REPUBLIC OF TURKEY

ALTINBAŞ UNIVERSITY

Institute of Graduate Studies

Information Technologies

APPLICATION OF DEEP NEURAL NETWORKS FOR NETWORK INTRUSION DETECTION SYSTEMS IN CYBER SECURITY

Thamer OTHMAN

Master of Science

Supervisor

Asst. Prof. Dr. Sefer KURNAZ

Istanbul, 2021

APPLICATION OF DEEP NEURAL NETWORKS FOR NETWORK INTRUSION DETECTION SYSTEMS IN CYBER SECURITY

by

Thamer OTHMAN

Information Technologies

Master of Science

ALTINBAŞ UNIVERSITY

2021

The thesis titled “APPLICATION OF DEEP NEURAL NETWORKS FOR NETWORK INTRUSION DETECTION SYSTEMS IN CYBER SECURITY” prepared and presented by “Thamer OTHMAN” was accepted as a Master of Science Thesis in Information Technologies.

Asst. Prof. Dr. Sefer KURNAZ

Supervisor

Thesis Defense Jury Members:

Prof. Dr. Mesut RAZBONYALI.	School of Engineering and Natural Sciences, Maltepe University	_____
Prof. Dr. Osman Nuri UÇAN.	School of Engineering and Natural Sciences, Altinbas University	_____
Asst. Prof. Dr. Sefer KURNAZ.	School of Engineering and Natural Sciences, Altinbas University	_____

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.

Approval Date of Institute of Graduate Studies: ____/____/____

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

Thamer OTHMAN

Signature

ACKNOWLEDGEMENTS

I might want to thank my administrators: Asst. Prof. Dr. Sefer KURNAZ Please let me express my profound feeling of appreciation and gratefulness to both of you for the information: direction and unrestricted help you have given me. I want you to enjoy all that life has to offer and further achievement and accomplishments throughout your life.

ABSTRACT

APPLICATION OF DEEP NEURAL NETWORKS FOR NETWORK INTRUSION DETECTION SYSTEMS IN CYBER SECURITY

Thamer OTHMAN,

M.Sc., Information Technologies, Altınbaş University,

Supervisor: Asst. Prof. Dr. Sefer KURNAZ

Date: 2021

Pages: 86

Network traffic is growing at an outpaced speed globally. According to the 2020 Cisco Annual Report, nearly two-thirds of the global population will have internet connectivity by the year 2023. The number of devices connected to IP networks will also triple the total world population's size by the same year. The vastness of forecasted network infrastructure opens opportunities for new technologies and businesses to take shape, but it also increases the surface of security vulnerabilities. The number of cyberattacks are growing worldwide and are becoming more diverse and sophisticated. Classic network intrusion detection architectures monitor a system to detect malicious activities and policy violations in its information stream using various signature libraries. Still, due to a heavy inflow of network traffic in modern network infrastructures, it becomes easier for cyber criminals to infiltrate systems undetected to steal or destroy information assets successfully. Classic network intrusion detection architectures' speed and efficiency also fail to meet expectations in a real-time processing scenario. Considering the above limitations, this research aims to present novel methodologies to design and architect network intrusion detection systems using applied deep learning techniques. Neural networks can derive patterns and signatures from a raw dataset and use the learned signatures to predict the nature and classify the forthcoming data at an outpaced speed. The robustness of neural network architecture can be augmented to build a real-time and efficient

network security framework. In this study, we will develop a hybrid network intrusion detection system using the CNN-LSTM architecture. Further, we will compare our results with the recent research in this field of study.

Keywords: Artificial Intelligence, Deep Learning, Network Intrusion Detection System, Artificial Neural Network, Transfer Learning.

TABLE OF CONTENTS

Pages

LIST OF TABLES

Pages

LIST OF FIGURES

Pages

LIST OF ABBREVIATIONS

AI	:	Artificial Intelligence
ANN	:	Artificial Neural Network
AUC	:	Area Under the Curve
DoS	:	Denial of Service
IDS	:	Intrusion Detection System
LSTM	:	Long Short-Term Memory

1. INTRODUCTION

1.1 GENERAL CONTEXT

Technology is becoming increasingly omnipresent, interconnected, and deeply integrated into our everyday life. As our world becomes more and more network-dependent, a whole range of critical infrastructure sectors such as health care, finance, transportation, and government rely on cyberspace to provide essential services and perform its many days to day functions. According to Cisco Annual Internet Report, nearly two-thirds of the world population will have internet access by 2023 [1]. The number of devices connected to an IP network will also proliferate to become three times the global population resulting in an expansion of 29.4 billion networked devices [1]. The network connections' speed is also accelerating as 5G wireless networks are making it possible to support extremely low latency and response times. It is projected that 5G technology will lead to a 1,000-fold gain in terms of capacity and connection for at least 100 billion devices and will make it possible for the network infrastructure to provide a 10 Gb/s user experience while its deployment continues to make progress worldwide between years 2020 and 2030 [2].

As we continue to move towards this high density, high-velocity data trend, we also require the evolution of the existing network security architectures to safeguard our personal and professional data. Cybersecurity breach incidences are on the rise and have started to gain traction over the last few years. Security in the age of hyper internet connectivity is not just another technology issue. It has become a business, and a societal safety imperative since disruption of critical services can cause economic harm and negatively impact a large section of the population's well-being. According to the 2019 survey by Canadian Internet Registration Authority, over 71 percent of government and business organizations reported at least one cyberattack in 2018 [3]. World Economic Forum identifies cyberattacks as one of the top 10 global risks of the highest concert for the next decade in its Global Risks Report 2019. As per their forecast, this risk's disruptive potential may cost up to $90 trillion in the net economic impact by 2030 if cybersecurity efforts do not keep pace with the growing interconnectedness [4].

1.2 PROBLEM STATEMENT

Deep learning machines can solve complex problems by learning sufficient data and have gained much success in big data related processing areas, such as image recognition [43], machine translation [3]. Some deep learning models have demonstrated even better performance than human experts. The

remarkable success motivates us to consider transferring the technology to the task of attack recognition. Like in other application areas, for using deep learning on network intrusion detection, two major tasks are involved: deep neural network (DNN) model construction, DNN model training and evaluation. We have identified two problems that are specific to the designs of DNN for network intrusion detection. One is that few training datasets are available and there are insufficient training data in the datasets, which may make network training and evaluation not effective. The second issue is that most existing deep learning models either have a low detection accuracy or can achieve a high accuracy but with a high false positive alarm rate, which as a whole presents a compromised detection efficiency. Therefore, in this thesis we particularly target these two problems.

1.3 MOTIVATION AND OBJECTIVE

In the 21st century, a major driving force behind economic growth worldwide is technological advancement. Many fields such as cloud computing, big data, social media, IoT, and artificial intelligence play a vital role in the digital transformation of leading world economies. Nonetheless, as previously conferred, the mass adoption of technology and heavy reliance on computer networks also leads to security vulnerabilities and intrusion attempts made by several bad actors who could gain access to critical infrastructures and institutions to either steal, destroy, or tamper the crucial data. Using the developments in technology and software design, the cyberattacks themselves are also becoming much sophisticated. In such settings, we need to create a cybersecurity culture in our existing networking systems to safeguard our data and privacy. Given the persistence of security threats, an efficient cybersecurity architecture demands a modern Network Intrusion Detection System (NIDS) to monitor the stream of data traversing through the network and recognize the intrusion attempts and malicious activity to block them and their data source before it can reach and debilitate the core network infrastructures. Classic network intrusion detection systems worked proficiently in an ecosystem where data has certain traffic thresholds. With the current explosion of data traffic, such systems also require development progress and incorporation with the current technological trends to continue being effective in securing and safeguarding modern networks.

This thesis's main objective is to present a novel methodology to architect an efficient, real- time network intrusion detection system that can recognize and detect malicious activity in a normal stream of network traffic flow using state-of-the-art deep learning algorithms.

1.4 CONTRIBUTION

In this thesis, we proposed a new system based on a unified CNN-LSTM system of network intrusion detection. We used transfer training techniques to increase the accuracy and speed of the applied model classification in real world environments, in which we transferred the domain expertise our model gains into a source domain. The goal domain is to simulate a resource-sparse real-world environment with less data and computer resources. In contrast with recent related works where the experimentation is performed in highly available and resource plentiful environments, our work focuses on securing infrastructures in domains where data and resource availability can be sparse, but the IDS model is still capable of performing optimally despite the limitations. Such methodology also ensures that the model is not overfitting in the source domain and can be tested for performance before deployment in live production environments with critical security needs. The overall effectiveness of our model, in terms of accuracy and speed performance, showcases the utility of the applied transfer learning methodology to design and implement efficient and real-time intrusion detection systems.

1.5 THESIS OUTLINE

The rest of the thesis is structured as follows. Chapter 2 provides a brief overview of network threats and general counter attack procedure, and a discussion on the work related to the network intrusion detection found in the literature. Chapter 3 presents a deep learning model, that uses transfer learning to improve the capability of the detection system for discovering new attacks. Chapter 4 discusses the design of the overall learning performance and alleviate the false positive rate/false alarms rate. Chapter 5 summarizes the thesis and suggests some future research directions on using deep learning for network intrusion detection

2. BACKGROUND

This chapter will present a brief review of background topics that are relevant to this thesis. We will cover four main sections in this chapter.

2.1 INTRUSION DETECTION SYSTEM

An Intrusion Detection System is a software system built to monitor and analyze a computer network system to detect intrusions and malicious activity before it can seriously damage the network system and corrupt the data assets. An effective security framework has an IDS as its core element because recognizing and detecting attacks before they can execute will save the system from substantial downtime and service loss.

2.1.1 Development of Intrusion Detection System

Research and development of intrusion detection goes back to 1980, beginning with a paper by Anderson[5] introducing the core concepts for monitoring and surveillance computer hazards. In 1986 Dorothy E. Denning proposed the earliest drawing of a reliable intrusion detection system [6]. The system detected a wide range of safety violations from outside the attempts to break in the system and abortive patterns and the incidence of data abuse. The system used a rule-based pattern matching scheme where normal behavior records were kept in a safe library, which was further compared with audited usage patterns to flag any abnormal behaviors. The standard operations monitored on the target system were logins, executed commands, file and device accesses, etc. The IDS could detect a wide range of intrusions, for instance, masquerading attempts, trojan horses, viruses, leakage, and other types of misuse by legitimate users.

This research further expanded on the IDES, abbreviated in 1988 by Teresa F. Lunt at SRI International to the Intrusion Detection Expert System. [7], which focused on safeguarding the system from the outside intrusion attempts by using thorough statistical anomaly detection. IDES's premise was to build historical profiles of various subjects such as users, remote hosts, and target system and use the profile data to detect unusual activity that deviates from them. The profiles were also updated daily, making IDES evolve to learn the subject’s behavior pattern adaptively. IDES also integrated a second component, which used a rule-based system to encode the known intrusions scenarios and various system vulnerabilities to build a knowledge base that further strengthens its detection capabilities. Lunt proposed

a neural network as its third component to further supplement the IDES, which was not fully implemented in this system's follow-up derivations.

2.1.2 Taxonomy of Intrusion Detection System

IDS can be evaluated and distinguished into several classes based on their nature and functionality. In general, we divide the IDS into two main categories, host-based IDS and network-based IDS, according to their data source. Based on the IDS detection method, we classify them between Signature-based IDS and Anomaly-based IDS as presented in Figure 2.1.

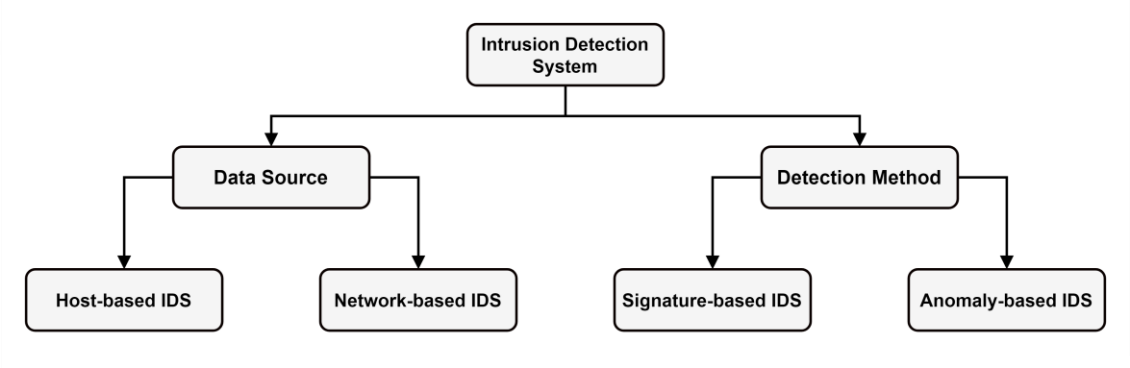


Figure 2.1: IDS Taxonomy Chart

2.1.3 Host-Based Intrusion Detection System

Host-based Intrusion Detection Systems (HIDS) are used to detect anomalies and misuse in the internals of a particular host they are installed on. HIDS was the original intrusion detection system which was designed to operate on the mainframe computers where external communication was rare and occasional. The input data which is used to derive the deviation patterns are collected by the operating system mechanism called audit trails. HIDS also uses other sources such as log files, filesystem data, and other process data generated by the single host. Because of many vendors and OS types, HIDS is required to be tailored to the design of the machine and OS it is integrated with, which limits its general efficacy due to lack of cross-platform support. This also increases the cost of developing the security infrastructure as with each iteration in manufacturer design. The HIDS also requires to be updated, making it economically unfeasible. HIDS are not designed to work with network traffic. They are limited in the scope of protecting the system which is connected to an external network interface.

2.1.4 Network-Based Intrusion Detection System

Network-based Intrusion Detection Systems (NIDS) monitor the network activity and analyzes it to detect malicious activities in the data traffic. The primary source of the examination for NIDS is the content and header information of incoming network packets. NIDS is situated strategically on the critical points in a network infrastructure that are receiving a large amount of external traffic. NIDS is effective to monitor a vast sized network, and because of the standardization of TCP/IP and UDP/IP network protocols worldwide, they are highly portable. They can be developed independently without constraining to any particular manufacturer and network device type.

As deliberated previously, with the vast adoption of the internet, each device today is in one form or another is connected to an external network to deliver services. Many software presents on the single host itself shares data with several external APIs for processing data. With the rise of cloud computing and serverless architectures, traditional hardware-based computing is becoming obsolete. It is gradually being taken over by external hardware provisioners that connect with the edge devices to enable access to computing services. As depicted in Figure 2.2, among both types of IDS architecture, this thesis will mainly focus on NIDS because it is imperative to protect the network system to circumvent any disruptions propagating itself into the local host system.

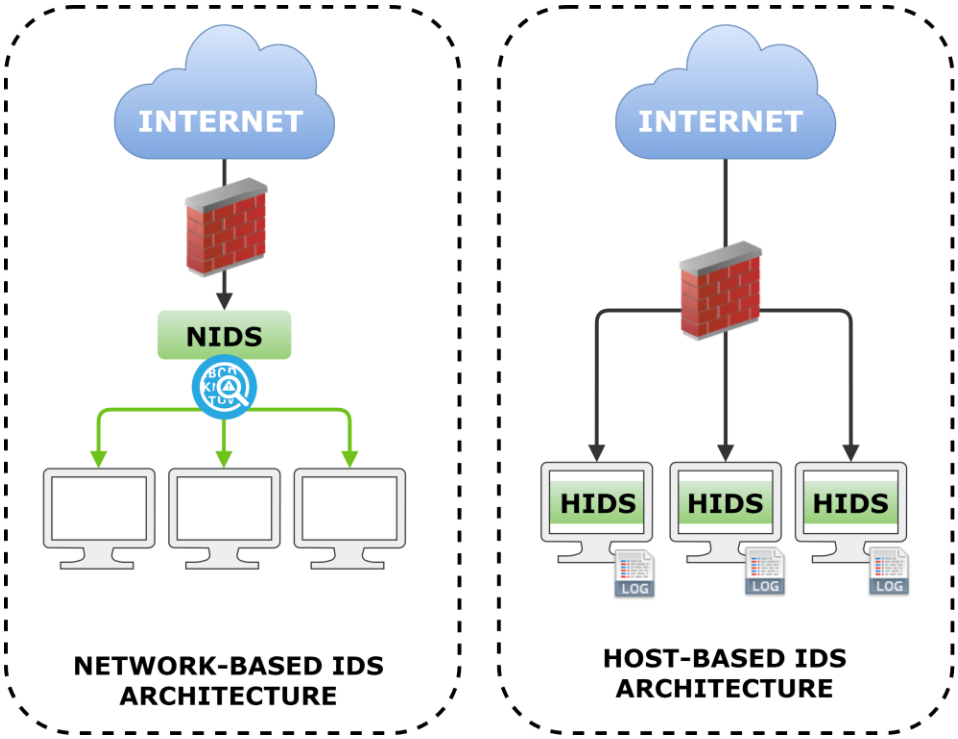


Figure 2.2: IDS Architectures

2.1.5 Detection Methods Used by IDS

There are two broad types of detection methodologies used by an IDS, namely, Signature detection and Anomaly detection, as shown in Figure 2.3.

A.Signature Based IDS: This type of IDS emphasizes the signature and patterns in the stream of data to detect intrusions. In computer security terminology, a signature is a pattern or footprint associated with computer network activity. Each type of hacking activity leaves a footprint behind, such as the nature of data packets, a hash of harmful files, or a code pattern. Using unique identifiers for known attacks and malicious activities, a database of such signatures is compiled, which is then used to find them in normal host or network activities. Signature-based IDS is essentially a knowledge system as it requires a knowledge base to draw inferences and match the activities [8]. The signature database must be updated regularly as if the signatures are not up-to-date, the system may fail to detect new types of intrusion attempts. Because of the specificity of the attacks the IDS is looking for, signature- based IDS has reasonably low false positive rates and false alarming incidences.

B. Anomaly Based IDS: This type of IDS focuses on deviations in the host system's normal behaviors or network traffic stream. Anomaly-based IDS essentially protects the system from unknown attacks that the system might not have encountered before. The IDS first establishes a baseline profile which is derived from the normal functioning of the information system by studying its traffic over a period of time. If the system behaves in a manner that deviates from the conventional baseline, the IDS raises the alarm. Anomaly-based IDS safeguard the system from two major types of anomalies.

1) Protocol Anomaly: This kind of anomaly refers to any deviated pattern in the internet protocol and standards. During the baseline establishment, the IDS learns normal patterns in the various aspects of the connection such as TCP segmentation, IP header flags, source and target ports being widely used, the presence of shellcodes in application protocol fields, checksum, IP fragmentation, and reassembly, etc. Using the plethora of these features recognized as normal, IDS guards against any deviations it may come across in these network protocols.

2) Traffic Anomaly: The flow of network traffic itself is a key signifier of the anomalies in an information system's operations. A stable network functions between the lower and upper bounds of traffic. When these thresholds are crossed, IDS will recognize that the system is at risk and does not function in the optimal baseline profile. Attacks such as Denial of Service (DoS) and Distributed Denial of Service (DDoS) are aimed to flood the network with fake traffic to overwhelm the infrastructure providing

certain services, leading to legitimate users being unable to access those services. The IDS can swiftly recognize such rapid disruption of the information flow as abnormal behavior, and measures can be put in place to block the source of such traffic.

