1-28-2021 11:30 AM

Building Effective Network Security Frameworks using Deep Transfer Learning Techniques

Harsh Dhillon, *The University of Western Ontario*

Supervisor: Haque, Anwar, *The University of Western Ontario*

A thesis submitted in partial fulfillment of the requirements for the Master of Science degree in Computer Science

Recommended Citation

Dhillon, Harsh, "Building Effective Network Security Frameworks using Deep Transfer Learning Techniques" (2021). *Electronic Thesis and Dissertation Repository*. 7658.
https://ir.lib.uwo.ca/etd/7658

Abstract

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 cybercriminals 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 thesis 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 paper, we will study various machine learning and deep learning concepts as well as techniques. Combining the strengths of the presented models for their latent feature extraction, memory retention, and classification abilities, 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

Network Intrusion Detection System, Artificial Neural Network, Convolutional Neural Network, Long Short-Term Memory, Transfer Learning.

Summary for Lay Audience

With the rise in network connectivity worldwide, we use network systems in all spheres of our society. The confidential data libraries of many businesses and government organizations are now stored on the network systems. Such data is prone to be stolen or destroyed by cybercriminals. The cyberattack activity has witnessed a rise with the mass adoption of communication networks globally. In such scenarios, the classic intrusion detection systems are not practical due to increased data traffic and speed as intrusion attempts may bypass the systems undetected. The fields of neural networks and deep learning have matured rapidly over the past decade. Neural networks are very efficient in recognizing and extracting patterns from a large dataset. Once we train a model to decipher various patterns and features, they become nominally fast in identifying and classifying the new data they encounter. Such recognition systems' efficiency and speed can also be increased using various novel methods and techniques during the developmental phase.

This thesis uses machine learning and deep learning techniques to build a novel and efficient network intrusion detection system, which can classify a malicious network activity from regular network activity. The proposed approach is much accurate and faster and can easily be integrated into modern network infrastructures to classify cyberattacks in real-time compared to classic intrusion detection systems.

Acknowledgments

First and foremost, I would like to thank and share my gratitude towards my supervisor, Dr. Anwar Haque, who gave me this opportunity to conduct research in the exciting field of network security and deep learning. I am thankful to Dr. Haque for his steady support, supervision, and inspiration throughout my master’s program at western university.

I would also like to thank my parents and my sisters who supported me during the uncertain time of the current pandemic and enabled me to keep working on my thesis.

I also want to thank all my professors who guided me throughout my coursework and during my appointment as a teaching assistant. Each course I enrolled in and assisted during the duration of my studies supplemented my research work in one way or another.

Lastly, I would like to kindly acknowledge the department of computer science, Western University, for giving me the opportunity to become part of its reputed institute and further develop my mind and skillsets for my future endeavors.

Table of Contents

List of Figures

List of Tables

Chapter 1

1 Introduction

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].

1.1 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.

Chapter 2

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. Section 2.1 will review the field of early intrusion detection systems and present the taxonomy of various intrusion detection methods. Section 2.2 will present the brief on cyberattack activity and their various types, which the IDS aims to counter. Section 2.3 will review the field of machine learning and its key concepts. Section 2.4 will cover the concept of deep learning, and vital architectures used to develop the novel network-based intrusion detection system deliberated in this thesis.

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

Intrusion detection research and development date back to 1980, starting with Anderson's paper [5] which introduced the principal concepts of computer threats monitoring and surveillance. The earliest sketch of a real-time intrusion detection system was proposed by Dorothy E. Denning in 1986 [6]. The system aimed to detect a wide range of security violations ranging from outside the system breaking-in attempts and inside the system abhorrent patterns and data abuse incidence. 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 augmented into IDES, abbreviated for Intrusion Detection Expert System developed by Teresa F. Lunt at SRI International in 1988 [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

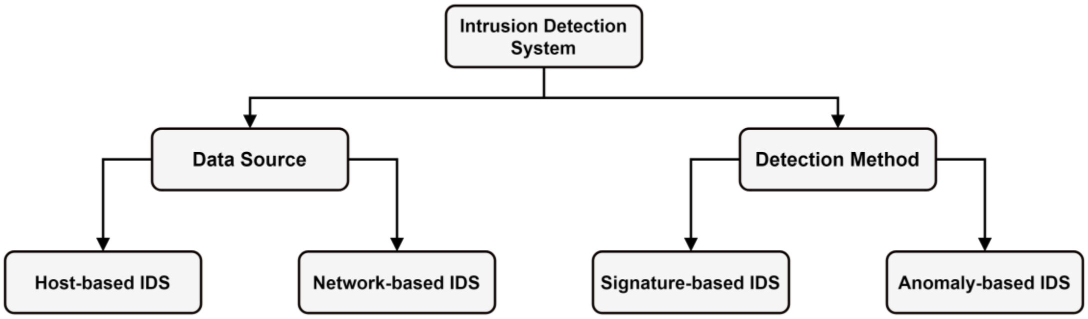


Figure 2.1: IDS Taxonomy Chart.

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.

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

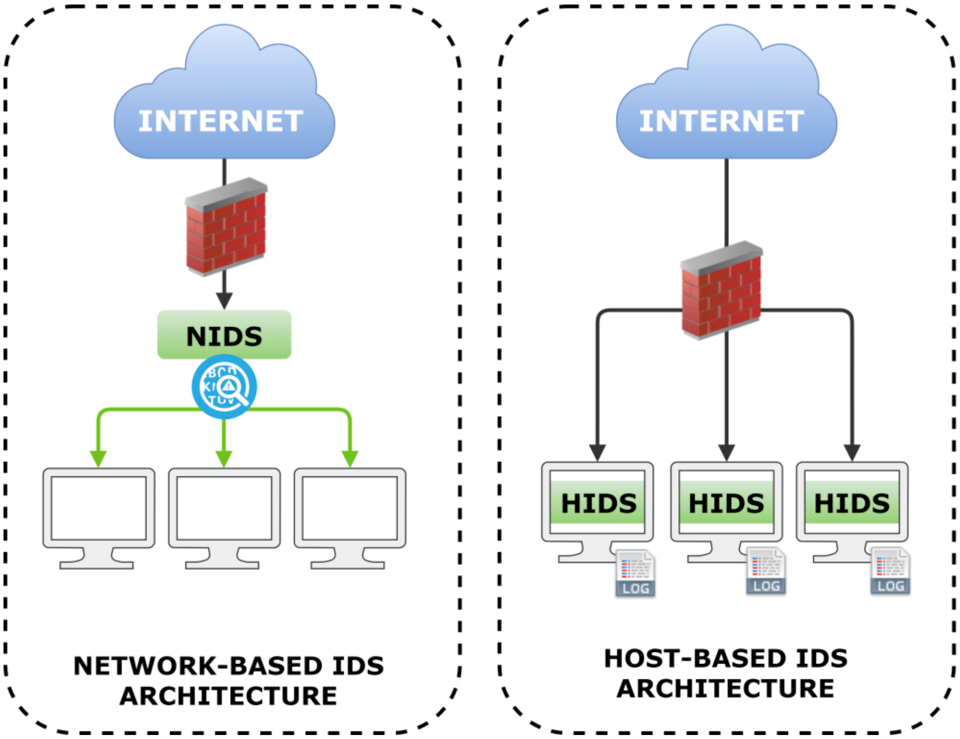


Figure 2.2: IDS Architectures.

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 present 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.

2.1.5 Detection Methods Used by IDS

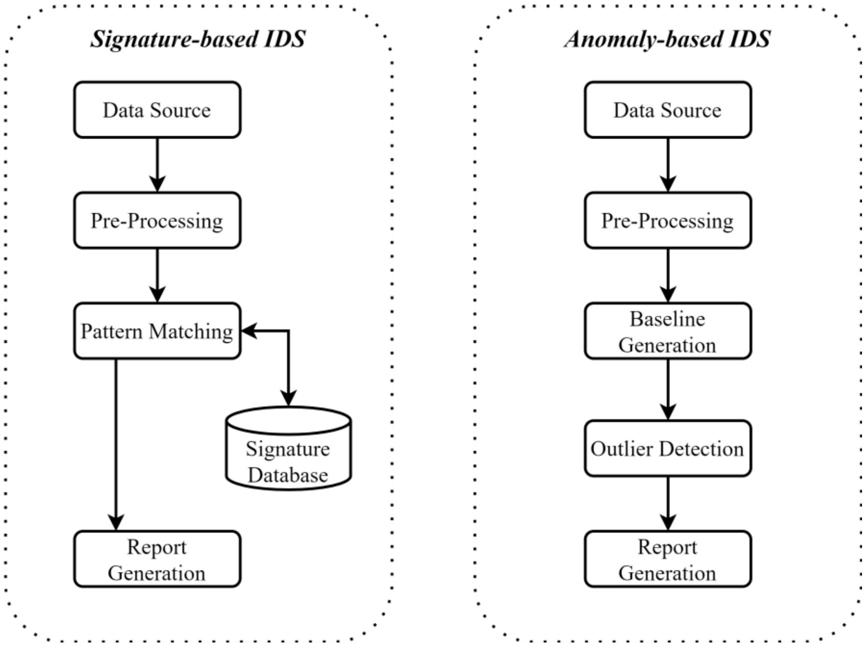


Figure 2.3: Detection Methods used by IDS.

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.

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

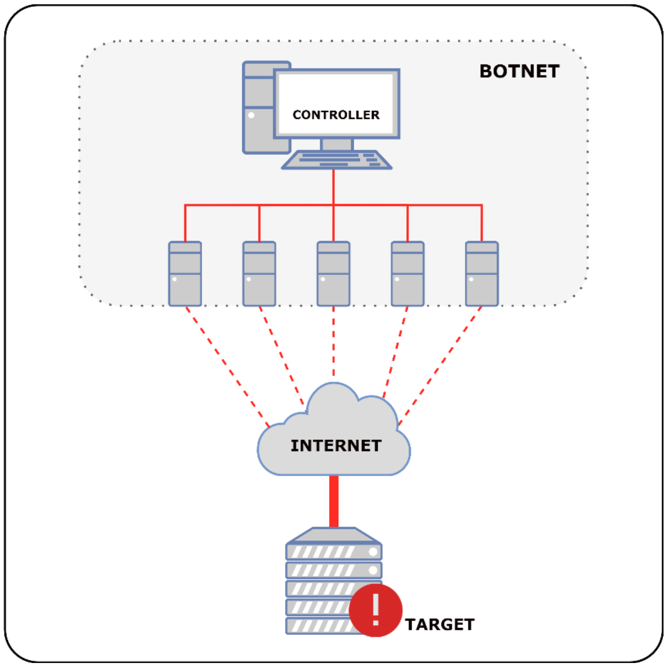


Figure 2.4: DDoS Attack.

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.

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 a field of Artificial Intelligence where we architect computer models capable of learning from a given dataset with minimal human intervention. According to Murphy [14], machine learning is a set of methods used to automatically detect patterns in data and then use the extracted patterns to predict future data or perform other kinds of decision-making tasks. A machine learning model can either be predictive if it is making forecasts for future conditions or descriptive if its objective is to gain knowledge from the given data or be both predictive and descriptive. Using the theory of statistics in building the mathematical models, machine learning algorithms' core task is to extrapolate inference from a given sample.

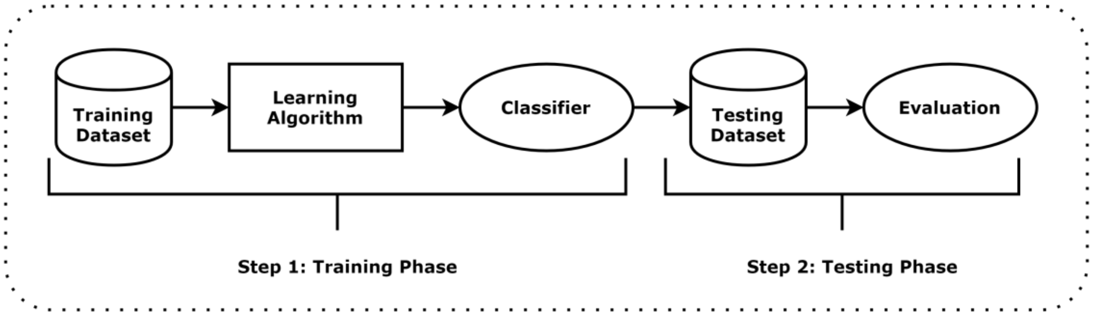


Figure 2.5: Fundamental Learning Process

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].

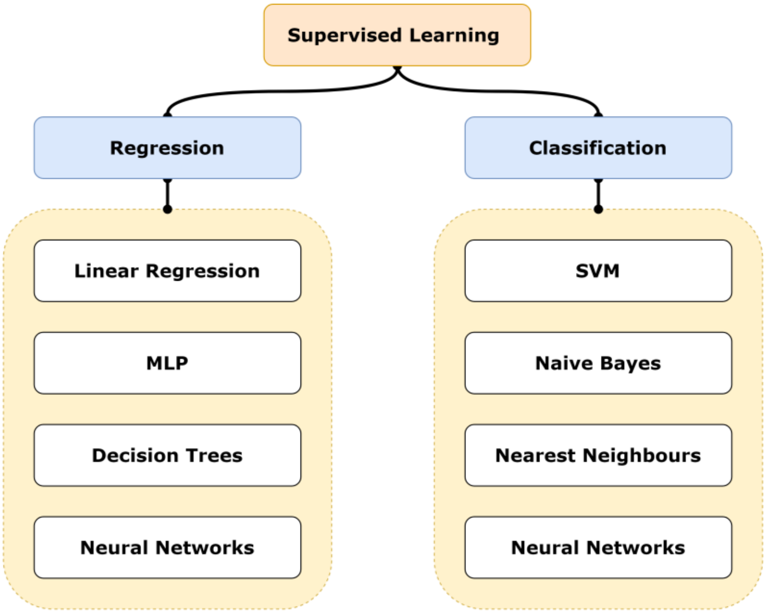


Figure 2.6: Taxonomy of Supervised Learning.

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 $\mathcal{D} = \{(x^i, y^i)\}_{i=1}^{N}$ 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 x^i is the $\mathcal{D}$-dimensional vector or numbers representing the simple features or attributes. However, x^i can also represent complex structured objects such as an image, time-series, e-mail, graphs, etc. [18]. The output y^i can also be of different forms depending on the problem.

