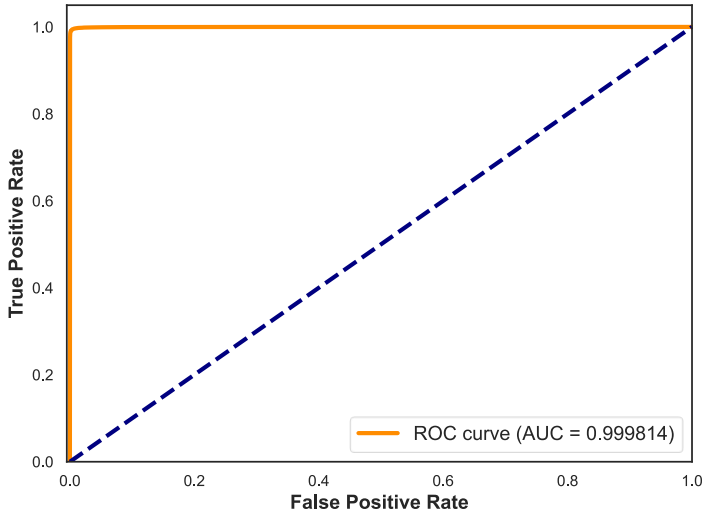


Figure 38: ROC curve for B-ANIDS-DNN with 28 input features

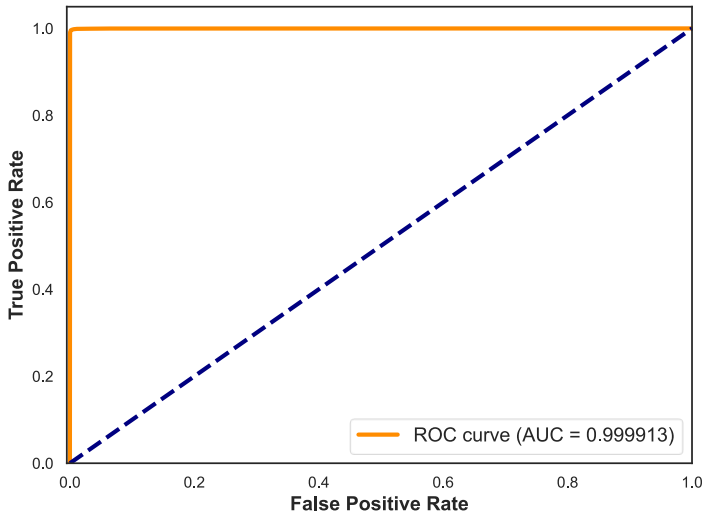


Figure 39: ROC curve for B-ANIDS-DNN with 45 input features

4.3.2 Multiclass ANIDS classifier (M-ANIDS-DNN) results:

The M-ANIDS-DNN model was trained on the train set. However, to overcome the limitation of class imbalance and to avoid bias, to majority classes a 10-fold stratified cross-validation technique was implemented. To evaluate the effectiveness and reliability of the model, it was evaluated on unseen data. The state-of-the-art metrics were used to evaluate and compare the results.

Table 15 shows the results of performance evaluation for multiclass classier M-ANIDS-DNN listed per class label and grouped by number of input features. The F1-score is the highest among all attack classes when 45 features are used. Moreover, the highest separable attacks are the anomaly ssh scan, UDP scan, DoS, and TCP Scan. The lowest detection rate is for Nerisbotnet attacks when only 15 input features are used.

Table 15: Performance evaluation metrics score [%] for M-ANIDS-DNN model listed per class label and grouped by number of input features

Number of Features	Performance Metric	Back-ground	Anomaly spam	Anomaly sshscan	Anomaly UDP scan	DoS	Neris-botnet	Scan
15	Precision	94.956	**90.305**	97.883	99.976	99.623	95.812	94.019
	Recall	98.017	94.636	99.485	100.000	100.000	**41.392**	99.835
	F1-Score	96.462	92.420	98.678	99.988	99.811	57.810	96.840
28	Precision	100.000	99.288	99.790	99.419	99.997	97.407	99.996
	Recall	99.999	99.760	99.749	99.920	100.000	**91.534**	99.875
	F1-Score	99.990	99.523	99.769	99.669	99.999	94.378	99.935
45	Precision	99.548	99.802	100.000	100.000	100.000	97.917	100.000
	Recall	99.817	99.826	99.977	99.999	99.999	**94.697**	99.898
	F1-Score	**99.682**	**99.814**	**99.989**	**99.999**	**99.999**	**96.280**	**99.949**

Figure 40 shows the precision for each class when different numbers of input features are used. Precision indicates that the classifier overestimates one class over another and then generates false positives. Therefore, higher precision means lower false positive rate. The false positive rate is reduced when number of input features was increased from 15 to 28 and then to 45.

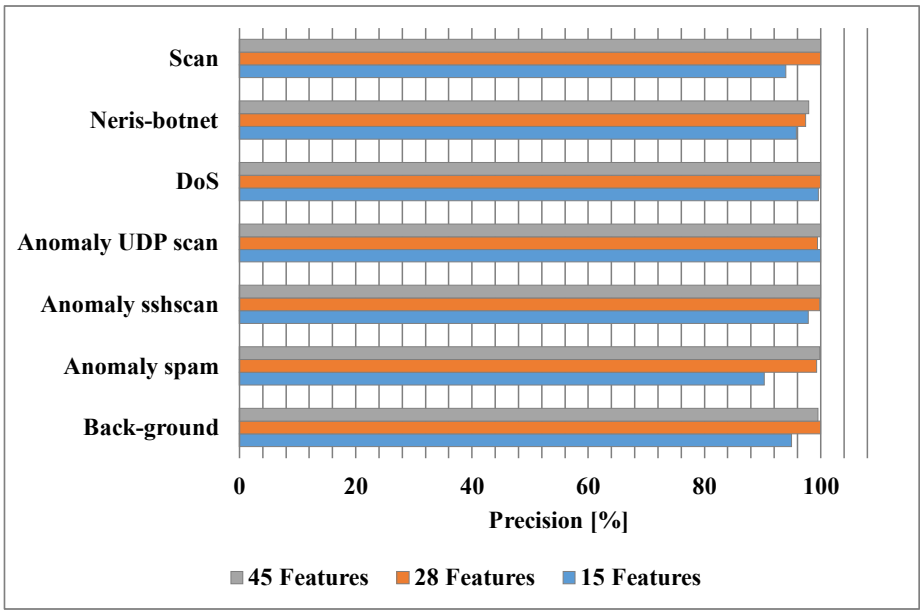


Figure 40:Precision [%] of M-ANIDS-DNN for individual classes & different number of input features

Figure 41 shows the recall or detection rate for each class when a different numbers of input features are used. The bar chart shows a high detection rate for Scan, DoS, and Anomaly UDP Scan attacks even when 15 input features are used. However, the detection rate for Anomaly Spam and Nerisbotnet attacks is improved when extra input features are used. Furthermore, the detection rate is improved slightly when number of input features is increased from 28 t0 45. The best recall for all classes is when 45 features are used as input for the model. A drop in Recall indicates that the model miss detected or underestimated a positive class and generates false negatives.