Anomaly Based IDS are more versatile to detect intrusions and malicious anomalies that the system has not encountered before. Still, it may also occasionally deem normal traffic with features unknown to the baseline as intrusions. This might lead to unnecessary false positives and alarms, leading to obstruction of genuine sources.

The central area of concern regarding the design of an IDS is its shortfall of generating many false-positive incidences, which leads to unnecessary interruptions. But suppose the IDS is designed unconventionally to remedy the high false positives incidence. In that case, it may let the actual intrusions pass over, which will become a real disruption to the whole system.

With the utility of Signature-based IDS, we can keep the false-positive results in a lower constraint. Still, the system needs to be manually updated for new signatures to be functional, or it might miss them out entirely. In this thesis, being mindful of the discussed strengths and limitation of both detection techniques, we are building a novel network- based IDS architecture which will use a hybrid model of detecting anomalies in the modern network traffic to minimize the false positive incidence as well as cover a large surface of diverse network attack types.

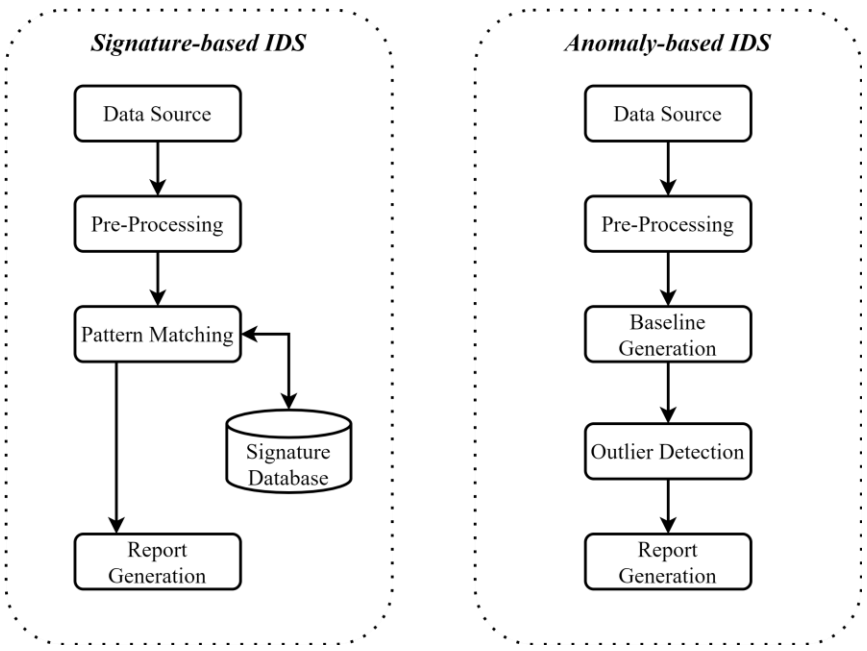


Figure 2.3: Detection Methods used by IDS

2.2 CYBER ATTACKS

In computer security terminology, a cyber-attack attempts to gain unauthorized access to an information asset with the intent to destroy, steal or alter the data asset. Using computer networks, the intent of such malicious activities can either be part of cyberwarfare or wide- ranging forms of cyberterrorism. As discussed in prior sections, the incidence of cyberattacks is on the rise, with cyber warfare becoming a new device for hostile global powers to commit to foreign government espionage and reconnaissance. According to the 2017 Word Threat Assessment report by US DNI, many countries view cyber capabilities as a way to project their global influence and are continuing to develop and fund their cyber arsenal [9].

2.2.1 Forms of Cyber Threats

1) DoS: A denial-of-service (DoS) is a common form of cyber threat that refers to the situation where the attackers aim to overflood the traffic on a host or network infrastructure to make the resources and services inaccessible for genuine users. The attack itself doesn't lead to the theft of data assets but costs the target victim organization time and money resources. The subsequent crashing and debilitating of services can also cause physical harm to systems if they are handling control networks and other critical infrastructures [10]. Distributed-denial-of-service (DDoS) attack is a variety of DoS attack which uses a distributed system called a botnet for orchestrating the cyber-attack as shown in Figure 2.4, increasing its overall severity and potential.

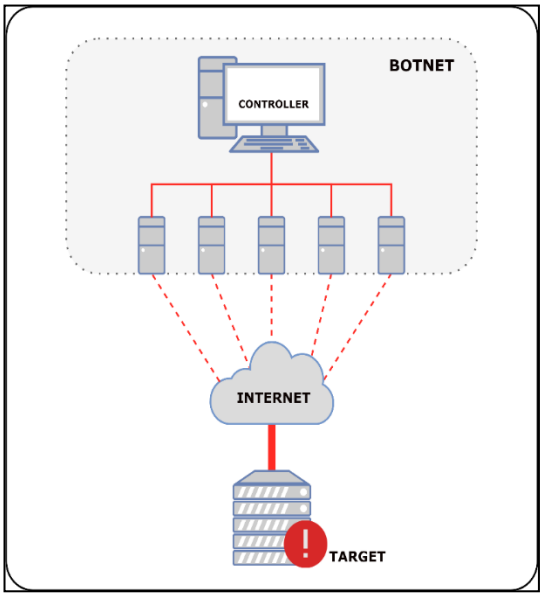


Figure 2.4: DDoS Attack

2) R2L: A remote-to-user is a cyber-attack where the attackers gain access as a local user to infiltrate the organization from a remote machine. The attackers send malicious packets to the local user's target host to find any vulnerabilities that can enable the attacker to exploit the local user's existing privileges [10]. This vulnerability is a prelude to more disruptive User-to-Root (U2R) attacks.

3) U2R: In a User-to-Root attack, the attacker first gains the foothold in the host machine as a local user with limited privileges and then proceeds to escalate the privileges using various methods to become the root user [10]. This enables the attacker to make more superuser accounts further and generate backdoors to re- enter the organization's network easily and undetected. The root privileges essentially give the attacker access to every list of commands in the system and enable them to manage the data assets present in the filesystem according to their directives.

4) Port Scanning: This cyber threat is a type of reconnaissance method used by attackers to thoroughly scan all the target host's open ports [11]. All the transmitted information the host is receiving and sending is using various ports dedicated to specific services. Using port-scanning, the attackers gain the ability to retrieve all the information for analysis and redirection to further entrap the targeted user in other forms of cyber-attacks. Mapping the ports, the attackers can also detect other vulnerabilities to exploit and further gain remote access.

5) Backdoor Attacks: These are a type of malware attacks aimed at giving attackers unrestricted access to the server and database of the compromised systems. Unlike other forms of access, backdoors remain discreet, and attackers utilize them to steal a large quantity of financial and competitive data while remaining undetected. According to the State of Malware Report 2019, backdoors continue to be a critical threat vector in cybercrime across all the government and business entities, with a staggering 173% rise in their detection rate in business organizations [12].

6) Fuzzers: As the name suggests, this attack type aims to fuzz or error out the normally operating host server by sending it various types of faulty commands in brute force mode, which will result in the systems to throw various error codes [13]. The aim is not to fail the system but to generate the error logs that can further be analyzed by the attackers to find the resources and locations that can be used for proceeding malicious activity to find vulnerabilities. Traditional fuzzer techniques are now being re-invented using machine learning algorithms to generate a wide range of test cases and seed files and cover a large surface of code to find additional vulnerabilities effectively.

7) Computer Worm: These are a type of malicious software that self-replicates themselves for propagating to other networks and systems in their vicinity. A computer work relies on systems existing vulnerabilities and backdoor exploits to stay hidden while continuing on their onslaught of the entire network. The core directive of this cyber threat is to gradually drain the resources of a system and congest the network infrastructure. Many types of worms also have payloads aimed at stealing sensitive data. Commonly worms are used first to gain access to the system and then escalate the privileges to proceed with other cyber-attacks.

In this thesis, including the discussed cyber threats, we aim to cover a large threat vector using extensive cybersecurity databases UNSW-15 to train and test our novel IDS architecture model.

2.3 MACHINE LEARNING CONCEPTS

The first-generation IDS deliberated in previous sections fundamentally used audit trail sources and pattern matching methods as their primary mode for intrusion detection. Using a formerly compiled signature knowledge base on a host system, the IDS could detect policy violation and any deviation from normal baseline usage by comparison. Over time, with the maturation of machine learning and driven by its many practical use cases, the researchers working in the field of computer security worked on integrating various machine learning and data mining techniques to augment the IDS design and essentially change its processing. The second-generation Intrusion Detection Systems principally used statistical analysis and data mining techniques to draw its core inferences.

2.3.1 Fundamentals of Machine Learning

Machine learning is an area of artificial intelligence in which we develop computer models that can learn with minimum human intervention from a given dataset. Murphy [14] states that machine learning is a set of methods used for the automatic detection of patterns in data, and then for predicting future data or for other decision-making tasks, it uses the extracted patterns. A master learning model can be either predictive if it makes predictions of the future or descriptive if it aims to acquire knowledge from the data or is predictive and descriptive at the same time. Use statistical theory in the construction of mathematical models, machine learning algorithms' core task is to extrapolate inference from a given sample.

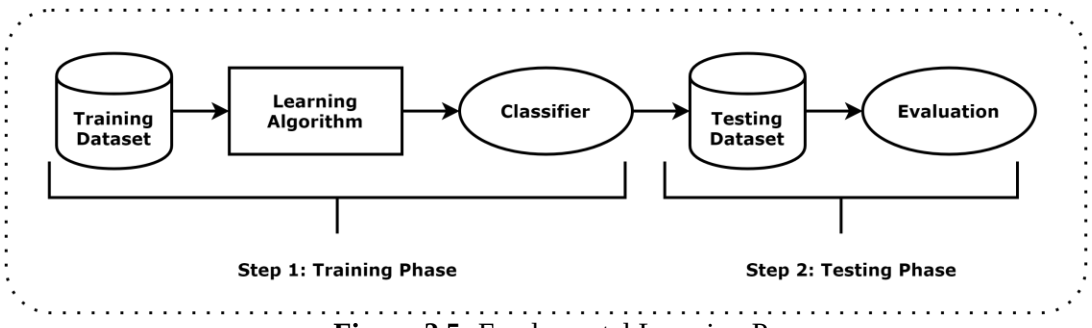


Figure 2.5: Fundamental Learning Process

As depicted in Figure 2.5, the fundamental learning process can be divided into two steps, a training phase and a testing phase, which require two kinds of separate data sets,

1. Training dataset: It is a subset of data used during the training phase. This data is labeled with pre-defined classes, so the learning algorithm can learn to produce associations of the data with the corresponding labeled classes.

2. Testing dataset: It is a subset of data used during the testing phase. It is used to evaluate the classification model generated by the learning algorithm during the training phase. This data is required to remain unseen by the algorithm during training to maintain the overall machine learning algorithm's veracity.

Machine learning algorithms are broadly divided into three main categories: supervised learning, unsupervised learning, and reinforcement learning. In this section, we will briefly overview supervised learning algorithms as they are an integral part of this thesis and further study various foundational algorithms that will build up the reader's knowledge base to be able to comprehend more complex algorithms in the field of deep learning.

2.3.2 Supervised Learning Algorithms

Supervised learning belongs to the category of predictive learning algorithms where we predict the label of unknown objects based on the label-based associations inferred by the algorithm during its training phase [14].

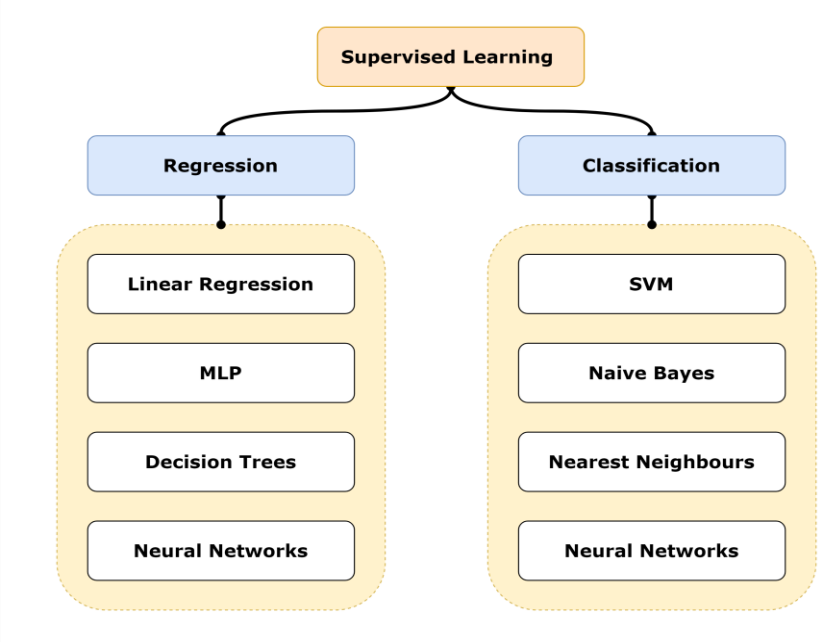


Figure 2.6: Taxonomy of Supervised Learning.

In the supervised learning approach, the goal of the algorithm is to learn mappings from input x to outputs y, given a labeled set of input-output pairs $\mathcal{D} = \{(xi, yi)\}N$ and produce a prediction function. Here $\mathcal{D}$ is referred to as the training set, and N is the number of training examples. The nature of training input depends on the kind of problem the algorithm is solving. The xi is the $\mathcal{D}$-dimensional vector or numbers representing the simple features or attributes. However, xi can also represent complex structured objects such as an image, time-series, e-mail, graphs, etc. [18]. The output yi can also be of different forms depending on the problem.

If the value of yi is a categorical variable from a finite set, $yi \in \{ 1, \ldots, C\}$, such as normal or malicious, then the problem is known as classification or pattern recognition. Similarly, when yi is a real value, the problem is considered as a regression. In simple terms, regression involves predicting a real value, leading to a label estimation whereas, classification involves identifying class membership of a given sample. The function learned during the training phase is also known as a classification model or simply a classifier. In Figure 2.6, we depicted supervised learning algorithms' taxonomy based on the concepts of regression and classification. In the proceeding sections, we will brief major types of regression-based supervised learning relevant to this thesis.

2.3.3 Linear Regression

Linear Regression is a monitored machine learning algorithm in which the forecast values are continuous and consistently sloped. Each remark consists of two values in linear regression. The dependent variable is one value, and the individual variable is one. In addition, the link between the dependent and the independent variables is drawn straight. Let yi be the predicted value of the dependent variable for a given value of the independent variable xi.

$$\mathrm{y}i = \beta 0 + \beta 1 xi + \varepsilon \qquad (2.1)$$

Here, $\beta 0$ represents the y-intercept of the regression line and $\beta 1$ represents the regression coefficient. The variable ε is the error of the estimate. In essence, linear regression tries to find the best line which we can fit through the data by searching for the regression coefficient $\beta 1$ which minimizes the overall error ε of the model.

A regression line can show three types of relationships between the x and y variables.

a. No relationship: When the graphed line is flat, not slopped, then we deduce that there is no relationship between the two variables.

b. Positive relationship: When the regression line pitches up, it is concluded that the two variables show a positive relationship when the lower end of the line at the y-intercept and the high end of the line pull up into the graph field. This means essentially that when one variable's value increases, it increases in synchrony as well.

c. Negative relationship. If the regression line is sliding down, in this case, we can infer that there is a negative relation between the two variables, in which the upper end of the line at the y-intercept is extended to the bottom of the line into the field of the graph.

As mentioned, we regulate the overall error of the algorithm to reach the best predictions. To do so, we use a loss function that determines the error or loss between the outcome of the learning algorithm and its expected outcome. In this example, let's examine Mean Squared Error (MSE), which is a sum of squared distances between our target variable and predicted values.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \qquad (2.2)$$

The variable $\hat{y}i$ is the predicted value and variable yi is the targeted value.

2.3.4 Logistic Regression

Logistic regression is a linear classification type supervised learning algorithm, where we aim to predict the class or category of the given sample based on its features. The nature of dependent variables is different when compared to regression problems as they are discreet with a finite set of outputs. Unlike linear regression, where the output is a continuous number of values, logistic regression transforms its output using a logistic sigmoid function to return a probability value, which can then be mapped to two or more discreet classes.

Logistic regression can be used for binary classifications, where there can only be two outputs i.e. 1 for malicious network packet or 0 for normal network packet, in case of an intrusion detection system. It can also be used for multi-class and ordinal classification problems. Consider a single input sample x, which is represented by a vector of features [x_1, x_2, , x_n]. Essentially, we want to compute the probability $P(y = 1| x)$, which infers that the observed sample is a member of the given class, whereas probability $P(y = 0| x)$ means the sample does not belong to the given class. In logistic regression, we first learn the

weights and a bias term from a training dataset. The weight w_i is a real number associated with a feature x_i, which represents how important that particular feature is to a classification decision. It can be either positive or negative depending on the assertion. The bias term or the intercept is another real value added to the weighted inputs. To decide on the observed sample, the algorithm after learning the weights from the training, we multiply each x_i by its weight w_i. Then we further sum up weighted features and add the bias term to the result. The resulting output z then can be given by the equation,

$$z = (\sum_{i=1}^{n} w_i\, x_i) + b \qquad (2.3)$$

To now create the probability value from the output, we would need to pass the z from asigmoid function $\sigma(z)$. The equation of the sigmoid function is,

$$\sigma(z) = \frac{1}{1+e^{-z}} \qquad (2.4)$$

This can further be graphed as shown in Figure 2.7 as follows. The sigmoid function takes real numbers and maps them in a range of [0,1]. Further, to make it into a probability, we use two cases, $P(y = 1)$ and $P(y = 0)$ as follows:

$$P(y = 1) = \sigma(w.x + b) = \frac{1}{1+e^{-(w.x+b)}} \qquad (2.5)$$

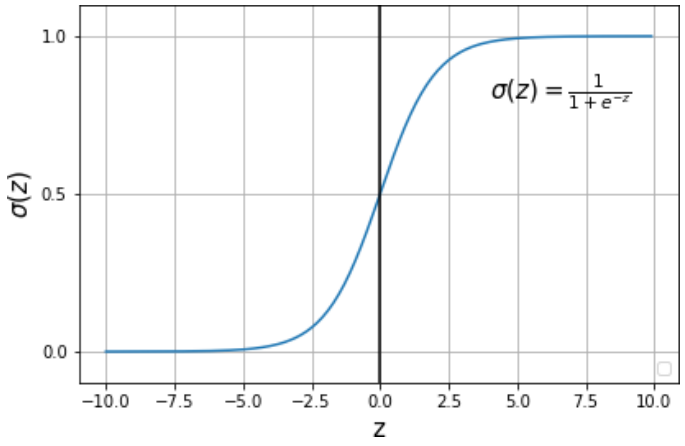


Figure 2.7: Sigmoid Function

$$P(y = 0) = 1 - \sigma(w.x + b) = \frac{e^{-(w.x+b)}}{1+e^{-(w.x+b)}} \qquad (2.6)$$

Where, if $P(y = 1|\, x)$ is more than 0.5, we infer the class to be 1, which we also call the decision boundary or threshold to determine the class membership. To summarize if,

$$\hat{y} = \begin{cases} 1 \; if \; p(y = 1) > 0.5 \\ 0 \; otherwise \end{cases} \tag{2.7}$$

We use the cross-entropy loss function with logistic regression, which is used to express how accurate the classifier's output results ($\hat{y} = \sigma(w.\, x + b)$) is for sample observation. An MSE loss function is not ideal for logistic regression problems as it assumes that the output value will follow a normal distribution, whereas in logistic regression, it follows a Bernoulli distribution. Primarily, cross-entropy is a measure to calculate the difference between two probability distributions for a given random variable or a set of events. In this case, the distributions are the true probability distribution y and the predicted probability distribution $\hat{y}$.