If the value of y^i is a categorical variable from a finite set, $y^i \in \{1, \dots, C\}$, such as normal or malicious, then the problem is known as classification or pattern recognition. Similarly, when y^i 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 supervised machine learning algorithm where predicted values are within a continuous range and have a constant slope . In linear regression, each observation consists of two values. One value is for the dependent variable, and one value is for the independent variable. Further, we chart a straight line to approximate the relationship between the dependent and independent variables. Let y_i be the predicted value of the dependent variable for a given value of the independent variable x_i.

$$\mathrm{y}_i = \beta_0 + \beta_1 x_i + \varepsilon$$

Here, β_0 represents the y-intercept of the regression line and β_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 β_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 slopes upward, we infer that there is a positive relationship between the two variables where the lower end of the line at the y-intercept and the upper end of the line extends upward into the graph field. This basically means that when the value of one variable increases, another variable's value also increases in synchrony.

c. Negative relationship: When the regression line slopes downwards, we infer in this case that there is a negative relationship between the two variables where the upper end of the line at the y-intercept and the lower end of the line extend downwards into the graph field., which means that as the value of one variable increases, the value of other variable decreases.

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.

$$\text{MSE } = \frac{1}{n}\sum_{i=1}^{n}(\mathrm{y}_i - \hat{\mathrm{y}}_i)^2$$

The variable $\hat{y}_i$ is the predicted value and variable y_i 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 $[x_1, x_2, \ldots, x_n]$. 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,

$$z = \left(\sum_{i=1}^{n} w_i x_i \right) + b$$

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

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

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:

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

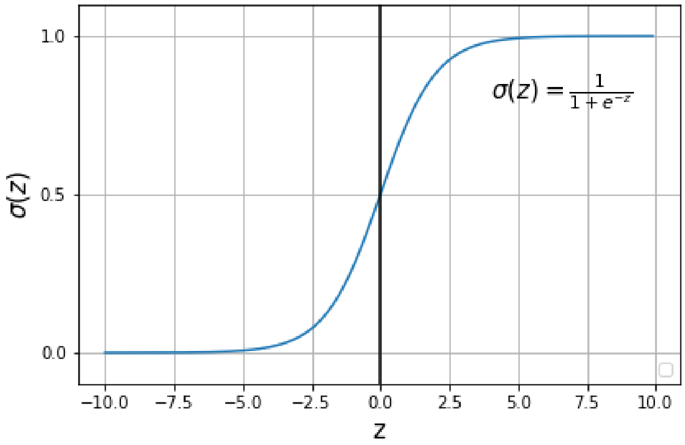


Figure 2.7: Sigmoid Function.

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

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,

$$\hat{y}\ =\ \begin{cases} 1\ \ if\ P(y=1|\ x) >\ 0.5 \\ 0\ \ otherwise \end{cases}$$

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 ŷ. The cross-entropy loss function for a binary classification can be expressed as,

$$\mathcal{L}(\hat{y}, y) =\ -\log p(y|x)$$

$$=\ -(ylog(\hat{y}) + (1-y)\log(1-\hat{y}))$$

Using the value of $\hat{y} =\ \sigma(w.x+b)$ in the equation as follows,

$$= -\left(ylog\left(\sigma(w.x+b)\right) + (1-y)\log\left(1-\sigma(w.x+b)\right)\right)$$

In tandem, cross-entropy loss function works with negative log-likelihood where when the true output y is 0, the equation reduces to $-\log(1-\hat{y})$ and when the true output of y is 1, the equation reduces to $-\,log(\hat{y})$. This ensures that correct answers are maximized, and the probability of incorrect answers is minimized. Further, we average the loss function over an entire training set of n examples, which is defined by a cost function $C(w,b)$ expressed as,

$$C(w,b) = \frac{1}{n}\sum_{i=1}^{n}\mathcal{L}\left(\hat{y}^{(i)}, y^{(i)}\right)$$

$$= -\frac{1}{n}\sum_{i=1}^{n}[(y^{(i)}log\left(\hat{y}^{(i)}\right) + \left(1 - y^{(i)}\right) + log\left(1-\hat{y}^{(i)}\right)]$$

2.3.5 Gradient Descent

We use a gradient descent algorithm to minimize the model's cost function, hence optimizing the overall prediction results. Gradient descent is an optimization technique used in machine learning and deep learning algorithms to create confident and accurate prediction models. Minimizing cost function is a convex optimization problem, and iterative algorithms such as gradient descent are used to find optimal weights [15]. The loss function $\mathcal{L}$ is parametrized by the weight parameters and bias in the case of our previous example of the logistic regression algorithm. Hence, we can refer to it as θ, where $\theta = w, b$. Gradient descent aims to find the minimum of θ, which can be referred to as,

$$\hat{\theta} = argmin\frac{1}{m}\sum_{i=1}^{m}\mathcal{L}_{CE}(y,x;\theta)$$

The way to find the minimum of the cost function is to find the direction where the slope of the function is rising too steeply and move in its opposite direction therefore the term descent. For logistic regression, the cost function is convex where there is only one local minimum, so the gradient descent is guaranteed to find the local minimum from any direction and find the minimum. In the case of a multi-layered neural network, the cost function is non-convex and gradient descent can get stuck in local minimum but fail to find the global optimum [16]. In Figure 2.8, we plotted the downward descent of parameters induced using gradient descent for optimization.

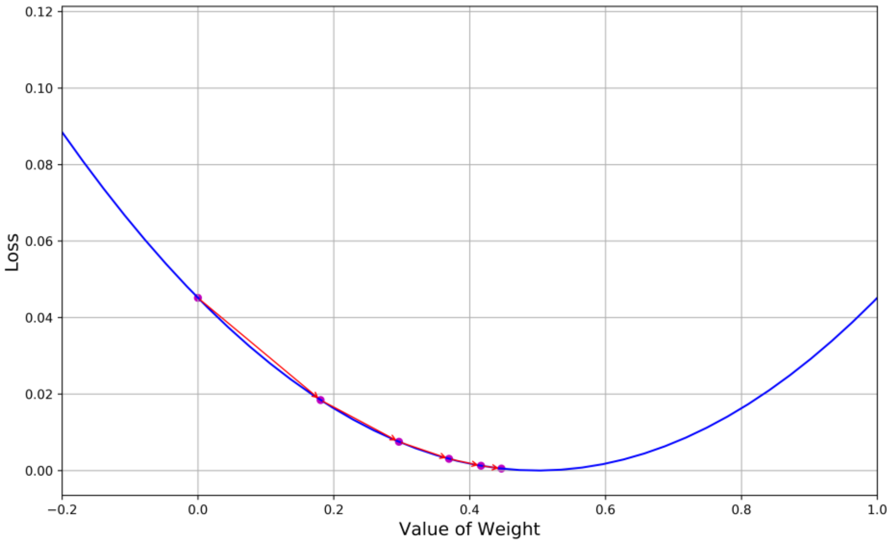


Figure 2.8: Gradient descent algorithm approaching local minimum.

The speed of descent of parameter w in a positive direction is the value of slope regulated by a learning rate η which is also called a step size. If the value of the learning rate is greater, the parameter w will move more each step, and the descent will be faster as well and vice versa. This can be summarized in the expression,

$$w^{t+1} = w^t - \eta \frac{d}{dw} f(x; w)$$

Here $\frac{d}{dw} f(x; w)$ is the slope's value, which also defines the magnitude of the amount to move w per step in gradient descent, multiplied by the learning rate η for regulation. Learning rate is one of the hyperparameters that need to be tuned accordingly. Making the learning rate faster can make the descent become haphazard and lead to erroneous predictive outputs as it may miss the minimum of the function by overshooting. In contrast, if the learning rate is too slow, it will take a long time to get to the minimum.

Gradient descent can be distinguished based on the amount of training data being used for the algorithm. We call the method batch gradient descent if we use all the training data for the algorithm to compute the gradient. In contrast, if we use a subset of training batches smaller than the entire training dataset and process each batch size to compute the gradient, the method is called as mini-batch gradient descent. Stochastic gradient descent is an online algorithm where we minimize the loss function by computing its gradient after each training example.

2.3.6 Neural Networks

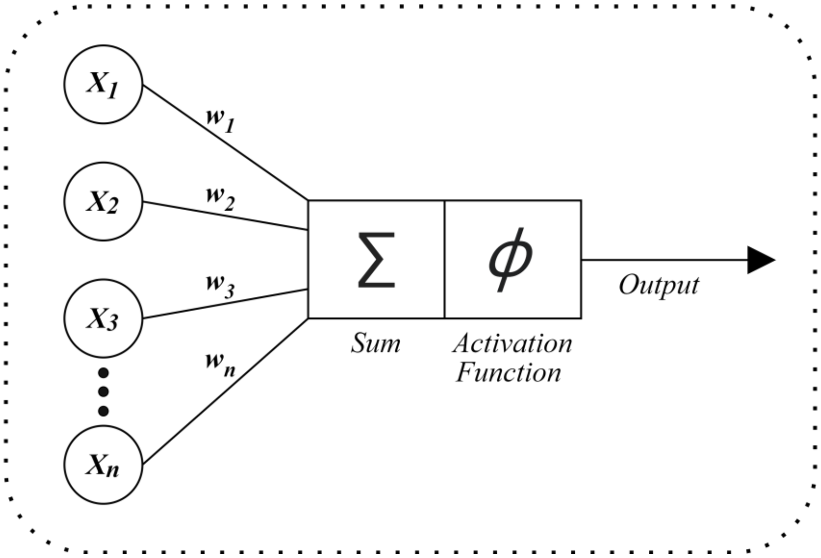


Figure 2.9: Perceptron Architecture.

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.

2.3.7 Multi-Layered Perceptron

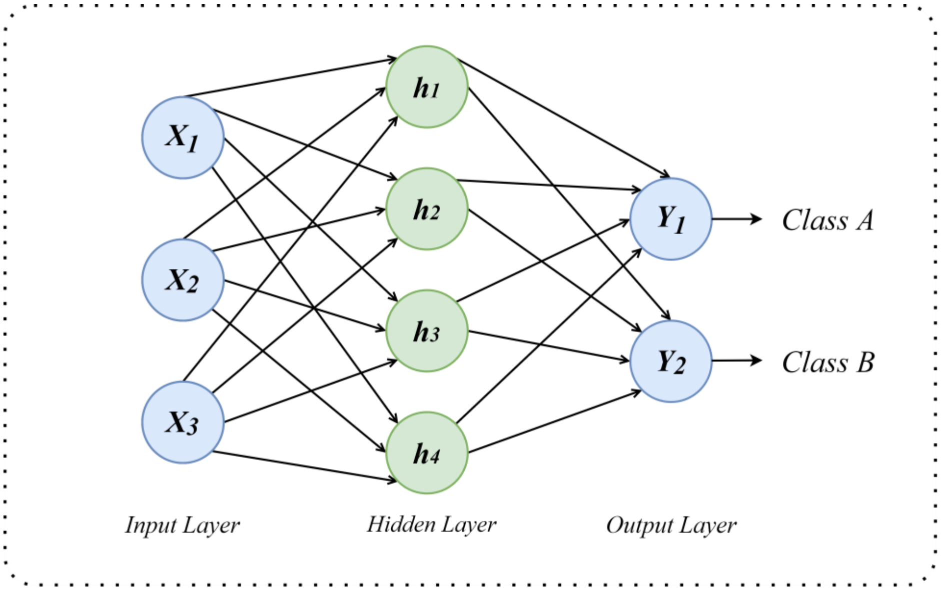


Figure 2.10: Multi-Layered Perceptron Architecture.

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 x_n 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,

$$h = \sigma(Wx + b)$$

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 w_{ij} 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,

$$z = U \times h$$

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,

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

A softmax function converts the logits, which is basically another term for the numerical output of the last linear layer of a multi-class neural network, into probabilities by taking exponents of each output and normalizing it by the sum of all the 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.

2.3.8 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,

$$\delta = z - y$$

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 $\frac{\partial L}{\partial a}$ gives how much the 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,

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

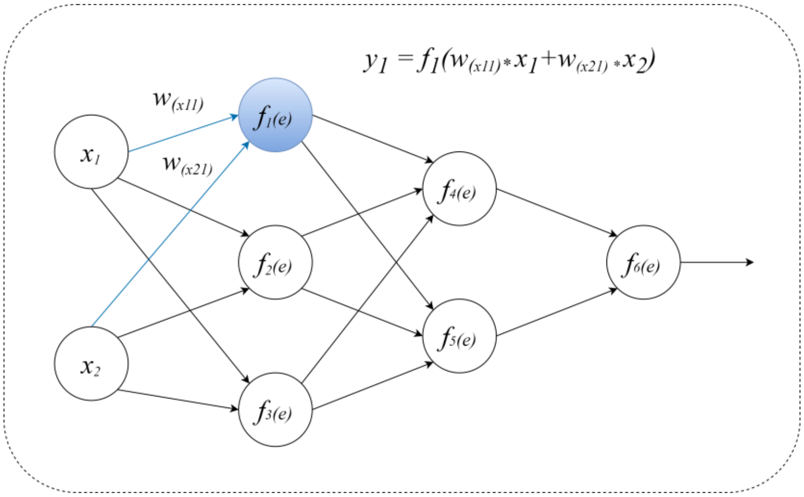


Figure 2.11: Computation of hidden layer units in neural network.

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 *f*₁*(e)* in the hidden layer, which uses the connection weights $w(x_{11})$ and $w(x_{21})$ between input units x_1 and x_2 where *e* = $w(x_{11}) * x_1 + w(x_{21}) * x_2$. The output of function unit *f*₁*(e)* then further becomes the input for computing function units *f*₄*(e)* and *f*₅*(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 δ where,

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

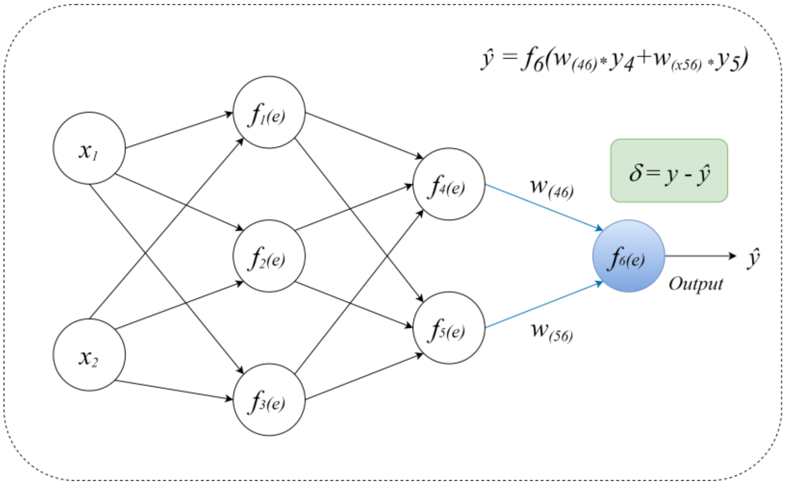


Figure 2.12: Computation of the error signal to propagate backwards.

