OGOBUCHI DANIEL OKEY

MULTI-PHASE OPTIMIZED INTRUSION DETECTION SYSTEM BASED ON DEEP LEARNING ALGORITHMS FOR COMPUTER NETWORKS

LAVRAS – MG

2022

OGOBUCHI DANIEL OKEY

MULTI-PHASE OPTIMIZED INTRUSION DETECTION SYSTEM BASED ON DEEP LEARNING ALGORITHMS FOR COMPUTER NETWORKS

Dissertation presented to the Federal University of Lavras, as part of the requirements for the degree of Master of Science in the Graduate Program of Systems and Automation Engineering.

Prof. DSc. Demóstenes Zegarra Rodríguez
Advisor

Prof. Muhammad Saadi
Co-Advisor

LAVRAS – MG

2022

Ficha catalográfica elaborada pelo Sistema de Geração de Ficha Catalográfica da Biblioteca Universitária da UFLA, com dados informados pelo(a) próprio(a) autor(a).

Okey, Daniel Ogobuchi.
Multi-Phase Optimized Intrusion Detection System Based on Deep Learning Algorithms for Computer Networks / Ogobuchi Daniel Okey. – 2022.
163 p. : il.

Advisor: Demóstenes Zegarra Rodríguez.
Co-advisor: Muhammad Saadi.
Dissertation (Academic Masters) – Universidade Federal de Lavras, 2022.
Bibliography.

1. Machine Learning. 2. Deep Learning. 3. Intrusion Detection Systems. I. Rodríguez, Demóstenes Zegarra. II. Saadi, Muhammad. III. Título.

OGOBUCHI DANIEL OKEY

MULTI-PHASE OPTIMIZED INTRUSION DETECTION SYSTEM BASED ON DEEP LEARNING ALGORITHMS FOR COMPUTER NETWORKS

Dissertation presented to the Federal University of Lavras, as part of the requirements for the degree of Master of Science in the Graduate Program of Systems and Automation Engineering.

Approved on September 5, 2022.

Prof. DSc. Demóstenes Zegarra Rodríguez	UFLA
Prof. Muhammad Saadi	UCP
Prof. Dante Coaquira Begazo	USP
Profa. Renata Lopes Rosa	UFLA

Prof. DSc. Demóstenes Zegarra Rodríguez
Advisor

Prof. Muhammad Saadi
Co-Advisor

LAVRAS – MG
2022

This research is dedicated to my dear father of blessed memory, late Chief. Francis Ogbonnaya Okey who through thick and thin ensured that I got the best of education that I needed. He had always prayed to see me in my current position but could not. Keep resting my mentor.

ACKNOWLEDGEMENTS

I express my sincere gratitude to the Almighty God who has been my strength through all my academic journey. He has always made everything so easy and comfortable for me. It pays to serve Jesus. My sincere appreciation goes to my advisor Prof. DSc. Demóstenes Zegarra Rodríguez for guiding me through the course of my research, providing me with all necessary instructions on how to achieve the objective. Also, I appreciate my co-advisor, Prof. Muhammad Saadi and all my teachers who taught me during the course. I acknowledge the efforts of my program coordinator, Prof. Ricardo Rodriguez Magalhaes for his efforts towards my comfort during the classes.

To my dear wife Mrs. Chinecherem Harriet Daniel-Okey, my mother, Mrs. Christiana Nwankwo Okey, Philip Osondu Okey (Brother), Sisters and friends who stood by me through this course, thank you. I appreciate the efforts of my flatmates, Julian and Ruth, Samuel, Luiz, Fernando, Bruno, Douglas, Silvia and all Nigerian Scholars in UFLA, you guys made my stay in Brazil. God bless you.

I thank the Forum for Agricultural Research in Africa (FARA) and Tertiary Education Trust Fund (TETFund) who through the Agricultural Research and Innovation for Africa (ARIFA) programme in collaboration with the UFV/FARA/TETFund initiative funded this research. To Joyce in the international Relation Office, (DRI), thank you for the amazing support you gave me to ensure my success through this program.

RESUMO

As redes de computadores revolucionaram todo o espaço de trabalho nos últimos tempos, de modo que seus potenciais e contribuições não podem ser subestimados. Sequência das imensas vantagens das redes de computadores, muitas organizações e empresas dependem delas para atividades cotidianas que vão desde a busca de recursos até a disseminação de informações. A grande dependência de serviços de Internet tem enfrentado o desafio de privacidade e segurança. Isso se deve ao fato de que indivíduos com intenção maliciosa elaboram algumas estratégias para explorar as redes e roubar informações causando danos. Para isso, diversas técnicas e tecnologias, como firewalls, estão sendo utilizadas para impedir a ocorrência de ataques cibernéticos. Um desafio com essa abordagem é a questão dos falsos positivos, onde informações reais são identificadas como ameaças. Uma maneira de resolver isso é o uso de um Sistema de Detecção de Intrusão (IDS - Intrusion Detection System) que monitora e inspeciona as atividades da rede para detectar ameaças. IDS desenvolvidos usando algoritmos de Aprendizado de Máquina (ML - Machine Learning) e Aprendizado Profundo (DL - Deep Learning) têm mostrado prevalência sobre IDS baseados em conhecimento. Neste trabalho, aproveitamos as capacidades do ML e DL para desenvolver IDS para redes de computadores. Especificamente, dois modelos de IDS são desenvolvidos com base em dados tabulares e dados de imagem. Primeiro, pré-processamos os dados em um formato compatível e lidamos com o desequilíbrio com a Synthetic Minority Oversampling Techinque (SMOTE). Nos dados tabulares, usamos a Rede Neural Convolucional Unidimensional (1D CNN - One-Dimensional Convolutional Neural Network) e alguns classificadores de ML, enquanto o Transferência de Aprendizado (TL - Transfer Learning) é usado nos dados da imagem. Os dados de imagem são gerados pela transformação do conjunto de dados amostrado em uma imagem RGB 64x64x3. Essas imagens são alimentadas na CNN que tem um excelente desempenho na extração de características das imagens utilizadas no processo de aprendizagem. Essa capacidade da CNN de extrair automaticamente recursos relevantes do tráfego de rede é usada para classificar o tráfego em diferentes categorias. Cinco diferentes modelos pré-treinados baseados em CNN: Visual Geometry Group (VGG16 e VGG19), InceptionV3 (IV3), MobileNetV3Small (MNV3S) e EfficientNetV2B0 (ENV2B0) são usados para desenvolver o IDS baseado em imagens geradas a partir do conjunto de dados. Finalmente, desenvolvemos um IDS usando um modelo otimizado Ensemble baseado em Transferência de Aprendizado (ELETL-IDS - Ensemble Lightweight Transfer Learning IDS) capaz de detectar e classificar o tráfego de rede segundo o tipo de ataque, tais como DDoS, DoS, Bot, Força Bruta, Infiltração, PortScan, Heartbleed e Web Attacks. Na avaliação, os modelos apresentam alto desempenho com 1D-CNN atingindo uma precisão média ponderada de 99,11% e ELETL-IDS tem 100% de precisão na classificação de cada classe. Realizamos a quantização do modelo original para reduzir cerca de 77% de seu tamanho (aproximadamente, 4 vezes menor que o tamanho do modelo original) com uma queda de 1,1% na precisão, tornando os modelos IDS altamente eficientes e adequados em diferentes domínios de aplicação.

Palavras-chave: Aprendizado de Máquina. Aprendizado Profundo. Sistemas de Detecção de Intrusão. Redes de Computadores. Transferência de Aprendizado. Redes de Convolucionais

ABSTRACT

Computer networks have revolutionized the entire workspace in recent times, so their potentials and contributions cannot be underestimated. As a result of the immense advantages of computer networks, many organizations and companies depend on them for everyday activities that range from searching for resources to disseminating information. The large dependency on Internet services has faced the challenge of privacy and security. This is due to the fact that individuals with malicious intent devise some strategies to exploit the networks and nodes to steal information thereby causing damage. To this end, several techniques and technologies such as firewalls are being used to deter cyber-attacks from occurring. One challenge with this approach is the issue of False Positives where real information is identified as threats. One way to solve this is the use of an Intrusion Detection System (IDS) that monitors and inspects network activities to detect threats. IDS developed using Machine Learning (ML) and Deep Learning (DL) algorithms have shown prevalence over knowledge-based IDS. In this work, we leverage the capabilities of ML and DL to develop IDS for computer networks. Specifically, two IDS models are developed based on Tabular data and Image data. First, we preprocess the data into a compatible format and handle the imbalance with Synthetic Minority Oversampling Technique (SMOTE). On the tabular data, we use One-Dimensional Convolution Neural Network (1D-CNN) and some ML classifiers while Transfer Learning (TL) is used on the image data. Image data are generated by transforming the sampled dataset into a 64x64x3 RGB image. These images are fed into the CNN, which has an excellent performance in extracting features from images used in the learning process. This ability of CNN to automatically extract relevant features from network traffic is used to classify the traffic into different categories. Five different pre-trained models based on CNN: Visual Geometry Group (VGG16 and VGG19), InceptionV3 (IV3), MobileNetV3Small (MNV3S), and EfficientNetV2B0 (ENV2B0) are used to develop the IDS based on images generated from the datasets and in the end, we develop an optimized Ensemble Lightweight Transfer Learning IDS (ELETL-IDS) capable of detecting and classifying network traffic into its attack type such DDoS, DoS, Bot, Brute force, Infiltration, PortScan, Heartbleed and Web Attacks. On evaluation, the models show high performance with 1D-CNN reaching a weighted average accuracy of 99.11% and ELETL-IDS has 100% accuracy in classifying each of the classes. We perform model quantization to reduce the model size to about 77% (about 4x smaller than the original model size) with a drop of 1.1% in accuracy, making the IDS models highly efficient and suitable in different application domains.

Keywords: Machine Learning. Deep Learning. Intrusion Detection Systems. Computer Networks. Transfer Learning. Convolutional Neural Networks

LIST OF FIGURES

LIST OF TABLES

CONTENTS

1 INTRODUCTION

Communication systems play important roles in the daily life of everybody as most human activities depend on them for efficiency and productivity. Computer networks are widely used for commercial and individual purposes such data processing, education and learning, large-scale data acquisition, agriculture, governance and entertainment. Nowadays, the number of connected devices on the internet increases continuously (CHEN, 2012; NGUYEN *et al.*, 2021). The increase can be attributed to the advancement in technology featuring process automation, such as in the use of Internet of Things (IoT), smart city, smart transportation, smart agriculture, smart banking, among others (SU; LI; FU, 2011; KHAN *et al.*, 2021). Due to this huge expansion of the network and anonymous structure of the Internet, ensuring the confidentiality, integrity and availability (CIA) of both information and communication systems has been challenging.

Network Security is an increasing challenge in modern times due to the rapid growth in technological advancement according to Liu *et al.* (2021). The Internet provides all knowledge that has been accumulated over time to users and with the advent of mobile computing at every person's fingertips, cyber attacks and cyber crimes have become all too popular. There have been well-known traditional techniques of dealing with cyber attacks in the past, but there has been an increase in cyber attacks and how exploits are carried out over the last decades. Consequently, network security techniques are also undergoing a revolution into more intelligent mechanisms.

Furthermore, many hackers with malevolent intent attempt to gain unauthorized access to the network systems in order to access the information. These persons are well-known programmers who are regarded to be hackers, invaders, or cyber criminals, according to Wall (2007). Some internet cyber attacks have resulted in massive data loss and, as a result, significant financial cost have been acquired. Many preventative measures, such as the installation of firewalls, use of anti-virus software (LYU; LAU, 2000), Intrusion detection systems (IDS) (RADOGLOU-GRAMMATIKIS; SARIGIANNIDIS, 2019), have been used as a counter-measure to detect network intrusion. However, nowadays some cyber attacks are still successful leading to increasing demand for a more intelligent fast-decision-making mechanism to combat cyber-criminals constant attempts to find new ways to bypass the systems' prevention mechanisms. The most practical solution is to create reliable, cost-effective and easily maintainable IDSs. To this end, many Machine Learning (ML) and

Deep Learning (DL) algorithms have been proposed in the literature such as DL based on Restricted Boltzmann Machine (RBM) and Sparse Auto-Encoder (SAE) in Van, Thinh *et al.* (2017), Random Forest in Farnaaz e Jabbar (2016) and Decision Tree in Panda, Abraham e Patra (2012).

Intrusion is as an unauthorized activity that causes damage to an information system (KHRAISAT *et al.*, 2019b). This means that any attack that could pose a possible threat to the CIA triad of information is considered an intrusion. Intrusion Detection (ID) implies a way to monitor, identify and react to all possible attacks or abnormal activities on a computer network. Common intrusion attempts include Botnets, Denial of Service (DoS) (CHEN *et al.*, 2019), Distributed Denial of Service (DDoS) (SHARAFALDIN *et al.*, 2019), Brute force attack (STIAWAN *et al.*, 2019), Infiltration (ZHANG *et al.*, 2021), Zero-day attack (KIM; BU; CHO, 2018), Pharming (CHOI, 2019), Eavesdropping attacks (VASHIST *et al.*, 2019), Phishing and Spamming (BORKAR; DONODE; KUMARI, 2017) among others.

IDS have the capability of detecting several types of network attacks as discussed in Smaha *et al.* (1988). The IDS monitors network traffic and decides if the activities are normal or have malicious intent. When unusual traffic or activity is perceived, an alarm is registered against such activity by the system. Knowledge-based IDS have shown the inability to detect more complex abnormal network activities as they are based on pre-defined rules, as a result, there is a greater requirement for IDS built with advanced technologies (ML/DL) that are capable of detecting attacks quicker and more precise. In general, IDSs are divided into two categories based on the kind of attack *(misuse and anomaly IDS)* and the type of data *(host-based, network-based and hybrid IDS)* (PHARATE *et al.*, 2015; LAKSHMINARAYANA; PHILIPS; TABRIZI, 2019). When IDS are designed to function according to known attack signatures, they are referred to as misuse-based IDS or signature-based IDS where as in anomaly situation, IDS are designed to function in relation to changing environmental demands in respect to network traffic.

Machine and Deep learning algorithms have shown great capabilities in aiding the quest to proffer solution to the vast challenges of cyber intrusion in computer networks. Consequently, several IDS models based on ML/DL techniques have been proposed in the past according to Sahu e Mehtre (2015), Aljawarneh, Yassein e Aljundi (2019). ML algorithms including the decision tree Sahu e Mehtre (2015), random forest, extra tree classifiers, boosting classifiers, extreme gradient boosting are some of the commonly used

ML techniques to develop an IDS model to classify network packets and detect intrusion. Like the ML approach, DL algorithms such as the Convolutional Neural Network (CNN), Deep Belief Networks (DBN), AutoEncoders (AE) proposed by Kannari, Shariff e Biradar (2021), Recurrent Neural Networks (RNN) have also been implemented by several authors but the challenge of high False Alarm Rate (FAR) and high computation resource requirement for training DL models have remained an issue that demands more investigation.

IDS models are usually developed on generated streams of network traffics which are either collected in real-time or simulated using a test-bed. Many of these databases have been generated and made publicly available for research purposes. While some are specific to a particular application domain, many of the datasets can be used to develop IDS that can be adapted to different environments such IoT systems, Internet of Vehicle (IoV) and computer networks in general. Some of the many datasets for IDS research include KDD Cup'99, NSL-KDD, AWID, UNSW-15, WSN-DS, CIC-IDS2017, CSE-CIC-IS2018 and more as discussed in Section 2.4. In Kannari, Shariff e Biradar (2021), authors proposed an IDS based on sparse autoencoder and using the CIC-IDS2017, NSL-KDD and AWID datasets for the model training. Some advances have been observed with IDS based on ML/DL. However, there is still a need for improvement in computational speed and Detection Rate (DR) of new attack types using robust datasets that contains real-time network traffics. To tackle this, we develop models that are light-weight, optimized and has the capacity to adapt to changing conditions.

Transfer Learning (TL) is an advanced ML/DL approach, where knowledge (e.g., weights) obtained from previously trained (pre-trained) models' are transferred for learning improvement in a new task. Over the years, TL using pre-trained models has been applied successfully for image classification to reduce computational cost and improve model performance. AlexNet, VGG-16, VGG-19, GoogLeNet or Inception, MobileNet, Xception, ResNet and EfficientNets are the most widely used pre-trained models that were mainly introduced for image classification challenges in the ILSVRC ImageNet competition according to Russakovsky *et al.* (2015). These models leverage more in-depth network architecture implemented on the ImageNet dataset that contains 14 million images belonging to 1000 classes to produce better classification performance. Ensemble Learning (EL) is a technique where weak models (learners) are aggregated using defined approach to obtain a more efficient model with improved performance. Integrating the concept of EL with transfer

learning, tends to yield more reliable and efficient models. In this work, these techniques are combined to develop IDS models that can be adapted to devices of varying memory capacities in computer network domain.

1.1 Objective

The objective of this work is to develop an IDS to identify various network traffics; detecting if they are malicious or normal traffic and classify them according to their types with reduced computational cost and False Alarm Rate (FAR). The IDS is developed and evaluated using robust datasets of recent intrusion attacks signatures including CIC-IDS2017 and CSE-CIC-IDS2018 using 1D-CNN and ensemble transfer learning approaches.

1.1.1 Specific Objectives

The specific objective of this research is to develop IDS model trained on most recent datasets to detect modern day attacks according to their types including DDoS, DoS, Brute Force, Web attacks and Infiltration.

1. Selecting the most recent dataset from available databases and building a more comprehensive dataset with detailed network features suitable for our research.
2. Design the architecture of the proposed IDS model for the selected algorithms.
3. Transform the numerical dataset into an image format to be used for TL implementation based on selected CNN architectures.
4. Train a model on tabular data and image data that not only achieve high detection accuracy but also has faster detection rate and evaluate the model performance using the evaluation parameters.
5. Implement tuning to determine the most important Hyper parameters for the model in order to reduce complexity, validate and compare the performance of the developed model with existing solutions.
6. Perform model size reduction using quantization such that the IDS model can be compatible with most devices in the network.

1.2 Justification

So much concern have been raised regarding the security of enterprise and individual profiles that are transmitted regularly over the Internet. Business organizations, IoT systems, government agencies, educational sectors and financial institutions generate and transmit huge amount of data daily; and these data are stored in the cloud over the network. As the data size increases, there is also corresponding increase in the network size, and subsequently the activities of attackers exploiting vulnerabilities to steal or destroy such data. To ensure data and personal privacy is secured, IDS systems are being utilized.

Most of the existing attack detection methods are not programmed to accommodate the increasing high volume of data. Thus, more solutions that can adapt to varying network size are needed. These solution are such that are designed to operate with a high level of intelligence. IDS systems typically functions in four mechanisms including the decoder, preprocessor, decision system and the detection system. The most important being the decision and detection systems. Hence, it is expected that IDS should be able to function independently in monitoring network packets.

We used a dataset that contains wide range of network flows collected over a long period of time and consider a higher degree of network flow types as opposed to some works in the literature. Hence, the model developed has the capability to detect and classify different attacks as identified in Sharafaldin, Lashkari e Ghorbani (2018) with higher degree of accuracy, precision, F-1 score, recall, Area Under the Curve (AUC) and lower latency. The datasets used contains streams of flow packets (features) among which some are not dominant and does not affect the behaviour of the traffic, hence, the need to reduce the dimensionality of the dataset by determining the most important features of the network traffic which defines the network traffic.