2.3.5 Neural Networks

Neural networks are a family of machine learning models inspired by neurons functioning in a brain system. In metaphor, a neuron in a machine learning sense is a computational unit that has scalar inputs and outputs. Each neuron also has a weight parameter associated with it. The neuron multiplies each input unit by its weight, sums all the input units, and then applies a nonlinear function to the result to produce an output [17]. The simplest neural network architecture, consisting of just two layers of input and output layers, is called a perceptron as depicted in Figure 2.9 where we have Xn input units, each with a weight association. We pass the input units to the next layer, which, as discussed, sums them and applies the activation function such as a sigmoid function like in the case of logistic regression deliberated previously, to find the $\hat{y}$ output based on a boundary decision criterion. Perceptron is limited to linear classification, where we can only classify linearly separable sets of vectors. If the vectors are not linearly separable, then perceptron will not be able to give correct prediction results. Whereas, if we add more layers to the perceptron architecture, also known as hidden layers, we progress towards a multi-layered perceptron or MLP architecture that can also do non-linear classifications and solve much more complex problems.

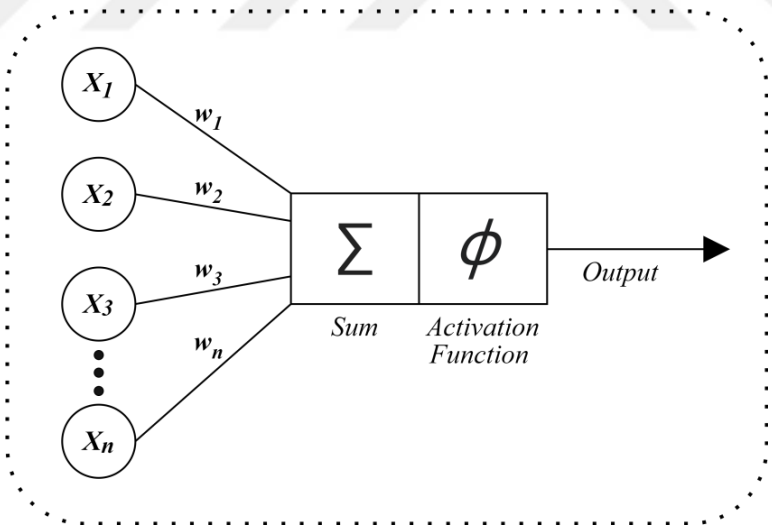


Figure 2.8: Perceptron Architecture

2.3.6 Multi-Layered Perceptron

A multi-layered perceptron is the augmentation of a perceptron but with more intermediate layers referred commonly as the hidden layers. MLP is a feed-forward neural network as the computation process moves iteratively to the next layers without being in a cycle of loops. Each layer's output becomes the input of the proceeding layers where no outputs are ever passed back to the previous layers. In this fashion, the data seems to be moving forward; hence we classify MLP as a feedforward network. The feedforward MLP has three central units: input layer units, hidden layer units, and output layer units. Units in each layer are connected to all the units in its previous layers. This way, the architecture is also known as a fully connected network, as shown in Figure 2.10, where we have one hidden layer in-between the input and output layers.

The input layer has xn units, each with a weight association and bias, connected to each unit in the hidden layer. The hidden layer can be represented as a vector h whose output can be expressed as,

$$h = \sigma(Wx + b) \tag{2.8}$$

The function represented by σ is an activation function, W represents the single matrix of weight associations between the input layer and hidden layer units, whereas the b represents the bias vector for the whole layer. The combination of the weight vector wij which represents the weight of the connectivity between ith input layer unit and jth hidden layer unit into a single matrix W makes the computation for the hidden layer in the feedforward network reliant on simpler and efficient matrix operations. Further, the hidden layer output becomes the input of the output layer. The weight matrix between these two layers is represented as U. The output z can now be computed as,

$$z = U \times h \tag{2.9}$$

In addition, we would need to normalize the output z, which is a vector of real number values, into an encoding of probability distribution ŷ to predict the class labels. We generally use the Softmax function for normalizing the output layer in neural networks where,

$$softmax(z_j) = \frac{exp(z_i)}{\sum_j exp(z_j)} \tag{2.10}$$

Softmax functions transform the logits into probabilties by taking representatives of each output, which is basically another term for the figure output for the last linear layer in a multi-class neural network,

and normalizing it with the sum of all exponents. This way, the entire vector adds up to one, giving us a probability distribution to map the correct prediction labels.

Neural networks can be thought of as a series of stacked logistic regression classifier units that learn the representations in the data and induce them into the neural network's further layers. This makes the neural networks classifier more powerful in learning data representations on its own without anyone handpicking the features templates for the network. The self-organization of neural networks sets them apart from various classical machine learning algorithms. In this section, we described a neural network architecture with one hidden layer. Such neural networks are known as shallow neural networks. In forthcoming sections, we will deliberate architectures that use several hidden layers, also known as deep neural networks.

2.3.7 Backpropagation Algorithm

To optimize neural networks, we use the backpropagation algorithm which aims to minimize the weights present in the neural network by using backward differentiation to update their values. The core directive of backpropagation is to compute the gradient of the loss function with respect to each unit present in the neural network layers. As deliberated in previous sections, in the case of logistic regression, we could directly compute the derivative of the loss function with respect to individual weight or bias [18]. Still, neural networks have in many cases millions of such parameters present in their overall architecture. In such a case, we cannot directly optimize weights in a particular layer as there are many more layers in precedence that influences its parameters. To optimize weights in a multi-layered paradigm, we make use of error backpropagation or backward differentiation to propagate the error signal δ back to the input neurons using partial derivatives and chain rule to define the relationship between a given unit in a neural network's individual weight and the overall computed cost function of the network. We express the error signal δ as,

$$\delta = z - y \tag{2.11}$$

Where y is referred to as the computed output of the neural network and z is the real and correct output during the training cycle.

To showcase the utility of chain rule for backprop, suppose we compute the derivative of an output function L with respect to the variables a. The derivative ∂L gives how much the ∂a change in parameter a impacts the overall output of function L. Now say we have a composite function $f(x)$

= $u(v(x))$. According to the chain rule, the derivative of $f(x)$ is the derivative $u(x)$ with respect to $v(x)$ times the derivative of $v(x)$ with respect to x, which can be expressed as,

$$\frac{\partial f}{\partial x} = \frac{\partial u}{\partial v} \cdot \frac{\partial v}{\partial x} \tag{2.12}$$

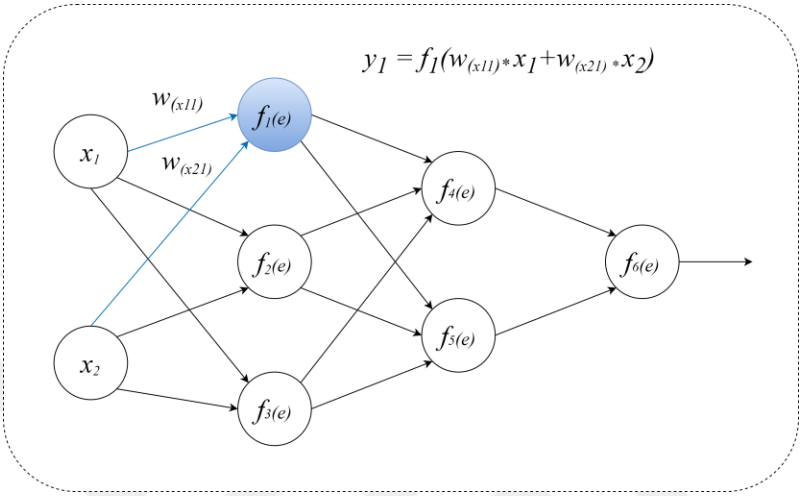


Figure 2.9: Computation of hidden layer units in neural network

The computation and updating of weights in a neural network can be further demonstrated step by step using an example of a neural network with two hidden layers so as to breakdown the idea behind the working of the backpropagation algorithm. In Figure 2.11, we are computing the result of function unit f1(e) in the hidden layer, which uses the connection weights $w(x11)$ and $w(x21)$ between input units $x1$ and $x2$ where e= $w(x11) * x1 + w(x21) * x2$. The output of function unit f1(e) then further becomes the input for computing function units f4(e) and f5(e). In Figure 2.12, we are computing the result of the output layer unit which uses the forward cascading results of the neural network to reach an output $\hat{y}$. The algorithm now compares the output $\hat{y}$ with the correct output y. The difference is called an error signal and is represented by □ where□

$$\delta = \hat{y} - y \tag{2.13}$$

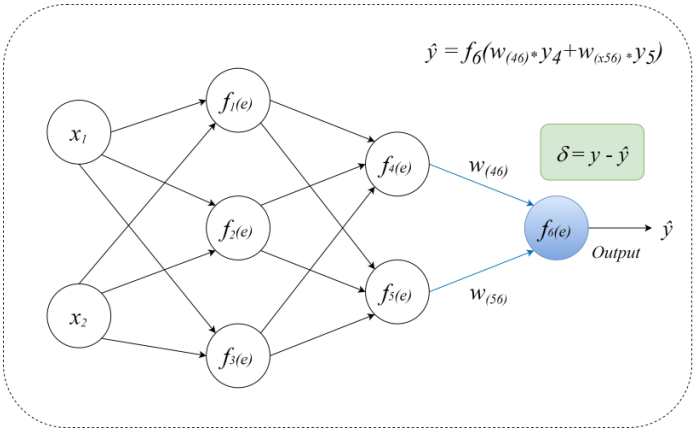


Figure 2.10: Computation of the error signal to propagate backwards

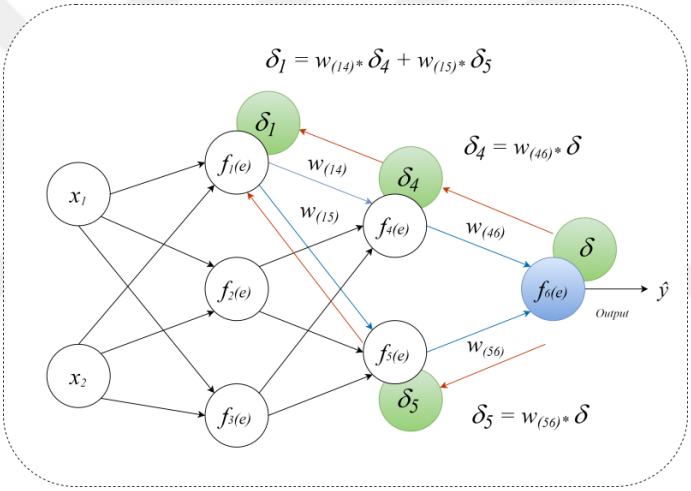


Figure 2.11: Backward propagation of the error signal in the neural network

The computed error signal is propagated backward in the network to the very first hidden layer units in the neural network as shown in Figure 2.13, where each unit in the neural network has an error signal computed using the same weight coefficients utilized during the forward pass but the direction is changed to flow backward. If the error signal is coming from multiple sources, they are summed to get the unit's overall error signal flowing.

When the error signal for every unit in the neural network is computed, we update the input weight coefficients of each neuron with the following equation,

$$w'(x_{11}) = w(x_{11}) + \eta \delta_1 \frac{df_1}{de} x_1 \qquad (2.14)$$

Where $w'(x11)$ is the updated weight for connection between the input unit $x1$ and hidden layer unit $f1(e)$, coefficient η represents the learning speed, $\delta 1$ represents the error signal computed for the unit, the equation $df1(e)$ represents the derivative of the neuron activation de of the hidden unit $f1(e)$ whose weights are being updated. Each iteration of passing all the training examples through a backpropagation algorithm is referred to as an epoch. We continue to run the epochs until the algorithm converges towards a global optimal minimum, which leads to more accurate results and a lower value of overall error signal δ.

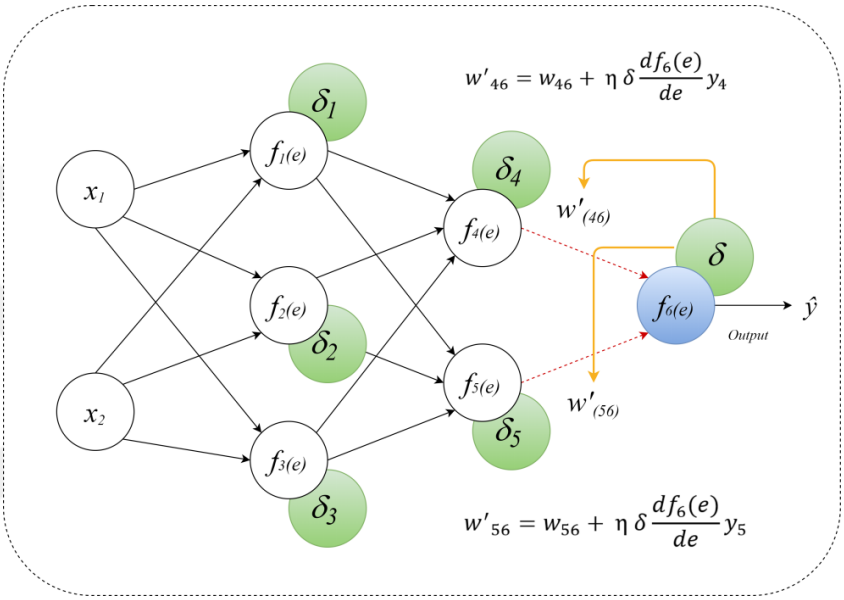


Figure 2.12: Weight updating using Backpropagation Algorithm

Figure 2.14 shows updating of weights by backprop in the neural network until the final output unit $f6(e)$ is reached. The algorithms again compute the error signal and back propagates the signal to update the network's weights again depending on the epoch numbers chosen.

2.4 DEEP LEARNING ARCHITECTURES

This section discusses the architectures of deep learning that are important for the proposed IDS architecture. Deep learning enables computer models consisting of multiple layers to develop data representations with multiple levels of abstraction [19]. A deep learning model consists of numerous fully connected hidden layers, hence we refer to such models being deep learning models as compared to models with just a couple of hidden layers referred to as shallow learning models. Deep neural networks can be classified based on the information flow. If the information flows from an input layer to an output layer without any feedback responses, such a network is called a feedforward-DNN. In contrast, if a neural network architecture is integrated to function with various feedback loops, we refer to such networks as a recurrent neural network. One of the vital utility of a DNN is to learn representations from a raw dataset. A neural network model's ability to automatically discover the representations in data required for feature detection and classification is known as a representation or feature learning [20]. As shown in Figure 2.15, we replace the manual hand-picking of domain-specific features using deep learning networks, which is a vital necessity for various data mining and machine learning techniques. Deep learning can simply be defined as a class of machine learning algorithms that uses multiple layers of functional units to progressively learn and extract features from a raw dataset, whereas we move from the lower end to the higher end of the layers, the features being extracted start becoming more and more pronounced for the learning model to infer accurate solutions for the given prediction or classification task. We will briefly deliberate two types of deep neural networks relevant to the IDS architecture in the proceeding sections: Convolutional Neural Network and Recurrent Neural Network.

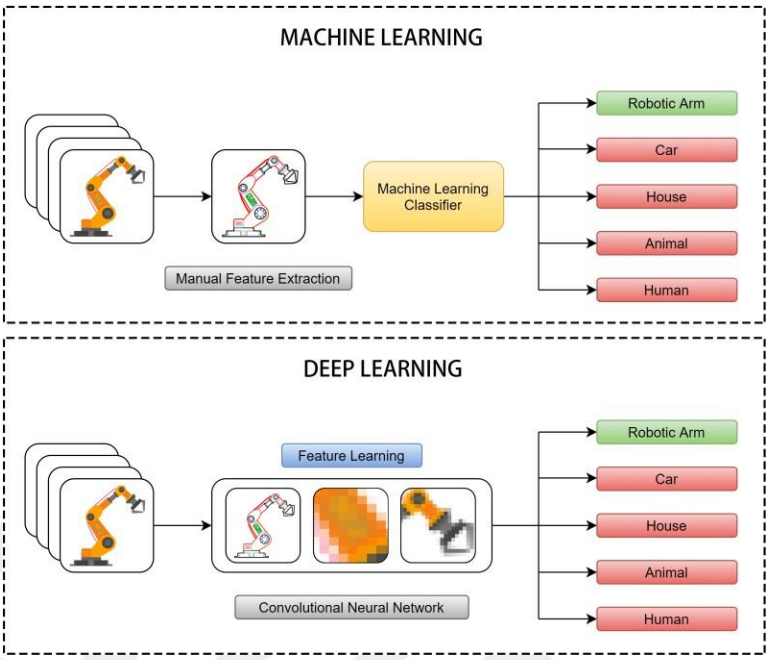


Figure 2.13: Comparison between machine learning and deep learning classification

2.4.1 Convolutional Neural Networks

Convolutional neural network abbreviated as CNN is a class of feed-forward deep learning networks applied to various visual analysis and text-based problems. The architecture of a CNN is inspired by the pioneering work of Hubel and Wiesel [21] which aimed at analyzing the neurons in the visual cortex of mammals to understand how neurons in visual pathways extract information from patterns cast on a retina of an eye and transform it on the way to cerebral cortex which evaluates and recognizes an image. This research inspired the architecture of Neocognitron by Kunihiko Fukushima [22], a type of multilayered artificial neural network consisting of cascading layers composed of two components: the S-cell layer and the C-cell layer. S-cell layers are the main feature extraction units in Neocognitron, whereas C-cell layers pools the information coming from the preceding simple cells and transmits the result to the successive simple cell layers in a feed-forward manner. A modern CNN is a successive iteration of Noncognition architecture, with the exception of backpropagation being the primary mode for being the learning algorithm.