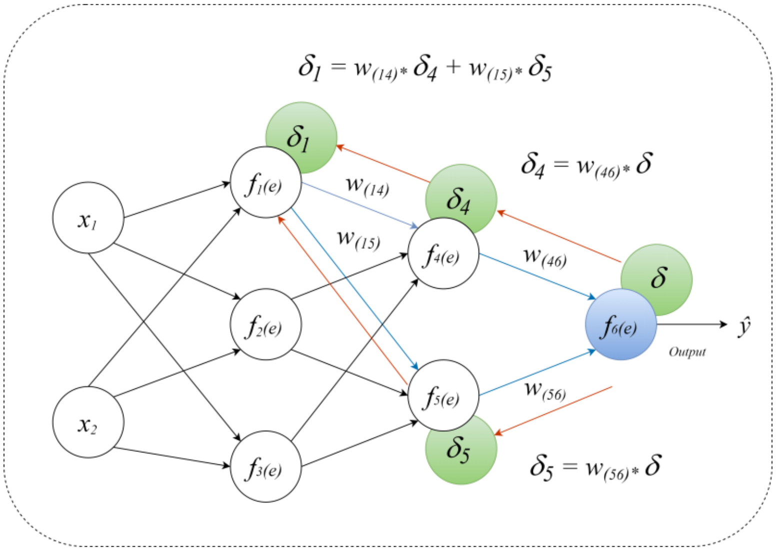


Figure 2.13: Backward propagation of the error signal in the neural network.

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,

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

Where $w'(x_{11})$ is the updated weight for connection between the input unit x_1 and hidden layer unit $f_1(e)$, coefficient η represents the learning speed, δ_1 represents the error signal computed for the unit, the equation $\frac{df_1(e)}{de}$ represents the derivative of the neuron activation of the hidden unit $f_1(e)$ 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.

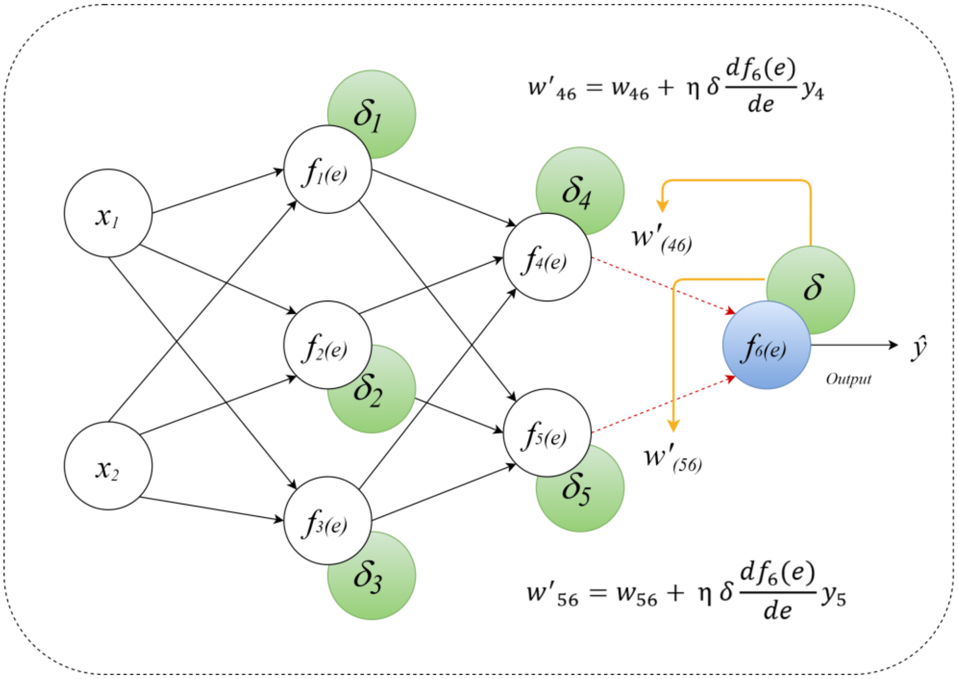


Figure 2.14: Weight updation using Backpropagation Algorithm.

Figure 2.14 shows updating of weights by backprop in the neural network until the final output unit $f_6(e)$ 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

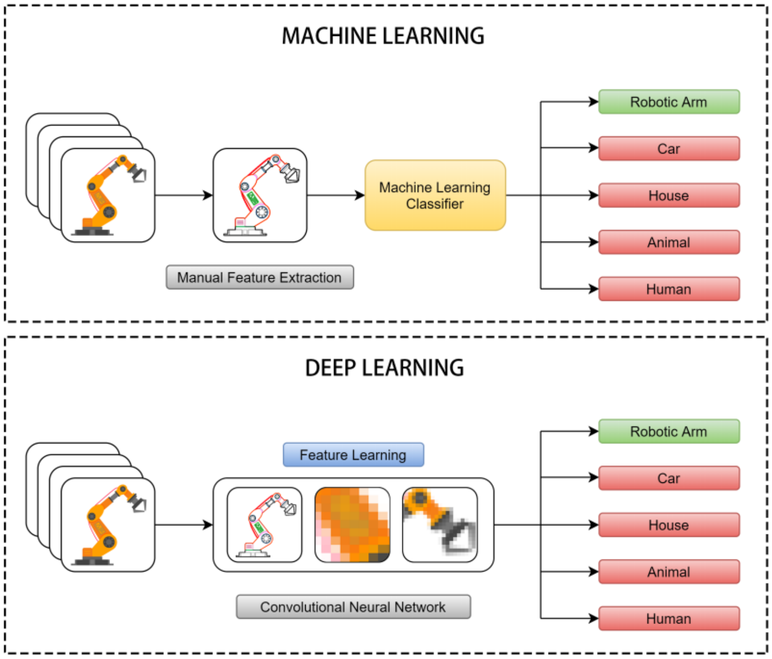


Figure 2.15 : Comparison between machine learning and deep learning classification.

This section will briefly discuss the deep learning architectures that are significant to the IDS architecture proposed in this thesis. Deep learning allows computational models composed of multiple processing layers to learn representations in the data with multiple abstraction levels [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.

2.4.1 Convolutional Neural Networks

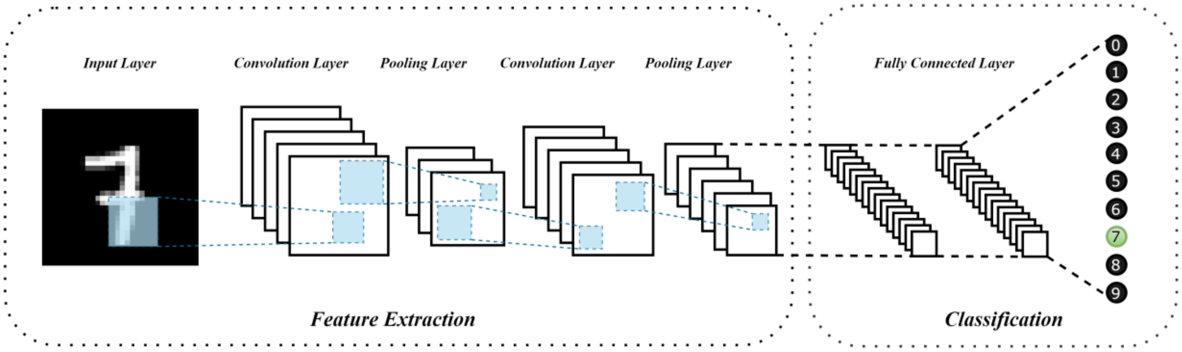


Figure 2.16: CNN Architecture

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 Neocognitron architecture, with the exception of backpropagation being the primary mode for being the learning algorithm. 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 2.16, 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 2.17, 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.

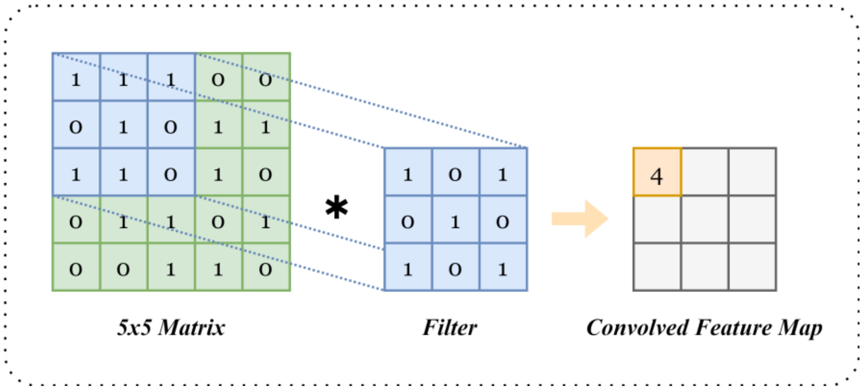


Figure 2.17: Feature Map computation by Convolution Layer

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 2.18 demonstrates the subsampling of the feature map using max pooling.

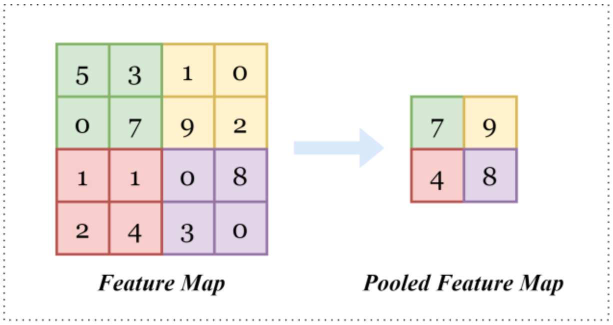


Figure 2.18: Max Pooling operation with 2x2 Filter and Stride value 2.

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 2.19. 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

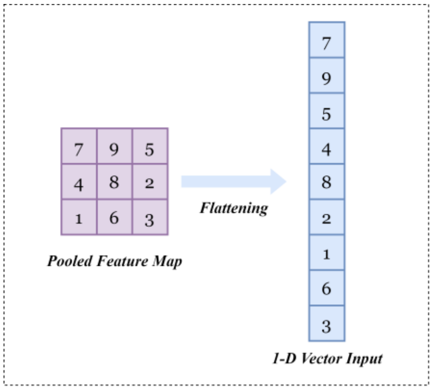


Figure 2.19: Flatten operation converting feature matrix into 1-D vector input.

function such as a sigmoid or softmax activation function to generate the class label predictions, which is also the CNN's final output.

2.4.2 Recurrent Neural Network

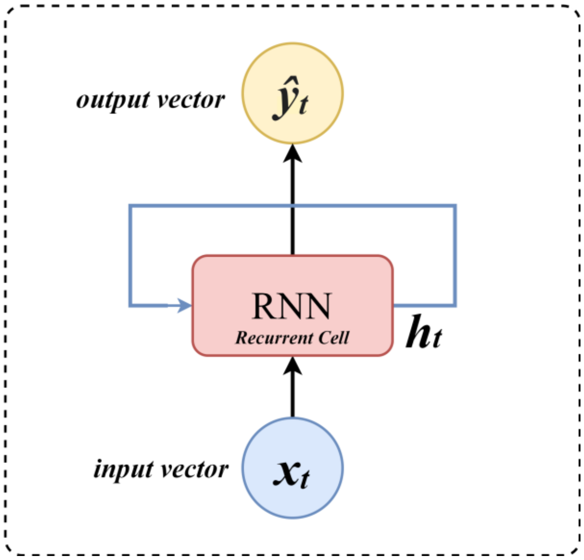


Figure 2.20: A simple RNN cell Architecture.

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. In contrast, Recurrent Neural Network abbreviated as RNN are a type of deep learning architecture that, in addition to feedforward connections, also has looping feedback connections that allow the model to store persistent information over time [24].

As shown in Figure 2.20, an RNN takes input x_t 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 h_t, which it passes from one-time step to another internally within the network where,

$$h_t = f_w(h_{t-1}, x_t)$$

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

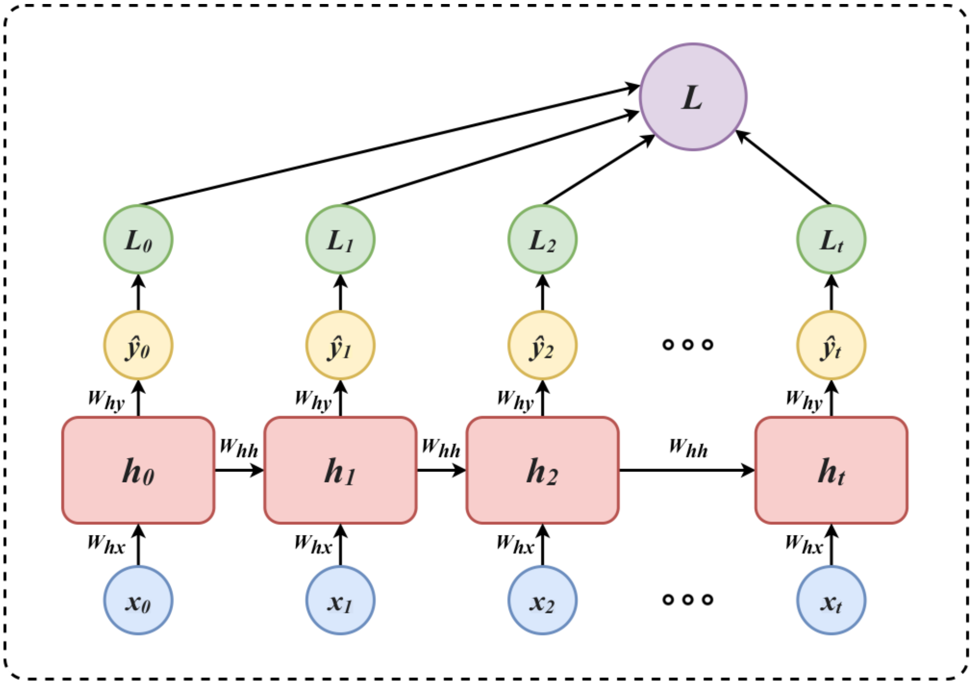


Figure 2.21: Unrolled RNN Architecture.