1.3 Organization of Work

This work is organized into the following form:

In Chapter 1, the main concept of computer networks and IDS, identifying the problems in this domain and the subjects related to the topic. In Chapter 2, we described in details the theoretical backgrounder. Detailed analysis of the datasets available for IDS design and the various algorithms leveraged to implement them were also presented. In the

end, related works were reviewed and summarized. In Chapter 3, the materials and method adopted for the research was discussed. In Chapter 4, we performed the presentation and discussion of the results obtained during the work. Finally, in Chapter 5, drew the conclusion on the current work and identified gaps for future work in this area.

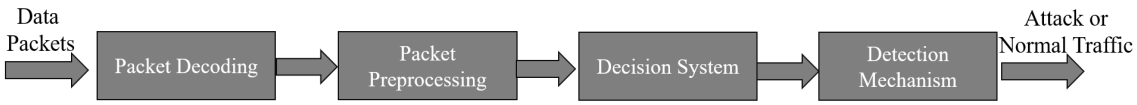
2 THEORETICAL BACKGROUND AND RELATED WORKS

In this section, a general review of the classification of IDS, types of computer network attacks, different approaches used in implementing IDS, publicly available IDS datasets and related works are presented.

2.1 The Concept and Classification of Intrusion Detection Systems (IDS)

Intrusion Detection Systems (IDSs) are software or hardware devices installed at designated locations to monitor the network or devices for possible malicious activities or policy violations that may be harmful to the network. These violations when detected are either reported to an administrator for possible response or collected through a central interface called System Information and Event Management (SIEM) for possible actions. The SIEM distinguishes anomalous event from false alarm through the use of filtering techniques over outputs collected from multiple sources (AXELSSON, 2000). IDS keeps constant check in real-time on data flow in a network or hosts systems to detect malicious behaviors. A typical IDS usually involve a four-stage action phases which includes: Decoder, preprocessor, decision system and detection mechanism. Network traffic such as requests sent by users are received and interpreted by the decoder; after which the information is preprocessed for subsequent actions. Based on the preprocessed network packet, the IDS takes a decision on the data and the final mechanism interprets it as attack or normal traffic. This process is shown in Figure 2.1.

Figure 2.1 – Phases of IDS mechanism in a typical operation environment.



Source: Author (2022)

James Anderson in 1980 at the National Security Agency (NSA) *national-level intelligence agency of the United States Department of Defense, under the authority of the Director of National Intelligence (DNI)* introduced the earliest preliminary IDS concept which was composed of different tool aimed at aiding network administrators review audit trails such as User access logs, system event logs and file access logs (ANDERSON, 1980). The first published model of an IDS was in 1986 by Dorothy E. D. with assistance from Peter G. Neumann. The IDS model used statistical approach for anomaly detection and became the

benchmark for IDS research today (DENNING, 1987). Also, the work resulted in an early IDS at SRI International named the Intrusion Detection Expert System (IDES), capable of considering both user and network data and runs on sun workstations (LUNT, 1990). IDES had a dual approach with a rule-based Expert System to detect known types of intrusions plus a statistical anomaly detection component based on profiles of users, host systems, and target systems. The author maintained that the inclusion of Artificial Neural Networks (ANN) as a third component to the existing IDES could present a better resolution to the challenges of intrusion. The Next-generation Intrusion Detection Expert System (NIDES) followed the IDES in 1993 through the efforts of the SRI (LUNT, 1993).

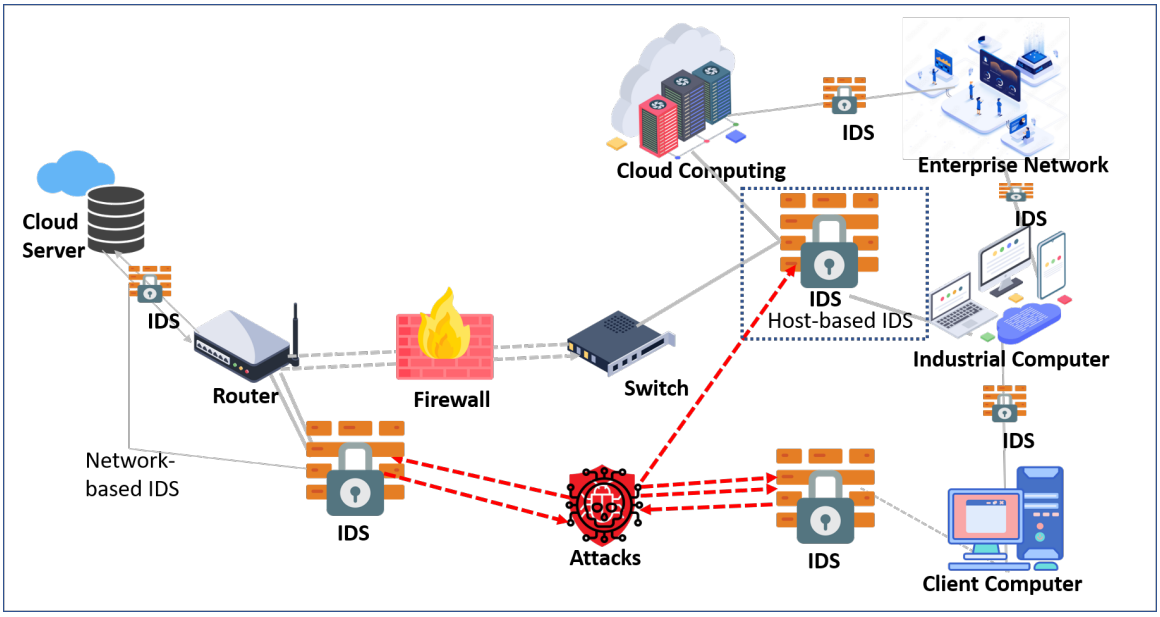
Following the successes achieved with the forgoing IDS method, other IDS were proposed using other methods including Expert Systems to develop IDS such as Network Security Monitor (NSM) (HEBERLEIN *et al.*, 1989) that performed masking on access matrices for anomaly detection on a Sun-3/50 workstation; the Information Security Officer's Assistant (ISOA), a 1990 prototype that considered a variety of strategies including statistics, a profile checker, and an expert system, (WINKLER, 1990) and ComputerWatch at ATT Bell Labs used statistics and rules for audit data reduction and intrusion detection (DOWELL, 1990). Machine Learning was implemented by Viegas and his colleagues in 2015 to propose an anomaly-based ID engine targeted on System-on-Chip for use on Internet of Things (IoT) devices using Decision Tree, Naive-Bayes and K-Nearest Neighbors and achieved more energy-efficiency compared to other IDS models (VIEGAS *et al.*, 2016).

IDS can be implemented on computer systems or on the network interface to keep track of event. While those on the computer systems monitors the events mostly related to the behaviour of the operating systems, IDS on the network interface analyzes incoming traffics/packets. Some ID systems are designed with intelligence, thus, they can take appropriate actions when necessary. We present a typical implementation of an IDS in a real scenario in Figure 2.2. These systems function autonomously (CENKERAMADDI *et al.*, 2020; AL-GARADI *et al.*, 2020). According to Figure 2.3, IDS can be categorized into two types (PHARATE *et al.*, 2015; LAKSHMINARAYANA; PHILIPS; TABRIZI, 2019), namely:

- *Types of attacks*: Based on the types of network attacks each IDS is designed to handle (detection approach), it can be subdivided into Misuse detection system and anomaly detection system (MILENKOSKI *et al.*, 2015).

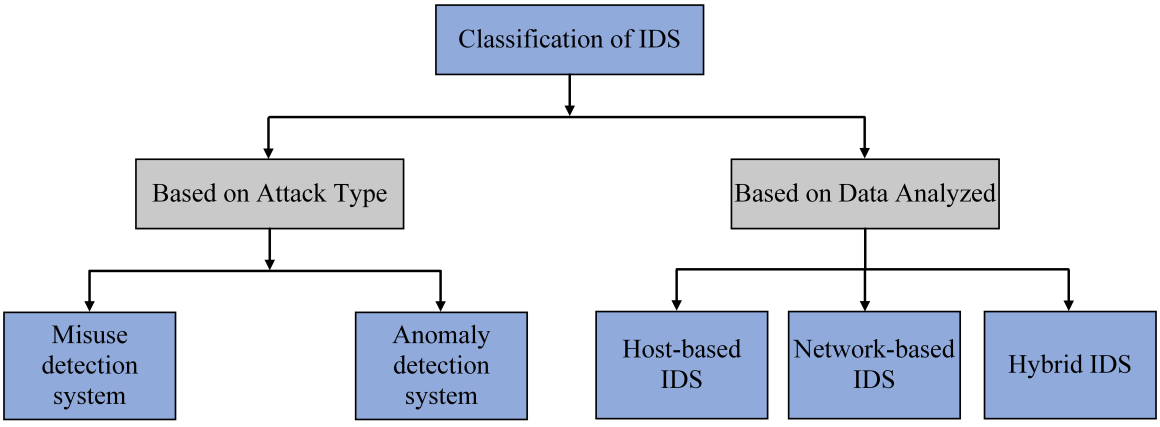
- *Types of analyzed data*: Here, the emphasis is laid on the type of data the IDS analyzes. As a result, under this domain, IDS are divided into the following subcategories: Network-based Intrusion Detection System (NIDS)(VIGNA; KEMMERER, 1999), host-based Intrusion Detection System (HIDS) (LIU *et al.*, 2018; VOKOROKOS; BALÁŽ, 2010), and hybrid Intrusion Detection System (H-IDS) (MILENKOSKI *et al.*, 2015)

Figure 2.2 – A typical IDS implementation scenario showing the Network-based and Host-based IDS



Source: Author (2022)

Figure 2.3 – A general classification of IDS methods



Source: Author (2022)

2.1.1 Categorization Based on the Attack Types

The category which can also be referred to as classification according to detection mechanism has the following variants:

a) **Misuse Intrusion Detection System (MIDS)**

MIDS, also known as signature-based IDS (SIDS) (WANG *et al.*, 2015), create libraries of known attack signatures and raise alarms when network traffic or system operations match any of the attack signatures in the library. The library, which is predefined by the network administrator, attempts to list and save all potential anomalous network and system activity. Other behaviors, both known and unknown, are accepted as normal (GHARIB *et al.*, 2019).

As a result, the abuse detection system can successfully discover previously identified attack methods. In misuse-based detection, attacks are represented by signatures or attributes, according to (MUKKAMALA; SUNG; ABRAHAM, 2005). However, in terms of detecting zero-day attacks, this strategy is ineffective. The key problem is determining how to create permanent signatures with all conceivable variants and non-intrusive actions to reduce false-negative and false-positive alarms.

One major limitation of the MIDS is the frequent chastisement for its high rate of false and missing alarms. As the number of zero-day attacks grows, this method is becoming increasingly obsolete. To get around the established intrusion libraries, intruders can simply conceal their attack methods.

b) **Anomaly Intrusion Detection System (AIDS)**

AIDS does not necessitate awareness of previously unknown attacks. Chandola et al. and Bhuyan et al. presented a thorough examination of the anomaly detection system (CHANDOLA; BANERJEE; KUMAR, 2009; BHUYAN; BHATTACHARYYA; KALITA, 2013). In the field of IDS, AIDS is sub-categorized into: network-based anomaly detection systems and host-based anomaly detection systems . Network-based anomaly detection systems detects abnormalities in regular network traffic. Host-based anomaly detection systems monitors hosts to detect deviations from regular system activity. ID systems where the regular behavior does not alter frequently implements host-based anomaly detection systems (NOBLE; COOK, 2003). Host-based anomaly detection systems according to Mutz *et al.* (2006), Tandon e Chan (2005) uses system

call traces to construct normal databases or data mining models of normal system responses. Then, these datasets or models may be employed as criteria for anomaly detection as opined in Jose *et al.* (2018). As a result, zero-day-attacks can easily be detected by the anomaly IDS.

The major challenge of host-based anomaly detection systems is a high false-alarm rate (FAR). As the number of new Linux programs grows, so does the quantity of new system call traces. Because host-based anomaly detection systems only stores normal databases or data mining models of known activities, new normal system call traces that do not match the databases or models may be misidentified as intrusions. A comparison of the advantages and disadvantages of MIDS and AIDS is presented in Table 2.1

Table 2.1 – Comparison between Misuse Intrusion Detection System (MIDS) and Anomaly Intrusion Detection System (AIDS)

Type	Advantages	Disadvantages
MIDS	Very excellent in detecting intrusions with little false alarms	If a prior attack is significantly modified to become a new version, the system will not detect this new form of an identical attack.
	Very apt in detecting existing attacks.	Requires regular update with a new signature.
	Has simple design principle	Does not have the capacity of detect the zero-day attack.
	SIDS is a security system that detects attacks based on recognized signatures.	Not suitable for detecting multi-step attacks.
	Easily identifies the intrusions.	Little understanding of the insight of the attacks.
AIDS	Can be used to detect new forms of attacks.	Because AIDS cannot handle encrypted packets, the attack can go unnoticed and pose a hazard.
	Can be used to create intrusion signature	Has high false positive alarm rates
		Needs initial training
		Uneasy to develop normal profile for a very dynamic computer system.
		Unclassified alerts.

Source: Adapted from Khraisat *et al.* (2019b)

2.1.2 IDS Based on the Analyzed Data

Depending on the packets the IDS will analyze or implementation interface, we have the following:

a) **Host-Based Intrusion Detection System (HIDS)**

This software keeps track of events on hosts, such as system or shell logs, in order to detect unwanted action. On host auditing data, HIDS may use different data mining methods, such as artificial neural networks, to detect threats.

A HIDS's execution speed is usually calculated by combining all of the training and testing durations together. As a result of having to process a large number of fine-grained system call traces, the HIDS's performance may be limited. Meanwhile, maintaining and upgrading a large number of classic HIDS software installed on every host or virtual host in a network is time-consuming when compared to the number of NIDS devices. As a result, innovative HIDS approaches that may reduce execution time while keeping appropriate detection accuracy are required (WUNDERLICH *et al.*, 2019; LIU *et al.*, 2018). Furthermore, classical HIDS does not demonstrate sufficient robustness in the face of modern persistent threats. As a result, future HIDS will need to work in tandem with other security mechanisms to achieve better performance.

b) **Network-Based Intrusion Detection System (NIDS)**

NIDS monitors network communications for intrusions, by applying data mining algorithms to network traffic data to discover abnormalities. Currently, NIDS-related topics are being aggressively investigated, with remarkable results. Tan *et al.* (2013), for example, suggested a technique for detecting denial-of-service (DoS) issues on networked server systems using multivariate correlation analysis. A distributed system made up of numerous hosts and network connections is the source of events (and the object) for the analysis in NIDS. The purpose of NIDS is to identify network-based threats that may span several hosts. Consequent upon network-level monitoring and distribution complexities, the following new parameters are necessary in setting up an IDS (VIGNA; KEMMERER, 1999):

(i) Networks generate a lot of data (events). Due to this, a NIDS should include approaches that allow the event collectors to listen for only the relevant events. This configuration can be performed by the Network Security Officer (NSO).

(ii) Only a portion of the network is usually accessible to relevant events (especially in the case of large networks). As a result, some means of selecting where to check for events should be included in an NIDS.

(iii) Local processing of events (information) is important as to reduce the amount of network data generated by the NIDS.

(iv) A NIDS must be scalable having the easy of communicating with other NIDS at least at a hierarchical level.

(v) To be most effective, NIDS must communicate with host-based IDS, allowing usage patterns to include network and OS events.

NIDS is often deployed between an organization's Internet and Intranet as hardware. This is useful since the NIDS can detect infiltration very immediately. However, because it is difficult for a typical NIDS to handle encrypted packets sent over the network, NIDS implementation may cause network flow speed to be decreased. Internal threats are also difficult to detect with NIDS in this case.

c) **Hybrid Intrusion Detection Systems**:

These are the IDS formed by the joining of HIDS and NIDS and sometimes, Misuse Intrusion Detection System (MIDS) and Anomaly Intrusion Detection System (AIDS). This is done to increase the performance of the IDS as indicted in the work of Meryem e Ouahidi (2020). Also, Khraisat *et al.* (2019a) combined the benefits of the MIDS and AIDS to propose a hybrid IDS framework able to detect both well-known intrusions and zero-day attacks yielding high level of detection accuracy and low rates of false-alarm. The proposed framework was evaluated using the Bot-IoT dataset, which contains both genuine IoT network traffic and various sorts of attacks. The results reveal that the hybrid IDS has a greater detection performance and a lower false-positive rate than the other approaches (KHRAISAT *et al.*, 2019a).

2.2 Computer Network and Different Types of Network Attack

A computer network is a connection of computers or computing systems to share resources, such as printers, applications, storage and processing units. The computers use common communication protocols over digital interconnections which can either be wired, optical or wireless radio-frequency medium to communicate with each other. Nodes of a computer network are usually identified by specific streams of numbers called network address or sometimes hostnames.The transmission medium used to carry signals, bandwidth, communications protocols to organize network traffic, the network size, the topology, traffic control mechanism, and organizational intent are common criteria used in the classification of computer Networks.

Network attacks can be initialized from an external environment or from inside of the network. The attack in the Wide Area Network (WAN) is the most challenging one due to the vast number of networked devices that can be affected. The IDS is one of the tools which try to ensure the safety of network communication from different network attack types. Understanding the reasons for the various types of attacks is imperative to knowing the target platforms and the purpose of the attack. These primarily are the determinants of the type of attack a system will be exposed to at any given time. For example, a phishing attack (attack type) is carried out on the web or via e-mail (attack platform) and is configured to steal personal information, such as card numbers and security passwords. Other attacks are designed to simply disrupt network traffic, hence cause annoyance.

The most common attack profiles are presented and discussed as follows.

2.2.1 Denial of Service (DoS)

This is an attack-type that is intended to make a service on the network become completely or temporarily unavailable, thus not responding to users and other systems requests (PENG; LECKIE; RAMAMOHANARAO, 2007; GASTI *et al.*, 2013). Vulnerabilities in operating systems or services are mostly exploited in this type of attack. To perform this attack, the attacker sends several queries to network service to deplete the resources of the target system. Queries sent over the network may come from either a single source or multiple sources. With the use of only one source address in the attack act, the attack can easily be detected and blocked by the firewall. However, protection becomes more difficult

when the network attacker employs address list to conceal the attack. In this case, a more standard measure is required to prevent the attack.

2.2.2 Distributed Denial of Service (DDoS)

Similar to DoS, this attack has the same objective, however, it uses distributed systems to carry out the requests to exhaust the target's resources (DOULIGERIS; MITROKOTSA, 2004). For example, Honda *et al.* (2015) used the Internet Control Message Protocol (ICMP) to send requests which were fired repeatedly against a target by implementing the help of the ping software. The attack was performed through distributed zombie computer systems.

The dispersion of attack sources is a method to make identification and blocking more difficult since the firewall may block requests that are not related to incursion. Botnets are self-spreading and self-organizing networks of hacked devices called (bots) that may be used to carry out damaging actions in a coordinated way under the supervision of a botmaster (PIJPKER; VRANKEN, 2016). Because of the large number of vulnerable devices linked to the Internet, attackers may install systems on several computers and execute coordinated attacks on their victims from these infected machines.

2.2.3 Infiltration

This happens from within the network usually through the exploitation of some software flaw in one of the network devices. Authors in Sharafaldin, Lashkari e Ghorbani (2018) showed that after successfully exploiting a network or computer device vulnerability, a backdoor is usually setup to initiate different network attacks. Internet Protocol (IP) address scanning, Port Scan, and service enumeration are common examples of such attacks that may be carried out with the open-source Network Mapper (Nmap) software. Nmap has several tools for probing computer networks, such as host discovery and service and operating system identification. Scripts that enable more advanced service detection can be used to extend these functionalities vulnerability detection, and other features (LYON, 2014).