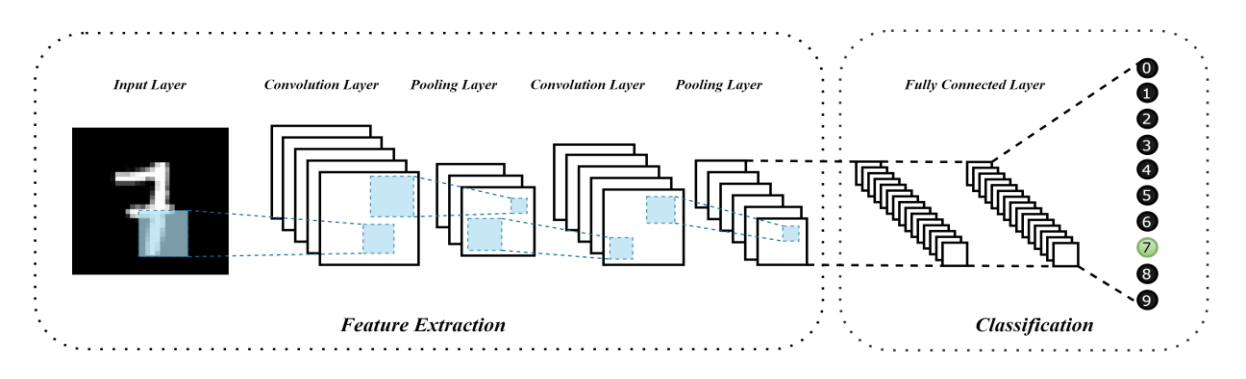


Figure 2.14: CNN Architecture

Yann LeCun et al. [23] demonstrated one of the early implementations of a CNN architecture known as LeNet-5 to the task of hand-written digit recognition using the MNIST dataset. As shown in Figure 2.14, a CNN architecture consists of the stack of various types of layers organized into two main components, a convolution and pooling layers unit which extracts the features of the input layer and a fully-connected layer which is used for classifying the results of the preceding feature extraction units into a predictive label output. Each layer in CNN has a specific operation, which is briefly described as follows,

1. Convolution Layer: The convolutional layer is the core building block of a CNN which generates feature activation maps from the input layer using various receptive fields commonly referred to as filters, by moving the particular filter across the width and height of the input layer so as to compute the dot product between the input layer and entries in the filter. As shown in Figure 2.15, the convolution operation results in the generation of various two-dimensional activation maps, which are later fed into subsequent pooling layers. The amount of movement of the filter per step is determined by the stride's value, which defaults to one. The convolution layer also uses an activation function ReLU, which converts all the negative values into value zero.

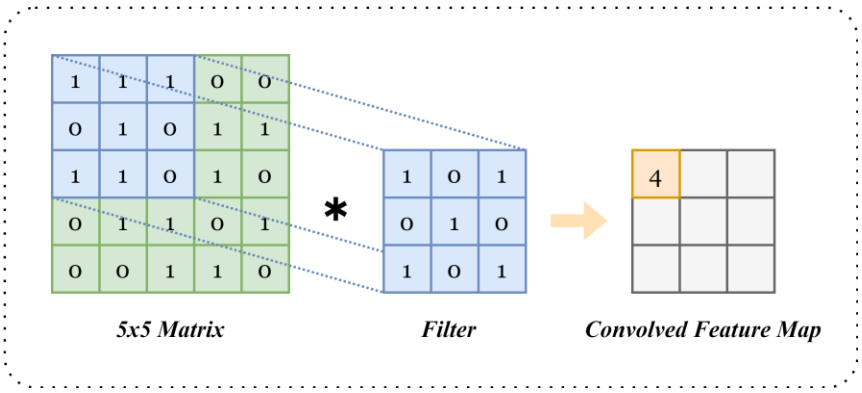


Figure 2.15: Feature Map computation by Convolution Layer

2. Pooling Layer: This layer is used for the downsampling operation, which reduces the spatial size of generated feature maps by convolution layer by reducing their dimension based on a chosen criteria. Pooling aims to extract the most dominant feature from the feature maps and optimize the overall computation needed to process the data. There are various types of pooling criteria, such as max pooling, average pooling, and sum pooling. Figure 2.16 demonstrates the subsampling of the feature map using max pooling

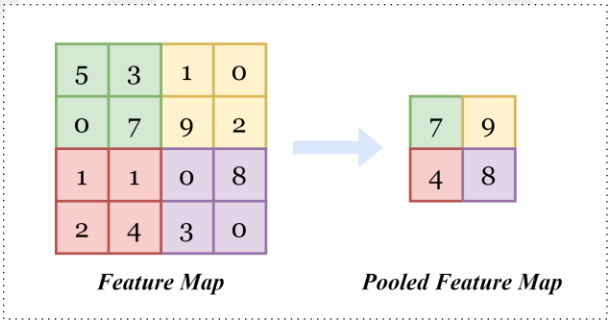


Figure 2.16: Max Pooling operation with 2x2 Filter and Stride value 2.

3.Fully Connected Layer: The last unit in the CNN architecture is a fully connected layer reminiscent of the previously deliberated artificial neural networks. After inferring the features from the input layer's matrix space, the final pooling layer's output is flattened into a 1-D vector space, as shown in Figure 2.17. The flattened column vector then becomes the input for the fully connected layer to interpret further the features, which is done by training the network using backpropagation over a series of epochs. The

last unit of the fully connected layer uses an activation function such as a sigmoid or softmax activation function to generate the class label predictions, which is also the CNN's final output.

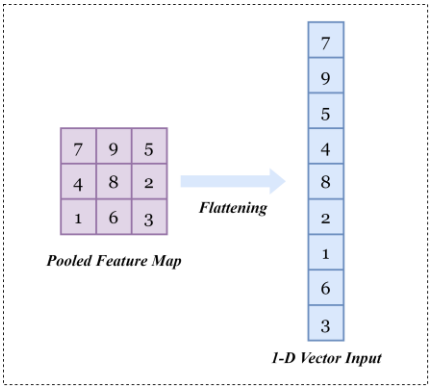


Figure 2.17: Flatten operation converting feature matrix into 1-D vector input

2.4.2 Recurrent Neural Network

As discussed so far, in standard neural networks, the information flows in one forward direction. The network does not maintain information about its previous states in any sequence of events. Recurrent neural networks, in contrast, abbreviated as RNNs are a type of profound learning architecture with looping feedback connections that enable the model over time to saves persistent information [24].

As shown in Figure 2.18, an RNN takes input xt at a time stamp t to produce $\hat{y}t$ which is the output of this network. In addition, the network is also computing an internal state at time stamp t denoted by ht, which it passes from one-time step to another internally within the network where,

$$h_t = fw(h_{t-1}, x_t) \tag{2.15}$$

In this equation, we are computing the recurrence relation in the network at every time step. The value of ht is determined by function f which is parametrized by a weight w, the older state of the network donated as $ht-1$ and the input vector xt at time t.

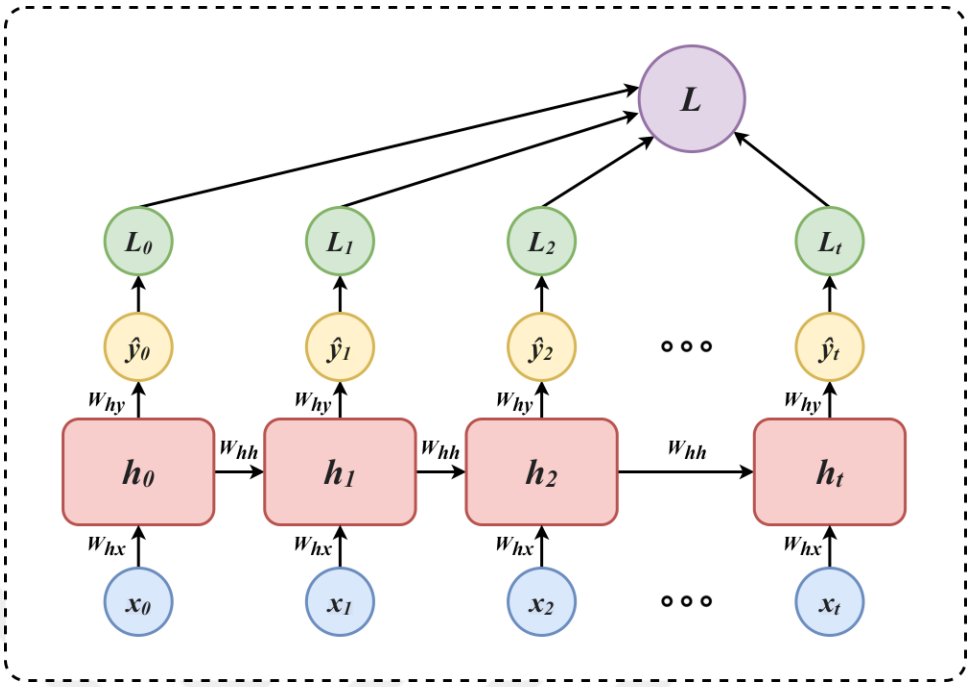


Figure 2.18: Unrolled RNN Architecture

To understand the inner workings of an RNN when it is processing data, we can unroll it to understand how it computes the output of its network which is shown in Figure 2.19, where we can explicitly comprehend the flow of weight matrices that remain the same through the network for a particular time step. Further, we compute the loss value from each unit in RNN, concluding a single iteration of forwarding pass through the network. All the computed loss values from the individual time steps are then summed into a single loss value *L* which also defines the total loss of the network. Now the updated hidden state of each step in the forward pass can be expressed as follows,

$$h_1 = tanh(W_{hh}^T h_{t-1} + W_{hx}^T x_t) \tag{2.16}$$

Where $tanh$ represents the hyperbolic non-linear function used with RNN whose value can bothe negative or positive, allowing for a decrease or increase in states as a contrast to a sigmoid function that only outputs non-negative values. As we are feeding two separate inputs, one from the previous state and another from the input xt, we use two weight matrices represented by WT and WT as shown in Figure 2.21. Now the output vector for *hh* each timestamp is expressed as,

$$\hat{y}_t = W_{hy}^T h_t \tag{2.17}$$

Where *ht* represents the computed hidden state and *WT* represents the weight matrix between the hidden state and the output unit.

Training an RNN requires updating each weight present in the network at each time step, for which we use the variant of backpropagation called backpropagation through time (BPTT) algorithm, where the errors are propagated backward at each individual step and then finally across all the time steps to the beginning of the data sequence as shown in the Figure 2.19.

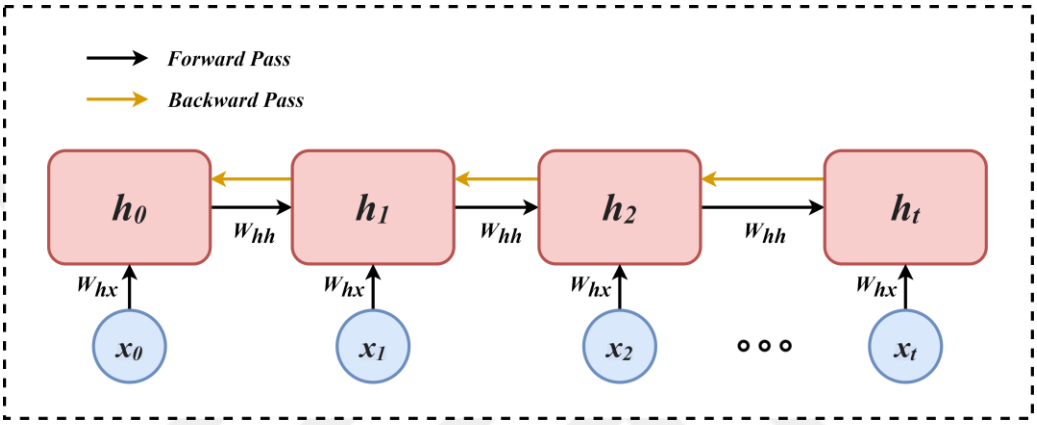


Figure 2.19: RNN Gradient Flow

In the case of deeper RNN architectures, computing the gradient in the network with respect to cell state *h*0 involves several repeated multiplication of the weight matrix as well as repeated gradient computation using the activation functions. This results in the issue of exploding gradients where gradients become increasingly large due to constant accumulation per step and the network are unable to optimize them leading to the overall instability of the network due to the extreme weight updates. The other common issue faced by RNN architecture is vanishing gradients, where the gradients become increasingly smaller in the midst of repeated matrix multiplications leading to the network being unable to be trained and optimized after a few number of epoch cycles.

2.4.3 Long-Short Term Memory

In order to mitigate the problem of exploding and vanishing gradients, Hochreiter and Schmidhuber [25] developed long short-term memory (LSTM) units that are retrofitted with simple RNN cells to enable them to control the information flowing through them selectively. The core component of LSTM units is the information gates, which can selectively add or remove information from its cell state. Gates basically consist of a sigmoid neural network layer and a pointwise multiplier unit. The sigmoid layer constricts the retention of information flowing through the cell from zero and one, which essentially gates the flow of information. As shown in Figure 2.20, an LSTM unit is made of three key gate components briefly described below,

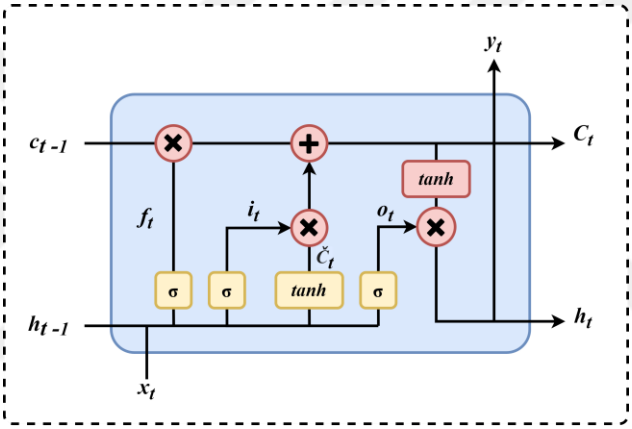


Figure 2.20: Structure of a LSTM unit.

A.Forget Gate: This gate determines which information should be discarded from the cell state. The sigmoid-layer is used to analyze ht−1 and xt values to give the cell-state Ct−1 a number between 0 and 1. The output shows the level of information to be maintained. A value of 1 represents keep everything, whereas the value of 0 represents completely forget this information. This gate can be expressed by,

$$ft = \sigma(Wf \cdot [ht{-}1, xt] + bf) \qquad (2.18)$$

B. Store Gate: This gate determines what information we are going to store in the new cell state. In this two-part process, first, a sigmoid layer also denoted as the input gate layer it decides which values we will be updating. The next layer $tanh$ creates a vector of new candidate $\tilde{C}t$ which will be added to the new state. These steps can be expressed as follows,

$$i_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_i) \qquad (2.19)$$

$$\tilde{C}t = tanh(W_C \cdot [h_{t-1}, x_t] + b_C) \qquad (2.20)$$

Now we update the old cell state $Ct{-}1$ into the next cell state Ct based on the computation of the last two gates. We multiply the old state ft hence forgetting the information earlier, then we add it with the information from store gate i.e. the value derived from $it * \tilde{C}t$. This step is expressed by the equation,

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}t \quad (2.21)$$

C. Output Gate: Finally, the cell needs to determine what information it will be output at the current cell state. Using the gate's sigmoid layer, we decide how much information of the cell state will be outputted. Further, the cell state is put through $tanh$ unit, which squashes the values between -1 and 1, which is multiplied by the output of the sigmoid gate. The process can be expressed in the following equations,

$$ot = \sigma(Wo \cdot [ht{-}1, xt] + bo) \quad (2.22)$$

$$ht = ot * tanh(Ct) \quad (2.23)$$

The primary instinct behind LSTM is its ability to create an uninterrupted gradient flow between various cell states by maintaining independence for each cell in the network, which alleviates the problems of vanishing and exploding gradients seen in simple recurrent neural networks. This enables the network to create long-term and short-term retention dependencies without losing essential information and filtering the non-important information.

2.5 RELATED WORK

In this section, we review the literature which is significant for the development of this thesis. The section is divided into three distinct areas related to the nature of algorithms applied in the design of intrusion detection systems.

2.5.1 Statistical based Approach

The early sketch of 1986 was suggested by Dorothy E. Denning to create an architecture for general usage independent of any particular system, application environment, or type of intrusion[5]. Denning was responsible for creating an intrusion system. Her work was inspired by a 1980 study of Jim Anderson, which developed a way to examine data from the computer to identify abnormal patterns of use at the end of every day. Anderson's method primarily used a statistical analysis approach using large dump files consolidated from all the infrastructure machines [6]. The study was further extended to the IDES, abbreviated in 1988 by Teresa F. Lunt at SRI International for the intrusion detection expert system [7]. Two main components were present in IDES. The first component acquires the user's normal pattern of behavior and detects different patterns. The second component uses a rule-based approach to encrypt and store vulnerabilities in a knowledge base. As a third component of the IDES follow-up derivations Lunt proposed to integrate the neural artificial network into the expert system. Different research laboratories and business computing companies, including AT&T Bell Labs, started implementing intrusion detection Systems by the 1990ies with their own versions of detection systems based on IDES on multiple other hardware and different programming languages. By introducing the well labeled data detection system KDD-99, researchers have been able to use data mining and machine learning algorithms with computer security to create many powerful and widespread IDSs.

2.5.2 Data Mining, Machine Learning based Approach

Tamas Abraham used the IDDM, abbreviated for Intrusion Detection Using the Architecture of Data Mining, in 2001. Data mining systems have traditionally been running on large off-line data sets. IDDM architecture was designed to use data mining to identify anomalies and misuse in real-time environments. IDDM's rule-based components evolved continuously as the system observed and identified a new type of attack. For updating the rule-set, IDDM used meta-mining, which derives new rules from the database's snapshots containing rule-sets at a given time. In order to identify anomalous and normal network traffic in order to recognize UDP flood attacks Z. Zhang et al. suggested a hierarchical network

intrusion detection system called HIDE, used by a Perceptron-backpropagation hybrid model[27]. The architecture of HIDE is divided into various levels where each tier contains Intrusion Detection Agents that monitor hosts and networks activities and multiple infrastructure units. Tier 1 agents would monitor the server's system activities and bridges present in a single department to generate reports for Tier 2 agents in the HIDE system. Tier 2 agents would monitor an entire LAN topology's network status and process the Tier 1 agents' information. Tier 3 agents collect data from the Tier 1 and Tier 2 agents to take necessary measures to detect potential security threats and maintain a user interface to give insight into the entire tiered topology.

Eskin et al. suggested a framework of unattended intrusion detection using SVM, K-Nearest Neighbor and Clustering Algorithms in 2002 [28]. The geometric framework for unsupervised anomaly detection introduced in this paper maps the normal usage data collected into a feature space. The system's newly observed data is also mapped into a feature space compared with the normal feature space to detect outliers and points present in the sparse regions. The framework can detect intrusions over unlabeled datasets, enabling the system to work with a large swath of raw collected system data without manual labeling. For a detection model of low calculative complexity and error rates, the weiming Hu et al. used an Adaboost-based algorithm with adaptive weight strategies[29]. J. Zhang et al. used data mining technology based on random wood algorithms to construct a hybrid IDS that is both a misuse and a malicious detection system [30]. The framework's misuse detection component builds and maintains the patterns of intrusions in a dataset during its offline phase, which is used for juxtaposing with the live data during the online phase. The anomaly detection component is used to detect anomalies and outliers in the data flow using supervised learning. The hybrid IDS first applies the misuse detection component to filter out the known intrusions before the anomaly detection component observes novel attacks. Chandrasekhar et al. used k- to create their variations of IDS[31] that claimed better test results than the backpropagation Neural Networks and other famous methods of learning machines, which means clustering, fugitive neural networks and radial vector support machines. The framework was shown to attain higher detection rates with boosted speed due to the fact that in each step of the designed IDS framework, the subset of data’s complexity is reduced with the application of each algorithm successively.