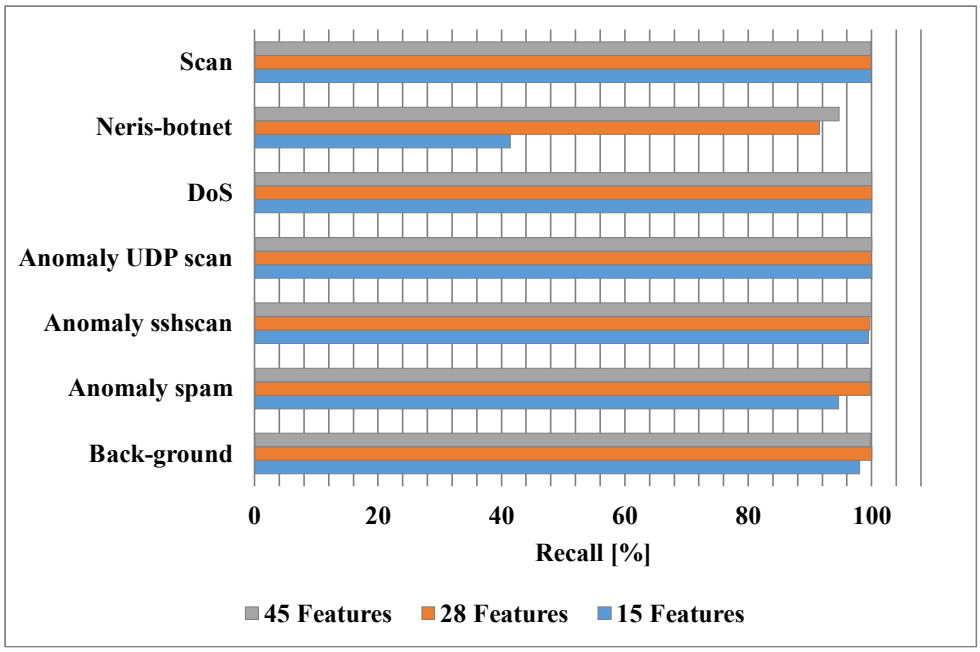


Figure 41: Recall [%] of M-ANIDS-DNN for individual classes & different numbers of input features

Figures 42,43,44 show the confusion matrix for the M-ANIDS-DNN classifier when 45, 28, 15 features are used respectively. The diagonal of the matrix shows the number of samples predicted as the true label. The depth of the color relates to the number of samples. The confusion matrix in figure 42 shows that 815 background samples are misclassified as nerisbotnet (false positives) and 3695 nerisbotnet samples are misclassified as background (false negatives). This indicates that the Nerisbotnet anomaly is not fully separable. Also, 76 background samples are misclassified as anomaly spam. And 76 anomaly spam samples are classified as background. Other classes are almost fully separable.

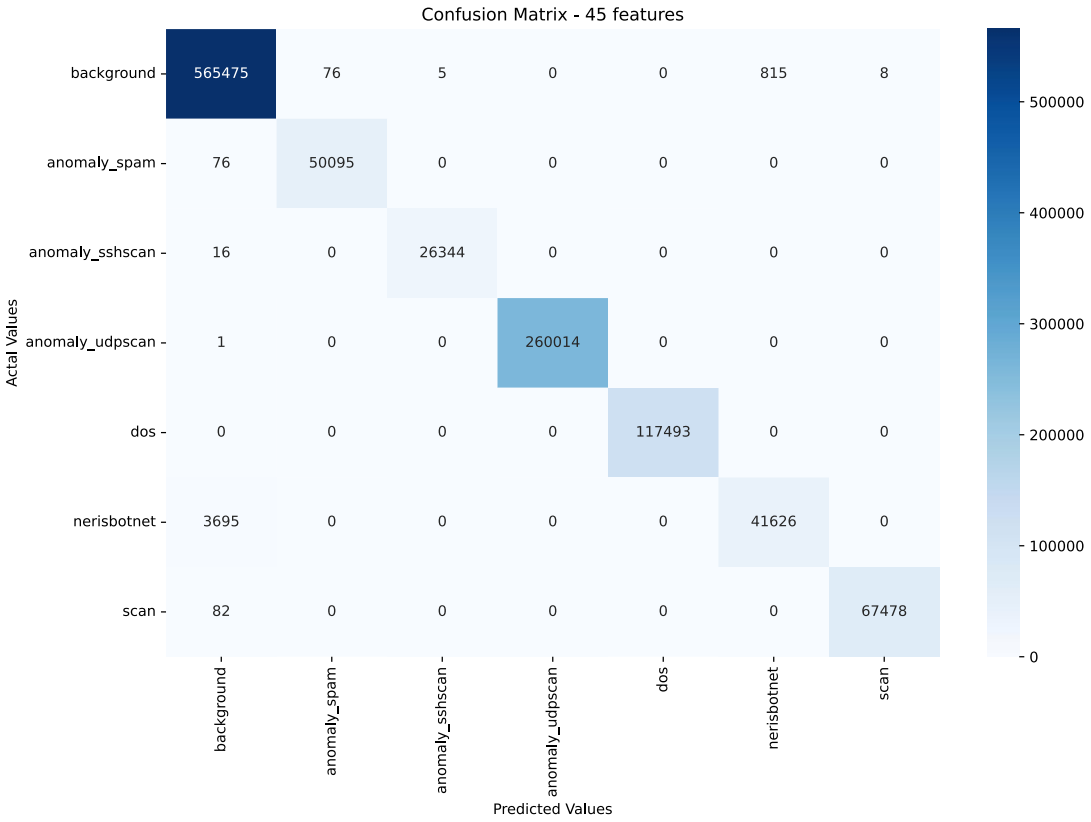


Figure 42: Confusion matrix for M-ANIDS-DNN (45 features)