During a scan, Nmap can adapt to network constraints like latency and congestion. Kaur e Kaur (2017) utilized the NMap tool to perform penetration testing on networking infrastructure to discover security flaws and exploit target Operating Systems (OS). The

information acquired about the OS was tested using the Linux operating system program, which exposed the network multiple security flaws.

2.2.4 Web attack

This form of attack is more visible on the Internet resulting from a large number of web applications available on the Internet. During web Application development, the attacker can insert malicious code into web page forms and upload the codes to the server setting attack mode in the active state. Cleaning the data entry handling of server requests are two important steps the App developer takes to handle the occurrence of this attack. Meanwhile, it is the most widely used approach to perform network invasion due to high vulnerabilities it presents. Some common types of this class of attack are SQL Injection (KAREEM *et al.*, 2021) and Cross-Site Scripting (XSS) (RODRÍGUEZ *et al.*, 2020).

During the execution of SQL Injection attack, the network intruder passes a string in the form of theoretical query to the application's database, thereby forcing the server to return true values, hence, breaking the authentication protocol of the database (KAREEM *et al.*, 2021). In XSS, an attacker manipulates user interactions with susceptible applications thereby bypassing the network's original policy which is intended to separate the websites into various categories. XSS flaws typically allow an attacker to impersonate a target user, conduct any activities that the user is capable of performing, and access any of the user's data. The attacker can gain full access control over application functionality and data of any website where the XSS attack is successful if the target user has privileged rights to the website (RODRÍGUEZ *et al.*, 2020).

2.2.5 Brute force

This type of attack occurs when the attacker illegally tries to obtain credentials to access systems (HONDA *et al.*, 2015). In addition to stealing credentials, an attacker can try to discover hidden documents through brute force (SHARAFALDIN; LASHKARI; GHORBANI, 2018). Due to the number of attempts, this is one of the least intelligent methods of attack as it is easily detectable by security monitoring systems (PARK *et al.*, 2021).

In brute-force cyberattacks, the attacker attempts to gain the system's account information by submitting all conceivable values as account inputs. The techniques used in brute-force attacks are classified as dictionary attack methods, which attempt all strings in a

pre-arranged listing, and random sequence methods, which attempt all possible string combinations in a sequence (PARK *et al.*, 2006). Controlling access when incorrect passwords are typed more than a given number of times is a standard method of preventing brute-force attacks. In the case of servers, the account lock policy defines the threshold for login failure, and if the threshold is surpassed, access to the account is restricted (PARK; KANG, 2016).

Additionally, the complexity of the password considering different type of characters, password usage duration, and minimum password length are set in the password policy to protect the system and its account from being snatched due to a simple password. While network systems do not block access following login failures exceeding the threshold, they do prevent unauthorized access through the Access Control List (ACL). Thus, Internet routers have logged on many attempts of unauthorized external access, and these can serve as a valuable reference for the access control policy.

2.2.6 Probing/Port scan

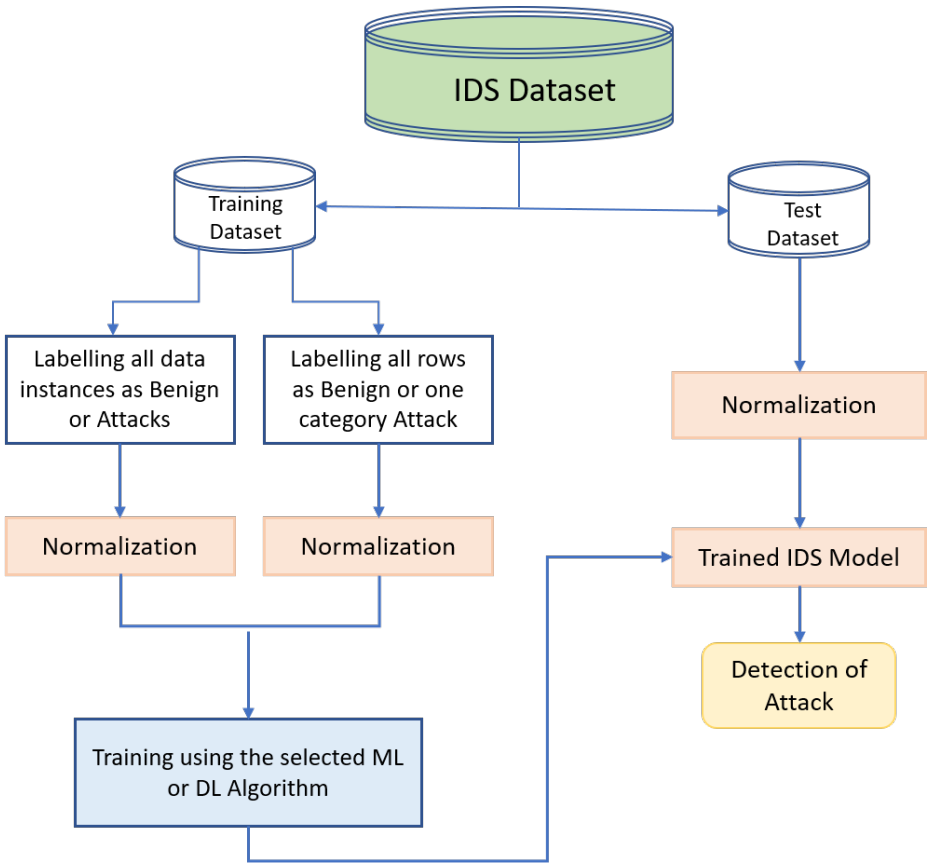
This type of attack aims to obtain information from the network to be used later by some other type of attack. In this case, the network is scanned for open Transmission Control Protocol (TCP) (POSTEL *et al.*, 1981) and User Datagram Protocol (UDP) (POSTEL, 1980) ports, thus mapping them for further exploitation of vulnerabilities of some service on these ports. Among the Probing techniques used are: UDPScan and SYNScan (BALRAM; WISCY, 2008).

2.3 Frameworks for Implementing IDS

This section presents an overview of the various approaches implemented in recent years to improve the detection accuracy of IDS and reduce false alarms which have been very problematic as long as cyber security is concerned. Figure 2.4 shows the flowchart for the design of an ML or DL IDS-based system. The Process begins with data acquisition. A machine or deep learning engineer identifies the most appropriate dataset required to train the model for the specific task. In the case of IDS, there is a publicly available dataset that can be used to train and test the models. These datasets are discussed extensively in Section 2.4 of this work. After the data is acquired, data preprocessing is performed and further split into the training and testing dataset. The model is then trained with the training set after normalization using the most appropriate algorithm. To validate the performance

of the model some metrics are used, such as the accuracy, $F-1$ score, AUC, sensitivity, among others. The trained model is then tested with the test dataset, and then it can be deployed for industrial application.

Figure 2.4 – A General Flowchart for developing an Intrusion Detection System using Deep Learning



Source: Adapted from Chandrashekhar, Tamane e Bharati (2021)

IDS methods can be categorized into four main groups: Statistics-based (LIN; KE; TSAI, 2015; MEHROTRA; SAXENA, 2018), Knowledge-based (LUNT *et al.*, 1989; MORE *et al.*, 2012), machine Learning-based (HALIMAA; SUNDARAKANTHAM, 2019; VERMA; RANGA, 2020) and Deep Learning-based (WANG *et al.*, 2020; LEE *et al.*, 2021). The Statistics-based strategy entails collecting and analyzing each data record in a group of objects, as well as developing a statistical model of typical user behavior. Knowledge-based methods, on the other hand, attempt to identify the requested actions from existing system data such as protocol specifications and network traffic instances, whereas ML and DL methods acquire complex pattern-matching capabilities using diverse artificial neural network (ANN) algorithms for training data to determine classification models. We leverage the special functionalities of the ML and DL category to develop our own IDS model in this work.

2.3.1 Statistics-based techniques

In a Statistics-based IDS, a low probability occurring attack behaviour profile are detected and flagged as probable intrusion or threats by the model which has been developed using probability distribution approach. Usually, the measures of central tendency (median, mean, mode) and standard deviation of transmitted packets are frequently considered by statistical IDS. Instead of monitoring data traffic, each packet is examined to establish the identity of the flow. Statistical IDS are used to detect any deviations from typical behavior in present conduct (JOSE *et al.*, 2018). For effective performance, a statistical IDS uses one of the following approaches in its implementation:

1. Univariate: This approach is utilized when a statistical normal profile is constructed for only one measure of behavior in computer systems, as the term "Uni"signifies one. Univariate IDS searches for anomalies in each particular metric (BRAEI; WAGNER, 2020).

2. Multivariate: It is based on the relationships between two or more measurements to understand the connections between variables. This approach would be useful if experimental evidence showed that combining correlated measures yielded better categorization than analyzing them independently. Braei e Wagner (2020) investigated a multivariate quality control approach for detecting intrusions by constructing a long-term profile of normal activity. Estimating distributions for high-dimensional data has proven to be a significant challenge in the implementation of multivariate statistical-based IDS.

3. Time series model: A time series is a collection of observations performed over a specific period. A fresh observation can be judged abnormal if the likelihood of its occurrence at that moment is too low. Braei e Wagner (2020) processed intrusion detection alert aggregates using time series, (BRAEI; WAGNER, 2020). Chauhan e Agarwal (2021) provided a technique for identifying network irregularities by studying the sudden fluctuation detected in time series data .

2.3.2 Knowledge-based techniques

In this approach, otherwise referred to as expert method, a knowledge base containing legitimate traffic profiles of all network activities is required. Any action that deviates

from these standard profiles is considered an intrusion. In contrast to the other types of IDS, the standard profile model is typically built on human knowledge in the form of a set of rules that attempt to define normal system activity. The main advantage of knowledge-based techniques is the ability to reduce false-positive alarms because the system is aware of all normal behaviors (WAN *et al.*, 2019).

In a continuously evolving computing ecosystem, however, this type of IDS requires consistent updating on expertise about predicting normal behavior, which is a time-consuming operation due to the difficulty of acquiring information regarding various usual behaviors. One of the following strategies can be used to conduct knowledge-based IDS training.

1. Finite State Machine (FSM): The FSM is a computational paradigm for describing and controlling the application logic. This model is used to develop an Intelligent system for IDS. States, transitions, and activities are commonly used to depict the model. The data from the past is examined by a state. Any differences in input data, for example, are noted, and a transition is made considering the aforementioned variation (RAMESH; MENEN, 2020). An FSM can represent the normal system behavior, and any variation beyond this FSM is considered a breach.

2. Description Language: The syntax of rules that are used to express the characteristics of a specific attack is specified by the description language. N-grammars and universal modelling language (UML) are examples of description languages that can be used to create these rules according to Fiorese e Montino (2021).

3. Expert System: An expert system comprises several rules that define attacks, having the rules manually defined by a knowledge engineer working in collaboration with a domain expert. This technique when applied to the IDS does not yield very accurate, and precise result due to human inefficiencies which may likely occur as observed by Chauhan e Agarwal (2021).

4. Signature Analysis: It is the earliest IDS methodology to be used. It is based on the concept of pattern or string matching. An incoming data packet is analyzed word by word with a distinct signature pattern matching. A warning is raised if a signature is matched. If not, the information in the traffic is matched to the signature database's next to the signature (MASDARI; KHEZRI, 2020; LEEVY; KHOSHGOFTAAR, 2020).

The major drawback of this approach is that malicious codes can be embedded into the strings and transmitted over the network. This information can be interpreted as normal because of the inability of the signature to detect the embedded codes.

2.3.3 IDS based on Machine Learning techniques

ML involves the design of computer algorithms that aid computing devices to improve in performance automatically through experience and by the implementation of data (HUSSAIN *et al.*, 2020b). ML techniques have been applied extensively in the area of IDS. Several algorithms and techniques, such as clustering, ANN, association rules, decision trees, genetic algorithms, and nearest neighbor methods, have been applied for discovering the knowledge from IDS-datasets (AHMAD; ANWAR; HAQUE, 2020; BISWAS *et al.*, 2018).

Figure 2.5 presents the broad classification of ML techniques showing that an algorithm can be a shallow learning algorithm or deep learning algorithm. Further to this, ML methods can also be either supervised, unsupervised or reinforcement depending on its learning pattern.

Using labeled training data, supervised learning-based IDS algorithms detect intrusions. Training and testing are usually the two stages of a supervised learning approach. The selection of characteristics and classifications, as well as the construction of an algorithm that learns from the data sample, are all part of the training step. Every data record is a sample pair of a network or host data source and an associated output (label) that can be classed as an attack or normal. (SARAVANAN; SUJATHA, 2018).

Next, feature selection is applied to eliminate unnecessary features from the data (CAI *et al.*, 2018; AL-TASHI *et al.*, 2020). A selected supervised learning method is used afterwards to train the classifier to learn the existing connection between the input data and the labeled output value using the training data from the selected features. In the literature, a large range of supervised learning strategies has been investigated, each with its own set of benefits and drawbacks. During the testing step, the trained model is utilized to categorize the unknown data as intrusion or normal. The resulting classifier is then transformed into a model that predicts the class to which the input data may belong given a collection of feature values. A classifier's performance in predicting the proper class is measured using a variety of criteria set.

Unsupervised learning, often referred as unsupervised ML, analyzes and clusters unlabeled datasets using machine learning algorithms. Without the need for human intervention, these algorithms uncover hidden patterns or data groupings. Its capacity to find similarities and contrasts in data makes it an excellent choice for exploratory data analysis, cross-selling techniques, consumer segmentation, and image identification, among other applications.

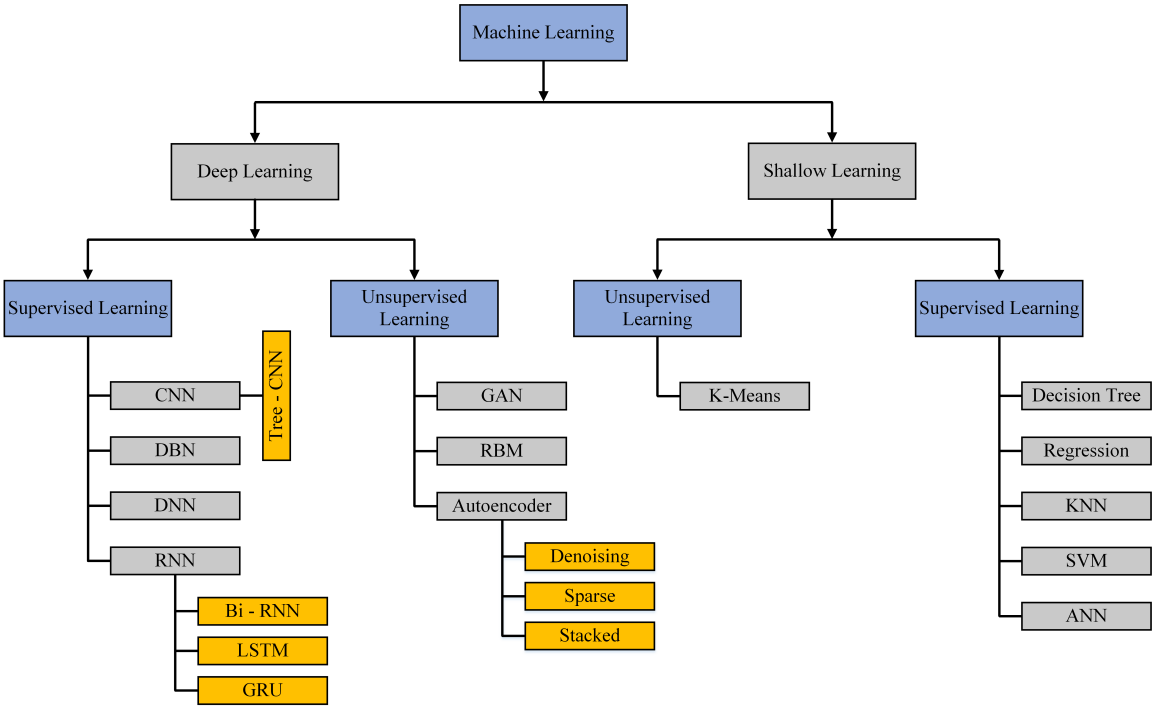
Unsupervised learning models are utilized for three main tasks: clustering, association, and dimensionality reduction. Clustering is a data mining technique that groups unlabeled data based on their similarities or differences. Clustering algorithms are used to process raw, unclassified data objects into groups represented by structures or patterns in the information. Clustering algorithms can be categorized into a few types, specifically exclusive, agglomerative, overlapping, and probabilistic. Exclusive clustering is a type of grouping in which a set of data can only be found in one cluster at a time. This type of clustering is often known as "hard"clustering. A well known example of the exclusive clustering is the K-Means algorithm. In the agglomerative clustering technique, every data is treated as a cluster and the iterative unions between two nearest clusters are used to reduce the number of total clusters. The Hierarchical clustering algorithm represents a typical example of agglomerative. Unlike the agglomerative clustering, the overlapping clustering uses fuzzy sets to cluster data, having each point belonging to either two or more clusters but with variant degrees of membership. The probabilistic clustering uses probability distribution to create clusters. Other Important clustering types are: (i)Hierarchical clustering (ii) K-means clustering (iii) K-NN (iv) Singular Value Decomposition (v) Principal Component Analysis (vi) Independent Component Analysis (GENTLEMAN; CAREY, 2008; JOHNSON, 2022).

Association rules enable you to create associations between data elements in huge databases. The goal of this unsupervised approach is to find interesting correlations between variables in massive databases. In dimensionality reduction, latent variable models are widely used for data preprocessing, feature reduction or dataset decomposition into multiple data components are achieved with latent variable models approach.

Reinforcement learning refers to a ML training method based on rewarding desired behaviors and/or punishing undesired ones. This involves a developed algorithm or pattern to reward every accurate behavior exhibited by the model. The program assigns positive values to all accurately predicted results and negative values to undesired results. In regards

to IDS, many authors have applied the approach to train models that obtained good performance results (SETHI *et al.*, 2020; LOPEZ-MARTIN; CARRO; SANCHEZ-ESGUEVILLAS, 2020; SERVIN; KUDENKO, 2005).

Figure 2.5 – General Taxonomy of Machine Learning Techniques



Source: Adapted from Liu e Lang (2019)

Table 2.2 – Advantages, Disadvantages and Improvement procedures for Shallow ML Algorithms

Algorithms	Advantages	Disadvantages	Improvement Measures
ANN	handles non linear data; performs excellent fitting	responds aptly to overfitting; tendency to get stuck with local optimum; Model training requires a lot of time	used improved optimizer, activation functions, and loss functions
SVM	Learn valuable information from small dataset; High capability	Do not perform well on big data or multiple classification tasks; Sensitive to kernel function parameters	Optimized parameters by Particle Swarm Optimization (PSO)
KNN	Apply to large data; Good response to noise; Train quickly; Able to work with nonlinear data	minority class has low accuracy; takes a long time to perform testing; Sensitive to the parameter K	Optimization of parameters via Particle Swarm Optimization (PSO); Balanced datasets based on Synthetic Minority Oversampling TEchnique (SMOTE)
Naïve Bayes	Robust to noise; Able to learn incrementally	Low response with attribute-related data	Imported latent variables to relax the independent assumption
LR	Simple can be trained rapidly; Automatically scale features	Has low performance with nonlinear data; Apt to overfitting	Imported regularization to avoid overfitting
Decision Tree	Automatically select features; Strong interpretation	Majority class has preference; Correlation of data is not considered	Balanced datasets with SMOTE; Introduced latent variables
K-mean	Simple, can be trained rapidly; Strong scalability; Can fit to big data	Low performance on non-convex data; Initialization set-up is important; Sensitive to K (parameter)	Improved initialization method

Source: Liu e Lang (2019)

2.3.3.1 Shallow ML Algorithms

Here, we explain some of the traditional ML methods generally called (shallow models) for IDS which can either be supervised or supervised such as Support Vector Machine (SVM), Artificial Neural Network (ANN), K-nearest Neighbor (KNN), Decision Tree, Naïve Bayes, Logistic Regression (LR), K-mean clustering, and combined or hybrid methods (LIN;

KE; TSAI, 2015; MISHRA *et al.*, 2018; ALMSEIDIN *et al.*, 2017; KAYACIK; ZINCIR-HEYWOOD, 2005).