2.5.3 Deep Learning based Approach

The victory of ImageNet 2012 led by Hinton et al. showed that deep neural networks were capable of outputting complex models for image recognition [32]. The neural network was able to overcome state-of-the-art algorithms with an overwhelming 10.8 error rate and a renewed interest in the field of deep learning. The team's researchers trained an extensive deep convolutional neural network to classify 1.2 million high-resolution images with more than 1000 different classes. The neural network itself had 60 million parameters and 650,000 neurons consisting of convolutional layers with a final 1000-way SoftMax layer to determine the output. Such an extensive neural network would take a long time to train, so to make the overall network faster, the researchers used GPU-powered machines and regularization method dropout. In the course of the following years, computer security scientists also began to integrate deep neural networks into their study [33]. Their method combines the Deep Belief Networks with Genetic Algorithms (GA) to reduce network structure complexity. The framework applies multiple iterations of GA on the network flow data to produce an optimal network structure used by DBN as an intrusion detection model to classify the attacks. This method is shown to improve the classification accuracy and generalization of the model. The model also acts as self-adapting where different types of attacks can change the network structure to produce associated results and maintain high detection rates. The field was revitalized by N. Moustafa and others[34] through a UNSW-NB15 dataset that contains hybrid records of real-world, modern synthesized network attacks. The UNSW-NB15 Network Dataset has a better function than the decade-old data sets, such as the KDD-99 and NSLKDD, to evaluate NIDS performance.

In 2018 Moustafa et al. used UNSW-NB15 to create IoT traffic data NIDS, using AdaBoost ensemble technology to classify normal and suspicious instances[35]. The AdaBoost set includes the Decision Trees, Naïve Bayes and Artificial Neural Network techniques. The framework focused on MQTT, DNS, and HTTP protocols and their flow identifiers in order to develop NIDS specific for IoT network operations. A. Ahim et al. [36] combined three different approaches to classification based on the decision-making treaties and different rules to create a new ID Service with a CICIDS2017 dataset. In this hierarchal framework, two classifiers operate in parallel and feed their output to the third classifier. The framework has relatively low computational time making the system ideal for real-time intrusion detection.

In 2019, Y. Xiao et al. [37] introduced a CNN-based IDS using KDD99 Dataset Batch Normalization. The proposed framework also removed unused and redundant features using an auto-encoder (AE) network as a dimensionality reductional technique. A hybrid IDS was created to monitor networking and host level activities by Vinayakumar et al.[38]. DNN demonstrated that it outperforms other traditional classifications when conducting a thorough comparative study with diverse machine education and classification classifications. B. Riyaz et al. [39] has designed the KDD-99 dataset IDS for use in CNN wireless networks. The frame used a new CRF-LCfS function selection algorithm that improved detecting accuracy and calculation times for the model. The researcher's proposed method demonstrated a 98.9% detection accuracy and a less than 1% false alarm rate. M. Injadat et al. [40] proposed a multi-stage optimized machine learning framework for Network Intrusion Detection. Their technique showcased a 99% detection accuracy on CICIDS 2017 as well as UNSW-15 datasets and reduced the false alarm rate by 1-2%.

2.6 CONCLUSION

As a conclusion of this chapter, the key points drawn from our study on the related work are given below: • Rule-based systems are outdated. Rule-based detection solutions should be replaced with mathematical-based detection methods based on intelligently computational methods. • For construction of an abstract detection engine, misuse detection is more practical than anomaly detection. • Deep supervised learning has a high potential to detect intrusions in the large scale network and is well worth investigating. • However, the research and development of using deep supervised learning for network intrusion detection is far from mature. Insufficient training data and high false alarm rate generated by the detection system should be addressed. This thesis addresses these issues and proposes three deep learning models, which are presented in the following three chapters.

3. PROPOSED MODEL AND METHODOLOGY

This chapter will discuss the proposed model and techniques applied to build for our novel IDS architecture. Section 3.1 will describe the unified deep learning architecture illustrated in this thesis. Section 3.2 will discuss the transfer learning methodology used to make our applied model perform with optimal accuracy and speed in a real-world environment. Section 3.3 will explore the system architecture and data pipeline of the proposed novel methodology. In Section 3.4, we will examine the UNSW-15 Dataset as well as various data-preprocessing techniques applied. Section 3.5 will discuss the evaluation principles we will be using to judge our candidate IDS model.

3.1 UNIFIED DEEP LEARNING ARCHITECTURE

This thesis includes a CNN, LSTM present in its hidden layer and fully connected layer units in order to prevent classification markings in our selective deep learning architecture for IDS. As illustrated in Figure 3.1, a modular approach is taken in the suggested unified IDS model by combining the architecture of the three distinct deep study models and their latent function extraction, memory retention and classification capacity to achieve greater accuracy than the models used separately.

A CNN is able to learn and identify patterns through an input space, while LSTM units can learn and recognize patterns over time. A DNN or a fully connected layer, on the other hand, can learn to create accurate class outputs from the input vector. The two feedforward networks include CNN and DNN, in which the data can only flow forward. In order to extract its features, CNN can use a 2D input into internal vector representations. By contrast, LSTM provides the possibility of using the CNN functionality vector output and builds internal conditions which weights can be updated repeatedly as data flows in LSTM in a recurring manner. In contrast, we have LSTM and CNN. The CNN extracts the intrinsic features from the input during this whole process. LSTM interprets these features in various time stages, making the architecture more efficient to gather more detailed data representations and relationships, in contrast to any individually applied network architecture.

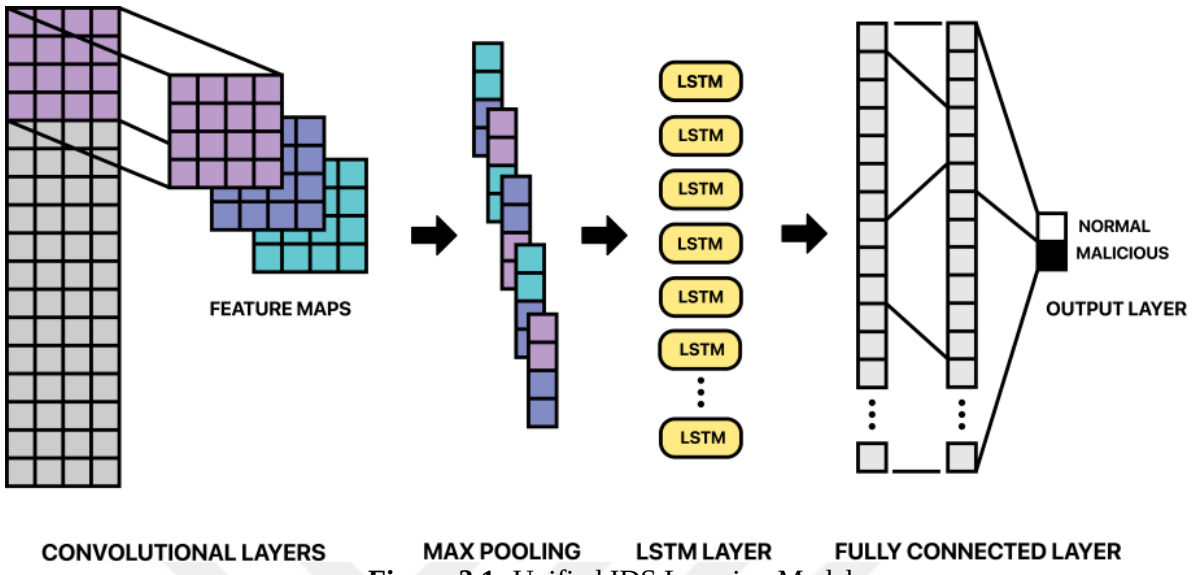


Figure 3.1: Unified IDS Learning Model

In the past the combination of DNN, CNN and LSTM was examined[41], where models are trained individually and subsequently combined with output. In order to ensure the successful models within the scheme, we are training the unified model with each model that provides its processed feature output.

In this thesis, we will be using a modular approach to create a novel deep learning model for our Intrusion Detection System. During our research progression, we applied various machine learning and deep learning techniques to select the candidate model for our IDS. After benchmarking each technique's performance, we used a modular approach of assorting distinct layers of distinct deep learning models and combining them to create a unified model. The unified model was able to outperform other applied models, as it was able to draw on the strengths and advantages of other models. The unified model consists of feature extraction layers of CNN known as convolutional layers, the temporal sequencing layers of LSTM, and fully connected layers of DNN for label classification.

Table 1 shows a summary of our CNN-LSTM candidate, where the contextual features in the training set are first extracted using CNN layers. The usefulness of CNN to sample input while retaining the main characteristics during extraction reduces the general dimension of the feature

parameters. CNN's output will then be fed into the LSTM layers in order to model the signal and use the BPTT algorithm to train weights quickly. Lastly, the output is modeled in the LSTM layers into fully connected layers to obtain higher order feature representation that separates the output into different class labels.

Table 3.1: CNN-LSTM IDS Model Architecture

Layer Type	Output Shape	Total Units
conv1d_1 (Conv1D)	(None, 32, 64)	256
conv1d_2 (Conv1D)	(None, 32, 64)	12352
max_pooling1d_1 (Pooling)	(None, 16, 64)	0
conv1d_3 (Conv1D)	(None, 16, 128)	24704
conv1d_4 (Conv1D)	(None, 16, 128)	49280
max_pooling1d_2 (Pooling)	(None, 8, 128)	0
conv1d_5 (Conv1D)	(None, 8, 256)	98560
conv1d_6 (Conv1D)	(None, 8, 256)	196864
max_pooling1d_3 (Pooling)	(None, 4, 256)	0
lstm_1 (LSTM)	(None, 100)	142800
dense_1 (Dense)	(None, 256)	25856
dropout_1 (Dropout)	(None, 256)	0
dense_2 (Dense)	(None, 128)	32896
dropout_2 (Dropout)	(None, 128)	0
dense_3 (Dense)	(None, 1)	129
Total Trainable Units		583,697

3.2 TRANSFER LEARNING

Transfer learning is a concept in which the knowledge from previous associated tasks is used again to facilitate the learning process for a new task [12]. There are a large variety of real-world applications in which knowledge gained from previous tasks can be transferred, including developing intruder detection systems that can perform optimally even with limited data and computer resources. Deep transfer learning reduces the massive dependence of data on profound learning algorithms to learn about the underlying data patterns. In general, our objective is to transfer knowledge from a source field into a target area by relaxing the assumption that the education data and test data are independently and identically distributed and that the real world data are uncommon. Figure 3.2 shows how the network architecture of a model is transferred into a target area with smaller data sets and limited calculation resources.

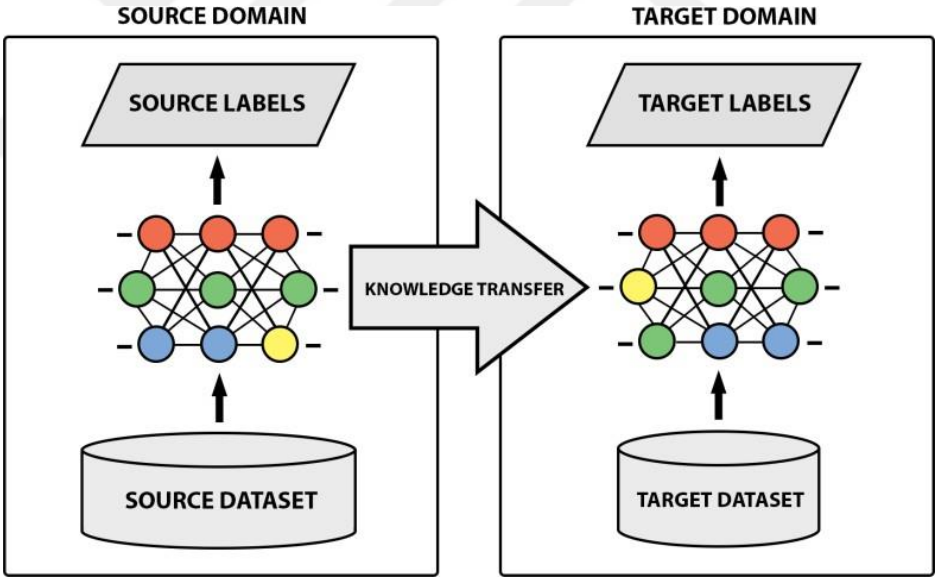


Figure 3.2: Transfer Learning Process

A domain can be represented as, $D = \{X, P(X)\}$, which consists of two parts: the feature space X and a margin distribution P(X), Where X = $\{x1, x2, \ldots, xn\}$, $xi \in X$.

Whereas A task can be represented as, $T = \{Y, P(Y|X)\} = \{Y, \eta\}$, Y = $\{y1, y2, \ldots, yn\}$, $yi \in Y$, where Y is a label space, and η represents the predictive function which can be learned from the

training data including pairs $\{xi, yi\}$, where $xi \in X$, $yi \in Y$; for each feature vector in the domain, η predicts its corresponding label as $\eta(xi) = yi$ [43].

we consider our source domain as DS, and target domain as DT. The source domain data is denoted as $DS = \{(xS1 , yS1), \ldots , (xSn , ySn)\}$, where $xSi \in XS$ is the data instance and $ySi \in YS$ is the corresponding class label. In our IDS, DS is the set of term vectors together with their associated attack and malicious labels. Similarly, we denote the target domain data as $DT = \{(xT1 , yT1), \ldots , (xTn, yTn)\}$, where the input xTi is in XT and $yTi \in YT$ is the corresponding output. We can now give the transfer learning definitions as follows, Given a source domain DS, learning task TS, a target domain DT and learning task TT, transfer learning aims to help improve the learning of the target predictive function ηt by using the knowledge in the source domain DS and learning task TS, where $DT \neq DS$, or $TS \neq TT$. The size of DS is much bigger than DT in various applied situations. Additionally, when there is some relationship, explicit or implicit, between the two domain's feature spaces, we say that the source and target domains are related. In this study, the two domains are related as they share a similar feature space from intrusion datasets. A transfer learning task defined by $(DS, TS, DT, TT, \eta t)$ becomes a deep transfer learning task if ηt is a non-linear function represented by a deep neural network.

There are two extreme forms of transfer learning referred to as one-shot learning and zero- shot learning, which were also studied during this study's progression. In one-shot learning, only one labeled example of the transfer learning task is given to the model to learn and make inferences on future data in a separate domain, whereas in zero-shot learning, no labeled examples are given at all for learning the task. These forms of transfer learning work in the scope of different use cases and specifically if we are using unsupervised deep learning where the model has to find the underlying structure and nature of the given data or the amount of training data at hand is of less size. In the case of our use case, because we are interested in a number of cybersecurity attacks and the data at our disposal is of large quantity, we used the standard approach towards transfer learning.

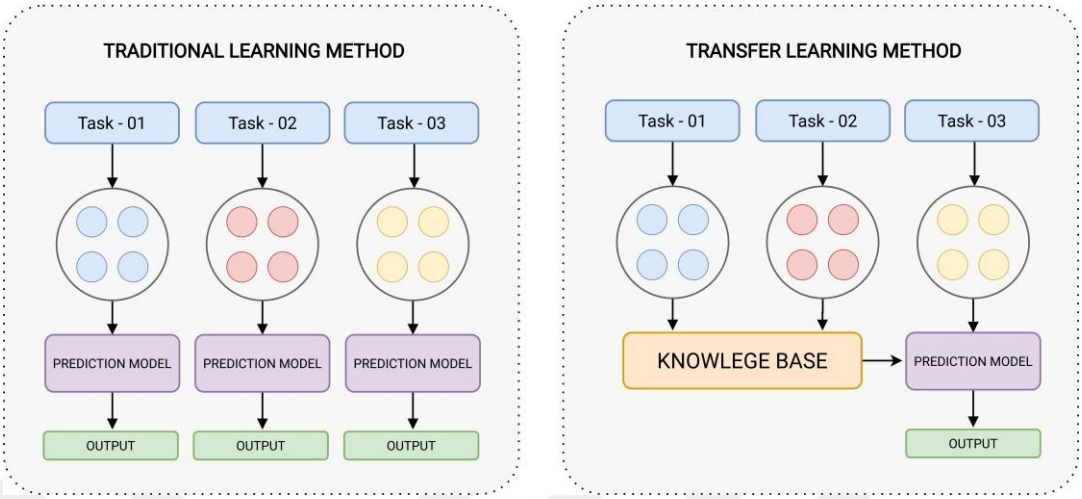


Figure 3.3: Comparison between Traditional learning and Transfer learning method.

As shown in figure 3.3, transfer learning methodology is fundamentally different from the traditional learning methods and systems. The figure represents the tasks and domains where we have a similar distribution and type of data, in the case of IDS, a network flow that shares a similar type of labels and datapoints. In traditional methodology, we construct a neural network model and use the same model to perform different tasks of similar nature independently. The model will perform optimally as long as the data which it is classifying is found to have an underlying structure it learned to detect in its training phase, but as deliberated previously, the results will falter when the data observed by the neural network is entirely new that might not have been present in the training dataset. In transfer learning methodology, we extract the knowledge learned from a model in one or more task setting to build a knowledge base in the form of a neural network architecture and learned weights to apply them for other similar tasks. The advantage this provides the system is that now we have the ability to run simulations in a lab setting to evolve our models to improve their performance each time before we deploy them in real-world environments. The model learns underlying patterns in a different segment of data with similar distribution in each simulation, which optimizes its weights to accommodate all the knowledge learned from the previous tasks for the application in the future tasks. The performance is also not just limited to the accuracy measure. The model already has a primary structure intact from previous tasks and

does not take more time to start anew, which speeds up the overall system. The transfer learning methodology reduces the time taken by the model to give its output results. These large neural network models usually take a longer time to test an entire dataset work faster to provide their classification results. This enables the conception of real-time based neural network architectures that work in live production environments to give classification results on impulse.

In this research's, we applied the transfer learning methodology to augment large neural networks to classify the network traffic flow, aimed to find pervasive intrusion and cybersecurity attacks to safeguard the modern network infrastructure. The design of a robust intrusion detection system requires it to continuously monitor network traffic and drive the defense mechanisms to detect any suspicious activities or threat patterns in the network flow. We previously established that neural networks are capable of detecting such threats at a greater granularity compared to the traditional data mining and machine learning methods, but for their full utility, we also need deep neural networks to work at a robust pace as the entire paradigm of training, validating and testing takes more time compared to other rudimentary methods. The transfer learning methodology enables the system to improve its speed and accuracy performance to become viable in an event-driven, real-time environment.

3.3 SYSTEM ARCHITECTURE

This section will discuss the system architecture of the proposed network classification and intrusion detection system. Figure 3.4 outlines the system architecture of the end-to-end deep learning pipeline applied to network classification tasks using different subcomponents.