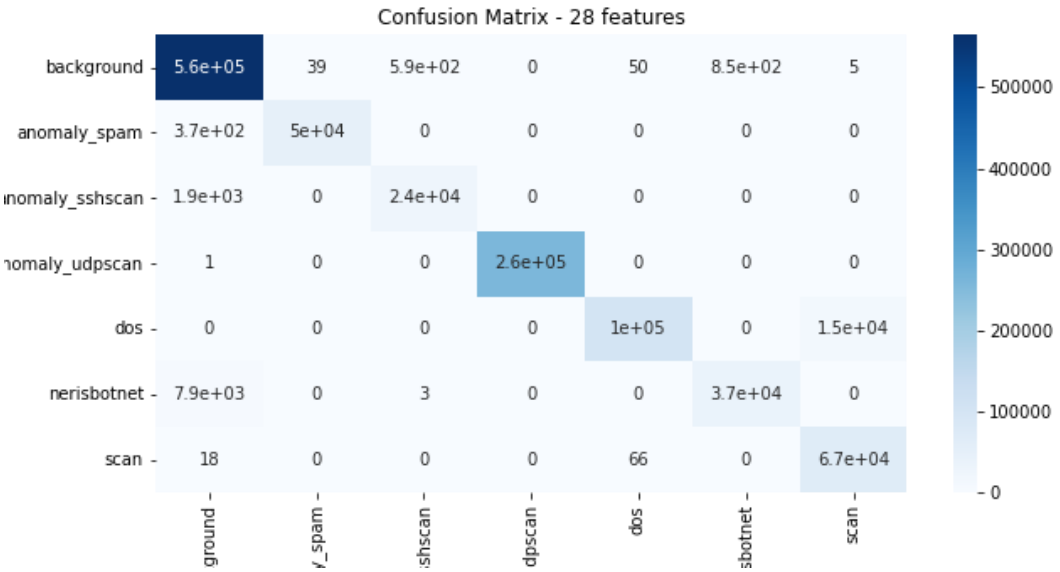


Figure 43: Confusion matrix for multiclass ANIDS model (28 features)

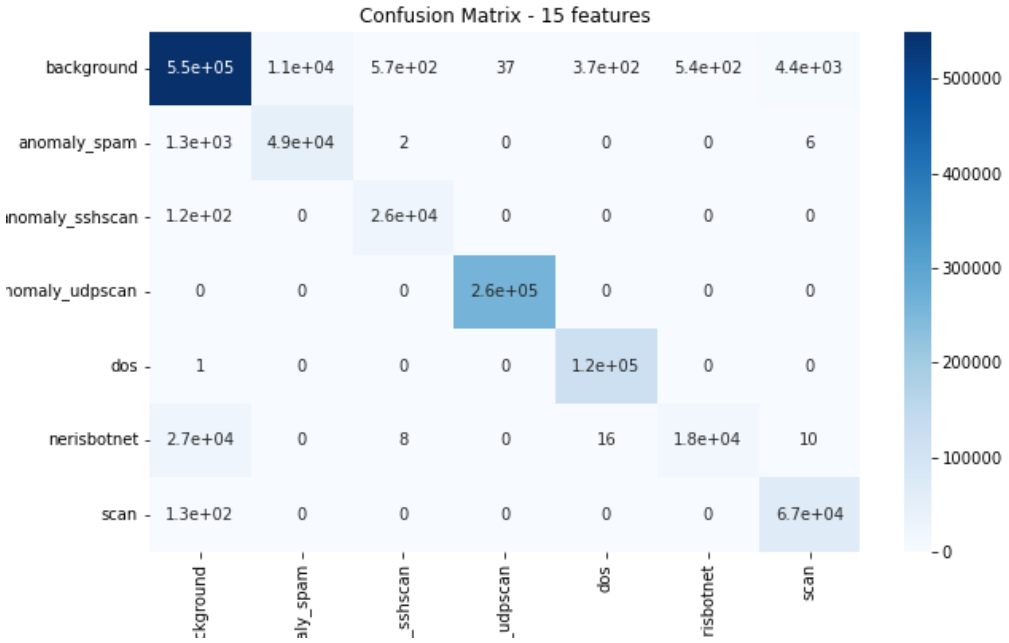


Figure 44: Confusion matrix for multiclass ANIDS model (15 features)

4.3.3 Performance Comparison with State-of-the-art Models

In order to assess the effectiveness and efficiency of the proposed approach, the experimental results will be compared with state-of-the-art models for anomaly-based intrusion detection systems evaluated on same UGR'16 dataset.

Magan et al. [33] proposed four ANIDS models based on four of the machine algorithms. The algorithms are multinomial Logistic Regression (LR), Random Forest (RF), Support Vector Machine with linear kernel (SVC-L), and Support Vector Machine with radial basis function kernel (SVC-RBF). They used a technique called Feature as a Counter (FaaC) [51] to aggregate the flows from UGR'16 dataset to capture the frequency patterns of the flows. The results of their models outperformed other statistical, unsupervised, and semi-supervised previous models. However, the experimental results of the proposed ANIDS-DNN model outperform the results of their models. Table 16 shows a comparison between one of their models which is the RF classifier with the proposed model ANIDS-DNN. Also figure 45 Compares the F1-score between the two models.

Table 16: Performance evaluation [%] comparison of ANIDS-DNN proposed model with Magan et al. model [33] using the same dataset UGR'16

Model	Algorithm	Metric	Back-ground	Anomaly spam	DoS	Neris-botnet	Scan
Magan	RF	Precision	88.50	91.70	97.30	97.70	93.20
		Recall	90.60	74.90	88.40	92.20	92.50
		F1-Score	92.10	82.40	92.50	94.80	92.80
ANIDS-DNN	DNN	Precision	99.55	99.80	100.00	97.92	100.00
		Recall	99.82	99.83	100.00	94.70	99.90
		F1-Score	99.68	99.81	100.00	96.28	99.95