Table 2.2 gives details on the advantages, disadvantages and improvement techniques for the shallow ML algorithms.

a) **Support Vector Machines (SVM)**

A splitting hyperplane defines SVM, a discriminative classifier. SVMs employ a kernel function to translate training data to a higher dimension space, allowing incursion to be categorized linearly (MEYER; WIEN, 2015). SVMs are widely renowned for their capacity to generalize and are most useful when the number of characteristics is big but the number of data points is modest. By using a kernel, you may separate different kinds of hyperplanes, such as linear, Gaussian Radial Basis Function (RBF), polynomial, and hyperbolic tangent. Many characteristics in IDS datasets are unimportant or have little influence on classifying data items. As a result, feature selection are considered during SVM model training. SVM can be used as well to classify data into numerous categories. (BYVATOV; SCHNEIDER, 2003; HEBA *et al.*, 2010; BHATI; RAI, 2020a).

b) **K-Nearest Neighbors (KNN) classifier**

The k-Nearest Neighbor (k-NN) technique is a traditional non-parametric classifier used in ML (PETERSON, 2009). The goal behind these methods is to give an unlabeled data sample a name considering the classification of its k nearest neighbours (where k represents an integer defining the total number of neighbors which are to be considered) (HAN; KARYPIS; KUMAR, 2001).

c) **Naïve Bayes** This method is based on the Bayes' principle, with robust independence assumptions for the characteristics. Using conditional probability equations, Naïve Bayes solves queries like *what is the likelihood that a specific type of attack is occurring, given the observed system activities?* The Naïve Bayes method is based on traits that have different odds of arising in assaults and regular activity. The Naïve Bayes classification model is one of the most popular in IDS because of its ease of use and computation efficiency, both of which are derived from its conditional independence assumption characteristic. However, if this independence assumption is not valid,

the system does not perform well, as proved on the KDD'99 intrusion detection dataset, which includes complicated attribute relationships. Many research works implemented this approach with good classification accuracy (VEMBANDASAMY; SASIPRIYA; DEEPA, 2015; PELING *et al.*, 2017; SARITAS; YASAR, 2019; SHINDE *et al.*, 2015).

d) **Logistic Regression (LR)** The LR algorithm is widely used in binary classification problems. LR utilizes gradient descent method to solve the optimal parameters of loss function and improve the global convergence of the optimization algorithm. This algorithm presents many iterations and it has a slow response to train large amounts of data (ZOU *et al.*, 2019). Thus, in the literature, there are some proposals to improve the global performance of the LR algorithm (ZOU *et al.*, 2019; LIU, 2018; YANG; LI, 2019)

e) **Decision Tree**

Decision Trees (DTs) are data structures composed of elements called nodes. Following a hierarchical model, the tree has a root node, where the tree is started, sequentially the tree is composed of child nodes, where each node can have other children or sub-trees. When a node has no children, it is known as a leaf or terminal node. The initial data enters at the root of the tree and passes through the decision nodes until reaching the leaf node, which in turn, presents the result of the processing. According to Khraisat e Alazab (2021), DT is a sequential model, which logically combines a sequence of simple tests. Each test compares a numeric attribute with a threshold (set) value or a nominal attribute with a set of possible values.

A DT classifies an unseen test instance through a series of decisions (MUSA *et al.*, 2020). Due to its simple configuration and ease of implementation, DT is particularly well known as a single classifier. DT can be divided into 2 categories: (i) Regression tree, which considers a range of class labels based on numerical values; and (ii) Classification tree, having a range of symbolic class labels (MUSA *et al.*, 2020). There are many different decision trees algorithms including ID3 (QUINLAN, 1986), C4.5 (QUINLAN, 2014) and CART (BREIMAN, 1996).

f) **K-Means Clustering Algorithm** K-Means algorithm is an unsupervised learning technique for pattern recognition that is widely used for clustering tasks. However, this algorithm presents some disadvantages, such as the need for setting the number of clusters (K), the arbitrary location of initial cluster centers, and the influence of the noise points on data treated. To avoid these shortcomings, some improvements were proposed in the literature (SINAGA; YANG, 2020; WANG; SU, 2011).

g) **Random Forest (RF)** RF is a well-known machine learning method (ensemble learning algorithm). It is referred to as an ensemble learning algorithm because it constructs several decision trees during the training phase of the algorithm and then uses the most popular result as its categorization. This is why it is called a forest. Because of its capacity to respond quickly and reliably, it may also be employed for regression or prediction issues (ATAWODI, 2019). The random forest creates a forest of independent subsets of the dataset.

   Random forest generates N decision trees from training set data. During this process, it randomly resamples the training set for each tree. Thus, N decision trees are obtained, each of which is different from the other. Finally, voting is performed by selecting new estimates from estimates made by N trees. The value with the highest rating is determined as the final value (KOSTAS, 2018).

   RF has the advantage of working well with very large and complicated datasets, reducing overfitting which is a common problem in ordinary DT classifiers, and handles missing values by replacing them with its own values. It also calculates and uses the importance level of the variables when making the classification. This feature is leveraged in selecting important features for machine learning problems. We have also implemented this in training our ML classifiers in this work.

h) **Artificial Neural Network (ANN)**

   The ANN algorithm attempts to be one of the most extensively used ML approaches (TEODORO *et al.*, 2021b), and it has been demonstrated to be effective in identifying various malware and network intrusion attempts. The backpropagation (BP) pattern is the most often used learning approach in solving supervised learning problems. The BP method examines the error of the network gradient concerning its changing weights. However, ANN-based IDS attack detection accuracy and precision still need

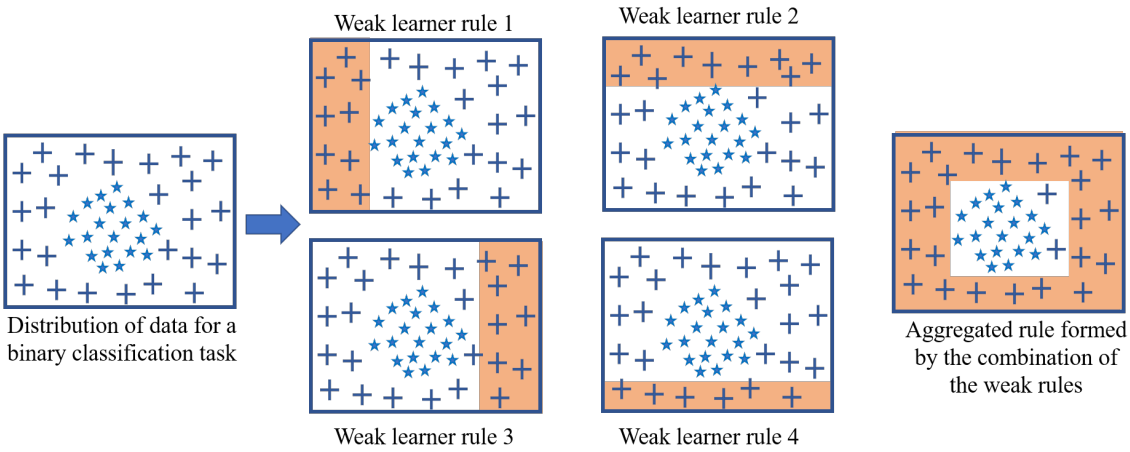
to be improved, particularly for less frequent attacks. The training dataset for less frequent intrusion is less than that for more frequent attacks, making it challenging for the ANN to appropriately understand the features of the attacks. Consequent upon this, accuracy of attack detection for fewer frequent attacks is reduced.

In this area of information security, low-frequency attacks can create critical damage to a security network if they are not aptly detected and overcome. For example, if User to Root (U2R) attacks go undetected, a malicious user can gain the root user's authorization permissions and thus undertake malicious activities on the victim's computer networks. Most often, outliers tend to be the less common attacks (MISHRA; SRIVASTAVA, 2014). Because ANNs frequently struggle with local minima, learning can also become extremely time-consuming. Meanwhile, the advantage with ANN is its ability to produce highly nonlinear models using one or more convolution layer that capture complex relationships between input attributes and classification labels. ANNs have become effective tools in many classification problems, including IDS, with the development of many variants such as recurrent and convolutional NNs. (BRASPENNING; THUIJSMAN; WEIJTERS, 1995).

i) **AdaBoost** (Adaptive Boosting), is a machine learning algorithm developed to improve classification performance. It is an ensemble algorithm which can be implemented with the use of the sklearn library in Python. The working principle of this algorithm can be explained thus: First, certain rules called "rough draft rules"are used to divide the data into groups. Each time the algorithm is executed, a new set of rules is created and added to the rough draft rules. This leads to obtaining many different weak and low performance rules generally referred to as "Basic rules".

   Over time as the algorithm continues to work many times, these weak rules are combined to achieve a stronger rule having better performance and more successful probability of yielding a better result. During this process, the Algorithm assigns a weighting coefficient to each weak rule, giving the highest coefficient value to the lowest error rate. These weight values come into play when final rules are selected. The final rule is created by giving priority to the high scored weak rules (SCHAPIRE, 2003; KOSTAS, 2018). Figure 2.6 demonstrates the basic operation of the AdaBoost classification algorithm.

Figure 2.6 – Demonstration of basic operation of AdaBoost Algorithm



Source: Adapted from (KOSTAS, 2018)

j) **Extra Tree (ET) Classifier**

This algorithm improves the performance of DT and RF by incorporating a more significant number of trees into its network. As a result, compared with other ML algorithms, it has the highest number of trees and computational resource requirements. This algorithm works on the principle of meta estimator and applies an averaging rule to increase predicted accuracy and reduce overfitting. First, the meta estimator fits several randomized decision trees on different sub-samples of the same dataset. Then, it aggregates the results of multiple de-correlated decision trees collected in a forest to output a classification result. The package is available in the $sklearn.ensemble.ExtraTreesClassifier$ library for use in any ML tasks (BHATI; RAI, 2020b).

k) **eXtreme Gradient Boosting (XGBoost)**

XGBoost is an extension of the implementation of Gradient Boosting Tree proposed by Friedman (2001). Because it offers parallel computation, cache awareness, a built-in regularization strategy to avoid overfitting, and tree optimization by a split-finding algorithm, XGBoost generally outperforms gradient boosting in terms of performance as it has a quick training and inference time according to Chen e Guestrin (2016). In Dhaliwal, Nahid e Abbas (2018), an efficient IDS model based on XGBoost was proposed for computer networks. The model was trained and evaluated on the

network socket layer-knowledge discovery in databases (NSL-KDD) dataset with an accuracy of 98.70%.

l) **Light Gradient Boosting Machine (LightGBM)**

Observing the high training time requirement for Gradient Boosting Decision Trees (GBDT), Ke *et al.* (2017) proposed two novel techniques to overcome the challenge based on *Gradient-based One-Side Sampling* (GOSS) and *Exclusive Feature Bundling* (EFB). This new implementation was named LightGBM which improved training and inference time of GBDT by 20%. Since its development, it has shown highly impressive results even in IDS systems as shown in Liu, Gao e Hu (2021) and Yao *et al.* (2021).

2.3.3.2 IDS Based on Deep Learning Techniques

DL is a branch of ML that deals with ANNs, which are algorithms motivated by the study of functions and biological organization of the human brain(ZHANG; PATRAS; HADDADI, 2019). It utilizes a hierarchical level of ANNs to carry out the process of ML leveraging the functioning pattern of the human brain. It was built on ML to improve the performance of information processing. Like ML, DL can be supervised, unsupervised or reinforcement in information processing. (KARATAS; DEMIR; SAHINGOZ, 2018; AMINANTO; KIM, 2016; KIM, 2017b).

DL networks are made up of numerous interconnected layers, the first of which is referred to as the input layer, the final as the output layer, and all levels in between as hidden layers. Each hidden layer is made up of multiple neurons, and the bias, weight, and activation function are the primary components that influence the network's strength.

DL techniques are further categorized into Generative, Descriptive, and hybrid methods (AL-GARADI *et al.*, 2020), in which Deep Boltzmann Machines (DBMs), Deep auto-encoders, Deep Belief Networks (DBNs), and Generative Adversarial Networks (GANs) are generative models while CNNs are considered a descriptive networks (SALLOUM *et al.*, 2020). Also, RNNs are categorized as both generative and descriptive models. Table 2.3 presents a deep comparison of the various deep learning algorithms classifying them as either supervised or unsupervised. The major function performed is also outlined. From the comparison, the autoencoders, RBM and GAN are unsupervised deep learning algorithms

while the DBN, Deep Neural Network (DNN), CNN and RNN are classified as supervised deep learning algorithms. The latter four are best used for classification problems in cases where labeled datasets are available.

Table 2.3 – Comparison of various Deep Learning Algorithms showing their basic functions

Algorithms	Learning Types	Suitable Data Types	Application
AE	unsupervised	Raw data; Feature vectors	used for Denoising, feature extraction and Feature reduction
RBM	unsupervised	Feature Vectors	Denoising; feature extraction and Feature reduction
DBN	supervised	Feature Vectors	Feature extraction; Classification
DNN	supervised	Feature Vectors	Feature extraction; Classification
CNN	supervised	Raw data; vectors of features; Matrices	Feature extraction; Classification
RNN	supervised	Raw data; vectors of features; Sequence data	Feature extraction; Classification
GAN	unsupervised	Raw data; Feature vectors	Data augmentation; GAN training

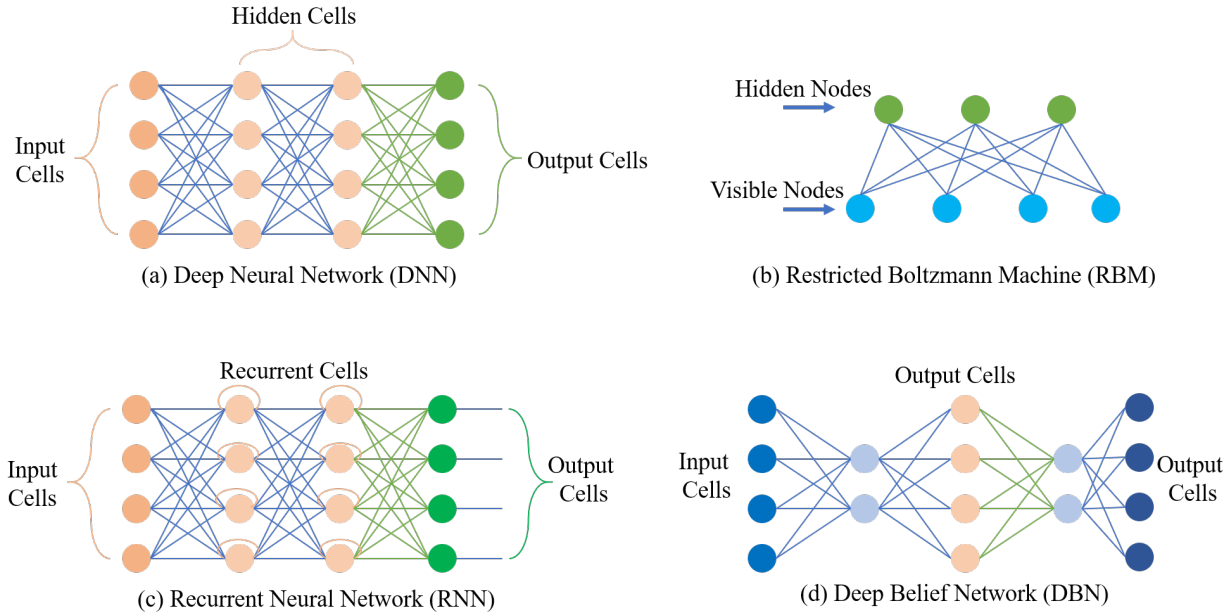
Source: Author (2022)

a) **Deep Neural Network**

The architecture of DNN is as shown in Figure 2.7a. The network contains three main layers which include: the input cells, the hidden cells and the output cell. These cells or networks are connected deeply with each other such that the inputs and outputs of the network are well represented. In Software Defined Networks (SDN), Tang *et al.* (2016) proposed an intrusion detection system based on a DNN. The proposed IDS system is implemented in the SDN controller which can monitor all the OpenFlow switches. The authors used the NSL-KDD dataset which consisted of four different attack categories, namely, (i) DoS attacks, (ii) R2L attacks, (iii) U2R attacks, and (iv) Probe attacks. A two-class classification was performed during the process to classify the attacks as either normal or anomaly. The major success recorded in the experiment was achieving a lower learning rate which was of better performance than others with the higher receiver operating characteristic curve (ROC).

Using the same dataset as Tang *et al.* (2016), Potluri e Diedrich (2016) developed a DNN model to handle deep-category classification problems in huge network data. The dataset which contained 39 different network attacks were grouped into four

Figure 2.7 – Basic Deep learning architecture for: 2.7 a: Deep Neural Network, 2.7 b: Restricted Boltzmann Machine, 2.7 c: Recurrent Neural Network and 2.7 d: Deep Belief Network



Source: Author (2022)

categories (TANG *et al.*, 2016), but regarding this, the developed model achieved a high attack detection accuracy of about 96%.

As the network continues to enlarge, the transport systems are also prone to attacks. Hence, Kang e Kang (2016) proposed an intrusion detection system based on the DNN for vehicular networks. Malicious data packets were injected into an in-vehicle controller area network bus to perform the attack scenario. The injected malicious packets caused network obstruction in the vehicular movement. The proposed system inputs the feature vector to the input nodes to classify packets into two classes (a normal packet and an attack packet). The ReLU activation function was used to compute the output of the network which was linked to the other hidden layers of the network. With the false-positive error less than 1–2%, the proposed system achieves a detection accuracy of about 98%.

In cyberspace, attempting to classify cyber-attacks, Zhou *et al.* (2018) proposed an intrusion detection system based on the DNN. Specifically, the system uses three phases, namely, (i) data acquisition (DAQ), (ii) data pre-processing, and (iii) deep neural network classification. The system achieves an accuracy of 96.3% for the SVM model with a learning rate of 0.01, training epochs 10, and input units 86. The results

show this approach to outperform the following three machine learning approaches (i) linear regression, (ii) random forest, and (iii) k-nearest neighborhood.