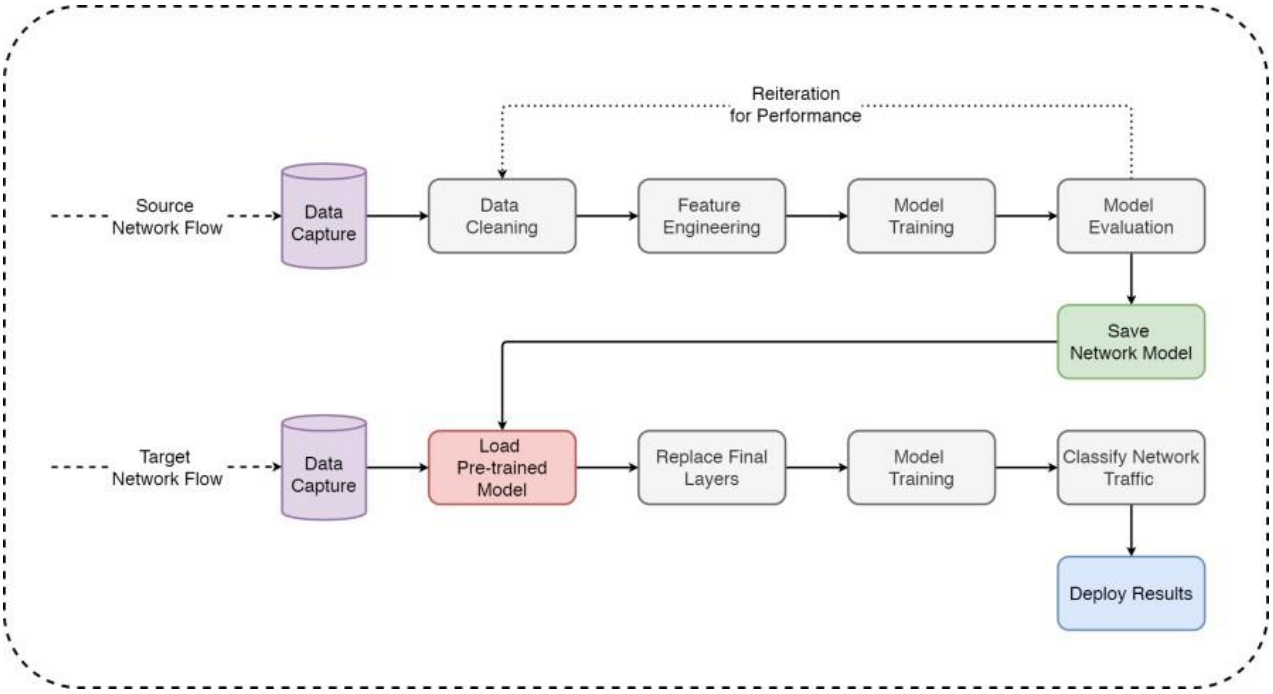


Figure 3.4: System Architecture Flowchart

The pipeline's system architecture can be divided into 7 main steps briefed as follows,

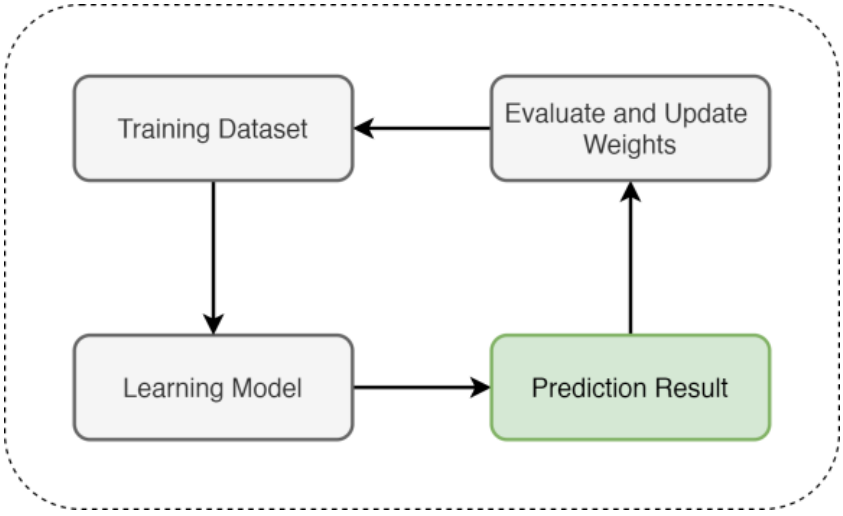
1.Data Capture: The pipeline begins with capturing data from the source domain as well as the target domain's network flows. The UNSW-15 dataset, which is also further discussed in more depth, used the IXIA Perfect Storm tool to capture the real network traffic and the synthetic contemporary cyber-attacks in the form of packet data. Further, the TCPdump tool is used to

generate Pcap files, which is further fragmented from the 100 GB of captured data into 1000 MB segmentations.

2.Data Cleaning: The raw Pcap files are then synthesized to generate reliable features using Argus and Bro-IDS toolsets. Argus tool processes the Pcap files and generates the network flow features as outlined in Table 2. The open-source Bro-IDS tool analyzes network traffic using the raw Pcap files and generates connection information such as HTTP, FTP requests, and replies. These tool's output is then matched and combined to create a full length of a feature set, including both flow-based and packet-based features.

3.Feature Engineering: To further improve our data's efficacy and its raw features, we use various data pre-processing techniques such as feature selection, feature scaling, and feature normalization, also discussed at length in proceeding sections. The main aim of feature engineering is to get the best speed and accuracy performance when the data is used with a model to draw inferences. Feature engineering creates the most accurate representation of the underlying patterns in the data flow.

4.Model Training: During this phase, we use the deep learning frameworks alongside the formatted data from the previous steps to train and build analytical models capable of learning semantic relationships in the data. Learning the data's fundamental structure enables the model to predict the newly seen data's nature, which can be utilized for various classification tasks. In our case, we



are using network flow data consisting of both normal and malicious packets for training our model so that the model becomes efficient in recognizing and classifying new network flow data based on that criteria. This is an iterative process where we incrementally improve our model's classification abilities using labeled data until the model can give accurate prediction results.

Figure 3.5: Model Training Process

5.Model Evaluation: Once we are satisfied with our analytical model results from the training phase, we evaluate the model using a subset of unseen data that was not used during its training. We use the predefined evaluation criteria to judge the performance and efficacy of the applied model. we can further fine-tune the applied model's various hyperparameters to retrain the model, improving our results with each iteration as shown in Figure 3.5.

6.Transfer Learning Methodology: When the model provides satisfactory results based on the defined evaluation criteria, we will save the model and its weights in the HDF5 format designed to store large and complex data hierarchically. The model is then transferred to our target domain with an entirely different network flow with similar engineered features. If the output labels are required to be different in the target domain, we will unfreeze the last layers of the model and train them again. Using the pre-trained model with intact weight parameters, we utilize the derived knowledge from the source domain, which cuts down the training time and required computational resources. If the source domain model were large and powerful, trained with an extensive amount of training examples, it would generalize appropriately in the target domain. In section 4.4, we would show our results from the transfer learning methodology experimentally.

7.Deployment: The architected model has been through various iterations in both source and target domains based on our set evaluation criteria and metrics. Once we are satisfied with the classification results, we can deploy the system in a live production environment. The advantage transfer learning methodology brings is that now we can iterate and evolve our model and augment its performance abilities with the new subset of network flow it observes and learns to classify.

This improves the model over time to recognize many types of packet data in the network traffic while working in the real world environment, which is not possible using the traditional deep learning methodology.

As shown, we train the unified model to classify network packets iteratively. The model then becomes an integral part of the Intrusion Detection System, which receives the network flow and performs various data pre-processing methods to augment its classification performance. This designed architecture is then transferred to a different domain with less data and computation resources using the transfer learning methodology, where it adapts to the target domain to maintain its performance on an unseen data flow, while improving its overall classification speed significantly. This outlined framework can be utilized to deploy large and powerful deep learning based intrusion detection systems on resource sparse edge devices to maintain their security, despite the data and computational limitations.

3.4 DATASET DESCRIPTION

The primary dataset used for architecting the intrusion detection system was the UNSW- 15 dataset created by capturing raw network packets using the IXIA PerfectStorm tool. The Cyber Range Lab made the Australian Center for Cyber Security (ACCS) dataset open source at the University of New South Wales, Australia. As shown in Figure 6, the dataset has nine types of cyber-attacks, specifically DOS, Reconnaissance, Generic, Fuzzers, Shellcode, Worms, and Backdoors, as well as packets with normal activity.

UNSW-15 dataset contains a total 2 million network packet records which is partitioned into four CSV files. The selected partition will be divided into a training set containing 154 603 documents and a subset of these data will be used. We will also use a validation and test set of 51,535 records to properly evaluate the performance of the applied profound learning model in various areas.

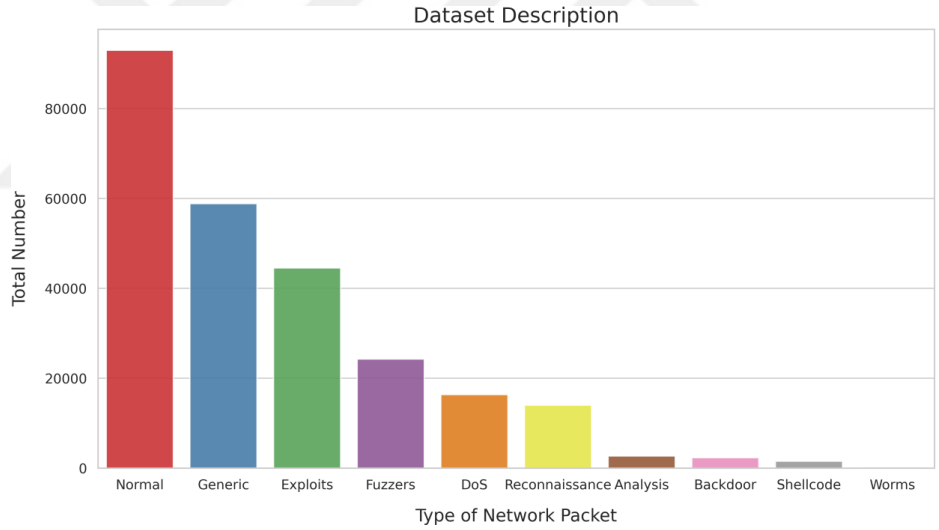


Figure 3.6: UNSW-15 Dataset Description

3.4.1 Data Pre-Processing

The first data pre-processing technique we will elucidate upon is feature selection. The features we use to train our model form the core of our model and significantly impact the model's overall performance and efficacy. In total, the UNSW-15 dataset has 49 features with appropriate class labels. To optimize a dataset with many features that may or may not improve the performance,

we clean the data, which is irrelevant to the task. We performed the feature importance test, which uses a filter-based method to extract the best features in the dataset as shown in Figure 3.7, where each feature has a scoring value that represents how important and relevant the feature is to the output variable.

On the basis of this calculation, we can also drop the unnecessary entries from the data set. Functional selection allows the model to allocate its computing resources and to accelerate training times by reducing data for processing and building the model. The presence of irrelevant and redundant data makes it much more difficult to discover knowledge.

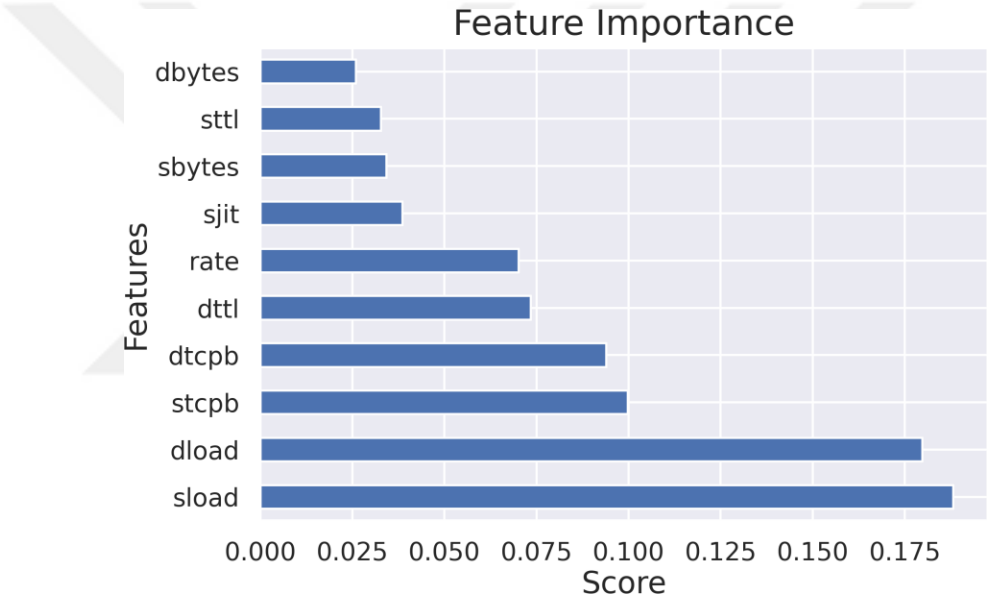


Figure 3.7: Feature Selection Plot

Table 3.2 shows few key features determined by feature selection as important well as their brief descriptions.

Table 3.2: Dataset Key Feature Descriptions

Feature Name	Data Type	Description
sload	Float	Source bits per second.
dload	Float	Destination bits per second.
stcpb	Integer	Source TCP base sequence number.
dtcpb	Integer	Destination TCP base sequence number.
sbytes	Integer	Source to destination transaction bytes.
dbytes	Integer	Destination to source transaction bytes.
sttl	Integer	Source to destination time to live value.
dttl	Integer	Destination to source time to live value.
swin	Integer	Source TCP window advertisement value.
dwin	Integer	Destination TCP window advertisement value.
sjit	Integer	Source jitter (millisecond).
djit	Float	Destination jitter (millisecond).
stcpb	Integer	Source TCP base sequence number.
dtcpb	Integer	Destination TCP base sequence number.
spkts	Integer	Source to destination packet count.
dpkts	Integer	Destination to source packet count.

Further, to visualize the correlation between each feature, we plotted the correlation heat map as shown in Figure 3.8. A correlation matrix shows the importance and relationship between two features in a dataset. The main aim of such visualization is to understand and see patterns in the data. It becomes clear which features are highly correlated to each other and have a linear relationship between each other, as the change in one feature will lead to a definite change in another. This is an important data-preprocessing step as these patterns can be further utilized to build predictive models which harness the co-related features to judge the unseen data with these similar label feature which makes it essential to establish before continuing on with any form of statistical modeling or analysis of the dataset.

Figure 3.8: Features Correlation Heat Map

3.4.2 Data Normalization

Data standardization or feature scaling is a method for data preprocessing that converts all input values for the study model to a standard scale. As usual, the data won't have a big effect on the Learning Model performance without a broad spectrum of features. This leads to other features which may be important, however, with a smaller range the general inferences drawn from the predictive model become less effective. It is important to scale data in order to achieve all characteristics equality. This helps to achieve convergence faster and also makes optimization much easier via the algorithm gradient descent.

While scaling helps to bring the ranges of features within a specific scale, normalization changes the shape of our dataset's distribution to become a normal distribution. A normal distribution, also known as a probability bell curve, is the statistical distribution where the observations are symmetrical around the mean. Normalization automatically rescues the data from its normal range to a standard range, in which for each characteristic, the minimum value becomes zero and the maximum value transforms in one and thus gives all the characteristics in data an equal basis for statistics.

This normalization technique is also known as min-max normalization which was used to rescale and normalize the UNSW-15 dataset for the IDS architecture. In comparison with other normalization, the min-max normalization keeps the form of the feature intact during scales. This particular data pre-processing stage is important because various algorithms such as logistic regression and neural networks are supposed to scale and normalize the data inputs for processing.

3.5 EVALUATION CRITERIA

This section will discuss the evaluation criteria for quantifying the performance and efficacy of our IDS machine learning and deep learning models.

3.5.1 Classification Accuracy

Accuracy is a classification model evaluation metric in which we compare the correct number of predictions with the number of predictions that are generated. The classification precision formula can be expressed as,

$$Accuracy = \frac{\text{Number of Correct Predictions}}{Total\ Number\ of\ Predictions} * 100 \quad (3.1)$$

3.5.2 Confusion Matrix

A confusion matrix is a visual representation of the performance of a classification model. It basically is a table with four different combinations of predicted and actual values. A classification model's outcome can be summarized into these four possible categories,

1.True Positive: This corresponds to the values which were predicted to be positive, and they turn out to be positive and correct. In the case of IDS, the model predicted the packet to be malicious, and it indeed is malicious. Hence the IDS made a correct prediction. A higher true positive value means the model is making good positive predictions.

2.False Positive: This corresponds to the values which were predictive to be positive, but they turn out to be negative and hence false. In the case of IDS, the model predicted the packet to be malicious, but the packet was actually a normal packet. A high false positivity of an IDS leads to unnecessary false alarms and causes needless disruption of services. A low false-positive value is an indicator of an accurate IDS model.

3.True Negative: This corresponds to the values which were predicted to be negative and they turn out to be negative and hence correct. In the case of IDS, the model predicted the packet to be

normal and it was indeed a normal packet. Again, a higher true negative is also deemed to be a positive indicator of the model's performance.

4.False Negative: This corresponds to the values which were predicted to be negative, but they were in actual positive values. In the case of IDS, the model predicted the packet to be normal, but the packet was actually a malicious packet. This is the most crucial indicator of an intrusion detection system's performance. This value represents how many wrong predictions the model made as each such instance can prove harmful to the infrastructure the IDS aims to protect and safeguard.

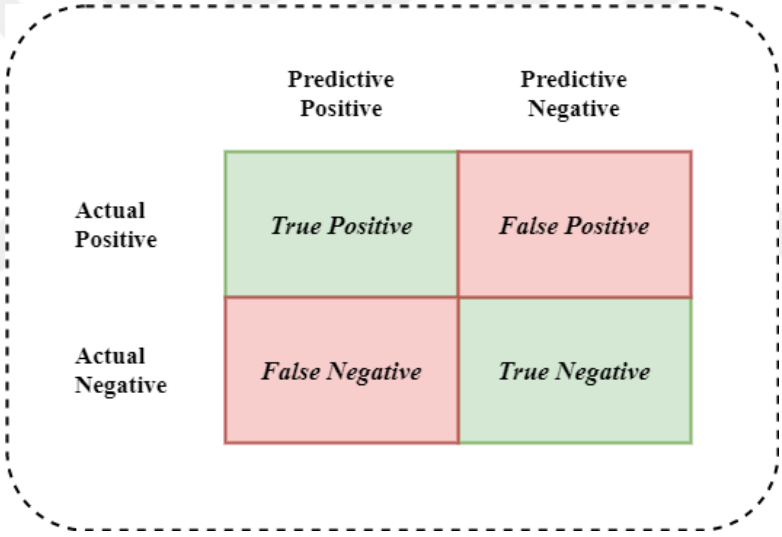


Figure 3.9: Confusion Matrix Sample

Figure 3.9 is a visual example of a sample confusion matrix. In essence, among these values, we are interested in the scope of a false positive and false negative, both of which cause the IDS to perform poorly in an applied sense. The research in part aims to mitigate and improve the score of the detection system's false positivity and false negativity.