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 2.21, 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,

$$h_t = tanh(W_{hh}^T h_{t-1} + W_{hx}^T x_t)$$

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 x_t, we use two weight matrices represented by W_{hh}^T and W_{hx}^T as shown in Figure 2.21. Now the output vector for each timestamp is expressed as,

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

Where h_t represents the computed hidden state and W_{hy}^T 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 2.22.

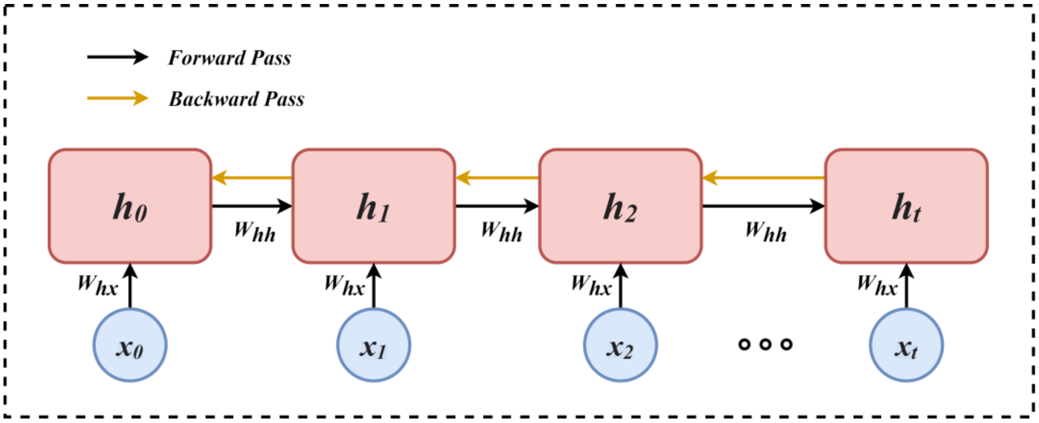


Figure 2.22: RNN Gradient Flow.

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

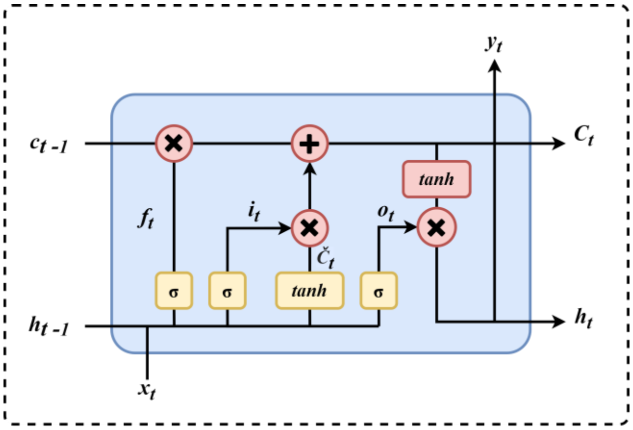


Figure 2.23: Structure of a LSTM unit.

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 2.23, an LSTM unit is made of three key gate components briefly described below,

A. Forget Gate: This gate determines what information is to be thrown away from the cell state. This decision is made by the sigmoid layer, which looks at the values of h_{t-1} and x_t to output a number between 0 and 1 for the cell state C_{t-1}. The output represents the degree to which information is to be kept. A value of 1 represents keep everything, whereas the value of 0 represents completely forget this information. This gate can be expressed by,

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$

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 i_t decides which values we will be updating. The next layer $tanh$ creates a vector of new candidate $\hat{C}_t$ which will be added to the new state. These steps can be expressed as follows,

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

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

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

$$C_t = f_t * C_{t-1} + i_t * \hat{C}_t$$

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,

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$$

$$h_t = o_t * tanh(C_t)$$

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.

Chapter 3

3 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.

3.1 Statistical based Approach

Dorothy E. Denning proposed the earliest sketch of a real-time intrusion detection system in 1986, which aimed to create a general-purpose architecture, independent from any particular system, application environment, or type of intrusion [5]. Her work took inspiration from a prior study of Jim Anderson in 1980, which formulated a way to audit a computer's data to identify abnormal usage patterns at the end of each day. Anderson's method primarily used a statistical analysis approach using large dump files consolidated from all the infrastructure machines [6]. This research further augmented into IDES, abbreviated for Intrusion Detection Expert System developed by Teresa F. Lunt at SRI International in 1988 [7]. IDES had two main components. The first component adaptively learns the user's normal behavior pattern and detects patterns that deviate from them. The second component uses a rule-based approach to encode the encountered system vulnerabilities and store them in a knowledge base. Lunt proposed integrating an artificial neural network in the expert system as a third component, which was not fully implemented in IDES' follow-up derivations. By the 1990s, intrusion detection systems were started to get implemented by various research labs and business computing firms, including AT&T Bell Labs, who built their own versions of detection systems, using IDES as a base on multiple other hardware and different programming languages. The introduction of a well-labeled KDD-99 intrusion detection dataset enabled researchers to work in computer security to apply data mining and machine learning algorithms to build many efficient and generalized IDS.

3.2 Data Mining, Machine Learning based Approach

In 2001, Tamas Abraham used data mining techniques to formulate the IDDM, abbreviated for Intrusion Detection Using Data Mining architecture [26]. Traditionally, data mining systems operated on large off-line data sets. IDDM architecture was designed to use data mining in real-time environments to identify anomalies and misuse. 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. Z. Zhang et al. proposed a hierarchical network intrusion detection system named HIDE, which used a Perceptron-Backpropagation hybrid model to classify anomalous and normal network traffic for recognizing UDP flood attacks [27]. The architecture of HIDE was divided into various tiers where each tier contains Intrusion Detection Agents, which are the components that monitor the activities of hosts and networks as well as multiple units that make up those infrastructures. 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.

In 2002, Eskin et al. proposed an unsupervised intrusion detection framework using SVM, K-Nearest Neighbor, and clustering algorithms [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. Weiming Hu et al. used an Adaboost-based algorithm with an adaptive weight strategy to build a detection model reporting low computational complexity and error rates [29]. J. Zhang et al. used random forest algorithm-based data mining techniques to build a hybrid IDS, which is capable of functioning as both a misuse and anomaly 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. applied k-means clustering, fuzzy neural networks, and radial support vector machine consequently to build their variation of IDS [31], which claimed better experimental results than the Backpropagation Neural Networks and other well-known machine learning methods. 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.

3.3 Deep Learning based Approach

The 2012 ImageNet victory led by Hinton et al. demonstrated that deep neural networks could outperform complex machine learning models in image recognition tasks [32]. The neural network was able to beat the state-of-the-art algorithms by a whopping 10.8 error percentage margin rate and creating 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 proceeding years, academics working in computer security also started integrating deep neural networks in their research.

In 2014, N. Gao et al. applied Deep Belief Networks (DBN), a class of DNN, which reported the lowest published false-positive results with the KDD-99 dataset [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. N. Moustafa et al. [34] reinvigorated the field by creating the UNSW-NB15 network dataset, which contains hybrid records of real normal and contemporary synthesized network attack activities. The UNSW-NB15 network dataset is more superior for evaluating NIDS performance as it reflects current traffic scenarios more fittingly than decade-old intrusion datasets such as KDD-99 and NSLKDD.

In 2018, N. Moustafa et al. used the UNSW-NB15 dataset to create NIDS for IoT traffic data for classifying normal and suspicious instances by applying AdaBoost ensemble techniques [35]. The applied AdaBoost ensemble consists of three techniques, namely Decision Trees, Naïve Bayes, and Artificial Neural Network. The framework focused on MQTT, DNS, and HTTP protocols and their flow identifiers to build the NIDS specific to detecting exploits in IoT networks. A. Ahim et al. [36] combined three different classifier approaches based on decision trees and various rules-based concepts to build a novel IDS using the 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] implemented a CNN-based IDS using Batch Normalization with KDD99 Dataset. The proposed framework also removed unused and redundant features using an auto-encoder (AE) network as a dimensionality reductionality technique. Vinayakumar et al. [38] created a hybrid IDS to monitor network and host level activities. Upon conducting an exhaustive comparative study with various machine learning and deep learning classifiers, DNN demonstrated to outperform other traditional machine learning classifiers. B. Riyaz et al. [39] designed an IDS for application in wireless networks with a CNN architecture using the KDD-99 dataset. The framework utilized a novel coefficient-based feature selection algorithm (CRF-LCFS), which enhanced the model’s performance in terms of detection accuracy and computation times. 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%.

In this thesis, we proposed a novel network intrusion detection system based on a unified CNN-LSTM model. To augment the applied model's classification accuracy and speed in real-world environments, we used transfer learning techniques where we transferred the domain knowledge learned by our model in a source domain to a target domain. The target domain is aimed at simulating a resource sparse real-world environment with moderately less amount of data and computational 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.

Chapter 4

4 Proposed Model and Methodology

This chapter will discuss the proposed model and techniques applied to build for our novel IDS architecture. Section 4.1 will describe the unified deep learning architecture illustrated in this thesis. Section 4.2 will discuss the transfer learning methodology used to make our applied model perform with optimal accuracy and speed in a real-world environment. Section 4.3 will explore the system architecture and data pipeline of the proposed novel methodology. Section 4.4 will explore the development environment for our research. In Section 4.5, we will examine the UNSW-15 Dataset as well as various data-preprocessing techniques applied. Section 4.6 will discuss the evaluation principles we will be using to judge our candidate IDS model.

4.1 Unified Deep Learning Architecture

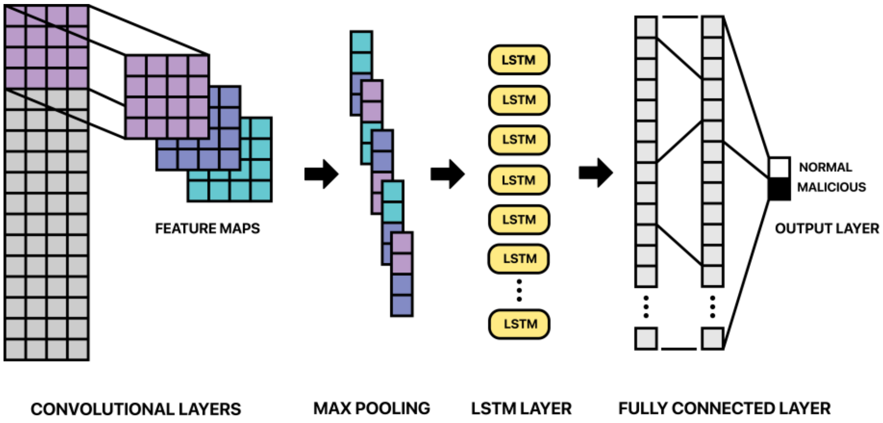


Figure 4.1: Unified IDS Learning Model

In this thesis, our chosen deep learning architecture for the IDS consists of a CNN with LSTM present in its hidden layers and fully connected layer units to predict the classification labels. As shown in Fig. 4.1, the proposed unified IDS model uses a modular approach of combining the three distinguished deep learning models' architecture and combines their latent feature extraction, memory retention, and classification abilities to give a higher accuracy score as compared to the models applied separately.

A CNN can learn and recognize patterns over an input space, whereas LSTM units can learn and recognize patterns across time. A DNN or a fully connected layer, on the other hand, is capable of learning mappings from an input vector to give precise class wise outputs. Both CNN and DNN belong to the feedforward networks class where data can only flow in the forward direction. CNN can use a 2D input and transform it into internal vector representations to further extract its features. In contrast, when we apply LSTM with CNN, LSTM provides the capability of using the feature vector output of the CNN and further build internal states whose weights can repeatedly be updated because data in LSTM flows in a recurrent manner. During this entire process, the CNN extracts the inherent features from the input. In contrast, LSTM interprets those features across various time steps, making the architecture more efficient to learn more in-depth representations and relationships in the data, in contrast with any network architecture applied separately.

Combining DNN, CNN and LSTM have been explored in the past in [41], where the models are being trained separately, and then their outputs are later combined. In our approach, we are training the unified model jointly with each model providing their processed feature outputs as an input to the subsequent models in the scheme.

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 I shows the summary of our candidate CNN-LSTM model, where we are first using CNN layers to extract the contextual features in the training set. The utility of CNN’s to downsample the input while conserving the essential features during the extraction process reduces the feature parameters' overall dimension. The output of CNN is then fed into the

LSTM layers to model the signal in time and train the weights using the backpropagation in time (BPTT) algorithm. Finally, after the signal is modeled in the LSTM layers, the output is passed into fully connected layers, which are used to learn higher-order feature representations suitable for separating the output into different class labels.

Table 4.1

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

4.2 Transfer learning

Transfer learning is a concept where a learning algorithm reuses the knowledge from the past related tasks to ease the process of learning to perform a new task [42]. The ability to transfer the knowledge gained from previous tasks has a wide range of real-world applications, including building real-time intrusion detection systems that can perform optimally even with scarcity of data and computing resources. Using deep transfer learning alleviates the massive data dependency of deep learning algorithms, which they require to learn the underlying patterns in the data. In general, terms, using transfer learning, we aim to transfer the knowledge from a source domain to a target domain by relaxing the assumption that the training data and the test data must be independent and identically distributed, which is rare for real-world data. Fig. 4.2 shows the process of transferring a model's network architecture and learned weights from a source domain with a large dataset and higher computational resources to a target domain with a smaller dataset and limited computational resources.

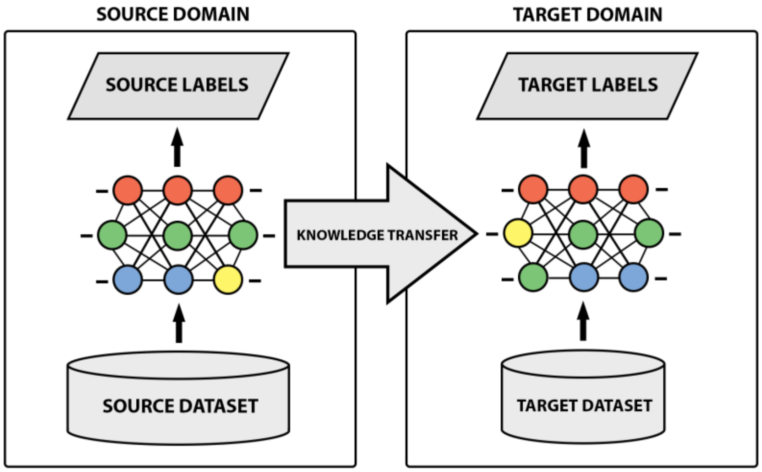


Figure 4.2: Transfer Learning Process

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 $X = \{x_1, x_2, \dots, x_n\}, x_i \in X$.