Another application of the DNN algorithm was carried out by Kim *et al.* (2017) to investigate evolving attacks on special networks using the KDD 1999 dataset. The proposed intrusion detection model uses two parameters, namely, four hidden layers and 100 hidden units. The ReLU function is used as the activation function and the stochastic optimization method for deep neural network training. The proposed model achieves an accuracy of approximately 99% with a false alarm rate of 0.08%

b) **Auto-Encoders**

Auto-encoders are feed-forward ANNs that maintain the same number of neurons in the input and output layers of the network. It can have several hidden layers and builds its inputs intending to minimize the differences between the output and the input. Each auto-encoder has two steps: an encoder that is used for mapping the input data into the code, and a decoder used for constructing input data from the code. In more specific terms, auto-encoders support unsupervised learning of dataset encoding for dimensionality reduction, by training the network to ignore the signal noise (LANGE; RIEDMILLER, 2010). During training, the diversity that exists between the input of the encoder and the output of the decoder is reduced. When the decoder succeeds in reconstructing the data via the extracted features, it means that the features extracted by the encoder represent the true nature of the data. There exist some variations of auto-encoder such as denoising autoencoders, stacked autoencoders, deep autoencoders, convolutional autoencoder, contractive autoencoders under complete autoencoders and variational autoencoders (**??**).

A non-symmetric deep autoencoder with multiple hidden layers that provides better classification results compared to Deep Belief Networks was proposed by Shone *et al.* (2018) for cybersecurity intrusion detection. The KDD Cup '99 and NSL-KDD datasets were used with five performance metrics, including accuracy, precision, detection, false alarm, and F-score. The results of evaluating the KDD Cup '99 dataset show that the proposed model has an average accuracy of 97.85%, which is better than the work of Alrawashdeh e Purdy (2016). On the other hand, the results of

the evaluation of the NSL-KDD dataset show that the proposed model achieves an accuracy of 85.42%, which is a 5% improvement over the Deep Belief Network model.

Khan *et al.* (2019) proposed an IDS based on a two-stage deep learning model called TSDL. The TSDL model uses a stacked auto-encoder with a soft-max classifier consisting of three main layers, namely (i) the input layer, (ii) the hidden layers, and (iii) the output layer. These three layers use a feed-forward neural network similar to a multilayer perceptron. The study uses two public datasets, including the KDD99 and UNSW-NB15 datasets. The results for the KDD99 dataset achieve high detection rates of up to 99.996%. In addition, the results for the UNSW-NB15 dataset achieve high detection rates of up to 89.134%.

A self-adaptive and autonomous IDS using auto-encoder techniques were developed by Papamartzivanos, Mármol e Kambourakis (2019). Specifically, the proposed system was designed to operate in four phases, including (i) monitoring, (ii) analyzing, (iii) planning, and (iv) execution of phases. The monitoring phase identifies any changes that require adjustments to the IDS adaptation. Network audit tools such as the Argus and CICFlowMeter were used to analyze the network data and transform the raw data into network flows. This process was performed in the analysis phase. In the planning phase, Sparse autoencoders are used to represent the input data and the data are stored for further processing. However, two datasets are used for performance evaluation, including KDDCup'99 and NSL-KDD, where an average accuracy of 59.71% was achieved for the static model and 77.99% for the adaptive model.

The model trained with the NSL-KDD and UNSW-NB15 datasets using a combination of an improved conditional variational autoencoder and a deep neural network was proposed by Yang *et al.* (2019) for cybersecurity intrusion detection. The datasets NSL-KDD and UNSW-NB15 are used to validate the proposed model, and the default learning rate of the Adam optimizer is 0.001, and the highest accuracy of 89.08% and a detection rate of 95.68% were obtained for the UNSW-NB15 dataset. The proposed study was conducted in three phases: (i) training, (ii) attack generation, and (iii) attack detection. The training phase consists of the optimization of the encoder and decoder losses. The distribution used in the new attack generation phase is a

multivariate Gaussian distribution. To detect attacks, a DNN is used in the detection phase.

Another network IDS using an unsupervised learning algorithm autoencoder was proposed by Choi *et al.* (2019) and the performance was evaluated using the accuracy, recall, specificity and F1-score measures. The results obtained show, that the model achieved an accuracy of 91.70%, which is a reliable performance.

c) **Restricted Boltzmann Machine (RBM)**

An RBM uses Markov random field approach based on an undirected probabilistic graphical model that contains one visible layer and one or several hidden layers, as shown in Figure 2.7b. An RBM through its learning approach learns the input's complex internal representations using very few labeled data for fine-tuning the representation created with a set of unlabeled input. The RBMs permit the application of top-down or bottom-up training and inference procedures, thereby allowing them to find the input's representations. Nonetheless, the issue of slow speed has limited their functionality and performance of RBM (SALAKHUTDINOV; HINTON, 2009; LIU; ZHANG; SUN, 2014).

The RBM was used for intrusion detection by Fiore *et al.* (2013). Combining the generative model's expressive power with discriminative RBM, authors achieved good classification result. The KDD Cup 1999 dataset was used with a set of 41 features and 97,278 instances and the trained model was found to accurately classify the attacks into the normal and anomaly categories.

Salama *et al.* (2011) combined the RBM and SVM to develop a deep intrusion detection model. In the research, the NSL-KDD dataset was used, setting the training set to contain a total number of 22 training attack instances, and an additional 17 instances in the testing set. The result of the study states that this combination shows a higher percentage of classification than when using a support vector machine.

Alrawashdeh e Purdy (2016) used the RMB with a deep belief network (DBN) on the KDD 1999 dataset, which contains 494,021 training records and 311,029 testing records. The detection algorithm is implemented using C++ and Microsoft Visual Studio 2013. The study shows that the RBM classifier classified 92% of the attacks

accurately. The paper compared the results to the work by Salama *et al.* (2011), which shows both a higher accuracy and detection speed.

Aldwairi, Perera e Novotny (2018) proposed a comparative study of RBM for cyber security intrusion detection. Specifically, the study demonstrates the performance of RBM to distinguish between normal and anomalous NetFlow traffic. The proposed study was applied to the ISCX dataset, which showed the highest accuracy of 78.7 ± 1.9% when the learning rate was set to 0.004. In addition, the true positive rate and true negative rates are 74.9 ± 4.6% and 82.4 ± 1.8%, respectively, at the learning rate 0.004.

Gao *et al.* (2014) attempted to integrate multilayer unsupervised learning networks and applied them to intrusion detection. The study used a hybrid of RBM and DNN algorithms in training the network following two training steps: (i) training the restricted Boltzmann machine of layers n, and (ii) fine-tuning the parameters of the entire RBM. The study shows that DBNs based on RBM outperform SVM and ANN applied on the KDD CUP'99 dataset.

d) **Deep Belief Networks (DBN)**

DBNs are probabilistic generative models comprising of multiple stacked RBM modules, where the output of each RBM is used as input to the subsequent RBM. In addition, neurons in the DBN layers have connections to the next layer, but not to neurons in the same layer. DBNs eliminate the training problems of ANNs and prevent problems such as falling into a local minimum, slow training, and the need for a large training dataset. Figure 2.7d shows the architecture of the DBN algorithm. DBNs are used in different domains, such as speech recognition, image identification, natural language processing, and intrusion detection. These deep learning networks have good feature classification and feature learning capabilities (SARIKAYA; HINTON; DEORAS, 2014).

The DBN was used for intrusion detection in Thamilarasu e Chawla (2019) where a feed-forward deep neural network was developed for the Internet of Things environments. In a more specific approach, the authors proposed a cost-effective model based on a binary cross-entropy loss function that can be used to train the model in a short period. For performance evaluation, the Keras library, Cooja network simulator,

and Texas Instruments CC2650 sensor tags were used. The Keras library was used to build a sequential DL model. The proposed model was tested against five different attacks: including;(i) sinkhole attack, (ii) wormhole attack, (iii) blackhole attack, (iv) opportunistic service attack, and (v) DDoS attack. The results show a higher precision of 96% and a recall rate of 98.7% for DDoS attack detection.

In another study, Zhao, Zhang e Zheng (2017) proposed an IDS framework using DBN and Probabilistic Neural Networks (PNN). The study carried out on the KDD CUP 1999 dataset, used 10% of the training dataset and 10% of the test dataset to evaluate the trained model. In their study, Zhao, Zhang e Zheng (2017) concluded that the proposed method performed better than three models, namely, (i) the traditional probabilistic neural network, (ii) principal component analysis with the traditional probabilistic neural network, and (iii) the unoptimized deep belief network with the probabilistic neural network.

Zhang *et al.* (2019) proposed a combination of improved genetic algorithms and DBNs for cybersecurity intrusion detection. Several RBMs were used in the study, which mainly performs unsupervised learning from preprocessed data. The DBNk module is divided into two steps in the training phase: (i) each restricted Boltzmann machine is trained separately and (ii) the final layer of the deep belief network is set as a backpropagation neural network. The performance evaluation with the dataset NSL-KDD shows a recognition rate of 99%.

An intrusion detection approach based on an improved DBN was proposed to mitigate the problems of overfitting, low classification accuracy, and high false-positive rate (FPR) in a large amount and variety of network data (TIAN *et al.*, 2020). Here, the dataset is processed by probabilistic mass function (PMF) and min-max normalization method to simplify the data preprocessing. To overcome the issue of feature homogeneity and overfitting, a penalty term based on the Kullback-Leibler divergence (KL) and the non-mean Gaussian distribution of the unsupervised training phase of DBN was combined, and the sparse distribution of the dataset was recovered by sparse constraints. In simulation experiments performed on the NSL-KDD and UNSW-NB15 datasets, an accuracy of 96.17% and 86.49% were achieved respectively.

e) **Recurrent Neural Networks (RNNs)**

RNNs can be considered as an improved version of feed-forward ANNs, since they can remember the data processed at each step to compute subsequent results. For this reason, in an RNN, the output of the neurons in each layer is connected to the input of the neurons in the other layer and also to itself. In this way, RNNs can use their internal memory to handle variable-length input sequences, such as time series, and learn a data sequence to generate its new members, as shown in Figure 2.7c. In RNNs, the input layer is unidirectionally connected to the hidden layers, while the neurons of the hidden layers are connected to themselves and all other neurons of the next layer to ensure complete information exchange. To this end, the RNNs can be trained with current and historical inputs, with the probability of an attack based on the current and previous states of the features (LEE *et al.*, 2019).

The RNN-based IDS model proposed by Yin *et al.* (2017), and trained on the NSL-KDD and KDDCup'99 datasets, was implemented to check the performance of the model on multiclass and binary classification problems. The authors also investigated the consequences of different learning rates and a number of neurons on the performance of their model. The results of this model were compared with the shallow classifiers such as SVM, J48, Random Forest, and ANN. From the obtained results, it was observed that the proposed model RNN-IDS gives good results with high accuracy on binary and multiclass intrusion detection problems.

Kaur e Singh (2020) proposed D-Sign, a hybrid deep learning-based IDS scheme that handles both anomalies and intrusions. This scheme, modeled to detect and generate signatures of web-based security attacks, specifically detected security attacks in network traffic by using an RNN with multiple Long Short-Term Memory (LSTM) layers. The LSTM was integrated into the model to overcome the vanishing gradient and long-term dependency problem, which is usually a challenge in RNN models. In addition, a softmax activation function that receives inputs from a two-layer LSTM network was used to detect new security attacks. The binary and multi-class classification was investigated using the trained D-Sign model on the NSL-KDD and CICIDS-2017 datasets. Metrics such as sensitivity, accuracy, specificity, false negatives and false positives were used to study the performance of the model. The hybrid model performed optimally compared to other classifiers.

In Lee *et al.* (2019), authors proposed an IDS approach to deal with high false positives in anomaly detection systems and to cope with attacks with unbalanced training sets. This scheme presents a feature selection model, referred to as Sequence Forward Selection Decision Tree (SFSDT), that generates the best possible subset of features. The SFSDT model consists of a decision tree and Sequence Forward Selection. Then, it trains classifiers such as RNN, LSTM, and Gated Recurrent Unit (GRU) on the selected features. Finally, the authors evaluated their anomaly detection scheme on the ISCX and NSL-KDD datasets. Authors indicated that their scheme outperforms other IDS approaches in terms of accuracy and detection rate and reduces the required computation time.

An RNN model using the gated recurrent unit was proposed for anomaly detection in the SDN environment by Tang *et al.* (2019). The GRU-RNN represents the relationship between previous and current events and can increase the anomaly detection rate. The authors concluded that the performance of the model was not affected by the introduction of six raw features into the test dataset. The model was trained and tested with the dataset NSL-KDD and achieved an accuracy of 89% without affecting the performance of the network.

f) **Generative Adversarial Networks (GAN)**

GANs provide a way to learn deep representations without extensively annotated training data. They achieve this by deriving backpropagation signals through a concurrent process involving a pair of networks. They consist of two competing ANNs that are trained in a zero-sum game with each other, where one ANN wins while the other loses. After a GAN is trained, it can learn the distribution of data and generate synthetic data instances that can be used as real data. GANs are used extensively in various applications of different fields, such as speech, video, image generation, and IDS (CRESWELL *et al.*, 2018).

Liu *et al.* (2019) proposed a GAN based approach to data augmentation in a wireless network scenario. With an Encryption Policy Intrusion Detection (EPID) using Channel State Information (CSI), the model was trained to achieve high accuracy in dataset acquisition and feature extraction. In addition, a sparse autoencoder (SAE) is used for feature extraction to reduce computational complexity and the risk of

false detections normally caused by redundant statistics. One-class classification using a SVM was performed on the general intrusion detection dataset. In the model evaluation, the EPID achieved a mean of 96.6% detection accuracy with practical feasibility.

Shu *et al.* (2020), in an attempt to improve network security in a vehicular environment, authors combined DL with GANs and investigated distributed SDN to develop a collaborative intrusion detection system (CIDS) for vehicular ad hoc networks (VANETs). This model was designed to allow multiple SDN controllers to collaboratively train a global intrusion detection model for the entire network without directly exchanging their sub-network flows. The authors confirmed the correctness of our CIDS in both Independent Identically Distribution (IID) and non-IID situations. The performance of the model was evaluated by both theoretical analysis and experimental evaluation of a real dataset. Detailed experimental results confirm that the CIDS is efficient and effective in detecting intrusion in VANETs, using metrics, such as precision, accuracy, F1-Score, recall, and AUC.

Shahriar *et al.* (2020) provide G- IDS, an IDS approach that uses GAN to generate synthetic data samples to address issues such as missing data and imbalanced classes when training the IDS. They evaluated their approach with the NSL-KDD dataset and evaluated it using the following metrics: precision, recall, and F1-score. They compared their approach with that of a stand-alone IDS in terms of the specified metrics. However, it was not evaluated on other datasets or in real-time environments.

In Huang e Lei (2020), Huang and Lei investigate the challenge of imbalanced classes in intrusion detection datasets and propose IGAN or imbalanced GAN. IGAN was achieved by adding convolutional layers and an imbalanced filter to the GAN architecture, creating new data samples for a minority class in the dataset. Afterward, using IGAN and the data samples it generated, an IDS approach called IGAN-IDS is provided, which consists of feature extraction, IGAN, and DNN. The authors used a feed-forward ANN to compute feature vectors from raw network features and evaluated the performance of IGAN-IDS using datasets such as CICIDS2017, UNSW-NB15 and NSL-KDD. The obtained results were compared using metrics such as F1 score, AUC and accuracy with some deep and shallow learning methods.

g) **Convolutional Neural Networks (CNN)**

CNN's are DL networks that have high-performance accuracy in image processing but have also been used in intrusion detection problems obtaining a good performance. In IDS, CNNs are often used for feature extraction from raw data. CNN is a multi-layer ANN, which consists of input and output layers as well as several hidden layers. It receives its input as a 2D image and then assigns some meaning to different parts of the image to recognize the output. A CNN benefits from several hidden layers such as a convolutional layer, a fully-connected layer, a nonlinearity layer, and a pooling layer, where the first two are parametric and the other two are nonparametric as shown in Figure 2.8.

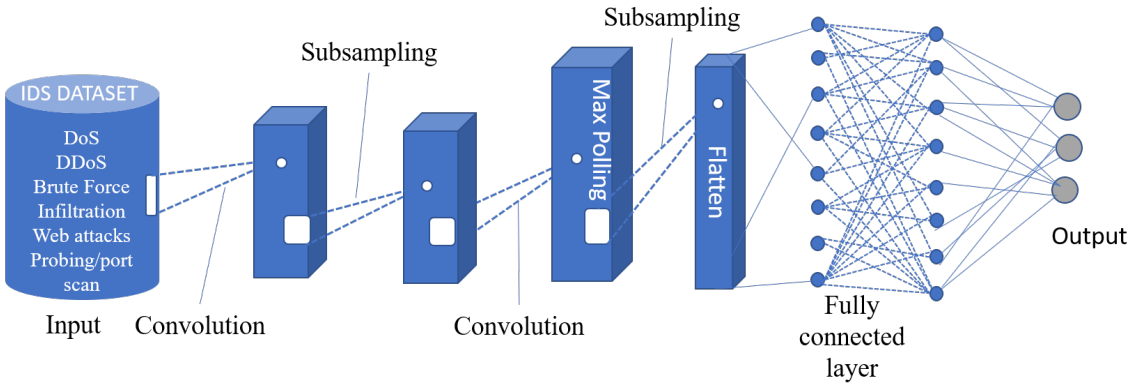
CNNs have been used for packet data anomaly detection by Mendonça *et al.* (2021) and phasor measurement units-based state estimators in Basumallik, Ma e Eftekharnejad (2019), Jo *et al.* (2020). They use a data filter based on a foldable neural network to extract event signatures (features) from phasor measurement units. The busses of the phasor measurement units are the IEEE -30 and IEEE118 bus systems. The study claims a probability of 0.5 with 512 neurons in a fully connected layer and an accuracy of 98.67%. The authors also claim that CNN-based filters outperform other machine learning techniques, such as RNN, LSTM, SVM, bagged, and boosted.

The system proposed by Fu *et al.* (2016) was a CNN model to capture the intrinsic patterns of fraud behavior, especially in credit card fraud detection. In the same research direction, Zhang *et al.* (2018) developed a model to train and test an IDS using a CNN and B2C online transaction data from a commercial bank. One month's data was split into a training set and a testing set. The study claims a 91% of precision rate and a 94% of recall rate. These results are improved by 26% and 2%, respectively, compared to Fu *et al.* (2016).

Nasr, Bahramali e Houmansadr (2018) introduced an IDS called DeepCorr, which builds on a CNN framework to train a correlation function. DeepCorr consists of two layers of CNN and three layers of fully connected neural networks. Experiments have shown that DeepCorr achieves a true-positive rate of nearly 0.8 with a learning rate of 0.0001 and a false-positive rate of $10^{-}3$, giving it a good performance rating for detecting intrusion.

An anomalous traffic detection model based on two layers of neural networks. The first layer is the improved LetNet-5 CNN. The second layer uses a long-term memory proposed by Zhang *et al.* (2019). The first layer is designed to extract the spatial features of the flow, while the second layer is designed to recover the temporal features of the flow. On the CICIDS2017 dataset, the performance was 94% accuracy. The proposed system achieves good accuracy, precision, recall and F1 measure compared to other ML techniques (Naïve Bayes, Logistic Regression, Random Forest and Decision Tree). As a result, Zhang *et al.* (2019) proposed a lightweight framework called deep-full range (DFR) for novel threat detection, encrypted traffic categorization, and traffic classification (ALBAWI; MOHAMMED; AL-ZAWI, 2017; KALCHBRENNER; GREFENSTETTE; BLUNSOM, 2014; KIM, 2017a).

Figure 2.8 – Architecture of a fully connected CNN that are implemented in IDS solutions



Source: Author (2022)

Table 2.4 presents the comparison of DL and ML techniques (SARKER, 2021), highlighting their main differences (XIN *et al.*, 2018; APRUZZESE *et al.*, 2018; FERRAG *et al.*, 2019). Meanwhile, the important issue in both methods is the quality of data, because it determines the accuracy, F1-score, recall and precision of the results in each case.