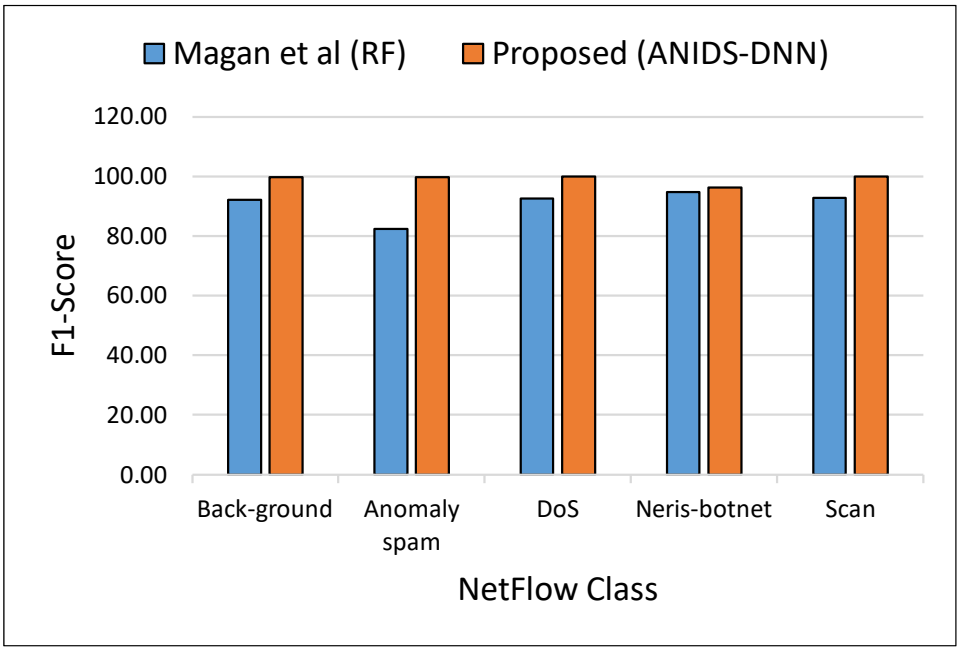


Figure 45: Comparison of F1-score between Magan et al. RF model and proposed ANIDS-DNN model when both evaluated on the UGR'16 dataset

Rajagopal et al.[52] proposed an ensemble stacker model where three base classifiers namely random forest, KNN, and logistic regression are trained on UGR'16 dataset. The output of the base classifiers then used to train a Support Vector Machine metaclassifier. The performance of their proposed model was evaluated on UGR'16 dataset. However, only two performance metrics are calculated precision and recall. The results of our proposed model outperform the results of their model as listed in Table 17.

Table 17: Performance evaluation [%] comparison of ANIDS-DNN proposed model with Rajagopal et al. [52] model using the same dataset UGR'16

Model	Algorithm	Metric	Anomaly spam	Anomaly sshscan	Anomaly UDP scan	DoS	Scan
Rajagopal et al.	**Ensemble Stacker**	Precision	98.90	99.00	98.10	98.70	98.90
		Recall	98.60	99.10	97.80	99.00	98.90
ANIDS-DNN	**DNN**	Precision	99.80	100.00	100.00	100.00	100.00
		Recall	99.83	99.98	100.00	100.00	99.90

As depicted in Figure 46 and Figure 47 the comparison of performance between ANIDS-DNN proposed model and the ensemble stacker shows that our proposed model achieved higher precision and recall for all classes of attacks.

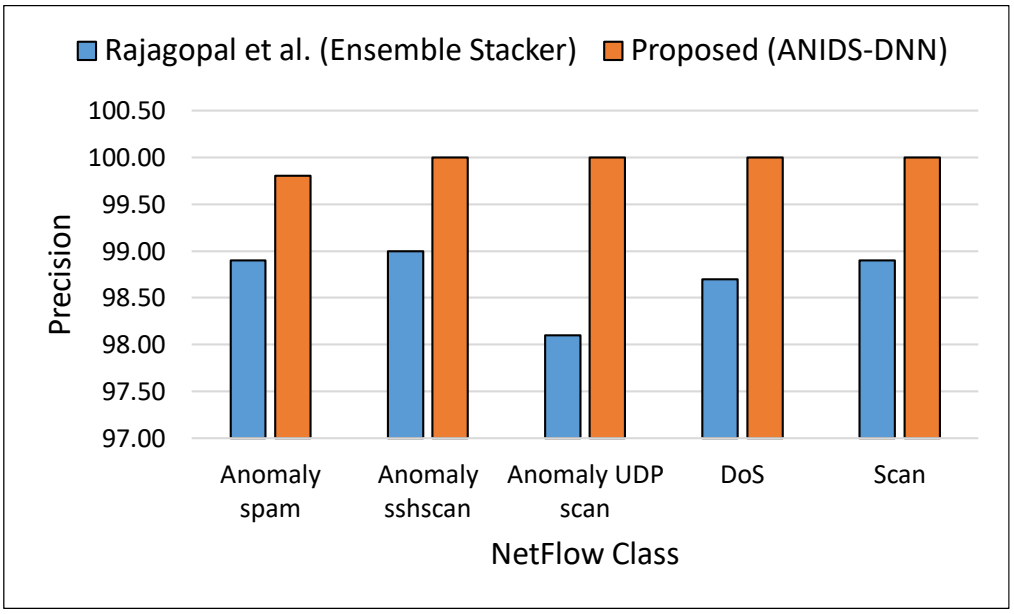


Figure 46: comparison of the precision of ANIDS-DNN proposed model with the precision of Rajagopal et al. model using the same dataset UGR'16

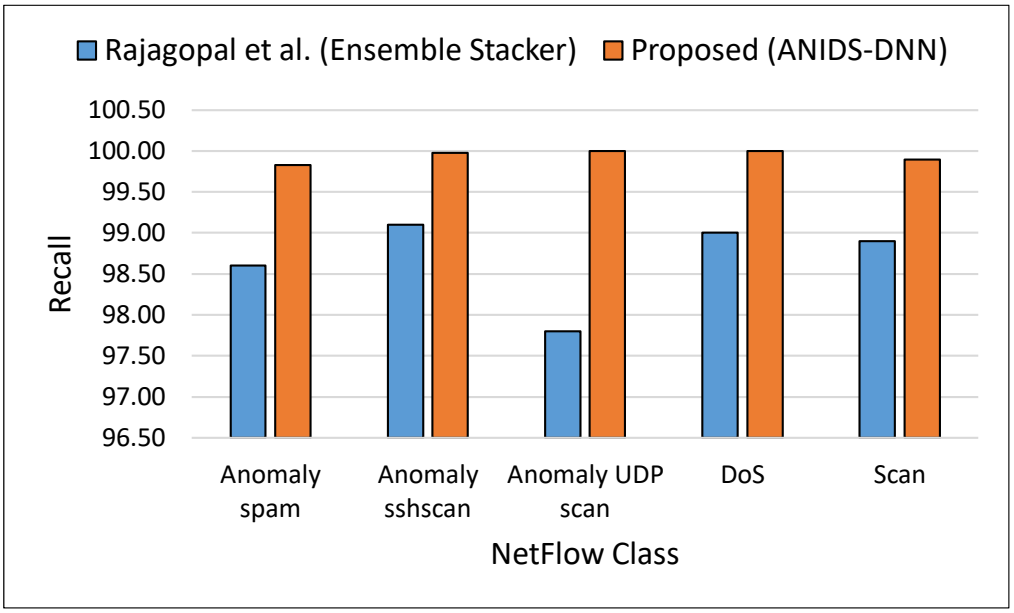


Figure 47: comparison of the recall of ANIDS-DNN proposed model with the recall of Rajagopal et al. model using the same dataset UGR'16