Whereas A *task* can be represented as, $T = \{Y, P(Y|X)\} = \{Y, \eta\}$, $Y = \{y_1, y_2, \dots, y_n\}, y_i \in Y$, where Y is a label space, and η represents the predictive function which can be learned from the training data including pairs $\{x_i, y_i\}$, where $x_i \in X, y_i \in Y$; for each feature vector in the domain, η predicts its corresponding label as $\eta(x_i)$ = y_i [43].

we consider our source domain as D_S, and target domain as D_T. The source domain data is denoted as D_S = $\{(x_{S_1}, y_{S_1}), \ldots, (x_{S_n}, y_{S_n})\}$, where $x_{S_i} \in X_S$ is the data instance and $y_{S_i} \in Y_S$ is the corresponding class label. In our IDS, D_S is the set of term vectors together with their associated attack and malicious labels. Similarly, we denote the target domain data as D_T = $\{(x_{T_1}, y_{T_1}), \ldots, (x_{T_n}, y_{T_n})\}$, where the input x_{T_i} is in X_T and $y_{T_i} \in Y_T$ is the corresponding output. We can now give the transfer learning definitions as follows,

Given a source domain D_S, learning task T_S, a target domain D_T and learning task T_T, transfer learning aims to help improve the learning of the target predictive function η_t by using the knowledge in the source domain D_S and learning task T_S, where $D_T \neq D_S$, or $T_S \neq T_T$. The size of D_S is much bigger than D_T 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 paper, the two domains are related as they share a similar feature space from intrusion datasets. A transfer learning task defined by $(D_S, T_S, D_T, T_T, \eta_t)$ becomes a deep transfer learning task if η_t is a non-linear function represented by a deep neural network.

Chuanqi Tan et al. [43] classified the deep transfer learning approach into four main categories, namely instance-based, mapping-based, network-based, and adversarial-based transfer learning. In this paper, we utilize the network-based transfer learning approach. Network transfer learning refers to the transfer of a partial network trained in the source domain, which includes its network structure and learned weights to the target domain, where it becomes part of its existing architecture. The network-based transfer learning architecture works with the notion that neural networks should become as iterative as human brains. Human brains use prior knowledge even when they are performing new tasks and often perform well with the new tasks by using the previously learned concepts. As discussed from a domain perspective, transfer learning can be understood as domain adaption where knowledge learned to perform a task in one setting, or distribution is utilized to improve the generalization of the task in another setting or distribution. In case of our IDS model, the task remains the same, but the input distribution becomes different with a forthcoming flow of network packet data. The main objective of transfer learning is to use the first domain setting and extract information that will be useful for making necessary predictions about the nature of new data.

There are two extreme forms of transfer learning referred to as one-shot learning and zero-shot learning, which were also studied during this thesis'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.

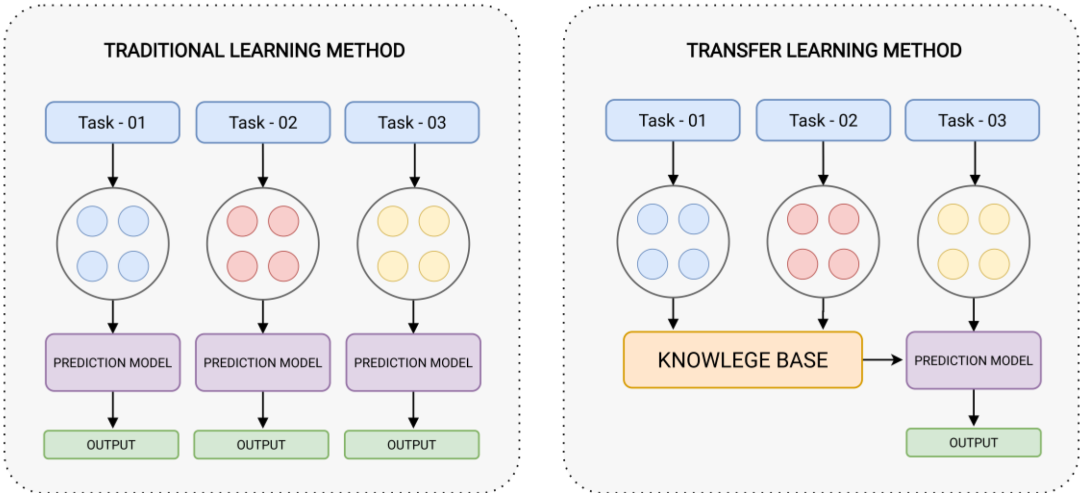


Figure 4.3: Comparison between Traditional learning and Transfer learning method.

As shown in figure 4.3, 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 thesis, 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.

4.3 System Architecture

This section will discuss the system architecture of the proposed network classification and intrusion detection system. Figure 4.4 outlines the system architecture of the end-to-end deep learning pipeline applied to network classification tasks using different subcomponents.

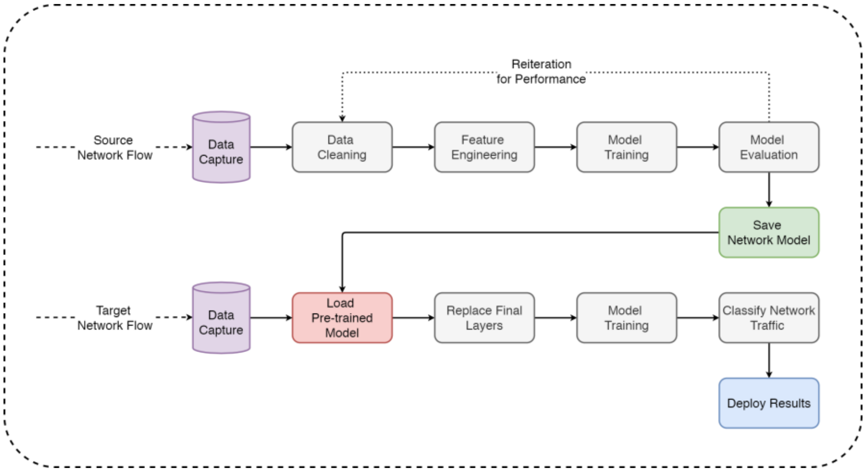


Figure 4.4: System Architecture Flowchart

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 PerfectStorm 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 4.2. 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.

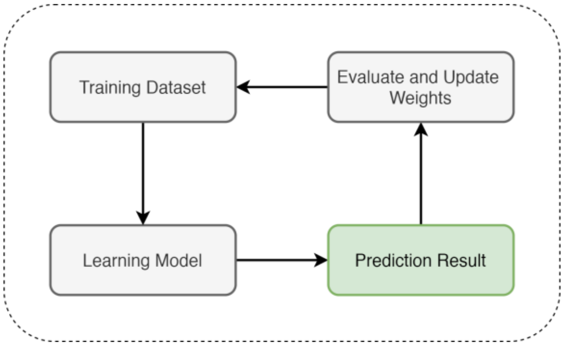


Figure 4.5 : Model Training Process

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. Section 4.6 lays out the evaluation criteria for the intrusion detection system defined in this thesis. Based on the evaluation results, 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 4.5.

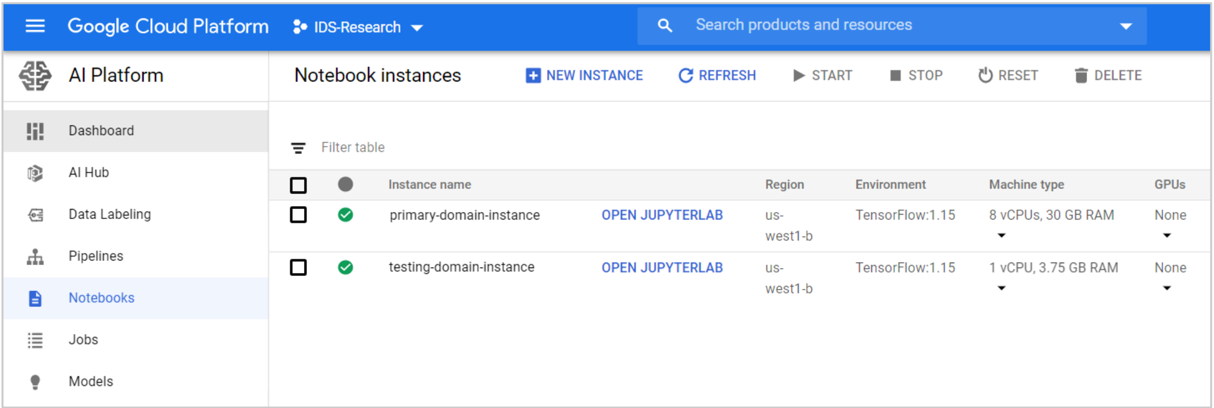
6. Transfer Learning Methodology: When the model provides satisfactory results based on the defined evaluation criteria, we will save the model and its weights in the HDF5 format designed to store large and complex data hierarchically. The model is then transferred to our target domain with an entirely 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 5.4, 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.

4.4 Development Environment

The development of IDS architecture was done using the google cloud platform. Offered by Google, the platform provides a number of services among which the most relevant to this thesis were google compute engine, which is an infrastructure as a service component for provisioning dynamic computing clusters, cloud AI platform which provides services for building and training machine learning, and deep learning models and various cloud network services such as cloud storage, DNS management, and cloud API.



□	●	Instance name		Region	Environment	Machine type	GPUs
□	✓	primary-domain-instance	OPEN JUPYTERLAB	us-west1-b	TensorFlow:1.15	8 vCPUs, 30 GB RAM	None
□	✓	testing-domain-instance	OPEN JUPYTERLAB	us-west1-b	TensorFlow:1.15	1 vCPU, 3.75 GB RAM	None

Figure 4.6: Research instance setup in Google Cloud Platform

As shown in Figure 4.6, we provisioned two separate clusters in the compute engine for our research. To experiment and build the model, we used machine type n1-standard-8, which is fitted with 8 vCPUs and a 30 GB memory. For domain-specific tests, we provisioned a cluster with machine type n1-standard-1, which comes with 1 vCPUs and a 3.75 GB memory. Both clusters used Debian GNU/Linux10 as their boot operating system. 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.

4.5 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 4.7, 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. We will use a subset of this data, which includes 257,673 records and will further divide the selected partition into a training set with 154,603 records. We will also use a validation set and a testing set, both with 51,535 records, to aptly evaluate the applied deep learning model's performance in the separate domains.

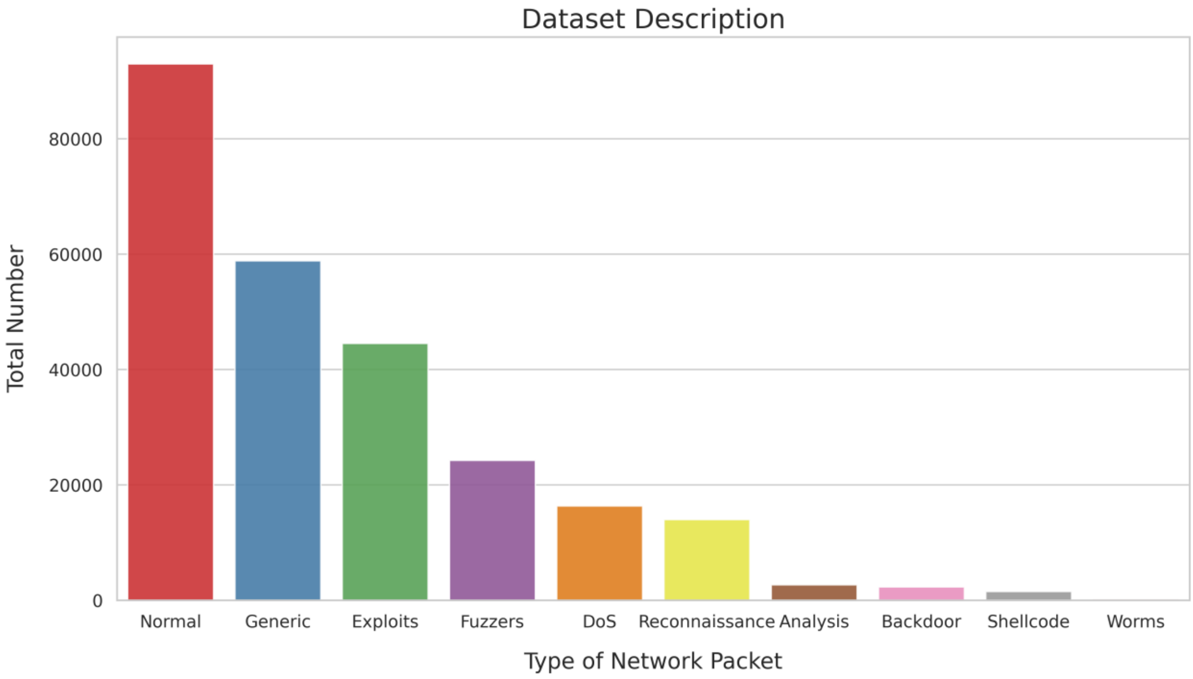


Figure 4.7: UNSW-15 Dataset Description

4.5.1 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 4.8, where each feature has a scoring value that represents how important and relevant the feature is to the output variable.

We can further drop the unnecessary feature entries from the dataset based on this computed scoring. Feature selection enables the model to allocate its computational resources appropriately, increasing the speed of training times because we are reducing down the data to process and construct the model. The presence of irrelevant and redundant data makes the ultimate goal of knowledge discovery much harder also. Table 4.2 shows few key features determined by feature selection as important well as their brief descriptions.