In Table 2.5, we present a summary of the key features of the various deep learning algorithms discussed earlier.

Table 2.4 – General comparison of deep learning and machine learning techniques

S/NO	Machine Learning	Deep Learning
1	ML models can be trained on small datasets.	DL networks need a large dataset for training.
2	ML algorithms need less time for execution.	DL networks require more execution time.
3	The output of ML models is mostly numeric.	The output of the DL models can be in various forms.
4	It is easier to interpret results obtained from solving a problem using ML algorithms such as Decision Trees.	The problem solving using deep learning methods, based on multilayered ANNs, are more intertwined and complex.
5	ML algorithms need to be retrained through human intervention.	DL networks do not require human intervention.
6	ML model training requires labeled data, it is not appropriate for handling large-scale problems that need a large set of labeled data.	Regarding the overheads of the deep learning methods, they are more appropriate for dealing with large-scale problems.
7	External intervention is necessary to provide the right input in ML.	DL networks can learn about features from the raw data, hence does not require intervention.
8	ML algorithms can be handled using CPU on low-end computer systems.	Solving large-scale problems using high-end computer systems or dedicated hardware such as Graphics Processing Unit (GPU) can increase the performance of the DL algorithms.

Source: Author (2022)

Table 2.5 – A summary of DL networks highlighting some key points

Deep Learning Techniques	Descriptive Key Points
RBM	An unsupervised learning algorithm that learns through statistical distribution It is probabilistic or stochastic Mostly applied to feature selection and feature extraction. Comprises the building blocks of DBNs
DBN	A probabilistic generative model containing multiple RBMs. The ability to encode richer and higher-order network structures It can be applied to an unsupervised or a supervised setting Can be used in a large number of high-dimensional data applications
GAN	A form of generative model typically used for unsupervised learning problems Generate new, synthetic instances of data with features close to the actual data input To make the DL models more robust
CNN	Regularized version of multi-layer perceptrons Can automatically learn or detect the key features from data Typically deal with the variability of 2D shapes, e.g., image
Auto Encoder	An unsupervised learning algorithm that learns a representation of the inputs it is deterministic To significantly reduce the noise in the input data Used typically for dimensionality reduction, very similar to PCA.

Source: Author (2022)

2.4 Publicly Available IDS Datasets

To develop an IDS, one of the basic requirements is a dataset on which the model can be trained and tested. Since the 1990s, several datasets have been developed. Usually, new datasets are improvements on existing datasets by adding new features. Some of the widely used datasets include KDD99 (TAVALLAEE *et al.*, 2009), NSL-KDD (INGRE; YADAV, 2015), Kyoto2006 , UGR2006 (PROTIĆ, 2018), UNSW-NB2015 (MOUSTAFA; SLAY, 2015a; ALASADI, 2019), CICIDS2017 (PANIGRAHI; BORAH, 2018), CSE-CICIDS2018 (SHARAFALDIN; LASHKARI; GHORBANI, 2018) and more. In this section, we provide a detailed review of all the available datasets from 1998 to 2022.

2.4.1 DARPA 1998 dataset

This dataset, which is based on network traffic and audit records, was made accessible for the first time in February 1998. The MIT Lincon developed the dataset whereas the project was funded by DARPA. The data used for model training is based on network-based attacks recorded in seven weeks. DARPA dataset was created for network security analysis and revealed vulnerabilities that show connection with artificial injection of malicious and benign traffic. Email, browsing, FTP, Telnet, IRC, and Simple Network Management Protocol (SNMP) activities are included in this dataset. It includes Rootkit threats, remote File Transfer Protocol, Nmap FTP, Guess password, Syn flood, DoS and Buffer overflow. It has the drawback of not representing real-world network traffic and also has various data inconsistencies. Also, the dataset is out of date for effective IDS evaluation on today's networks considering attack types as well as network infrastructure, as it lacks real attack records (MCHUGH, 2000; LIPPMANN *et al.*, 1999). Currently, the dataset has three different versions:

1. 1998 DARPA ID Assessment Dataset: containing 7 weeks of training data and 2 weeks of test data.

2. 1999 DARPA ID Assessment Dataset: inclusive of 3 weeks of training data and 2 weeks of test data.

3. 2000 DARPA ID Scenario-Specific Dataset: Includes LLDOS 1.0 Attack Scenario Data, LLDOS 2.0.2 Attack scenario data, Windows NT attack data.

2.4.2 KDD CUP 99 Datasets

The Knowledge Discovery and Data Mining Competition (KDD Cup) 99 dataset (TAVALLAEE *et al.*, 2009), which is based on the DARPA 1998 dataset, is one of the most extensively used training sets. On record, there are 4 900 000 repeated attacks in this dataset. There is one normal type with the identity normal and 22 attack types grouped into five primary categories: Denial of Service (DoS) attacks, User to Root attack (U2R), Root to Local attacks (R2L), Probe (Probing attacks), and Normal. The KDD Cup 99 training dataset provides 41 fixed feature characteristics and a class identifier for each record. Seven of the 41 fixed feature traits are of the symbolic kind, while the others are continuous.

Despite the wide application of this dataset, there are inherent challenges associated with usage as identified by (MCHUGH, 2000). This promoted the interest of the researcher to propose other datasets. Other KDD Cup datasets are also available (ACM, 2016).

2.4.3 National Security Lab Knowledge Discovery and Data (NSL-KDD) Dataset

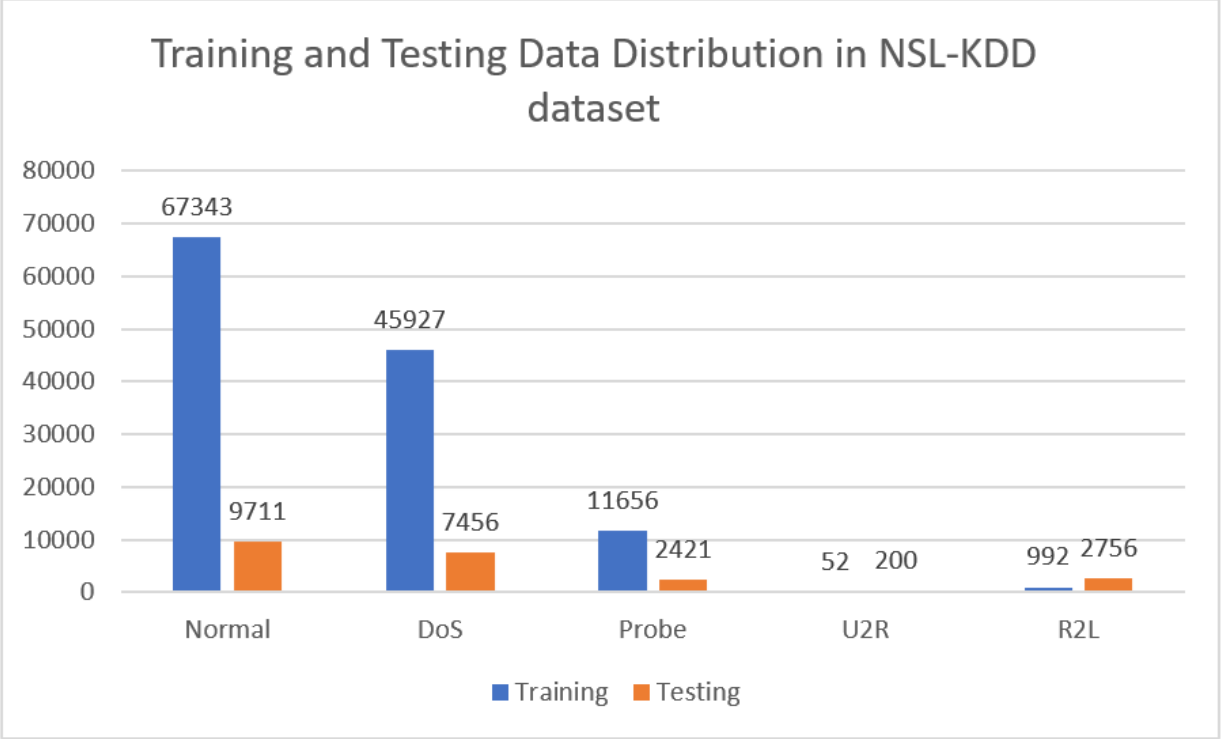
NSL-KDD is a dataset proposed to address most of the KDD'99 dataset's intrinsic flaws (MCHUGH, 2000). Although this new KDD dataset also contain some concerns as discussed by McHugh, thus does not offer perfect representation of established real networks. Nonetheless, it is an efficient standard dataset to assist scholars carry out comparative research on different ID techniques considering lack of publicly accessible datasets for network-based IDS. Also, the NSL-KDD train-test sets have a reasonable amount of data records. This advantage enhances the execution of the experiments on the whole dataset without having to pick a tiny sample at random. Hence, the evaluation outcomes of various research projects will give consistent and comparable results. The data file features are as shown in Table 2.6.

Table 2.6 – Data File Features of NSL-KDD

Parameter	**Meaning**
KDDTrain+.$ARFF$	NSL-KDD train set having the binary labels represented ARFF format
KDDTrain+.TXT	The NSL-KDD train set containing the labelled attacks and their difficulty level shown CSV format
KDDTrain+_20Percent.$ARFF$	20% subset of the KDDTrain+.$arff$ file
KDDTrain+_20Percent.TXT	20% subset of the KDDTrain+.txt file
KDDTest+.$ARFF$	NSL-KDD test set considering binary labels in ARFF files
KDDTest+.TXT	NSL-KDD test set including attack-type labels and difficulty level in CSV format
KDDTest-21.ARFF	A subset of the KDDTest+.$arff$ file that not include data with difficulty level of 21 / 21
KDDTest-21.TXT	A subset of the KDDTest+.txt file that not include data with difficulty level of 21 / 21

Source: Author (2022)

Figure 2.9 – Training and Testing data distribution in NSL-KDD dataset



Source: Adapted from Wu *et al.* (2022)

2.4.4 DEFCON dataset

This data was generated in two versions in different years: DEFCON 8 (2000) and DEFCON-10 (2002). The DEFCON-8 dataset contains ports scan and buffer overflow while the DEFCON-10 dataset contains probing and non-probing attacks (such as bad packets, ports scan, port sweeps, etc.). Both versions were used by Nehinbe Ojo Joshua (NEHINBE, 2011) for reclassifying network intrusions. One limitation of this dataset is that the traffic generated during the *Capture the Flag (CTF)* competition differs from real-world network activity in that it primarily consists of intrusive traffic rather than normal daily traffic. As a result, it is unreliable for detecting intrusions and,thus, is not widely used by researchers.

2.4.5 ADFA Dataset

The ADFA[1] dataset is a group of datasets from the Australian Defence Force Academy (ADFA) that are commonly utilized in training and testing of IDS. Various system calls in this dataset have been classified and labeled for the kind of attack. The global data contains two operating systems, Linux (ADFA-LD) and Windows (ADFA-WD), which record the sequence of system calls. The ADFA-LD keeps a record of the operating system's invocation for a certain amount of time. The kernel offers the user space program and the kernel space interacts with a set of standard interfaces, the interface to the user program can be restricted access hardware devices, such as use of system resources, operating equipment, etc. There are mainly 5 different attack variations and 2 normal attack types as shown in Table 2.7

Table 2.7 – Attack Types in ADFA-LA Dataset

Attack Type	**Category**	**Data Size**
Training	Normal	833
Validation	Normal	4373
Hydra-FTP	Attack	162
Hydra-SSH	Attack	148
Adduser	Attack	91
Java-Meterperter	Attack	75
Webshell	Attack	118

Source: Author (2022)

[1] https://research.unsw.edu.au/projects/adfa-ids-datasets

2.4.6 ISCXIDS2012

ISCXIDS2012 is developed based on the idea of profiles, that include vivid explanations of intrusions as well as abstract delivery mechanisms for lower-level network entities, protocols and applications. For simulating the user behavior, the profiles are applied. To generate some profiles for agents that create real traffic for POP3, SMTP, FTP, HTTP, IMAP and SSH, real traces of network activities were analyzed. These profiles are utilized for generating a dataset using the required test-bed. For producing the anomalous section, different scenarios for multi-stage attacks are used. Then, agents are used to running these profiles imitating the user activity. This dataset contains seven days of malicious and normal network activities as shown in Table 2.8. Some key features of this dataset include (SHIRAVI *et al.*, 2012):

1) **Labelled dataset**. The importance of labeled datasets in intrusion detection cannot be over-emphasized. This ensures the impractical processing of manually labeling the data before being employed for anomaly detection is eliminated.

2) **Complete Capture**: One of the primary bottlenecks for network security researchers has always been privacy concerns associated to releasing genuine network traces, as data providers are typically hesitant to give such information. As a result, the primary purpose is to develop network traces in a regulated testing environment, which eliminates the need for any proper cleaning and preserves the genuineness of the final dataset.

3) **Realistic network and traffic**: A dataset is not expected to exhibit any unwanted qualities, both for network and also traffic-wise, considering an ideal scenario. This is to give more information about the true consequences of network threats and the how the workstation responds to them. As a consequence, the traffic must appear and behave in a realistic manner. This applies to both normal and unusual traffic.

4) **Diverse Attack Scenarios**: In recent years, the frequency, scale, variety, and sophistication of attacks have all increased. Attacks have also evolved to include increasingly complicated strategies, such as service-network and application-targeted threats. The goal is to undertake a broad set of multi-stage attacks, with each being carefully developed and directed at recent developments in security concerns, by performing attack scenarios and applying anomalous behavior.

5) **Total interaction capture**: The availability of data to detection systems is critical since it gives them the ability to detect abnormal activity. To put it another way, this data is critical for post-evaluation and proper interpretation of the results. Therefore, all network interactions, whether within or between internal LANs, is considered a major criterion for a dataset.

Table 2.8 – Activity Description of the development of the ISCX-IDS-2012 dataset

Days	Date	Description	File Size (GB)
Friday	11/6/2010	Normal Activity. There was no malicious activity performed	16.1
Saturday	12/6/2010	Normal Activity. Contains no malicious events	4.22
Sunday	13/6/2010	Obtained network infiltration and some normal events	3.95
Monday	14/6/2010	HTTP DoS + Normal network Activity	7 6.85
Tuesday	15/6/2010	DDoS obtained an IRC Botnet	23.4
Wednesday	16/6/2010	contains normal network events fr the whole day	17.6
Thursday	17/6/2010	contains Brute Force SSH attacks + Normal network Activity	12.3

Source: Author (2022)

2.4.7 UNSW- NB15

The dataset is majorly used to evaluate network IDS due to their capabilities in detecting recent attacks. The software tool denoted as IXIA Perfect Storm is incorporated to create its abnormal and normal network traffic traces. The IXIA tool which can mimic nine types of security attacks and can also use new attacks which are updated from a site including some security vulnerabilities information is an efficient tool in monitoring and recording these attacks. Also, the Tcpdump is utilized for 16 h to capture 100 GBs of network traffic (MOUSTAFA; SLAY, 2015b). In Table 2.9, we show the data attributes of this dataset including the different attack types, train data and test data. The table shows over 93,000 normal attacks, 24,246 fuzzers, 2,677 analyses, 170 worms and more in the dataset.

Table 2.9 – Description of the properties of UNSW-NB15 Dataset

Class	Description	Train	Test
Normal	Normal connection record	56,000	37,000
Fuzzers	Attacks related to spams, html files penetrations and port scans	18,184	6,062
Analysis	Attacks related to html files penetration, spam and port scan	2,000	677
Backdoors	Backdoor is a mechanism used to gain access to a computer by evaluating the background existing security	1,746	583
DoS	Intruder aims at making network resources down and hence, resources are inaccessible to authorized users.	12,264	4,089
Exploits	The security hole of operating system or the application software is understood by an attacker with the aim to expoit vulnerabilities.	33,393	11,132
Generic	Attackers are related to block-cipher	40,000	18,871
Reconnaissance	A target system is observed by an attacker to gather information for vulnerability	10,491	3,496
Shell Code	A small part of program termed a payload used in exploitation of software	1,133	378
Worms	Worms replicate themselves and distribute to other system through the computer network	130	44
Total		**93,500**	**28,481**

Source: Author (2022)

2.4.8 ISCX-URL2016

This is a dataset composed of several Universal Resources Locators (URLs) developed to train IDS models to be able to detect URL attacks. The dataset contains 35,000 benign URLs collected from the top Alexa websites in which the domains were passed through a Heritrix web crawler to extract the needed URLs. Among over 500,000 URLs, obtained, 35,000 were selected after removing duplicates and domain names. The other content of the dataset are 12,000 spam URLs collected from the WEBSPAM-UK2007 dataset, phishing URLs having 10,000 addresses collected from the OpenPhish repository, malware URLs from the DNS-BH project which maintains a list of malware sites to a tune of 11,500 URLs and 45,450 URLs from Defacement category (MAMUN *et al.*, 2016).

2.4.9 CICIDS2017

The CIC-IDS2017 dataset includes benign and up-to-date known attacks, and it almost considers actual data Packet Captures (PCAPs). This dataset gives the analysis results of network traffic including labeled flows that are based on the source and destination IPs and also ports, protocols, and attack using CICFlowMeter, and considering time stamp. The most important goal in creating this dataset was to create similar traffic to real scenarios. The B-Profile technology was used to generate naturalistic innocuous background traffic by profiling the abstract behavior of human interactions. The abstract behavior of 25 users was developed for this dataset using HTTP, FTP, SSH, HTTPS and email protocols. The dataset was developed with the features shown in Table 2.10.

Table 2.10 – Features of CICIDS2017 Dataset

Feature	**Description**
Complete Interaction	By having two distinct networks and Internet communication, both within and between internal LANs are covered.
Complete Network configuration	The Firewall, Switches/Routers, and different OS, such as Windows, Linux, and Mac OS X represent a complete network topology.
Available Protocols	Shows all commonly available protocols, such as HTTPS, SSH, FTP, HTTP and email protocols.
Labeled Dataset	Shows that the dataset contains benign or attack labels. Also, information about the attack timing are also made available.
Heterogeneity	During the attack, network traffic from the main switch, as well as memory dumps and system calls from all target PCs are recorded
Complete Capture	All traffics events both normal and abnormal are recorded on an appropriated server.
MetaData	Contains complete details about the dataset such as attacks, the time, flows and labels in published articles
Attack Diversity	Containing the attacks of the 2016 McAfee report, such as Web-based, Infiltration, DoS, DDoS, Heart-bleed, Brute force, Bot and Scan.
Feature Set	The dataset considers more that 80 network features from network traffic generated by CICFlowMeter and saved in an CSV file.
Complete Traffic	Implementing user profiling agent, and twelve different nodes in networks with actual attacks.

Source: Author (2022)

2.4.10 CSE-CICIDS2018

In the year 2018, the Communications Security Establishment (CSE) and the Canadian Institute for Cybersecurity (CIC) collaborated on a project to develop a more recent database for intrusion detection. The main goal of building the dataset was to create user

profiles that contained abstract representations of events and actions as viewed on the network, as well as diverse and thorough information on innovative attacks. The final dataset includes seven different attack scenarios: Heartbleed, brute-force, Botnet, Web attacks, DoS, DDoS and network infiltration. The attack infrastructure consists of 50 machines, while the victim organization consists of 5 departments with 420 machines and 30 servers. Each machine's network traffic and system logs are included in the dataset, as well as 80 features extracted from the collected traffic using CICFlowMeter-V3. The seven different attack scenarios implemented in the development of the dataset as shown in Table 2.11 (SHARAFALDIN; LASHKARI; GHORBANI, 2018).