4.4 Summary:

From analysis of experimental results, the optimal network architecture was when three hidden layers are used to build the model with 90, 60, 45 neurons in the hidden layers respectively. Also, the best performance was when the 45 most important features were selected. The proposed B-ANIDS-DNN model achieves the value of classification accuracy of network flows up to 99.73%, and the ROC-AUC of 0.999915. The proposed multiclass classifier M-ANID-DNN model achieved F1-score of 99.682% in classifying background traffic, F1-score of 99.814% in detecting anomaly spam, F1-score of 99.989% in detecting sshscan, F1-score of almost 100% in classifying anomaly UDP scan and DoS attacks, F1-score of 96.280% in classifying nerisbotnet attacks, and F1-score of 99.949% in classifying scan attacks. Furthermore, the performance of the proposed model is compared with other state -of-the-art random forest and ensemble classifier which are evaluated on same UGR'16 dataset. The performance of the proposed model outperforms models proposed in [33] in terms of F1-score and model proposed in [52] in terms of both precision and recall.

Chapter Five: Discussion and Conclusion

Chapter Outline

Chapter Five: Discussion and Conclusion

5.1 Discussion of the Results

From the experimental results the following findings can be concluded:

- **Imbalanced dataset**: The deep learning ANIDS model can capture and learn normal and anomalous patterns from data by learning from examples. To minimize training error sufficient learning examples for each pattern or class must be provided to train the model. The deep learning algorithm will be biased toward the majority class and the detection accuracy will be low for minority classes. This issue is solved by under-sampling background traffic and increasing the samples of attack classes. To minimize generalization error the stratified 10-fold cross-validation can be used to train and validate the model on all samples of minority classes.
- **Feature Engineering**: The classification model maps the input features to a class label or a target. Input features should be relevant to target classes and extracted from raw data using feature engineering techniques. The relevancy of input features impacts the accuracy of the model. Figure 14 shows the improvement in all performance metrics when the model is trained on more relevant features.
- **Feature Selection**: The quantity of the input features affects both the performance and capacity of the model. Selecting important and uncorrelated features will improve detection accuracy and reduce the total number of model parameters and thus reduce the complexity of the model. Table 12 shows a slight performance improvement when 45 are used instead of 28 and considerable improvement when 28 features are used instead of 15 basic features. However, there is a tradeoff between detection accuracy and model complexity. Random forest is effective when used in the context of intrusion detection, so it was used for feature selection.

- **Hyperparameters tuning** choosing the right hyperparameters will improve both training and generalization error. From Table 10 it can be noted that increasing the number of hidden layers and hidden units affects the performance and complexity of the model and should be tuned so that the model can match the complexity of the problem to be solved, the number of input features, and availability of a sufficient number of training examples, especially for minority classes. The best hyperparameters could be searched automatically using grid search and random search techniques or manually based on heuristics and related previous research.
- **Performance metrics** using state-of-the-art performance metrics will help to diagnose problems and improve performance and provide comparative analysis. ROC curve and confusion matrix visualize the performance of the model. AUC is a measure of the discriminating power of the classifier. The closer the area is to one, the better the performance of the model in terms of recall and precision.

5.2 Conclusion

In this thesis, deep learning algorithms were utilized to build predictive models for anomaly-based network intrusion detection systems. Network flow features extracted from packet metadata were used as input to the models. In binary classifier, the output is a decision on either the flow normal or anomaly. In a multiclass classifier, the output is a class label for the flow. The class labels are learned from the training dataset.

One of the main objectives of this research is to investigate the efficacy of deep learning algorithms when applied to modeling anomaly-based network intrusion detection systems (ANIDS). Previous studies in the field highlighted some challenges such as low detection accuracy, high false alarms, and miss detection of unknown attacks.

Based on the experimental results, both binary and multiclass ANIDS-DNN models achieved high performance based on state-of-the-art performance metrics given a benchmark dataset UGR'16. It has been noted that to build, train, and evaluate an efficient and reliable ANIDS based on deep learning the following points should be considered. First, the supervised feedforward deep network should be trained on a sufficiently large and labeled dataset. The quality, accuracy, and currency of the dataset are highly important for the performance and generalization of the model. Also, the training set should have several samples for each class that is sufficient for the algorithm to learn and capture the pattern of that class. Sampling strategies and k-fold stratified cross-validation techniques can be used to solve the issue of class imbalance in the dataset.

Second, preparing data is of high impact on training error and generalization error of deep learning-based models. For instance, feature engineering can be used to extract or transform features from raw data that represent the temporal or spatial associations of data. Furthermore, deep learning algorithms can only process numerical data therefore input features should be preprocessed beforehand. Deep learning model capacity is dependent on the number of input features so selecting the uncorrelated and important features will improve model performance and efficiency and reduce model complexity.

Third, the hyperparameters of the model should be tuned to minimize both training error and generalization error. Although the parameters of the model are learned from data, hyperparameters should be set in advance. Grid search and random search are techniques to search for the best hyperparameters. Nevertheless, heuristics and literature may be used to manually adjust the hyperparameters of the model.

To conclude, deep learning algorithms are powerful tools for building predictive models. They are capable of learning patterns from data and approximate functions that map inputs to outputs efficiently. When they are applied to network security context to

model anomaly-based network intrusion detection systems (ANIDS), they have high performance as classifiers in terms of accuracy of detection and recognizing different patterns of anomalies in known and unknown network traffic given a benchmark dataset.