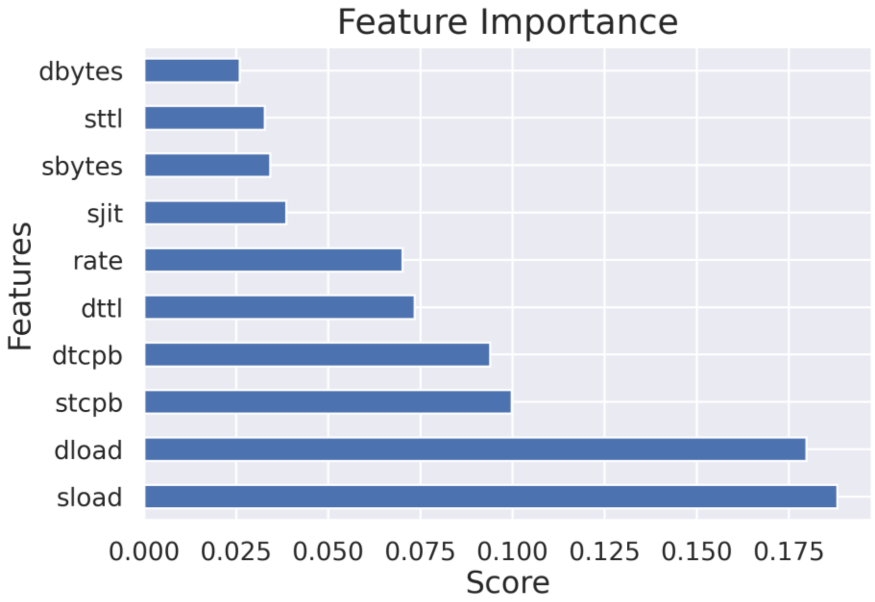


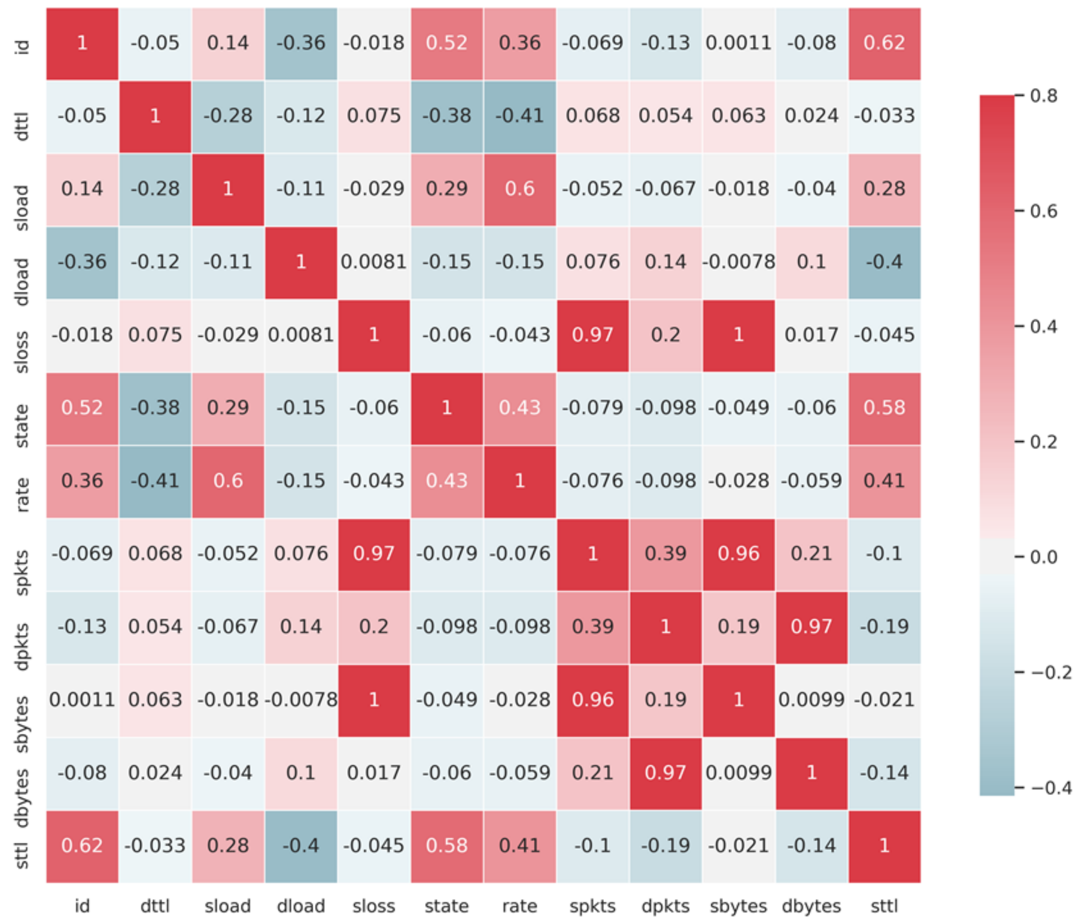
Figure 4.8: Feature Selection Plot

Table 4.2

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 4.9. 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.



	id	dttl	sload	dload	sloss	state	rate	spkts	dpkts	sbytes	dbytes	sttl
id	1	-0.05	0.14	-0.36	-0.018	0.52	0.36	-0.069	-0.13	0.0011	-0.08	0.62
dttl	-0.05	1	-0.28	-0.12	0.075	-0.38	-0.41	0.068	0.054	0.063	0.024	-0.033
sload	0.14	-0.28	1	-0.11	-0.029	0.29	0.6	-0.052	-0.067	-0.018	-0.04	0.28
dload	-0.36	-0.12	-0.11	1	0.0081	-0.15	-0.15	0.076	0.14	-0.0078	0.1	-0.4
sloss	-0.018	0.075	-0.029	0.0081	1	-0.06	-0.043	0.97	0.2	1	0.017	-0.045
state	0.52	-0.38	0.29	-0.15	-0.06	1	0.43	-0.079	-0.098	-0.049	-0.06	0.58
rate	0.36	-0.41	0.6	-0.15	-0.043	0.43	1	-0.076	-0.098	-0.028	-0.059	0.41
spkts	-0.069	0.068	-0.052	0.076	0.97	-0.079	-0.076	1	0.39	0.96	0.21	-0.1
dpkts	-0.13	0.054	-0.067	0.14	0.2	-0.098	-0.098	0.39	1	0.19	0.97	-0.19
sbytes	0.0011	0.063	-0.018	-0.0078	1	-0.049	-0.028	0.96	0.19	1	0.0099	-0.021
dbytes	-0.08	0.024	-0.04	0.1	0.017	-0.06	-0.059	0.21	0.97	0.0099	1	-0.14
sttl	0.62	-0.033	0.28	-0.4	-0.045	0.58	0.41	-0.1	-0.19	-0.021	-0.14	1

Figure 4.9: Features Correlation Heat Map

4.5.2 Data Normalization

Data normalization or feature scaling is a data preprocessing technique where we convert all input values to be used in the learning model to a standard scale. As commonly noticed, without scaling the data the features with a large range value will have a greater impact on the learning model's output. This leads to other features that may also be important but with a smaller range become less effective to the overall inferences drawn by the predictive model. To make all features equal, it is important to scale the data, which also helps the algorithm reach convergence faster, and optimizing also becomes much more comfortable using the gradient descent algorithm.

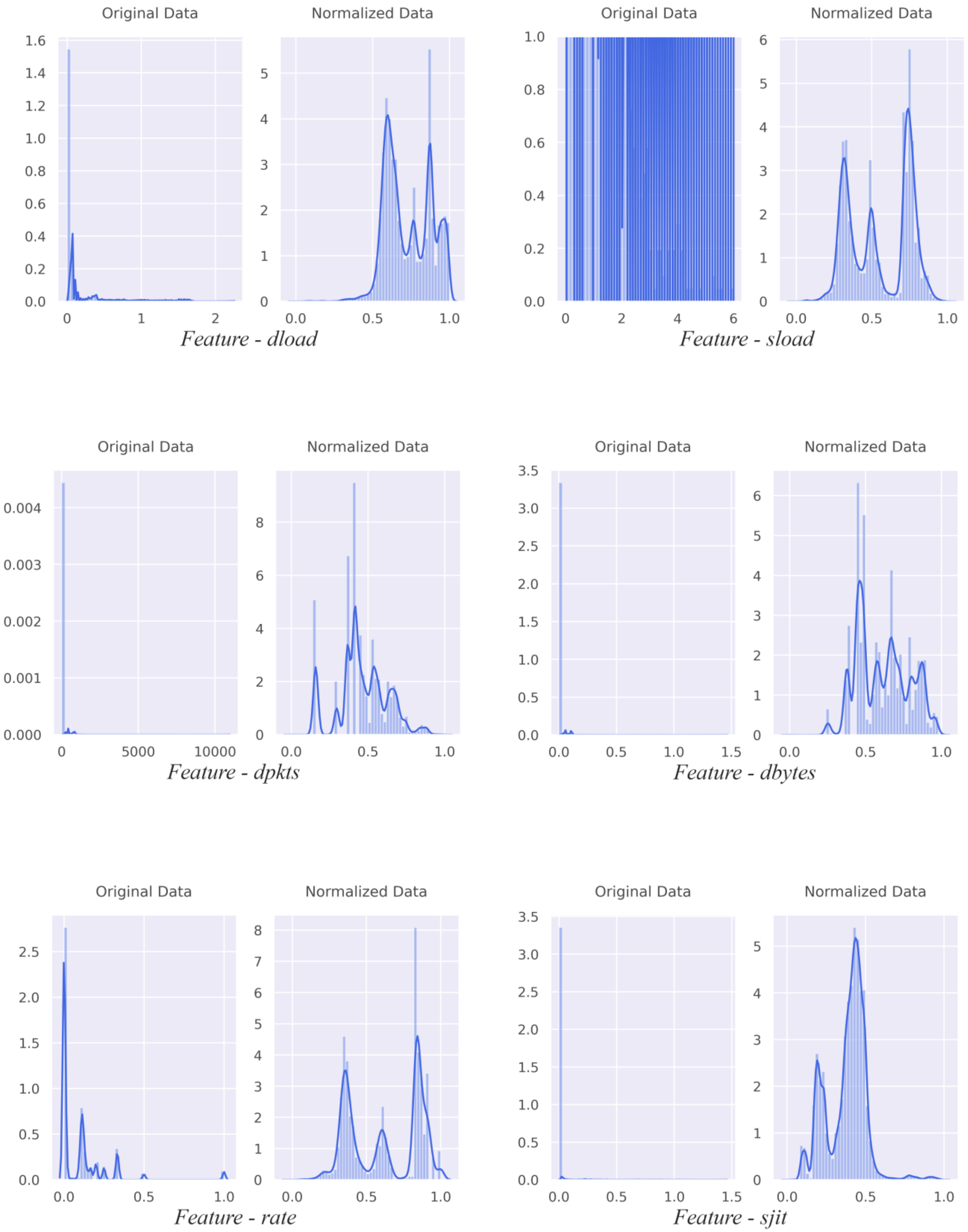


Figure 4.10: Data Normalization Visualization per Feature

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 independently rescales the data feature-wise from its natural range into a standard range where for every feature, the minimum value gets transformed into the value of zero, and the maximum value gets transformed into the value of one, hence giving all the features in data an equal footing for drawing the statistical inference. The formula for normalization can be expressed as,

$$\acute{x} = \frac{(x - x_{\min})}{(x_{\max} - x_{\min})}$$

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. The min-max normalization retains the shape of the feature intact during scale as compared to other normalization we tested during the course of design. Figure 4.10 visualizes how the normalization changed the natural range of raw features in the dataset to the standard range [0,1]. This particular data pre-processing step is vital as various algorithms such as logistic regression and neural networks etc. assume that the input data for processing will be scaled and normalized.

4.6 Evaluation Criteria

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

4.6.1 Classification Accuracy

Accuracy is an evaluation metric used for classification models where we compare the number of correct predictions drawn with the total number of predictions made by the model. The formula for classification accuracy can be expressed as,

$$Accuracy = \frac{Number\ of\ Correct\ Predictions}{Total\ Number\ of\ Predictions} * 100$$

The formula converts the model's accuracy into a percentile value that can be used to evaluate the model’s performance. But classification accuracy by itself is not a good indicator of the performance. It does not consider the class imbalance that might persist in a dataset, where there can be a large difference between the number of positive and negative labels. Hence, we need to judge a model by other metrics as well.

4.6.2 Confusion Matrix

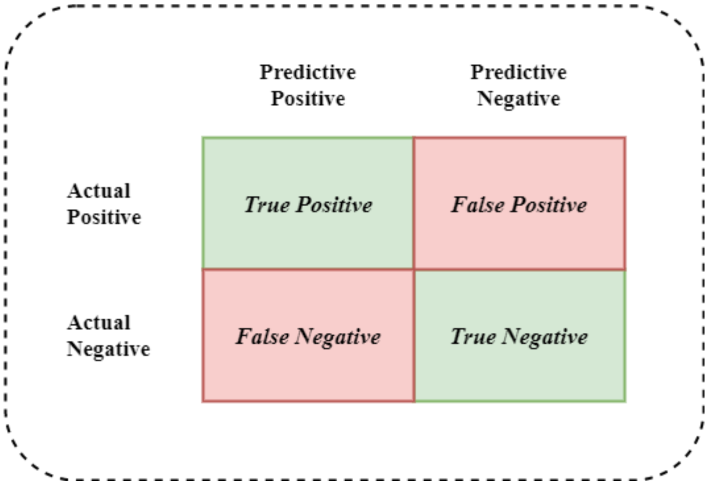
A confusion matrix is a visual representation of the performance of a classification model. It basically is a table with four different combinations of predicted and actual values. A classification model’s outcome can be summarized into these four possible categories,

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.



	Predictive Positive	Predictive Negative
Actual Positive	*True Positive*	*False Positive*
Actual Negative	*False Negative*	*True Negative*

Figure 4.11: Confusion Matrix Sample

Figure 4.11 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.

4.6.3 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. True Positive Rate is also known as sensitivity of a model which determines the proportion of the values which are positive and were indeed correctly identified as positive. This can be expressed as,

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

B. False Positive Rate is also known as the specificity of a model which determines the proportion of values that are negative and were also identified by the model as negative. This can be expressed as,

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

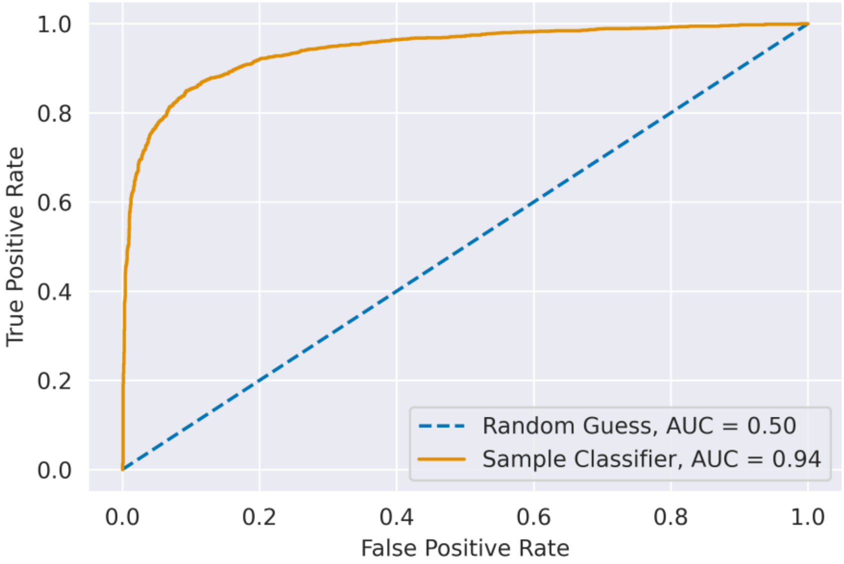


Figure 4.12: ROC Curve example with a Sample Classifier

ROC curve plots the True Positive Rate of a model with False Positive Rate at various classification thresholds as shown in Figure 4.12. The AUC value aggregates the performance of the model across all possible classification thresholds.