Table 2.11 – Attack Scenarios and Duration in CSE-CICIDS2018

Attack	**Tools**	**Duration**	**Attacker**	**Victim**
Bruteforce attack	FTP – Patator SSH – Patator	One day	Kali linux	Ubuntu 16.4 (Web Server)
DoS attack	Hulk, GoldenEye, Slowloris, Slowhttptest	One day	Kali linux	Ubuntu 16.4 (Apache)
DoS attack	Heartleech	One day	Kali linux	Ubuntu 12.04 (Open SSL)
Web attack	Damn Vulnerable Web App (DVWA) In-house selenium framework (XSS and Brute-force)	Two days	Kali linux	Ubuntu 16.4 (Web Server)
Infiltration attack	First level: Dropbox download in a windows machine Second Level: Nmap and portscan	Two days	Kali linux	Windows Vista and Macintosh
Botnet attack	Ares (developed by Python): remote shell, file upload/download, capturing screenshots and key logging	One day	Kali linux	Windows Vista, 7, 8.1, 10 (32-bit) and 10 (64-bit)
DDoS and PortScan	Low Orbit Ion Canon (LOIC) for UDP, TCP, or HTTP requests	Two days	Kali linux	Windows Vista, 7, 8.1, 10 (32-bit) and 10 (64-bit)

Source: Author (2022)

2.4.11 CAIDAs datasets

The Center of Applied Internet Data Analysis have proposed several dataset for IDS, including, RSDoS Attack Metadata (2018-09), CAIDA DDOS, and CAIDA Internet traces 2016. More specifically, the RSDoS Attack Metadata (2018-09) includes randomly spoofed denial-of-service attacks inferred from the backscatter packets collected by the UCSD Network Telescope while the CAIDA DDOS includes one-hour DDoS attack traffic distributed within 5-minute PCAP files which are passive traffic traces from CAIDA's Equinix-Chicago (GHARIB *et al.*, 2016).

2.4.12 CIC-DDoS2019

CIC-DDoS2019 (CIC, 2019) called the DDoS Evaluation Dataset was designed to solve the challenges of real-time detection of DDoS in earlier versions of the dataset. It was proposed and tested by Sharafaldin *et al.* (2019). The dataset contains 80 features collected on two different days using the network analysis tool, CICFlowMeter-V3 with labeled flows considering IPs and ports of source and and destination, protocols and attack, using the time stamp for each flow. It has benign and recent forms of DDoS attacks built from the abstract behavior of 25 users based on HTTP, FTP, e-mail, SSH and HTTPS protocols. The attacks in the dataset include among others the UDP-Lag, TFTP, DNS, MSSQL, NetBIOS,UDP,UDP-Lag,MSSQL,WebDDoS, SNMP,NetBIOS,LDAP,DNS,NTP, SSDP, SNMP, LDAP,PortMap, NTP, SYN, UDP and SYN.

2.4.13 CIC-InvesAndMal2019

CIC-InvesAndMal2019 is an improved version of the CICAndMal2017 dataset and it was made publicly available for research purposes in 2019 by Canadian Institute for Cybersecurity[2]. Taheri, Kadir e Lashkari (2019) developed the dataset specifically to address the issues of malware in Android devices. It includes permissions and intents as static features and API calls and all generated log files as dynamic features which were captured During installation, before restarting and after restarting the phone. 5000 samples of four different malware categories are contained in the dataset such as Adware, Ransomware, Scareware and SMS Malware. The authors trained a model using the data and succeeded in achieving 95.3% precision in Static-Based Malware Binary Classification at the first layer,

[2] https://www.unb.ca/cic/datasets/invesandmal2019.html

83.3% precision in Dynamic-Based Malware Category Classification and 59.7% precision in Dynamic-Based Malware Family Classification at the second layer.

2.4.14 CICDarknet2020 dataset

In the CICDarknet2020[3] dataset, a two-layered approach is used to generate benign and darknet traffic at the first layer. It was developed to detect the malicious activities that go into the darknet which is considered illegal. The darknet traffic is composed of activities such as Audio-Stream, Browsing, Chat, Email, P2P, Transfer, Video-Stream and VOIP which is generated at the second layer. The ISCXTor2016 and ISCXVPN2016 were joined together to generate the new dataset with improved features and combined the respective VPN and Tor traffic in corresponding Darknet categories (LASHKARI; KAUR; RAHALI, 2020). On training, the network achieved a detecting accuracy of 86%.

2.5 Activation Functions

Activation functions (AFs) also referred to as transfer functions in some literature are functions applied in neural networks to calculate the weighted sum of input and biases for neurons (NWANKPA *et al.*, 2018). These weights and biases are of importance to decide if a neuron gets activated or not. If the neuron is activated, it will be triggered, but if not, it stays. Through series of gradient processing, the AFs manipulates the data usually using gradient decent and thereafter produces an output for the neural network containing parameters of the input data. There are many AFs which have been applied in DL projects of which some are linear while others are non-linear depending on the type of function it represents and can either be used in the input, hidden or output layers of the neural network.

2.5.1 Sigmoid

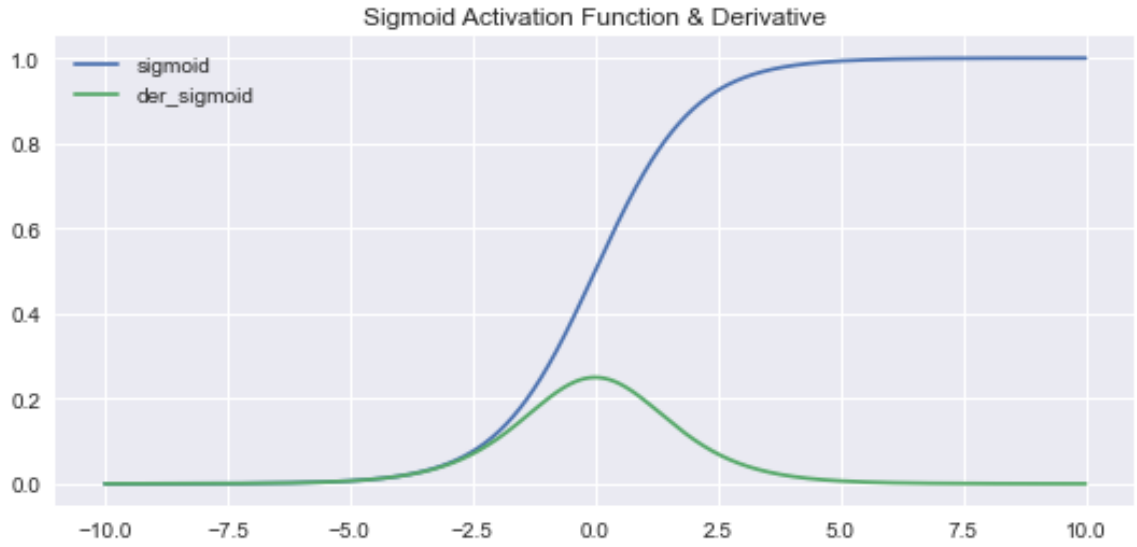
The Sigmoid is a non-linear AF which has been used frequently in feedforward neural networks. It is a bounded differentiable real function with specific for real input values, positive derivatives and having some degree of smoothness. The Sigmoid function is given

[3] https://www.unb.ca/cic/datasets/darknet2020.html

by the relationship shown in Equation 2.1 and Figure 2.10.

$$f(x) = \left(\frac{1}{1 + exp^{-x}} \right) \tag{2.1}$$

Figure 2.10 – Sigmoid Function and its Derivative



Source: Author (2022)

The sigmoid function is mostly used in the output layers of the DL architectures for predicting probability based output. It has been successfully applied in binary classification problems, modeling logistic regression tasks as well as other neural network domains. The AF is easy to understand and can be used in shallow networks which are its major advantages as highlighted by Neal (1992). However, the AF is not very good when initializing NNs from small random weights. Also, gradient saturation, slow convergence, sharp damp gradients during backpropagation from deeper hidden layers to the input layers and non-zero centred output thereby causing the gradient updates to propagate in different directions are some of the major drawbacks of the AF. There are three variants of the Sigmoid AF which include *Hard Sigmoid Function, Sigmoid-Weighted Linear Units (SiLU) and Derivative of Sigmoid-Weighted Linear Units (dSiLU)* (ELFWING; UCHIBE; DOYA, 2018).

2.5.2 Hyperbolic Tangent Function (Tanh)

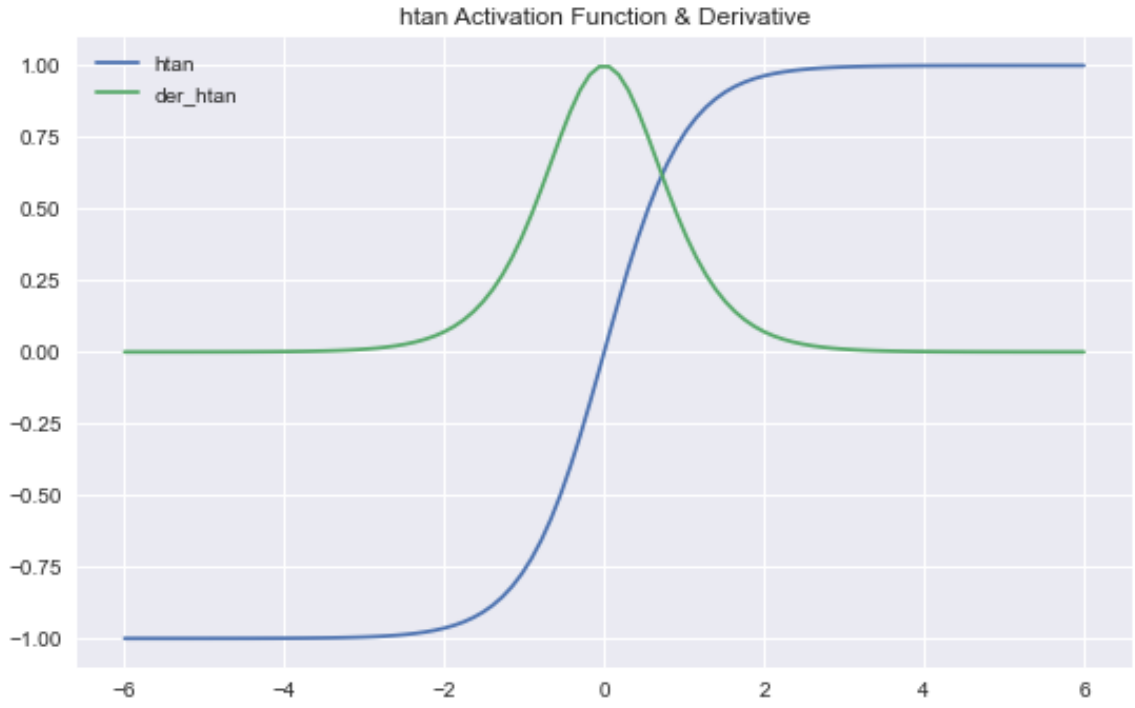
This was proposed as an alternative to overcome the draw back of the Sigmoid function. It is also called tanh function. The range of values for this AF is between -1 to

1, having a smoother and zero-centered feature. Equation 2.2 shows the mathematical representation of the function.

$$f(x) = \left(\frac{e^x - e^x}{e^x + e^{-x}} \right) \tag{2.2}$$

Tanh function gained more popularity over the predecessor (Sigmoid) because it gives better training performance for multi-layer neural networks (KARLIK; OLGAC, 2011). However, the problem of vanishing gradient experienced with Sigmoid was not solved by the tanh function. Meanwhile, tanh function had the main advantage of producing zero centered output thereby aiding back-propagation process. Figure 2.11 shows the normal and derivative plots of the tanh function

Figure 2.11 – Hyperbolic Tangent Function and its Derivative



Source: Author (2022)

2.5.3 Softmax

Another prominent AF used in DL projects is the Softmax function. It is used to compute probability distribution from a vector of real numbers. The Softmax function is used in multi-class models where it returns probabilities of each class, with the target class having the highest probability. Usually, the output of Sofmax function is in the range of

values between 0 and 1, and the sum of the probabilities equal to 1. The relationship in Equation 3.14 shows the mathematical representation of the function.

$$f(x_i) = \frac{exp(x_i)}{\sum_{i=1}^{N} exp(x_j)} \tag{2.3}$$

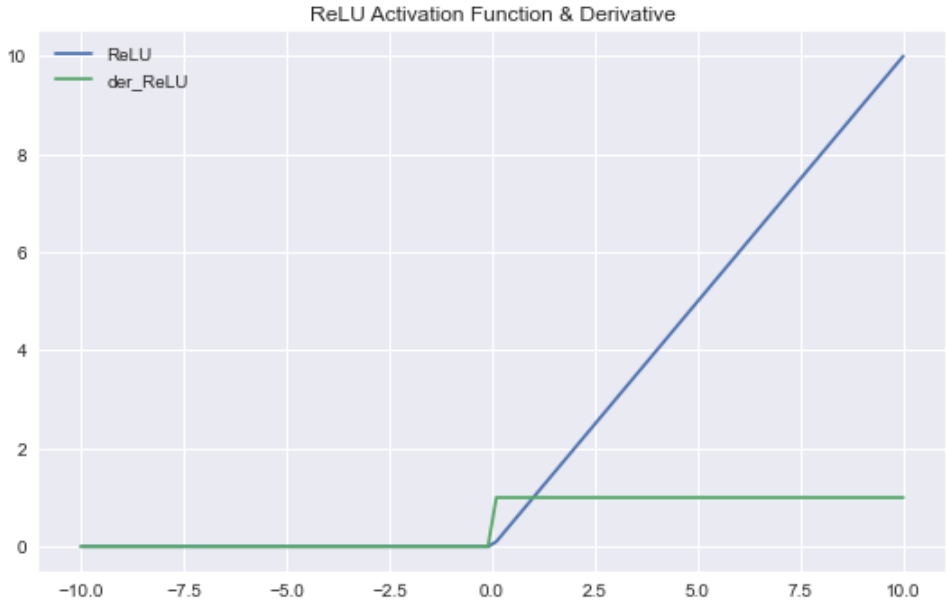
The Softmax function mostly appears in almost all the output layers of the deep learning architectures, where they are used (ROY *et al.*, 2017). The main difference between the Sigmoid and Softmax AF is that the Sigmoid is used in binary classification while the Softmax is used for multivariate classification tasks.

2.5.4 Rectified Linear Unit (ReLU)

Nair e Hinton (2010) proposed the Rectified Linear Unit (ReLU) activation function in 2010 and since then, it has been the most widely used AFs in deep learning computational problems especially in the hidden layers of the networks. The major advantages of the ReLU function includes faster learning, better performance and generalization and since it is nearly linear, it preserves the properties of linear models, thus making it easy for optimization with gradient descent methods. The ReLU activation function performs a threshold operation to each input element where values less than zero are set to zero thus the ReLU is given by Equation 2.4 and the corresponding graph showing the normal and derived function is shown in Figure 2.12.

$$f(x) = max(0, x) = \{^{0, for x_i 0}_{x_i for x_i \geq 0} \tag{2.4}$$

Figure 2.12 – Rectified Linear Unit Function and its Derivative



Source: Author (2022)

This function overcomes the vanishing gradient problem seen earlier by rectifying the values of the inputs less than zero thereby forcing them to zero. In addition to using the ReLU in hidden layers of DL networks, other AFs are normally used in the output layer. However, the ReLU has a limitation that it easily overfits compared to the sigmoid function although the dropout technique has been adopted to reduce the effect of overfitting of ReLUs and the rectified networks improved performances of the deep neural networks. ReLU and its variants have been used in RNNs and CNNs. Due to the fragility experienced with ReLU AF, there are tendencies for some neurons to die while training. This drawback has led to the development of some variants of the ReLU function to include: *Leaky ReLU, Parametric ReLU, Randomized Leaky ReLU, and S-shaped ReLU.*

2.5.5 Soft Root Square (SRS)

SRS adaptively adjusts a pair of independent trainable parameters to provide a zero-mean output, thereby achieving better generalization performance and faster learning speed. It also prevents the distribution of the output from being scattered in the non-negative real number space and corrects it to the positive real number space, making it more compatible with batch normalization (BN) and less sensitive to initialization as defined by Equation 2.5. Figure **??** shows the grap of the function with the maximum and minimum values which can be controlled by altering the values of α and β (ZHOU *et al.*, 2020).

$$SRS(p) = \frac{p}{\frac{p}{\alpha} + e^{-\frac{p}{\beta}}} \tag{2.5}$$

where α and β variables are a pair of trainable positive parameters. The SRS presents a non-monotonic region in which $p < 0$ provides the property with zero mean. When $p > 0$, it avoids and rectifies the output distribution. The SRS derivative is defined as follows:

$$SRS'(p) = \frac{\left(1 + \frac{p}{\beta}\right) e^{-\frac{p}{\beta}}}{\left(\frac{p}{\alpha} + e^{-\frac{p}{\beta}}\right)^2} \tag{2.6}$$

The output of an SRS is limited by the range

$$\frac{\alpha\beta}{\beta - \alpha\epsilon}, \alpha \tag{2.7}$$

2.6 Optimizer

During the training of Deep models, we aim at maximizing each neuron's weights and minimize loss functions. This objective is achieved through the application of optimizer. An optimizer is a function or an algorithm that modifies the attributes of the neural network, such as weights and learning rate. Thus, it helps in reducing the overall loss and improve the accuracy. The choice of the optimizer to be used in deep learning problems determines the performance of the model. There many optimizer functions used in deep learning tasks: Gradient Descent (GD), Stochastic Gradient Descent (SGD), Stochastic Gradient descent with momentum, Mini-Batch Gradient Descent, Adagrad, RMSProp, AdaDelta and Adam. The commonly used of these are the SGD, RMSProp and Adam, with Adam having wider application.

2.6.1 Stochastic Gradient Descent (SGD)

SGD is one of widely and most popular algorithm to implement for an optimizer to reduce a cost and optimize neural networks (RUDER, 2016). It is expressed mathematically as:

$$w_{(n+1)} = w_{(n)} - Lr * \hat{g} \tag{2.8}$$

where: $w(n)$ is variable update at time n, Lr is learning rate $\hat{g}$ is a gradient vector.

Processing in each epoch of neural networks has an aim to reduce an error which many of work are defined in term of cross entropy. SGD performs variable update in every epoch to minimize the error. The variable will update by using prior time step variable minus with a result from learning rate multiple with a gradient vector.