3.5.3 AUC - ROC

AUC-ROC curve is another applied performance metric criteria for the classification model. Term AUC is abbreviated for Area Under the Curve which measures the two- dimensional area underneath the ROC abbreviated for Receiver Operating Characteristic Curve at various threshold settings. To plot a ROC, we compare the parameters namely True Positive Rate and False Positive Rate which can be summarized as follows,

A. The true positive rate is also called the sensitivity of a model that determines the share of positives and positives. A. This can be said as,

$$TPR = \frac{True\ Positive}{True\ Positive + False\ Negative} \tag{3.2}$$

B. False Positive rate is known also as a model's particularity which determines the proportion of negative values that have also been identified as negative by the model. This can be said as,

$$FPR = \frac{False\ Positive}{False\ Positive + True\ Negative} \tag{3.3}$$

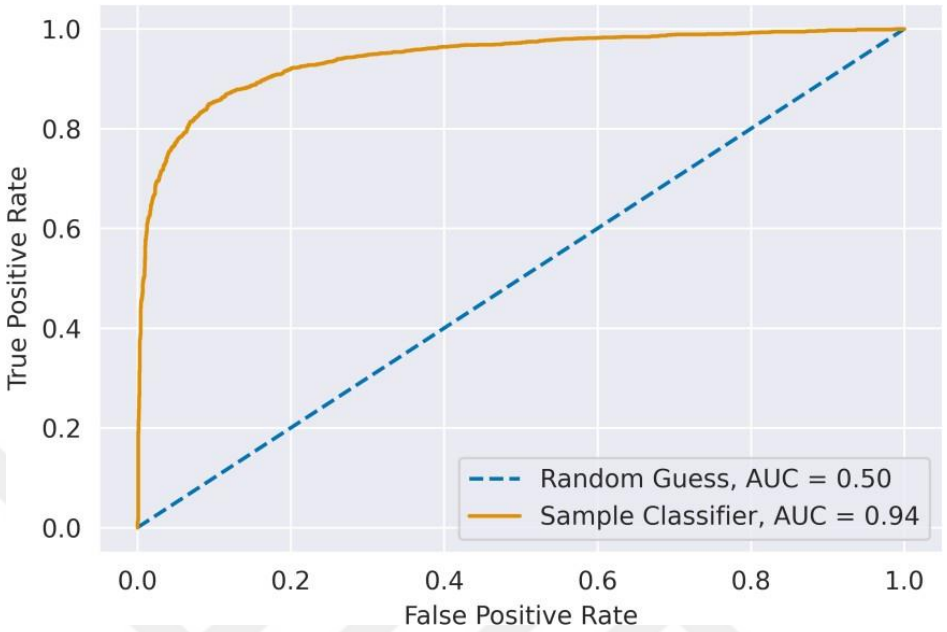


Figure 3.10: ROC Curve example with a Sample Classifier

ROC curve plots the True Positive Rate of a model with False Positive Rate at various classification thresholds as shown in Figure 3.10. The AUC value aggregates the performance of the model across all possible classification thresholds.

3.6 CONCLUSION

In this section, we have a profound neural network architecture, which further reduces the false positive rate of intrusion detection in the network. The model uses CNN to learn space characteristics for traffic and LSTM time characteristics. We sync both CNN and RNN to learn the data input at the same granular levels, to avoid the loss of knowledge due to different learning focus of CNN and RNN. We also include batch normalization in the design to enhance learning.

4. EXPERIMENT RESULTS

This section will show our results and their analysis from the experiments performed to guide our design of the Intrusion Detection System.

4.1 DEVELOPMENT ENVIRONMENT

The programming language primarily used in this research is Python 3.7 with deep learning framework TensorFlow 1.15 and Keras in the backend. The development environment used mostly throughout the research was Jupyter Notebook. This efficient web-based integrated platform enables various kinds of data processing and statistical modeling and provides a single place for all the libraries to be utilized in a project. We used Sci-Kit learn as our machine learning library, Pandas library for data analysis and manipulation, NumPy for n- dimensional array support, and Matplotlib to produce all the graphs for the results.

4.2 DEEP LEARNING METHODS

This section will discuss our experimentation and results in the field of deep learning. As previously discussed, deep learning architectures offer an ability to extract essential features in a given dataset by transforming its data iteratively. The algorithm aims to build and learn deeper representations and patterns using multi-layered network architectures. Unlike machine learning algorithms, which may require human intervention to be trained towards an accurate outcome, deep learning algorithms are self-adjusting. They don't require any explicit human intervention to hardcode the features for improving their results. In the deep learning space, we focused our efforts specifically on three main algorithms, namely

- Deep Neural Network.
- Convolutional Neural Network.
- Long Short-term Memory Network.

For training and validating our model, we will use the two preprocessed partitions of the training dataset and validation dataset. In total, we are using 206,138 packet observations for our experimentation in the source domain environment.

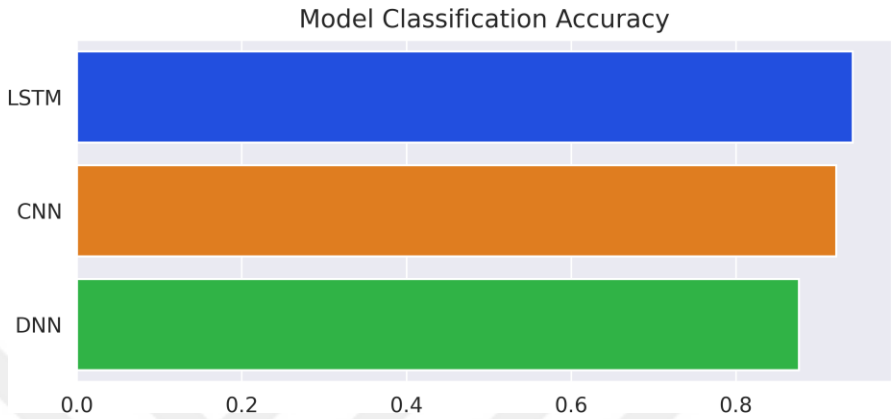


Figure 4.1: Classification Accuracy of Applied Deep Learning Models

Figure 4.1 plots the bar chart for the classification accuracy for each of the deep learning models applied to the task of intrusion detection. From our experimentation, we observed that LSTM demonstrated a 94.42% classification accuracy, CNN gave a 92.16% classification accuracy whereas, DNN gave an 87.66% classification accuracy.

We further studied our applied deep learning models using a ROC curve to visualize our results at various thresholds which are plotted in Figure 4.2.

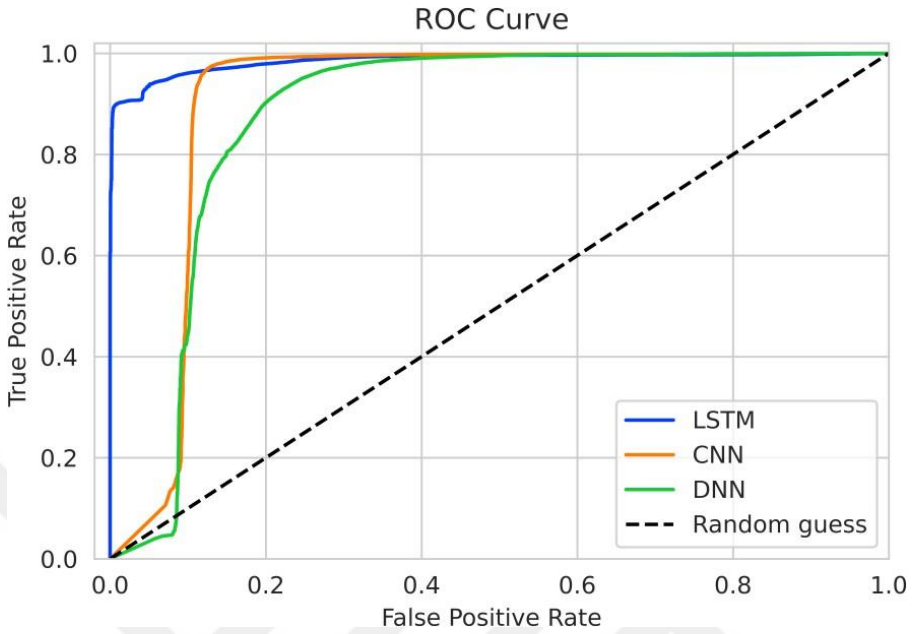


Figure 4.2: ROC curve visualization for applied Deep Learning Models

We will also consider each deep learning model based on the time it took for them to process an entire validation dataset partition to classify the data. Among the applied models, DNN took only 28 seconds, whereas CNN took a total of 2 minutes and 15 seconds. LSTM took 3 minutes and 15 seconds for its complete processing.

Table 4.1: Deep Learning Model Performance Summary

Model	Accuracy	Speed	AUC
Deep Neural Network	87.66%	28s	0.85
Convolutional Neural Network	92.16%	135s	0.91
Long Short-Term Memory	94.42%	195s	0.94

Table 4.1 summarizes our initial experimentation results in the area of deep learning to design and select our target IDS model. We decided to further study both CNN and LSTM models to design

our candidate Intrusion Detection System based on our experimental results. We used confusion matrix metrics for both of these models to thoroughly look into their precise prediction outcomes, as shown in Figure 5.6, which plots the confusion matrix for the applied CNN model. Figure 4.3 plots the confusion matrix for the LSTM model.

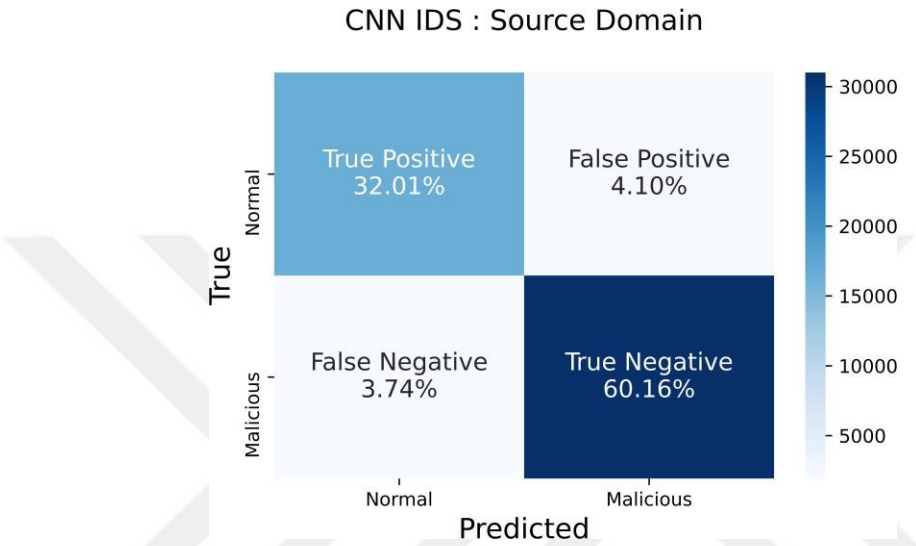


Figure 4.3: CNN based IDS - Confusion Matrix

According to the confusion matrix, the CNN-based IDS model has a 4.10% False Positive and a 3.74% False Negative value. This is an improvement in the prediction outcomes from our machine learning models, but we still require our candidate IDS to have even lower false outcomes

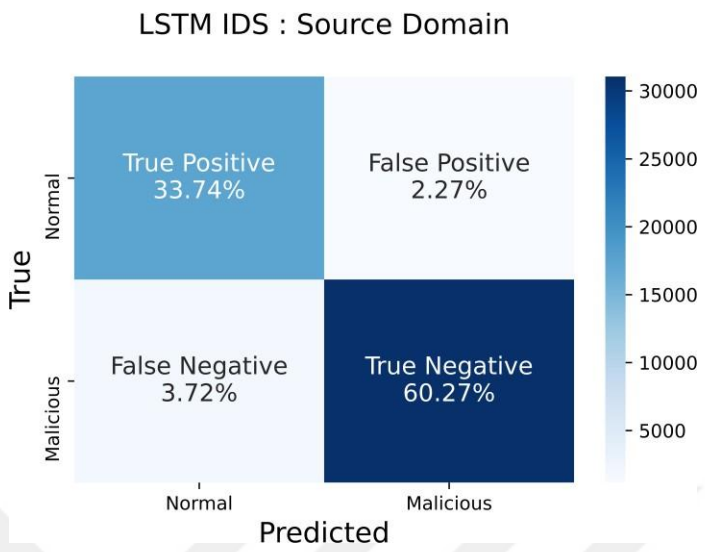


Figure 4.4: LSTM based IDS - Confusion Matrix

The LSTM based IDS model demonstrates an improvement in the False Negative and False Positive values when compared to the CNN model according to the confusion matrix. But a 3.72% False Negative value is still too high, as it means that the IDS based on the LSTM model will let that percentage of incoming malicious packets through its system. The LSTM model's 2.27% False Positive value on the other hand will lead to that percentage of incoming normal packets being dropped by the system due to misidentification as malicious packets.

The standard deep learning models performed much better in terms of their predictive outcomes and classification accuracy when compared with the applied machine learning models. But they still did not provide us the precise outcome results expected from an intrusion detection system aimed to be developed in this thesis.

4.3 UNIFIED DEEP LEARNING NETWORK

Our continued experimentation lead us to consider adopting a modular approach towards constructing our candidate IDS model, where we use three deep learning models' advantages and combine their latent extraction features, memory retention and classifying capabilities to achieve a higher level of accuracy and forecast results than those models are used independently. We have

discussed the overall architecture of our proposed deep learning model. This section will report our experimentation findings using the unified CNN-LSTM model and will compare our results with previously applied deep learning models.

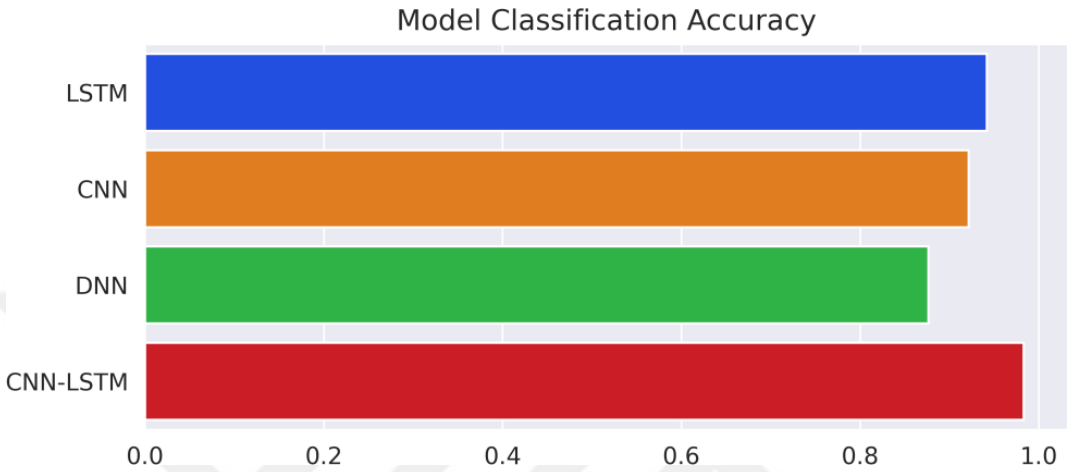


Figure 4.5: Classification Accuracy of Unified Model in comparison with DL Models

As shown in the bar chart plotted in Figure 4.5, our applied unified CNN-LSTM model demonstrated an improved 98.30% accuracy score which was the highest result when compared to other applied deep learning models. We further used the confusion matrix to study in-depth the individual classification of the unified model.

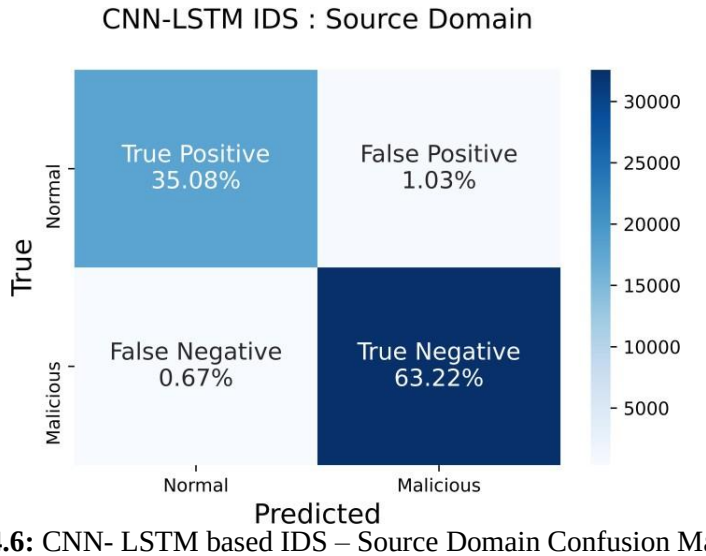


Figure 4.6: CNN- LSTM based IDS – Source Domain Confusion Matrix

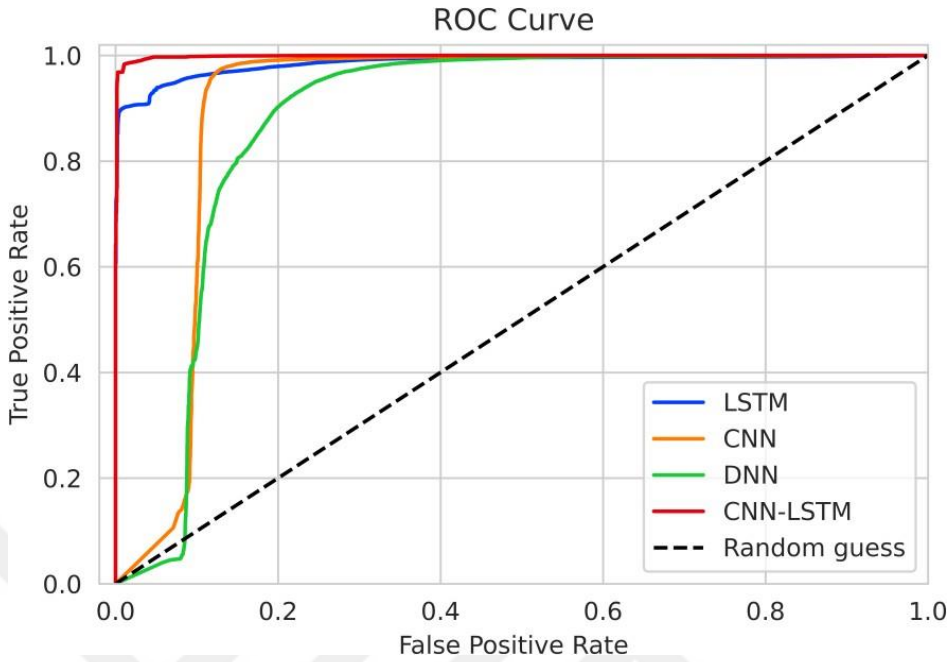


Figure 4.7: ROC curve visualization of Unified CNN-LSTM Model

Based on our confusion matrix metrics as shown in Figure 4.6, our unified model showed improvements in the overall classification of normal as well as malicious packets. The model demonstrated a 1.03% False Positive value and a 0.67% False Negative value. As per these values, the unified model performs much better at predicting the nature packets when we compare its results with the LSTM model which demonstrated a 2.27% False Positive and 3.72% False Negative value. We further plotted the unified model's ROC curve with the other deep learning models as shown in Figure 4.7. The ROC curve of the unified CNN-LSTM model covers the most area on the graph which represents its ability to correctly identify a larger number of packet samples when compared to other deep learning models.