Chapter 5

5 Experiment Results

This section will show our results and their analysis from the experiments performed to guide our design of the Intrusion Detection System. Section 5.1 discusses the initial design of the IDS model using various machine learning algorithms. In section 5.2, we will discuss the utility of deep learning algorithms in design our candidate model. Further, in section 5.3 we will demonstrate the experimentation result of the unified learning model proposed in this thesis. Section 5.4 will showcase the results and improvements in performance from applying the transfer learning methodology to our candidate IDS model. In section 5.5, we will discuss our overall results and findings.

5.1 Machine Learning Methods

During the progression of this thesis, we studied and applied various machine learning algorithms to design the initial IDS architecture. This section will elucidate our experimentation and results in this area. As previously deliberated, machine learning in an application sense means we are predicting the nature of data based on our prior analysis during a training phase. For the IDS, we are interested in knowing the nature of a data packet, especially whether the packet is a normal network packet, or it belongs to the class of nine distinguished cyber-attack types the model is trained to identify. The main mode used to build such a system is supervised learning where we are building a model with various training examples with both normal and malicious packets being used to draw signatures and patterns, which are then precedingly used to classify the new future data packets encountered by the model as either normal or malicious. In this case, the malicious packets will always seem like an anomaly to the system and in a statistical sense, their feature data will look like an outlier when compared to the normal baseline. The model helps us establish an optimal baseline of the normal network usage where during the normal network use, the packets flowing through the network will identify with the feature values that are recognized to belong to a normal network packet's features. In contrast, when the network is in the midst of an ongoing misusage that is deemed a cybercriminal activity, the packets flowing through the network will exhibit the feature values that mirror that of the malicious packet observations used in training the model. In essence, the model is continually looking for any outliers from the established normal baseline to filter the forthcoming network packets in terms of normal use or misuse.

We concentrated our efforts on three separate and distinct machine learning models, namely

- Logistic Regression.
- K-Nearest Neighbors.
- Decision Trees.

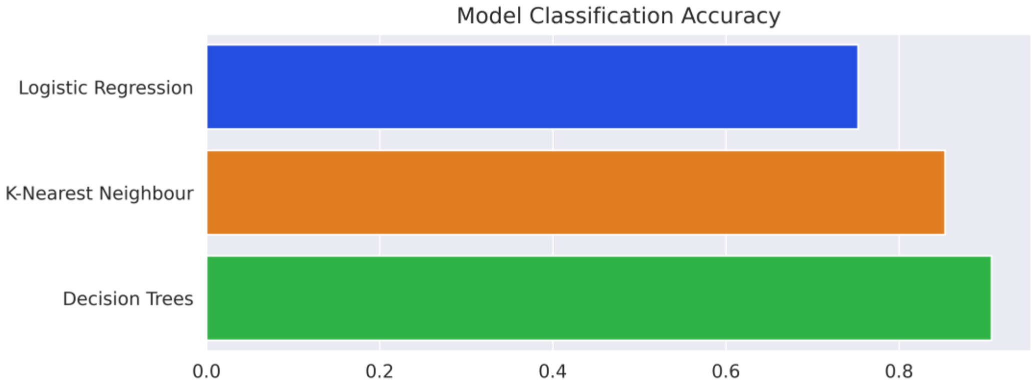


Figure 5.1: Classification Accuracy of Applied Machine Learning Models

Figure 5.1 plots the bar chart for the classification accuracy of each machine learning model applied for the task of intrusion detection. From the experimentation, we observed that Decision Trees performed best in terms of accuracy amongst the applied models with a 90.64% classification accuracy performance. K-Nearest Neighbor gave 85.32% classification accuracy, whereas Logistic Regression gave 75.27% classification accuracy. Because we also aim to design an IDS architecture that can classify the network data at a fast processing speed. We also considered each machine learning model based on the time it took for them to process an entire testing dataset partition to classify the data. Among the applied models, Decision Trees took 6.33 seconds whereas, K-Nearest Neighbor took 31.6 seconds. Logistic regression gave the best testing performance time with 3.01 seconds.

We further used the ROC curve to visualize each applied machine learning model's performance at different thresholds, as shown in Figure 5.2.

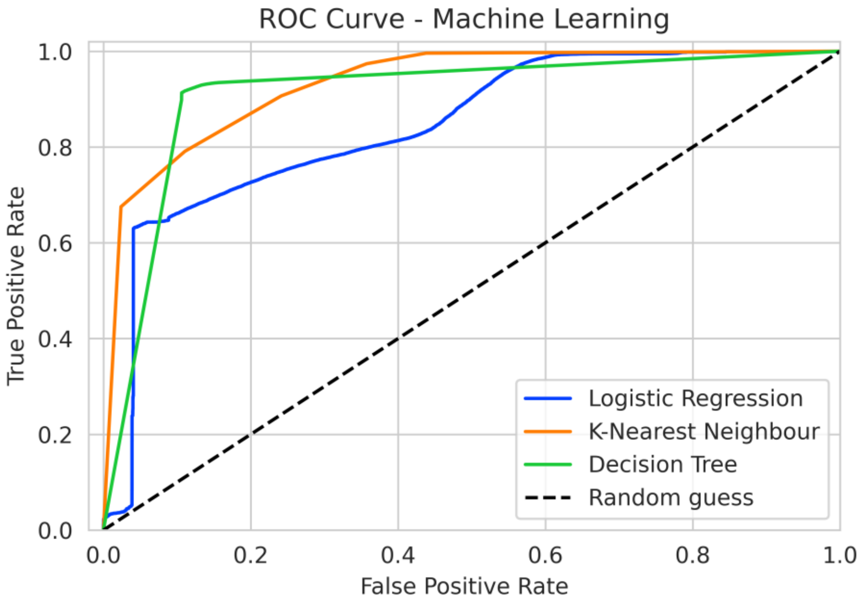


Figure 5.2: ROC curve visualization for applied Machine Learning Models

Table 5.1

Machine Learning Model Performance Summary

Model	Accuracy	Speed	AUC
Logistic Regression	75.27%	3.01s	0.84
K-Nearest Neighbors	85.32%	31.6s	0.93
Decision Trees	90.64%	6.33s	0.91

Table 5.1 summarizes our experimentation results in the area of machine learning to design and select our target IDS model. Based on our experimental results, we chose Decision

Trees as our target machine learning model to design the IDS. We further studied its results in-depth using the Confusion Matrix metrics, as shown in Figure 5.3.

According to the confusion matrix, the decision trees had 5.98% False Positive outcomes and 6.27% False Negative outcomes. As discussed before, false-positive determines the percentage of normal packets identified as malicious, and false-negative determines the percentage of malicious packets identified as normal by the Intrusion Detection System. So, in essence, what this means is that the decision trees based IDS are susceptible to allow 6.72% malicious attacks pass through its system undetected, which may open doors for more concealed attacks and identified 5.98% normal packets as malicious, which will lead to that percentage of packets being dropped or blocked by the system affecting the network quality of service.

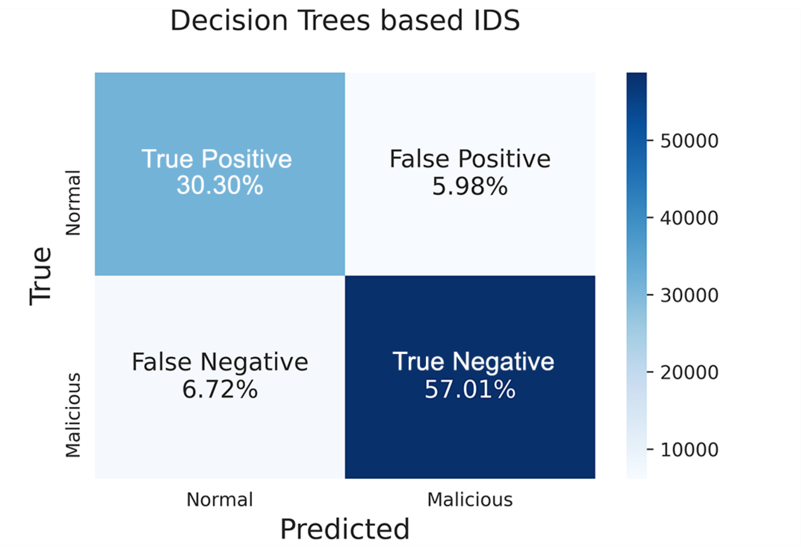


Figure 5.3: Decision Trees based IDS - Confusion Matrix

Despite the fast speed and high classification accuracy performance, we were not satisfied with the predicted outcomes of decision trees based IDS due to the fact of its high false positivity and high false negativity. After an exhaustive effort to improve the classification results, we chose to further investigate the field of deep learning to build our candidate IDS model to deliver high classification accuracy, speed performance, and precise prediction outcomes.

5.2 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 our experimentation, we will use our source domain which is modeled in the Google Cloud Platform's provisioned VM instance named n1-standard-8, which has a total number of 8 vCPUs and a 30 GB memory to simulate a computationally resource abundant environment.

For training and validating our model, we will use the two preprocessed partitions of the training dataset and validation dataset as described in section 4.4. In total, we are using 206,138 packet observations for our experimentation in the source domain environment.

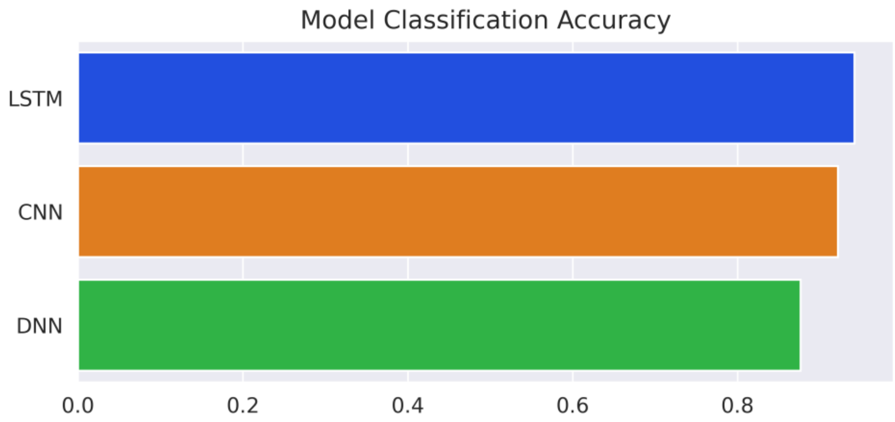


Figure 5.4: Classification Accuracy of Applied Deep Learning Models

Figure 5.4 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 5.5.

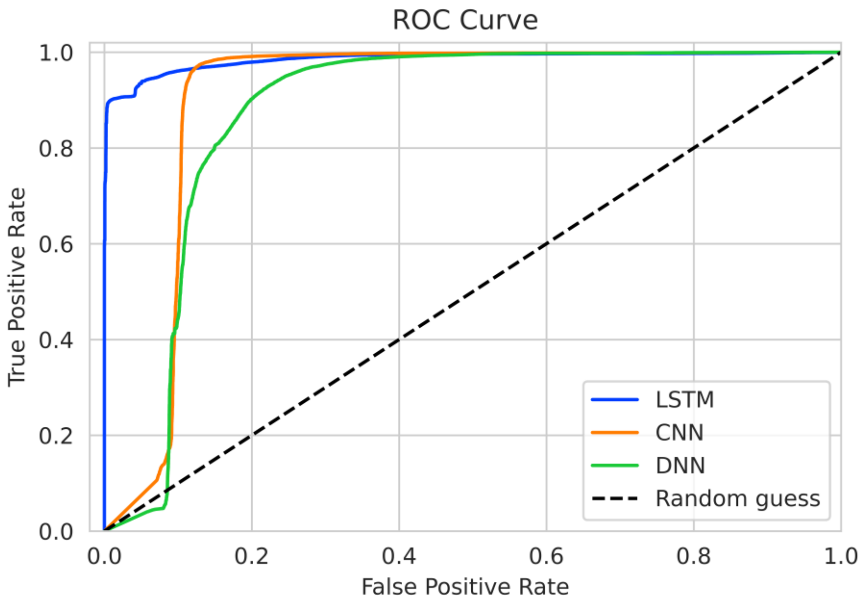


Figure 5.5: ROC curve visualization for applied Deep Learning Models

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 5.2

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 5.2 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 5.7 plots the confusion matrix for the LSTM model.

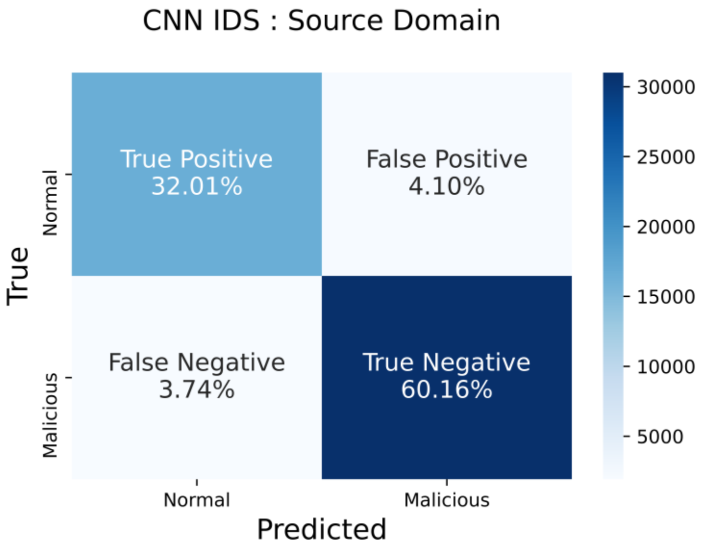


Figure 5.6 : CNN based IDS - Confusion Matrix

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.