5.3 Recommendations and Future Work

To extend this study, the following directions could be considered for future work. First, to integrate deep learning ANIDS models in the real world a big data platform such as Hadoop or Apache Spark in a distributed network is needed to overcome memory and computational constraints. Second, supervised learning models can only learn from labeled datasets. In the real world creating labeled intrusion detection datasets is time and resource-consuming. As an alternative approach, semi-supervised learning enables models to learn from both labeled and unlabeled data. Third, many attacks generate high traffic within a short or long temporal sequence. So, these types of attacks can be detected by capturing the sequence using recurrent neural networks (RNN) or short-term memory algorithms (LSTM) and thus improve the detection of unknown attacks. Fourth, Intrusion detection datasets are usually imbalanced. Attacks are usually minority classes compared to normal traffic. Though adversarial algorithms such as GANs and autoencoders can be used to model the probability distribution of minority classes and increase the number of samples of minority classes to improve the generalization of deep learning models.

References

[1] S. Northcutt and J. Novak, Network Intrusion Detection, Third Edition. Sams Publishing, 2002.

[2] R. Weaver, D. Weaver, and D. Farwood, Guide to Network Defense and Countermeasures, Third Edition. Cengage Learning, 2014.

[3] M. S. Abdel-Wahab, A. M. Neil, and A. Atia, "A Comparative Study of Machine Learning and Deep Learning in Network Anomaly-Based Intrusion Detection Systems," in Proceedings of ICCES 2020 - 2020 15th International Conference on Computer Engineering and Systems, Dec. 2020. doi: 10.1109/ICCES51560.2020.9334553.

[4] A. Khraisat, I. Gondal, P. Vamplew, and J. Kamruzzaman, "Survey of intrusion detection systems: techniques, datasets and challenges," Cybersecurity, vol. 2, no. 1, Dec. 2019, doi: 10.1186/s42400-019-0038-7.

[5] Z. Ahmad, A. Shahid Khan, C. Wai Shiang, J. Abdullah, and F. Ahmad, "Network intrusion detection system: A systematic study of machine learning and deep learning approaches," Transactions on Emerging Telecommunications Technologies, vol. 32, no. 1, Jan. 2021, doi: 10.1002/ett.4150.

[6] I. H. Sarker, A. S. M. Kayes, S. Badsha, H. Alqahtani, P. Watters, and A. Ng, "Cybersecurity data science: an overview from machine learning perspective," J Big Data, vol. 7, no. 1, Dec. 2020, doi: 10.1186/s40537-020-00318-5.

[7] H. Liu and B. Lang, "Machine learning and deep learning methods for intrusion detection systems: A survey," Applied Sciences (Switzerland), vol. 9, no. 20. MDPI AG, Oct. 01, 2019. doi: 10.3390/app9204396.

[8] M. Nkongolo, J. P. van Deventer, and S. M. Kasongo, "Ugransome1819: A novel dataset for anomaly detection and zero-day threats," Information (Switzerland), vol. 12, no. 10, Oct. 2021, doi: 10.3390/info12100405.

[9] P. García-Teodoro, J. Díaz-Verdejo, G. Maciá-Fernández, and E. Vázquez, "Anomaly-based network intrusion detection: Techniques, systems and challenges," Comput Secur, vol. 28, no. 1–2, pp. 18–28, Feb. 2009, doi: 10.1016/j.cose.2008.08.003.

[10] A. D. Lopez, A. P. Mohan, S. Nair, A. Lopez, and A. Mohan, "Network Traffic Behavioral Analytics for Detection of DDoS Attacks." [Online]. Available: https://scholar.smu.edu/datasciencereviewAvailableat:https://scholar.smu.edu/datasciencereview/vol2/iss1/14http://digitalrepository.smu.edu.

[11] W. Stallings, Data and computer communications, Tenth Edition. Pearson, 2014.

[12] J. F. Kurose and K. W. Ross, Computer Networking: A Top-Down Approach, Sixth Edition. Pearson, 2013.

[13] S. Jin, S. Guochun, and Z. Chunhua, “Research on the Anomaly Detection of Flow Streaming Technology in Network,” 2015.

[14] B. Li, J. Springer, G. Bebis, and M. Hadi Gunes, “A survey of network flow applications,” Journal of Network and Computer Applications, vol. 36, no. 2. pp. 567–581, Mar. 2013. doi: 10.1016/j.jnca.2012.12.020.

[15] L. García, G. Mací a-FernándezFern, R. Molina, and S. Member, “Leveraging a Probabilistic PCA model to Understand the Multivariate Statistical Network Monitoring Framework for Network Security Anomaly Detection.”

[16] S. Engle, Institute of Electrical and Electronics Engineers, and Ariz.) IEEE Conference on Visualization (2017 : Phoenix, VizSec 2017 : 2017 IEEE Symposium on Visualization for Cyber Security (VizSec) : Phoenix, Arizona, USA, October 2, 2017.

[17] M. H. Bhuyan, D. K. Bhattacharyya, and J. K. Kalita, “Network Traa c Anomaly Detection and Prevention.” Accessed: Nov. 02, 2022. [Online]. Available: http://www.springer.com/series/4198

[18] M. Stevanovic and J. M. Pedersen, “An analysis of network traffic classification for botnet detection.”

[19] K. G. Mehrotra, C. K. Mohan, and H. Huang, “Terrorism, Security, and Computation Anomaly Detection Principles and Algorithms.” [Online]. Available: http://www.springer.com/series/11955

[20] M. S. Abdel-Wahab, A. M. Neil, and A. Atia, “A Comparative Study of Machine Learning and Deep Learning in Network Anomaly-Based Intrusion Detection Systems,” in Proceedings of ICCES 2020 - 2020 15th International Conference on Computer Engineering and Systems, Dec. 2020. doi: 10.1109/ICCES51560.2020.9334553.

[21] F. Pedregosa FABIANPEDREGOSA et al., “Scikit-learn: Machine Learning in Python Gaël Varoquaux Bertrand Thirion Vincent Dubourg Alexandre Passos PEDREGOSA, VAROQUAUX, GRAMFORT ET AL. Matthieu Perrot,” 2011. [Online]. Available: http://scikit-learn.sourceforge.net.

[22] M. Abadi et al., “TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems.” [Online]. Available: www.tensorflow.org.

[23] I. Witten, E. Frank, M. Hall, and C. Pal, Data Mining, Fourth Edition. Elsevier, 2017.

[24] L. Breiman, “Random Forests,” 2001.

[25] I. Goodfellow, Y. Bengio, and A. Courville, Deep Learning. Massachusetts Institute of Technology, 2016.

[26] G. Zhong, L. N. Wang, X. Ling, and J. Dong, “An overview on data representation learning: From traditional feature learning to recent deep learning,” Journal of Finance and Data Science, vol. 2, no. 4. KeAi Communications Co., pp. 265–278, Dec. 01, 2016. doi: 10.1016/j.jfds.2017.05.001.