Because we aim to build a highly accurate model that also performs at a fast processing speed, we also need to consider our unified CNN-LSTM model based on the time it took to process the validation data set. Overall, the model took 3 minutes and 56 seconds for its entire processing. The results from all the deep learning models applied in our source domain results are summarized in Table 4.2.

Table 4.2: Model Performance Summary – Source Domain

Model	Accuracy	Speed	AUC
Deep Neural Network	87.66%	28s	0.85
Convolutional Neural Network	92.16%	135s	0.91
Long Short-Term Memory	94.42%	195s	0.94
CNN-LSTM Neural Network	98.30%	236s	0.98

As shown from our summarized experimentation results, the unified model was able to outperform the distinctly applied deep learning models. Overall, our candidate model reached a high accuracy of 98.30% and an AUC score of 0.98. The model demonstrated a satisfactory classification performance, but it took a longer time to process data due to the fact that it's a much deeper and larger model.

4.4 TRANSFER LEARNING RESULTS

One of the key criteria for our candidate model is that it should perform at the same accuracy and improve its overall performance speed in real-world environments. For ensuring this goal, we will be using transfer learning methodologies to transfer the learned weights and network architecture from our source domain to a resource sparse target domain. The target domain is simulated to act as a real-world environment. The transfer learning methodology is deliberated in section 3.2. This section will illustrate the experimental results in our simulated target domain using the Google Cloud Platform. We will also compare our deep learning model's performance in both the source and target domains.

We will use the unseen test data set in this domain to simulate the IDS model in a real environment where it meets completely new data to apply the learned knowledge in the target domain. This helps to evaluate how the model mainly reacts when used in a real-world network infrastructure.

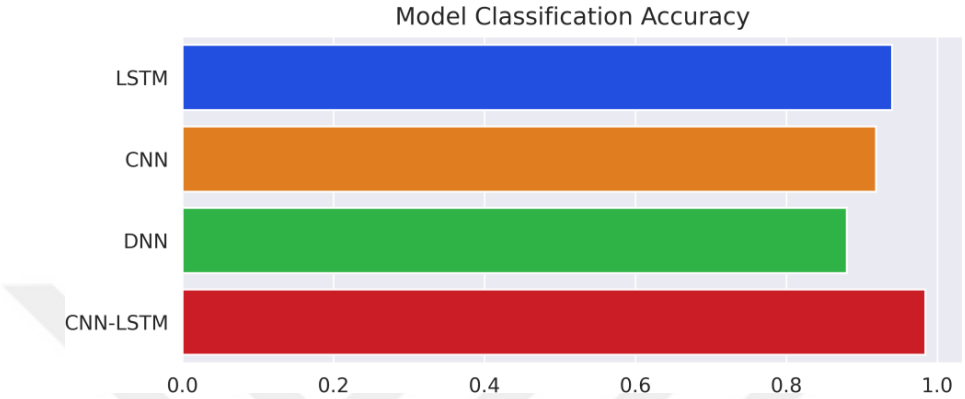


Figure 4.8: Classification Accuracy of Applied Deep Learning models in Target Domain

As shown in the bar plot illustrated in Figure 4.9, the deep learning models were able to maintain their accuracy performance in the target domain with an entirely new dataset unseen by each model. The unified model CNN-LSTM's accuracy improved to 98.43% whereas other models also reported an accuracy improvement in their results. The LSTM model reported an improved 94.18% accuracy while the DNN model reported an improved88% accuracy score percentage.

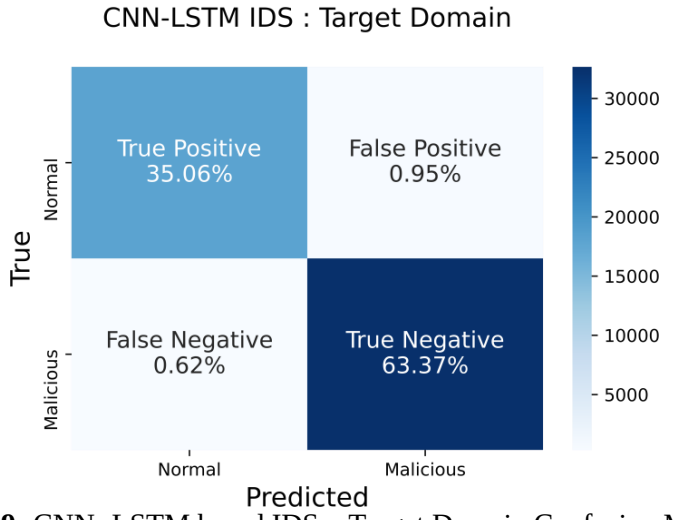


Figure 4.9: CNN- LSTM based IDS – Target Domain Confusion Matrix

In order to further investigate our results for the target domain our candidate CNN-LSTM model classification performance was visualized with confusion matrix metrics, as shown in Figure 4.10. The confusion matrix shows that our new unified CNN-LSTM model reached a false positive 0.95% and a false 0.62% value. This was the best performance in each model of neural networks used in both domains by far. The models demonstrated that they could classify the network packets at a high level of accuracy using their learned weights in the target domain. The ROC curve charted in Figure 4.11 shows that our IDS model's diagnostic ability remained comparable in the target domain.

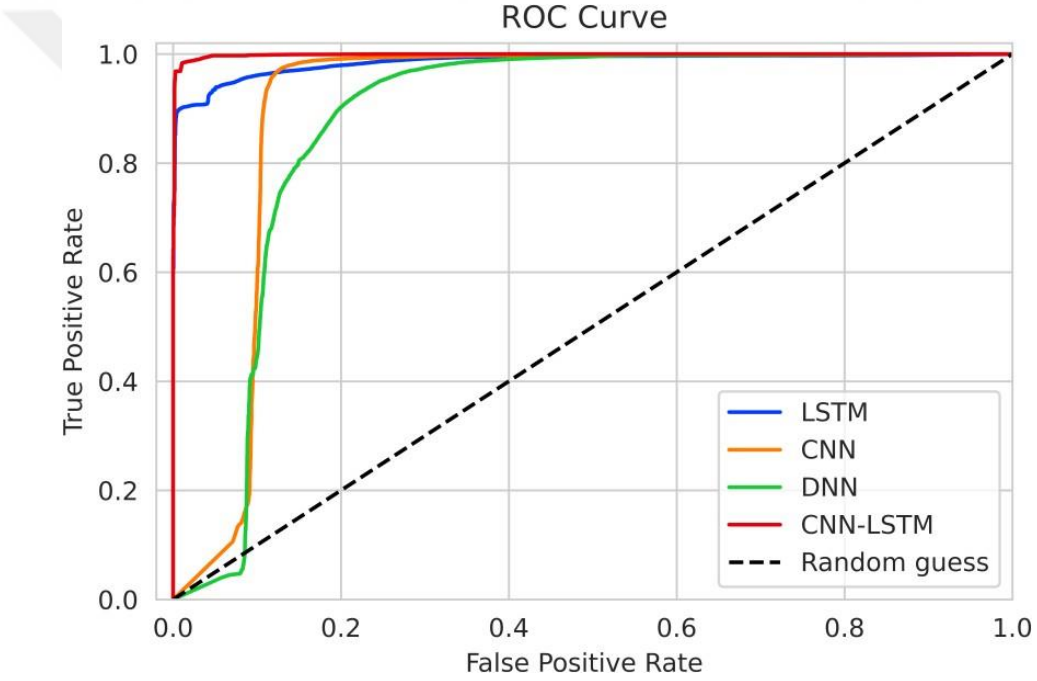


Figure 4.10: ROC curve visualization -Target Domain

Table 4.3: Model Performance Summary – Target Domain

Model	Accuracy	Speed	AUC
Deep Neural Network	88.05%	1.6s	0.85
Convolutional Neural Network	91.88%	18.1s	0.91
Long Short-Term Memory	94.00%	10.9s	0.94
CNN-LSTM Neural Network	98.43%	19.5s	0.98

Overall, in terms of the classification performance, each model applied in the target domain using the transfer learning approach maintained and slightly improved their accuracy on an entirely new and unseen dataset. In terms of the speed performance, the models showcase huge improvements that enable us to build real-time IDS models in real-world settings. Our candidate CNN-LSTM model took mere 19.5 seconds to process the entire dataset, which is a dramatic change from its 3 minutes and 56-second performance speed in the source domain. Table 4.3 showcases the summary of our results in the target domain for each neural network model applied.

4.5 DISCUSSION AND CONCLUSIONS

This chapter illustrated our experimentation and techniques to build a real-time, fast processing intrusion detection system that also demonstrates a high level of accuracy. We showcased the application of both machine learning and deep learning models to architect our model. Upon an exhaustive comparative study, we applied a novel modular approach towards building a unified CNN-LSTM model. Our candidate model outperforms other applied deep learning models for the task of packet classification. To further augment our model to work efficiently in real-world settings, we used transfer learning methodology to transfer our learned weights and model

architecture from our primary source domain to a target domain. The target domain is simulated as the real-world environment, with very low computational resources and data availability. Our results show that our models not only maintained their classification accuracy as well as improved their performance speed dramatically. The candidate CNN-LSTM unified model demonstrated a 98.30% classification accuracy in the source domain and a 98.43% classification accuracy in the target domain with a new and priorly unseen dataset. Our candidate model's speed also saw a boost, wherein the source domain the model processed the validation dataset in 3 minutes and 56 seconds. In the target domain, it processed the entire testing dataset in 19.5 seconds. Our results show that using our novel modular approach towards building IDS models enhances the overall classification ability of neural networks to identify potential intrusion attempts. Adding transfer learning methodology in our design further boosted our models' speed. It made our architecture promising to work efficiently with real-time processing power in the real-world settings on unseen data partitions.

5. CONCLUSIONS AND FUTURE WORKS

5.1 CONCLUSION

This thesis architected a novel intrusion detection system that uses state of the art deep learning algorithms and techniques to give highly accurate network packet classifications. For improving the efficacy of our overall architecture, we used our novel modular approach to develop a unified neural network model that outperforms other techniques illustrated in our research. We have used deep transfer learning methodologies to make our architecture work in real-world settings efficiently. Our research shows that the deep transfer study approach can be very effective for the development by means of knowledge transfer of an efficient, unified network intrusion detection system that maintains and improves its accuracy and speed in the simulated real world.

The proposed method allows us to train a big, powerful IDS model in a highly data and computerized data allocation source domain. Once the performance of our model is validated, its architecture and learned weights can then be transferred to a target area with reduced computing resources. We note that its efficiency is retained and its test speed improves. The objective field is to simulate the real world environment where we use a dataset partition that our models completely ignore while training and development The objective domain.

This thesis shows that high-powered deep-learning IDS architectures can be deployed on less-resourced, effective real-world devices and improve speed by means of transfer learning. The application of transfer learning in the overall design of an IDS improves its performance in a realistic environment. It mainly improves its classification speed, a feature which an IDS requires enormously to safeguard and secure modern network infrastructure. Our research is one of the first practical implementations for the integration of transfers into an IDS core architecture.

5.2 FUTURE WORKS

We would like to add stream processing in the overall design of our IDS architecture in the future. We also aim to use the models constructed in this thesis and apply them to a live network stream to provide our inferences in real-time. Exploring the IDS's design as a system daemon is also a noteworthy aim. The daemon mode will enable our IDS to work ubiquitously in the background as a process and oversee the live network traffic in a parallel, multitasking fashion. A real-time, stream-based IDS architecture can be further deployed on any edge device which uses networking for its day-to-day functioning. Adding GPU support in the source domain will also make the entire architecture dramatically faster in its processing. We would also like to add dimensionality reduction techniques as a pre- processing step in our design, making the architecture work with an even larger volume of datasets. As part of the future work, it would be interesting to use an ensemble approach for our models and compare the results with our current approach. In the future, we would also aim to build our own data sources and test our techniques on various modern network infrastructures.

REFERENCES

[1] *Cisco Annual Internet Report (2018–2023).* (2020). Cisco Systems. https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html.

[2] Wen, T., & Zhu, P. (2013). *5G: A technology vision*. Huawei. www. huawei. com/en/abouthuawei/publications/winwin-magazine/hw-329304.html.

[3] *CIRA Cybersecurity Report*. (2020). Canadian Internet Registration Authority. https://www.cira.ca/resources/cybersecurity/report/2019-cira-cybersecurity-survey.

[4] *The Global Risks Report.* (2019). World Economic Forum. https://www.weforum.org/reports/the-global-risks-report-2019.

[5] Denning, D. E. (1987). An intrusion-detection model. *IEEE Transactions on software engineering*, (2), 222-232.

[6] Anderson, J. P. (1980). Computer security threat monitoring and surveillance. *Technical Report, James P. Anderson Company*.

[7] Lunt, T. F., & Jagannathan, R. (1988, April). A prototype real-time intrusion-detection expert system. In *IEEE Symposium on Security and Privacy* (Vol. 59).

[8] Khraisat, A., Gondal, I., Vamplew, P., & Kamruzzaman, J. (2019). Survey of intrusion detection systems: techniques, datasets and challenges. *Cybersecurity*, *2*(1), 1-22.

[9] *Word Threat Assessment Report*. (2017). United States Intelligence Community. https://www.dni.gov/files/documents/Newsroom/Testimonies/SSCI Unclassified SFR- Final.pdf

[10] Paliwal, S., & Gupta, R. (2012). Denial-of-service, probing & remote to user (R2L) attack detection using genetic algorithm. *International Journal of Computer Applications*, *60*(19), 57-62.

[11] Lyon, G. F. (2008). Nmap network scanning: The official Nmap project guide to network discovery and security scanning. Insecure. Com LLC (US).

[12] 2020 State of Malware Report. (2020). Malwarebytes Lab. https://resources.malwarebytes.com/files/2020/02/2020_State-of-Malware-Report-1.pdf

[13] Li, J., Zhao, B., & Zhang, C. (2018). Fuzzing: a survey. Cybersecurity, 1(1), 1-13.

[14] Murphy, K. P. (2012). Machine learning: a probabilistic perspective. MIT press.

[15] Wu, G., Shen, D., & Sabuncu, M. (2016). Machine Learning and Medical Imaging (The MICCAI Society book Series) (1st ed.). Academic Press.

[16] Gupta, M. R., Bengio, S., & Weston, J. (2014). Training highly multiclass classifiers. The Journal of Machine Learning Research, 15(1), 1461-1492.

[17] Grosan, C., & Abraham, A. (2011). Intelligent systems. Berlin: Springer.

[18] Jurafsky, D., & Martin, J. H. (2019). Speech and Language Processing. Stanford. https://web.stanford.edu/~jurafsky/slp3/.

[19] LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. nature, 521(7553), 436-444.

[20] Bengio, Y., Courville, A., & Vincent, P. (2013). Representation learning: A review and new perspectives. IEEE transactions on pattern analysis and machine intelligence, 35(8), 1798-1828.

[21] Hubel, D. H. (1995). Eye, brain, and vision. Scientific American Library/Scientific American Books.

[22] Fukushima, K. (1988). Neocognitron: A hierarchical neural network capable of visual pattern recognition. Neural networks, 1(2), 119-130.

[23] LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11), 2278-2324.

[24] Graves, A. (2012). Supervised sequence labelling. In Supervised sequence labelling with recurrent neural networks (pp. 5-13). Springer, Berlin, Heidelberg.

[25] Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. Neural computation, 9(8), 1735-1780.

[26] Abraham, T. (2001). IDDM: Intrusion detection using data mining techniques. Defence Science and Technology Organization Salisbury (Australia) Electronics and Surveillance Research.

[27] Zhang, Z., Li, J., Manikopoulos, C. N., Jorgenson, J., & Ucles, J. (2001, June). HIDE: a hierarchical network intrusion detection system using statistical preprocessing and neural network classification. In Proc. IEEE Workshop on Information Assurance and Security (Vol. 85, p. 90).

[28] Eskin, E., Arnold, A., Prerau, M., Portnoy, L., & Stolfo, S. (2002). A geometric framework for unsupervised anomaly detection. In Applications of data mining in computer security (pp. 77-101). Springer, Boston, MA.

[29] Hu, W., Hu, W., & Maybank, S. (2008). Adaboost-based algorithm for network intrusion detection. IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics), 38(2), 577-583.

[30] Zhang, J., Zulkernine, M., & Haque, A. (2008). Random-forests-based network intrusion detection systems. IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews), 38(5), 649-659.

[31] Chandrasekhar, A. M., & Raghuveer, K. (2013, January). Intrusion detection technique by using k-means, fuzzy neural network and SVM classifiers. In 2013 International Conference on Computer Communication and Informatics (pp. 1-7). IEEE.

[32] Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. Advances in neural information processing systems, 25, 1097-1105.

[33] Gao, N., Gao, L., Gao, Q., & Wang, H. (2014, November). An intrusion detection model based on deep belief networks. In 2014 Second International Conference on Advanced Cloud and Big Data (pp. 247-252). IEEE.

[34] Moustafa, N., & Slay, J. (2015, November). UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set). In 2015 military communications and information systems conference (MilCIS) (pp. 1-6). IEEE.

[35] Moustafa, N., Turnbull, B., & Choo, K. K. R. (2018). An ensemble intrusion detection technique based on proposed statistical flow features for protecting network traffic of internet of things. IEEE Internet of Things Journal, 6(3), 4815-4830.

[36] Ahmim, A., Maglaras, L., Ferrag, M. A., Derdour, M., & Janicke, H. (2019, May). A novel hierarchical intrusion detection system based on decision tree and rules-based models. In 2019 15th International Conference on Distributed Computing in Sensor Systems (DCOSS) (pp. 228-233). IEEE.

[37] Xiao, Y., Xing, C., Zhang, T., & Zhao, Z. (2019). An intrusion detection model based on feature reduction and convolutional neural networks. IEEE Access, 7, 42210-42219.

[38] Vinayakumar, R., Alazab, M., Soman, K. P., Poornachandran, P., Al-Nemrat, A., & Venkatraman, S. (2019). Deep learning approach for intelligent intrusion detection system. IEEE Access, 7, 41525-41550.

[39] Riyaz, B., & Ganapathy, S. (2020). A deep learning approach for effective intrusion detection in wireless networks using CNN. Soft Computing, 24, 17265-17278.

[40] Injadat, M., Moubayed, A., Nassif, A. B., & Shami, A. (2020). Multi-stage optimized machine learning framework for network intrusion detection. IEEE Transactions on Network and Service Management.

[41] Deng, L., & Platt, J. C. (2014). Ensemble deep learning for speech recognition. In Fifteenth annual conference of the international speech communication association.

[42] Yang, L., Hanneke, S., & Carbonell, J. (2013). A theory of transfer learning with applications to active learning. Machine learning, 90(2), 161-189.

[43] Tan, C., Sun, F., Kong, T., Zhang, W., Yang, C., & Liu, C. (2018, October). A survey on deep transfer learning. In International conference on artificial neural networks (pp. 270-279). Springer.