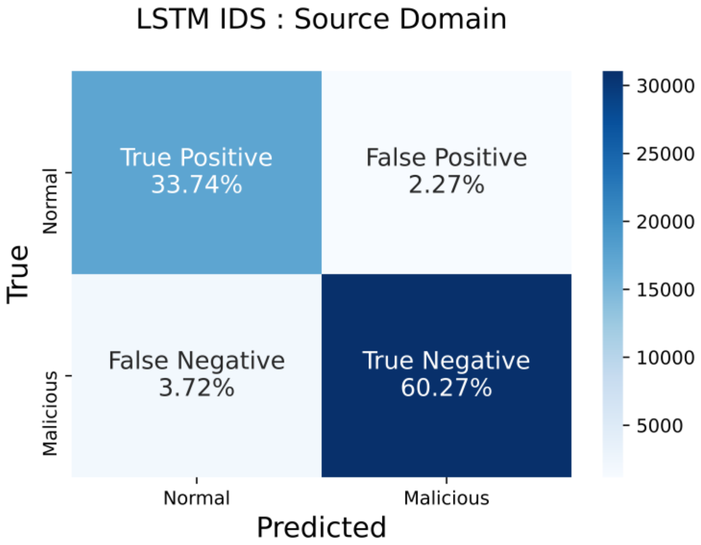


Figure 5.7 : LSTM based IDS - Confusion Matrix

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.

5.3 Unified Deep Learning Network

Our continued experimentation lead us to consider adopting a modular approach towards constructing our candidate IDS model, where we are using the advantages of the three applied deep learning models and combine their latent feature extraction, memory retention, and classification abilities to give a higher accuracy score and prediction outcomes as compared to these models being applied separately. In section 4, 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.

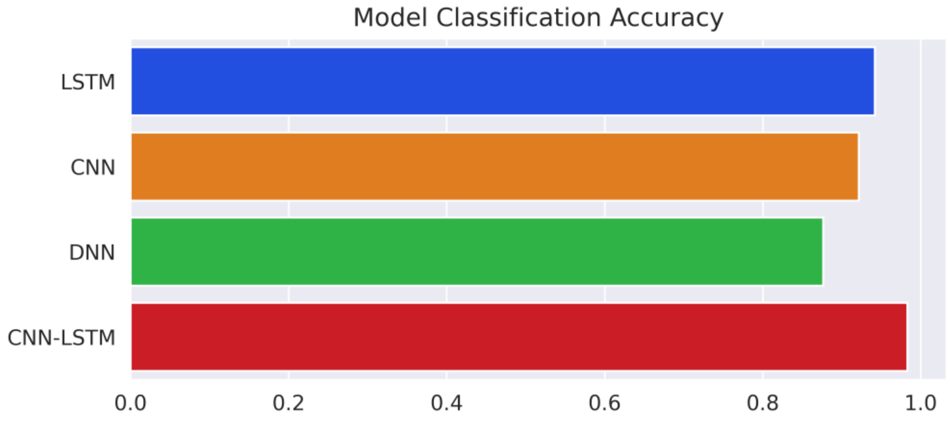


Figure 5.8 : Classification Accuracy of Unified Model in comparison with DL Models

As shown in the bar chart plotted in Figure 5.8, 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.

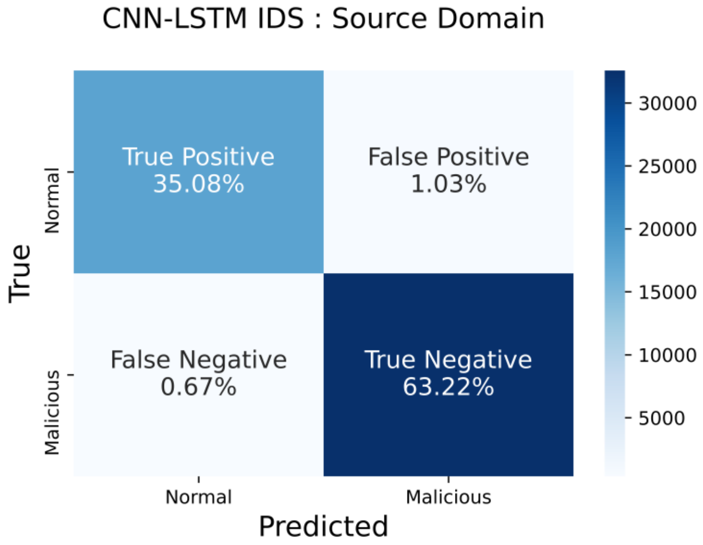


Figure 5.9: CNN- LSTM based IDS – Source Domain Confusion Matrix

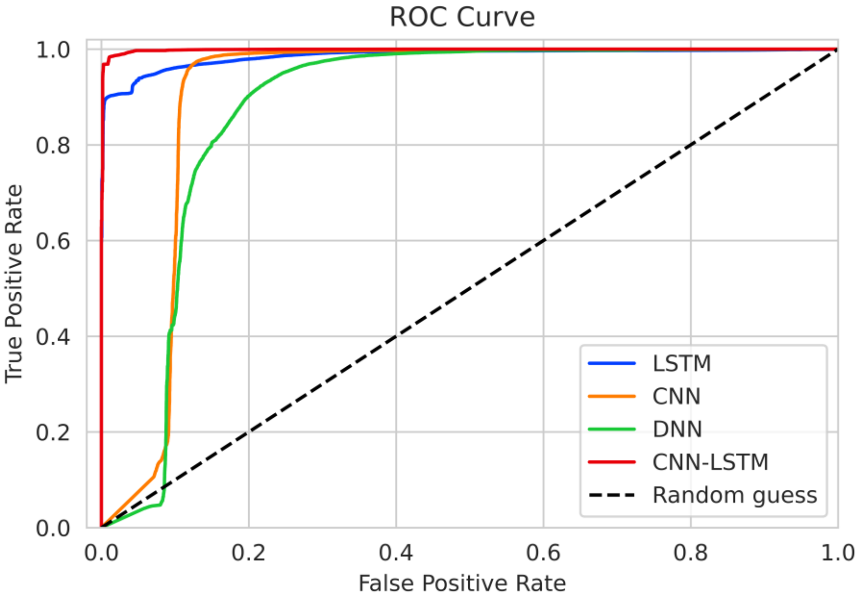


Figure 5.10: ROC curve visualization of Unified CNN-LSTM Model

Based on our confusion matrix metrics as shown in Figure 5.9, 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 5.10. 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 5.3.

Table 5.3

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.

5.4 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 4.2. 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.

To apply the learned knowledge in the target domain, we will use the unseen testing dataset in this domain to simulate the IDS model being in a real environment where it encounters entirely new data. This helps in evaluating how the model will essentially react when it is deployed in a real-world network infrastructure.

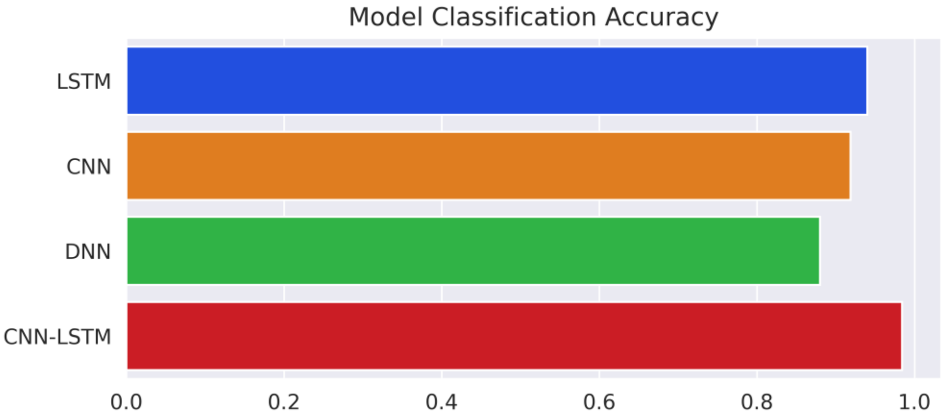


Figure 5.11: Classification Accuracy of Applied Deep Learning models in Target Domain

As shown in the bar plot illustrated in Figure 5.11, 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 improved 88% accuracy score percentage.

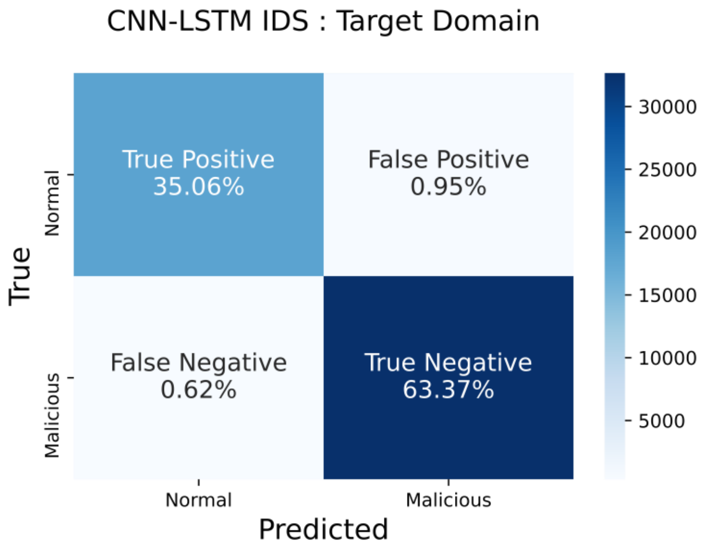


Figure 5.12: CNN- LSTM based IDS – Target Domain
Confusion Matrix

To further study our results in the target domain, we used confusion matrix metrics to visualize our candidate CNN-LSTM model's classification performance, as shown in Figure 5.12. According to the confusion matrix, our novel CNN-LSTM unified model reached a false positive value of 0.95% and a false negative value of 0.62%. This was by far the best classification performance amongst each neural network model applied in both domains. 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 5.13 shows that our IDS model’s diagnostic ability remained comparable in the target domain.

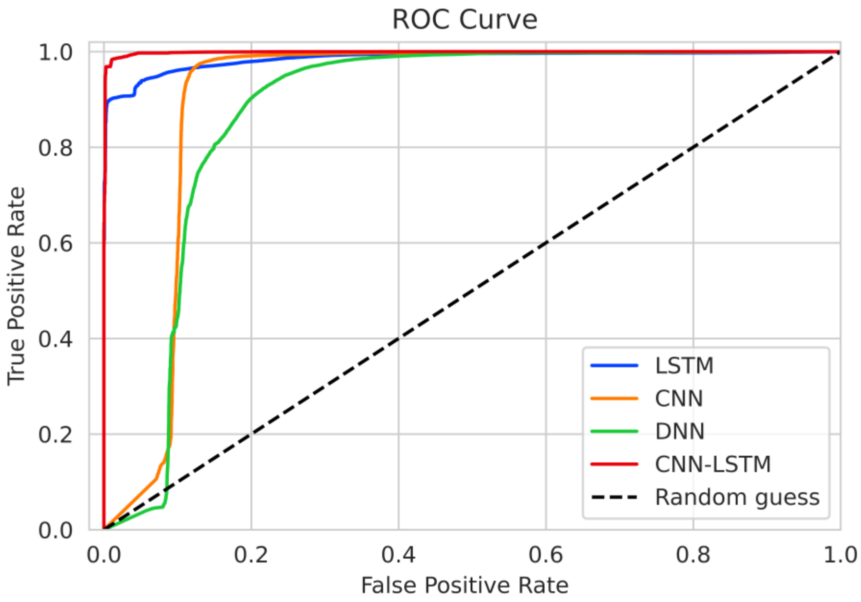


Figure 5.13: ROC curve visualization -Target Domain

Table 5.4

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 5.4 showcases the summary of our results in the target domain for each neural network model applied.

5.5 Discussion

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.

Chapter 6

6 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. To make our architecture work efficiently in real-world settings, we used deep transfer learning methodologies. Our research demonstrates that the deep transfer learning approach can be highly effective in developing an efficient, unified network intrusion detection system that maintains and improves its classification accuracy and speed in a simulated real-world setting via knowledge transfer.

Using the proposed method, we can train a large and powerful deep learning IDS model in a source domain with a high allocation of data and computational resources. After validating our model's performance, we can then transfer its architecture and learned weights in a target domain with reduced computational resources. We observe that the model maintains its efficiency and improves its testing speed. The target domain aims to simulate the real-world environment where we are using a partition of the dataset, which is entirely unseen by our models during their training and development.

This thesis showcases that high powered deep learning-based IDS architectures can be deployed on real-world devices with lesser resources, maintaining their efficiency and improving their speed using the transfer learning approach. Applying transfer learning in the overall design of an IDS enhances its performance in a real-world setting. It essentially increases its classification speed, which is a tremendously required feature demanded by an IDS to protect and secure modern network infrastructures. Our research is one of the earliest practical implementations of integrating transfer learning techniques in the core architecture of an IDS.

6.1 Limitations

Despite the showcased potential of deep learning and transfer learning methodologies to architect data-driven intrusion detection systems, certain limitations and challenges may present themselves when deploying the systems in live production environments. We have discussed the efficacy of transfer learning to improve learning models' performance in the target domain. Still, there may be certain times when transfer learning may lead to a drop in performance, also known as a negative transfer. This happens when the source domain’s data is fundamentally different from the type of data used in the target domain leading to the learning model not being able to build a semantic relationship between the domains appropriately. This can be avoided by carefully examining the data in the source as well as target domains and prudently planning the data ingestion and feature engineering subcomponents.

6.2 Future Work

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.

Curriculum Vitae

Name:	Harsh Dhillon
Post-secondary Education and Degrees:	Western University London, Ontario, Canada 2018 – 2020 M.Sc. (Computer Science) Seneca College of Applied Arts and Technology Toronto, Ontario, Canada 2015 – 2016 Graduate Certificate (Visual Effects for Film and Television) 2014 – 2015 Ontario Certificate (Art Fundamentals) Amity University Noida, Uttar Pradesh, India 2010 - 2014 Bachelor in Computer Application
Honors and Awards:	Western Graduate Research Scholarship (WGRS) 2019
Related Work Experience	Graduate Teaching Assistant University of Western Ontario 2018 - 2019

Publications:

Harsh Dhillon and Anwar Haque, “Towards Network Traffic Monitoring Using Deep Transfer Learning,” to appear in the proceedings of the 19th IEEE International Conference on Trust, Security and Privacy in Computing and Communications (IEEE TrustCom 2020), Dec 29 2020 – Jan 1 2021, China [acceptance rate: 25%]