[27] Y. Bengio, A. Courville, and P. Vincent, “Representation Learning: A Review and New Perspectives,” Jun. 2012, [Online]. Available: http://arxiv.org/abs/1206.5538

[28] I. Al-Turaiki and N. Altwaijry, “A Convolutional Neural Network for Improved Anomaly-Based Network Intrusion Detection,” Big Data, vol. 9, no. 3, pp. 233–252, Jun. 2021, doi: 10.1089/big.2020.0263.

[29] H. Hindy et al., “A Taxonomy of Network Threats and the Effect of Current Datasets on Intrusion Detection Systems,” IEEE Access, vol. 8, pp. 104650–104675, 2020, doi: 10.1109/ACCESS.2020.3000179.

[30] Canadian Institute for Cybersecurity, “NSL-KDD Dataset,” 2009. http://www.unb.ca/cic/datasets/nsl.html (accessed Oct. 22, 2021).

[31] N. Moustafa and J. Slay, “UNSW-NB15: A Comprehensive Data set for Network Intrusion Detection systems (UNSW-NB15 Network Data Set).” Accessed: Mar. 22, 2022. [Online]. Available: https://cve.mitre.org/

[32] G. Maciá-Fernández, J. Camacho, R. Magán-Carrión, P. García-Teodoro, and R. Therón, “UGR‘16: A new dataset for the evaluation of cyclostationarity-based network IDSs,” Comput Secur, vol. 73, pp. 411–424, Mar. 2018, doi: 10.1016/j.cose.2017.11.004.

[33] R. Magán-Carrión, D. Urda, I. Díaz-Cano, and B. Dorronsoro, “Towards a reliable comparison and evaluation of network intrusion detection systems based on machine learning approaches,” Applied Sciences (Switzerland), vol. 10, no. 5, Mar. 2020, doi: 10.3390/app10051775.

[34] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, “Toward generating a new intrusion detection dataset and intrusion traffic characterization,” in ICISSP 2018 - Proceedings of the 4th International Conference on Information Systems Security and Privacy, 2018, vol. 2018-January, pp. 108–116. doi: 10.5220/0006639801080116.

[35] C. Yin, Y. Zhu, J. Fei, and X. He, “A Deep Learning Approach for Intrusion Detection Using Recurrent Neural Networks,” IEEE Access, vol. 5, pp. 21954–21961, Oct. 2017, doi: 10.1109/ACCESS.2017.2762418.

[36] Y. Jia, M. Wang, and Y. Wang, “Network intrusion detection algorithm based on deep neural network,” IET Inf Secur, vol. 13, no. 1, pp. 48–53, Jan. 2019, doi: 10.1049/iet-ifs.2018.5258.

[37] M. Sarhan, S. Layeghy, N. Moustafa, and M. Portmann, “NetFlow Datasets for Machine Learning-based Network Intrusion Detection Systems,” Nov. 2020, doi: 10.1007/978-3-030-72802-1_9.

[38] R. Vinayakumar, M. Alazab, K. P. Soman, P. Poornachandran, A. Al-Nemrat, and S. Venkatraman, “Deep Learning Approach for Intelligent Intrusion Detection System,” IEEE Access, vol. 7, pp. 41525–41550, 2019, doi: 10.1109/ACCESS.2019.2895334.

[39] V. Sstla, V. K. K. Kolli, L. K. Voggu, R. Bhavanam, and S. Vallabhasoyula, “Predictive model for network intrusion detection system using deep learning,”

Revue d'Intelligence Artificielle, vol. 34, no. 3, pp. 323–330, Jun. 2020, doi: 10.18280/ria.340310.

[40] N. Altwaijry, A. ALQahtani, and I. AlTuraiki, "A Deep Learning Approach for Anomaly-Based Network Intrusion Detection," in Communications in Computer and Information Science, 2020, vol. 1210 CCIS, pp. 603–615. doi: 10.1007/978-981-15-7530-3_46.

[41] Institute of Electrical and Electronics Engineers. and IEEE Computational Intelligence Society., IEEE Symposium on Computational Intelligence for Security and Defense Applications (CISDA), 2009 : [Ottawa, Canada, 8-10 July 2009]. IEEE, 2009.

[42] Institute of Electrical and Electronics Engineers, SoutheastCon 2016 : 30 March-3 April 2016.

[43] S. Amari and S. Wu, "Improving support vector machine classifiers by modifying kernel functions." [Online]. Available: www.elsevier.com/locate/neunet

[44] M. Loecher, "Unbiased variable importance for random forests," Commun Stat Theory Methods, vol. 51, no. 5, pp. 1413–1425, 2022, doi: 10.1080/03610926.2020.1764042.

[45] Y. Z. Chen, Z. G. Huang, S. Xu, and Y. C. Lai, "Spatiotemporal patterns and predictability of cyberattacks," PLoS One, vol. 10, no. 5, May 2015, doi: 10.1371/journal.pone.0124472.

[46] S. Ullah et al., "A New Intrusion Detection System for the Internet of Things via Deep Convolutional Neural Network and Feature Engineering," Sensors, vol. 22, no. 10, May 2022, doi: 10.3390/s22103607.

[47] S. Johan Knapskog, S. Nilsen, and A. Hegna, "Master of Science in Communication Technology Visualizing Spatial and Temporal Dynamics of a Class of IRC-Based Botnets," 2010.

[48] S. García, M. Grill, J. Stiborek, and A. Zunino, "An empirical comparison of botnet detection methods," Comput Secur, vol. 45, pp. 100–123, 2014, doi: 10.1016/j.cose.2014.05.011.

[49] M. Heydarian, T. E. Doyle, and R. Samavi, "MLCM: Multi-Label Confusion Matrix," IEEE Access, vol. 10, pp. 19083–19095, 2022, doi: 10.1109/ACCESS.2022.3151048.

[50] R. Magán-Carrión, D. Urda, I. Díaz-Cano, and B. Dorronsoro, "Improving the Reliability of Network Intrusion Detection Systems through Dataset Integration," Dec. 2021, doi: 10.1109/TETC.2022.3178283.

[51] J. Camacho, J. M. García-Giménez, N. M. Fuentes-García, and G. Maciá-Fernández, "Multivariate Big Data Analysis for Intrusion Detection: 5 steps from the haystack to the needle," Jun. 2019, doi: 10.1016/j.cose.2019.101603.

[52] S. Rajagopal, P. P. Kundapur, and K. S. Hareesha, “A Stacking Ensemble for Network Intrusion Detection Using Heterogeneous Datasets,” Security and Communication Networks, vol. 2020, 2020, doi: 10.1155/2020/4586875.

Appendix A: Python Source Code

A.1 Python source code organization and dataset files

The python source code for building, training, and evaluating both binary and multiclass ANIDS models using UGR'16 datasets is organized as follows:

- Five CSV files with different time stamps are selected from UGR'16 dataset.
    1. april.week3.004.csv
    2. april.week3.024.csv
    3. july.week5.006.csv
    4. july.week5.007.csv
    5. august.week1.004.csv
- One preprocessed, scaled and balanced derived subset is generated and can be used to train and test other ANIDS classifiers.
    1. derived_dataset_scaled_45features.csv
- Cleaning and sampling and feature extraction Jupyter notebook.
    1. Derived_Dataset.ipynb
- Data Preprocessing and feature selection Jupyter notebook.
    1. Derived_Dataset_Preprocessing.ipynb
- Binary ANIDS-DNN model building, training, and evaluation Jupyter notebook.
    1. Binary-classification_45-features-batch512.ipynb
- Multiclass ANIDS-DNN model building, training, and evaluation Jupyter notebook.
    1. Multiclass-classification_45-features.ipynb

A.2 Sample Python code

 # Build ANIDS-DNN Model
 # Import libraries

```
import numpy as np

import pandas as pd

import time

from sklearn.model_selection import train_test_split

from sklearn.model_selection import GridSearchCV

from sklearn import svm

from sklearn import metrics

from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

from sklearn.metrics import classification_report

import tensorflow as tf

from tensorflow.keras.models import Sequential

from tensorflow.keras.models import load_model

from tensorflow.keras.layers import Dense

from tensorflow.keras.layers import Dropout

from tensorflow.keras.callbacks import EarlyStopping

from tensorflow.keras.callbacks import ModelCheckpoint

from tensorflow.keras.wrappers.scikit_learn import KerasClassifier

from tensorflow.keras.constraints import MaxNorm

import matplotlib.pyplot as plt

from sklearn.ensemble import RandomForestClassifier

from sklearn.feature_selection import SelectFromModel

from datetime import datetime, timedelta

from math import log
```

```
from sklearn.base import TransformerMixin


# Function to create model, required for KerasClassifier
def create_model():
    # create model
    model = Sequential()
    model.add(Dense(90, input_dim=45, kernel_initializer='uniform', activation='relu'))
    model.add(Dense(60, kernel_initializer='uniform', activation='relu'))
    model.add(Dropout(0.2))
    model.add(Dense(45, kernel_initializer='uniform', activation='relu'))
    model.add(Dropout(0.2))
    model.add(Dense(1, kernel_initializer='uniform', activation='sigmoid'))
    # Compile model
    model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=1e-3),
            loss=tf.keras.losses.BinaryCrossentropy(),
            metrics=[tf.keras.metrics.BinaryAccuracy(),
                    tf.keras.metrics.FalseNegatives(),
                    tf.keras.metrics.FalsePositives(),
                    tf.keras.metrics.Precision(),
                    tf.keras.metrics.Recall(),
                    tf.keras.metrics.AUC()])
    return model
```

Appendix B: UGR'16 Preprocessed Features

The details of selected input features used to train the ANIDS-DNN models

Table 18: Details of the 15 features in the UGR'16 dataset

No.	Feature name	Type
1	td	continuous
2	sp	discrete
3	dp	discrete
4	stos	discrete
5	pkt	continuous
6	byt	continuous
7	protocol_TCP	boolean
8	protocol_UDP	boolean
9	protocol_ICMP	boolean
10	protocol_other	boolean
11	flag_Ack	boolean
12	flag_Psh	boolean
13	flag_Rst	boolean
14	flag_Syn	boolean
15	flag_Fin	boolean

Table 19: Details of the 28 features in the UGR'16 dataset

No.	Feature name	Type
1	td	continuous
2	sa_nsessions_T	continuous
3	da_nsessions_T	continuous
4	flag_Ack	boolean
5	flag_Psh	boolean
6	flag_Rst	boolean
7	flag_Syn	boolean
8	flag_Fin	boolean
9	sport_reserved	boolean
10	sport_register	boolean
11	sport_private	boolean
12	sport_telnet	boolean
13	sport_smtp	boolean
14	sport_http	boolean
15	dport_reserved	boolean
16	dport_register	boolean
17	dport_private	boolean
18	dport_telnet	boolean
19	dport_smtp	boolean

20	dport_http	boolean
21	protocol_TCP	boolean
22	protocol_UDP	boolean
23	protocol_ICMP	boolean
24	protocol_other	boolean
25	stos_zero	boolean
26	stos_other	boolean
27	pkt	continuous
28	byt	Continuous

Table 20: Details of the 45 features in the UGR'16 dataset

No.	Feature name	Type
1	td	continuous
2	sa_nsessions_T	continuous
3	da_nsessions_T	continuous
4	sa_da_nsessions_T	continuous
5	flag_Ack	boolean
6	flag_Psh	boolean
7	flag_Rst	boolean
8	flag_Syn	boolean
9	flag_Fin	boolean
10	sport_reserved	boolean
11	sport_register	boolean
12	sport_private	boolean
13	sport_ftp_control	boolean
14	sport_ssh	boolean
15	sport_telnet	boolean
16	sport_smtp	boolean
17	sport_dns	boolean
18	sport_http	boolean
19	sport_pop3	boolean
20	sport_ntp	boolean
21	sport_https	boolean
22	sport_mds	boolean
23	sport_http2	boolean
24	dport_reserved	boolean
25	dport_register	boolean
26	dport_private	boolean
27	dport_ftp_control	boolean
28	dport_ssh	boolean
29	dport_telnet	boolean
30	dport_smtp	boolean
31	dport_dns	boolean
32	dport_http	boolean

33	dport_pop3	boolean
34	dport_ntp	boolean
35	dport_https	boolean
36	dport_mds	boolean
37	dport_http2	boolean
38	protocol_TCP	boolean
39	protocol_UDP	boolean
40	protocol_ICMP	boolean
41	protocol_other	boolean
42	stos_zero	boolean
43	stos_other	boolean
44	pkt	continuous
45	byt	continuous

Table 21: Target classes of multiclass ANIDS

No.	Class Label	Type
1	background	boolean
2	anomaly_spam	boolean
3	anomaly_sshscan	boolean
4	anomaly_udpscan	boolean
5	dos	boolean
6	nerisbotnet	boolean
7	scan	boolean