2.6.2 RMSProp

RMSProp was proposed by Hinton, Srivastava e Swersky (2012). There is always the case of differing magnitudes for gradients for different weights in SGD, and can also change during learning, hence it is difficult to choose a single general learning rate. This limitation is tackled with RMSProp by exponentially smoothing the average of squared gradients and adjusting the weights updates for each training phase as shown in Equations 2.9 and 2.10

$$E\left[g^2\right]_t = \beta E\left[g^2\right]_{t-1} + (1-\beta)\left(\frac{\delta C}{\delta W}\right)^2 \tag{2.9}$$

$$W_t = W_{t-1} - \left(\frac{\eta}{\sqrt{E\left[g^2\right]_t}}\right)\frac{\delta C}{\delta W} \tag{2.10}$$

2.6.3 Adams

Kingma e Ba (2014) proposed an adaptive learning rate optimizer algorithm, called Adam. It is an algorithm for optimizing stochastic objective functions, based on adaptive estimates of lower-order moments.The optimizer achieved fast convergence by merging the ability of AdaGrad to deal with sparse gradients, and the ability of RMSProp to deal with non-stationary objectives; hence, it becomes very resistant to model structures, making it the top choice when deciding which algorithm to be used. The optimizer performs according to Equation 2.11.

$$w_{(n+1)} = w_{(n)} - \frac{Lr}{\sqrt{\widehat{v_n}} + \varepsilon}\hat{m}_n \tag{2.11}$$

and

$$\hat{m}_n = \frac{\beta_1 m_{n-1}(1-\beta_1)\hat{g}_n}{1-\beta_1^n} \tag{2.12}$$

$$\hat{v}_n = \frac{\beta_2 v_{n-1} + (1-\beta_2)\hat{g}_n^2}{1-\beta_2^n} \tag{2.13}$$

where: $w(n)$ is variable update at time n, Lr is learning rate, $\hat{g}$ is a gradient vector, m_n is an estimation of the first moment (the mean) of the gradients and v_n is the second moment (the uncentered variance) of the gradients

By the default values of 0.9 for β_1, 0.999 for β_2, and 10^{-8} for ϵ. Adam optimizer is straightforward to implement and can efficiently solve practical deep learning problems with using large dataset.

2.7 Loss Function

ML/DL tasks usually either come as a regression or classification task. Where it is regression, the model predicts on continuous data while in classification tasks, the model predicts on categorical variables. Neural networks uses a strategy called stochastic gradient descent as an optimization technique to minimize error in the algorithm. Loss Function (LF) is used by the network to compute this error, the value of which quantifies how good or bad the model is performing. Based on the class of problem being solved, we can have Regression Loss and Classification Loss. There are several loss functions for ML/DL tasks including Mean Square Error (MSE), Mean Absolute Error (MAE), Mean Squared Logarithmic Error (MSLE), Cross-Entropy (CE), Hing Loss (HL) and Kullback Leibler Divergence Loss (KLDL). MSE, MAE and MSLE apply to regression task while CE, HL and KLDL are used for classification tasks. The most commonly used is the cross-entropy loss.

2.7.1 Cross Entropy

Cross-entropy measures the classification model whose probability in the range of 0 to 1, cross-entropy loss nearer to 0 results low loss and loss nearer to 1 results high loss. We calculate the individual loss for each class in a multi-class classification problem. When the output is probability distribution we use cross-entropy which uses softmax activation in the output layer. Consider a classification sample whose label y is associated with a class k,

and the model output shows that the sample is associated with class $\hat{y}$, so the cost function equation for CE can be given according to the Equation 2.14

$$l_{CE}(y,\hat{y}) = -\sum_{j} y_j \cdot \log(\hat{y}) \tag{2.14}$$

where the probabilities of more than one sample is required, the cost function for the CE can be obtained using the Equation 2.15

$$L_{CE} = -\sum_{i=1}^{N}\sum_{k=1}^{K} P(y_i = k) \cdot \log(Q(y_i = k)) \tag{2.15}$$

where $P(y_i = k)$ is the probability distribution of the i-th sample over the real class, and the probability distribution of the i-th observation being of the class k is $Q(y_i = k)$.

2.7.2 Binary Cross-Entropy

It gives the probability value between 0 and 1 for a classification task by calculating the average difference between the predicted and actual probabilities as shown in Equation 2.15

2.7.3 Categorical Cross-Entropy

This loss function is used for multi-class problems to output a probability over the class C for each image in the problem domain. In the specific (and usual) case of Multi-Class classification the labels are one-hot, so only the positive class C_p keeps its term in the loss. There is only one element of the Target vector t which is not zero t_i=t_p. So discarding the elements of the summation which are zero due to target labels, we can write the function as shown in Equation 2.16.

$$CE = -log\left(\frac{e^{s_p}}{\sum_j^C e^{s_j}}\right) \tag{2.16}$$

where S_p is the CNN score for the positive class.

Also, the derivative function with respect to the positive and negative classes can be expressed as given in Equation 2.17 and Equation 2.18 respectively.

$$\frac{\partial}{\partial_{S_p}}\left(-log\left(\frac{e^{s_p}}{\sum_j^C e^{s_j}}\right)\right) = \left(\frac{e^{s_p}}{\sum_j^C e^{s_j}} - 1\right) \tag{2.17}$$

$$\frac{\partial}{\partial_{S_n}}\left(-log\left(\frac{e^{s_p}}{\sum_j^C e^{s_j}}\right)\right)=\left(\frac{e^{s_p}}{\sum_j^C e^{s_j}}\right) \tag{2.18}$$

2.8 Feature Selection and Handling Data Imbalances

The performance of ML/DL classifiers is strongly related the features that are used to train them and the issues of balanced datasets. In this section, we will discuss some of the basic approaches used by ML/DL experts to ensure good models are achieved.

Feature Selection: In real-world model building, it is almost rare that all the variables of a dataset will contribute to the performance of the model. It becomes important that to reduce ambiguity and complexity of the resulting model, the most important features of the dataset are used to build the model. In the past, many procedures have been established to be used for this task. The following methods have been implemented viz: Filter Method, Wrapper Method, Embedded Method and Hybrid Method (FERREIRA; FIGUEIREDO, 2012).

In the Filter method, the model starts with all features and selects the best features subset based on statistical measures such as Pearson's Correlation, Linear Discriminant Analysis (LDA), ANOVA, Chi-square, Wilcoxon Mann Whitney test, and Mutual Information (MI). All these statistical methods depend on the response and feature variable present in the dataset. As the process continues, each feature is assigned a scoring value using statistical measures. Features are organized in descending order based on the scores and assign ranking for the features. A subset of features is selected using threshold value. Filter method takes less computational time for selecting the best features. Some of the drawbacks include none consideration of the correlation between the independent variables, this leads to selection of redundant features; Failure to recognize the patterns properly during the learning phase (VENKATESH; ANURADHA, 2019).

The wrapper method considers the correlation between the features and class labels as well as the dependencies between the features, hence, more accurate results are obtained compared to the Filter method. It is Computationally more complex and expensive and while Iteratively evaluating the selected feature subset, some features may not be considered for evaluation and dropped. It also has searching overhead which results to overfitting. The Forward Feature Selection (FFS), Backward Feature Elimination (BFE), Exhaustive feature Selection (EFS) and Recursive Feature Elimination (RFE) are the most common ways of implementing this technique (VENKATESH; ANURADHA, 2019).

The Embedded Method encompasses the Filter and Wrapper methods by including interactions of features. This method works in a way that the best features are selected during the learning process. The blending of feature selection during learning process has advantages of improving computational cost, classification accuracy and also avoids training the model each time when a new feature is added. Despite the advantages, its major drawback is that it cannot work well with high dimensionality data. Some of the well known implementation of the embedded method includes LASSO Regularization (L1) and the Random Forest Regressor (STAŃCZYK; JAIN, 2015).

Random Forests is a kind of a Bagging Algorithm that aggregates a specified number of decision trees. The tree-based strategies used by random forests naturally rank by how well they improve the purity of the node, or in other words a decrease in the impurity (Gini impurity) over all trees. Nodes with the greatest decrease in impurity happen at the start of the trees, while notes with the least decrease in impurity occur at the end of trees. Thus, by pruning trees below a particular node, we can create a subset of the most important features. This characteristics of Random Forest Regressor made it our choice in selecting the features for our model.

2.9 Related Works

There are numerous works that have been done by various researchers in the area of implementing CNN architecture in IDS. In this section, the most relevant related works to our research are presented. Developing a precise and accurate model requires a holistic approach consisting of the IDS type, ML algorithms, evaluation metrics and deployment. Riyaz e Ganapathy (2020) proposed an IDS model based on CNN network to detect intrusion in wireless networks. According to the authors, there was need to develop a new feature selection approach called Conditional Random Field and Linear Correlation Coefficient-based Feature Selection (CRF-LCFS) to choose the most important features for obtaining a better performing model. The work was implemented and test on KDDCup 99 dataset and achieved an accuracy of about 98.89%. Although the accuracy was high, the result is biased as it cannot be applied to a real-world intrusion scenario due to the dataset used in the model evaluation.

Wu, Chen e Li (2018) proposed an IDS framework which uses the CNNs to automatically select relevant features from the dataset and to solve data imbalance problem, authors

set the cost function weight coefficient of each class based on its numbers. However, this approach has some deficiencies as compared to SMOTE as it leads to data loss. Moreover, the developed model decreases the false alarm rate (FAR) and enhances the classification accuracy. The NSL-KDD dataset was used for evaluating the performance of the model. This presents another challenge with the work. Unlike our proposed system, this model was implemented for binary classification where the network traffic is identified as either normal or attack.

An ID model CNNs-based classifier for enhancing the precision of model have been proposed by Lin *et al.* (2018). In the proposed model, the LaNet-5 architecture was adapted and revised with the adaptive delta optimization algorithm to fine-tune the model parameters and minimize a classification error by using error derivatives of back-propagation and quick response to intrusion detection using Tensor flow. The results obtained showed an improvement in the accuracy of attack classification using the behaviour features of the trained CNNs. The KDDCup 99 dataset was used and the model obtained an accuracy of 97.53%. In Varanasi e Razia (2021), authors also proposed a DL-IDS using the CNN architecture for attack detection. The proposed model was evaluated on the CICIDS2017 dataset and compared with DNN model. The result showed that the CNN model achieved an accuracy of 99% compared to the DNN that showed an accuracy of 98%.

In Maseer *et al.* (2021), different ML models were proposed for IDS in computer networks using ANN, DT, k-NN, NB, RF, SVM, CNN, EM, K-means, and Self Organizing Map (SOM) were proposed and tested on CIC-IDS2017 which contained various attack types. The AID system based on K-NN, DT, and NB models achieved excellent performance, whereas the SOM and EM models achieved poor performance due to their high FP and FN alarms. In Sun *et al.* (2020), a DL-IDS which uses the hybrid network of CNN and LSTM to extract the spatial and temporal features of network traffic data and to provide a better IDS was proposed. Authors used a category weight optimization method to improve the robustness thereby overcoming the issues of unbalanced number of samples in the different attacks present in the dataset. The hybrid model was used for multi-class classification task and the DL-IDS reached 98.67% in overall accuracy, and the accuracy of each attack type was above 99.50%. Ho *et al.* (2021) also proposed an IDS based on CNN and evaluated on the CICIDS2017 dataset to classify network flows into normal or malicious activities. The proposed model reached a True Negative Rate of 98.98% on the test data and 99.01% on the

training data. Although the model showed high negative rate, it still suffers from bias as the data used contains imbalance which was not discussed or handled by the authors.

In Kim, Shin e Choi (2019), a CNN based IDS was proposed using different methods and developed an IDS model for the CICIDS2018 dataset, which is a dataset sharing the same feature set with CICIDS2017 but with larger sample counts. The training and test of the models in the study were performed on sub-datasets which included a subset of types of network traffic from CICIDS2018. Hence, the models were simulated for multi-class classification for certain classes in the dataset, not using all of them at once. According to the authors, the experimental results showed that the performance of the CNN based IDS could be higher than that of the recurrent neural network (RNN), which is another DL model popularly used for time series data analysis. The CNN model proposed in this study reached a 96.77% accuracy in the sub-dataset which was composed by the benign and DoS samples from CICIDS2018. On the other hand, the RNN model tested in this study reached a 82.84% accuracy in the same dataset, which was significantly lower than of the CNN model.

In Khan (2021), a convolutional recurrent neural network (CRNN) was proposed to create a DL-based hybrid ID framework that predicts and classifies malicious cyberattacks in a network. In the hybrid convolutional recurrent neural network intrusion detection system (HCRNNIDS), the CNN performs several convolution operation to capture local features, and the RNN captures temporal features to improve the ID system's performance and prediction. In the work, authors based their research on the CSE-CIC-IDS20018 dataset which is publicly available for research purposes. So, the efficacy of the model was evaluated on this dataset. The simulation outcomes prove that the proposed HCRNNIDS substantially outperformed other ID methodologies studied in the work, attaining a high malicious attack detection rate accuracy of up to 97.75% for CSE-CIC-IDS2018 data with 10-fold cross-validation. Meanwhile, the authors did not categorically handle the concerns with imbalanced data in their research.

A comparative study involving several DL algorithms for cyber security IDs was proposed by Ferrag *et al.* (2020). In the study, the authors implemented 10 different DL models including the CNN on the CSE-CIC-IDS2018 and Bot-IoT datasets using the Google Colaboratory[4] platform using Python and TensorFlow with GPU acceleration. Authors used only 5% of the entire datasets for their study and achieved good performances for the deep

[4] https://colab.research.google.com/

autoencoders and CNN models. The highest accuracy for the Bot-IoT dataset (98.39%) was obtained with a deep autoencoder, while the highest accuracy for CICIDS2018 (97.38%) was obtained with an RNN. The highest recall for the Bot-IoT dataset (97.01%) came from a CNN, whereas the highest recall for CICIDS2018 (98.18%) came from a deep autoencoder.

Many other authors have proposed several CNN models that where SMOTE was used to handle data imbalance in the datasets. Lin, Ye e Xu (2019) proposed an LSTM DL model for dynamic anomaly detection in networks. The author used the CSE-CIC-IDS2018 dataset to train and evaluate the model which achieved an accuracy of 96.2%. Meanwhile, to achieve this accuracy, the SMOTE technique together with improved cost function were used to handle class imbalance in the dataset, hence, obtaining a model that performs better that some other models while presenting more practical application. Zhang *et al.* (2020a) proposed a NIDS that combined the features of SMOTE and under-sampling for clustering based on Gaussian Mixture Model (GMM), and referred to as SGM. The SGM-CNN, which integrates imbalanced class processing with CNN, and investigate the impact of different numbers of convolution kernels and different learning rates on model performance was tested on UNSW-NB15 and CICIDS2017 datasets. The model achieved a detection rate of 99.74% and 96.54% for UNSW-NB15 and CIC-IDS2017 respectively, on a binary class classification and a detection rate of 99.85% for a multi-class classification.

Following the deficiencies identified in the related works such as the use of out-dated dataset, ignoring data imbalance which leads to inability of the model to generalize on the dataset, not considering multi-class approach to IDS design, model complexity and high computational resource usage, our developed IDS model addresses these concerns. In our contribution, we solved the challenge of data imbalance using two different techniques and compared the results on a more comprehensive dataset. Using a search algorithm, we obtained a more concise parameters to train our model thereby reducing training time. Through transfer learning, we leveraged the existing models as feature extractor, hence reducing computational cost. To ensure that our model can be deployed in any device, we perform model quantization that reduces the memory capacity of the IDS while maintaining a high level of accuracy.

3 MATERIALS AND METHOD

In this section, we discuss the materials and method used in our work including to achieve our objectives.

3.1 Materials (Tools Used)

3.1.1 Software

In the development of the project, we have used Python programming language, Jupyter Notebook integrated with the Anaconda IDE and other libraries discussed in this section. Python[1] is free and open source object-oriented programming language which is widely used for Machine learning and data science projects due to its simple syntax and dynamic structure. Writing and analyzing codes in Python is very easy. It also presents another major advantages of easily accessible documentation in books, articles, videos, and others . Also, Python can easily be integrated with other languages and libraries written in other languages are compatible. This work uses the latest version of Python 3.10 to develop the model.

Sklearn[2] - (Scikit-learn) is a machine learning library that can be used with the Python programming language. Sklearn offers a wide range of options to the user with its numerous machine learning algorithms having extensive documentation and contains all the algorithms needed for this work. Machine learning can be trained and evaluated with the functions available in the sklearn library.

Pandas[3] is a powerful data analysis library running on Python. When working with a large dataset, Pandas allows you to easily perform many operations such as filtering, bulk column / row deletion, addition, and replacement. Because of all these advantages, the Pandas library has been used. We have used Pandas to read the datasets and perform other functions associated with dataframes. Pandas-profiling is a feature of Pandas framework that generates a comprehensive analysis report of the dataset including the correlation, feature importance. Although this works best with small dataset, we used it to explore some important feature of our dataset.

[1] https://www.python.org/
[2] https://scikit-learn.org/stable/
[3] https://pandas.pydata.org/

Matplotlib[4] is a library that runs on Python, allowing visualization of data. This library is used to create graphs used in the study. We have used this library to create plots and visualization of the data points to understand the distributions of the datasets. Other Data visualization tools used include seaborn[5] and Plotly[6] packages. Plotly is s powerful visualization tool that enable interactive display of objects.

NumPy[7], a Python library that allows you to perform mathematical and logical operations quickly and easily, has been used in calculations in this work. All the mathematical functions associated with the data processing was done using numpy. We also used it to convert dataframes into arrays for the training of our model.

Tensorflow[8] and Keras[9] are deep learning frameworks integrated with Python language. TensorFlow is an end-to-end open source platform for deep learning tasks especially those involving large amount of datasets and requiring high computational power. These libraries have been used in the development of our DL models along with other dependencies. In this work, the TensorFlow is used as the brain to build all the DL models.

3.1.2 Hardware

One of the major evaluation criteria of ML/DL models is the execution time. However, the execution time may vary depending on the performance or configuration of the computing systems used in the training process. As a result, it is necessary to disclose the technical specification of devices used. For this work, the technical characteristics of the computer used in the implementation phase are discussed as follows:

- Central Processing Unit: Intel(R) Core(TM) i7-7700 CPU @ 3.60GHz, 3600Mhz, 4 Core(s)
- Random Access Memory: 16 GB (15.9 GB usable)
- Operating System: Windows 10 64-bit
- Graphics Processing Unit: NVIDIA GeForce GTX 1050 Ti

[4] https://matplotlib.org/
[5] https://seaborn.pydata.org/
[6] https://plotly.com/python/
[7] https://numpy.org/doc/stable/
[8] https://www.tensorflow.org/
[9] https://keras.io/

3.2 Performance Evaluation Metrics

There are many metrics that are used to measure the performance of ML/DL models which are trained to detect network intrusion. Key parameters applied to the metrics are defined as follow:

- True Positive (TP_{os})- represents the number of connection records that are correctly classified to the Normal class.
- True Negative (TN_{eg}) - the total number of connection records that are correctly classified to the Attack class.
- False Positive (FP_{os}) - the number of normal connection records that have been mistakenly categorized as an attack connection record.
- False Negative (FN_{eg}) - the amount of Attack connection records that have been incorrectly classified as normal connection entries.

1) **Accuracy:** The accuracy of a model estimates the ratio of the correctly recognized connection records to the entire test dataset. If the accuracy is higher, the machine learning model is better (Accuracy ϵ [0, 1]). Accuracy serves as a good measure for the test dataset that contains balanced classes and is defined by the equation given in Equation.

$$\text{Accuracy} = \frac{TP_{os} + TN_{eg}}{TP_{os} + TN_{eg} + FP_{os} + FN_{eg}} \tag{3.1}$$

2) **Precision**: It is used to estimate the ratio of the correctly identified attacks records to the overall number of identified attack connections. The higher the precision, the better the performance of the ML model (Precision ϵ [0, 1]). Precision is defined by the function

$$\text{Precision} = \frac{TP_{os}}{TP_{os} + FP_{os}} \tag{3.2}$$

3) **Recall or True Positive Rate (TPR):** It's also known as sensitivity and calculates the proportion of Attack connection records that are successfully categorized to the overall number of Attack connection records. A higher value for the TPR implies that the ML model is better (TPR ϵ [0, 1]). Mathematically, it is represented by the relation shown in Equation 3.3.

$$\text{Recall} = \frac{TP_{os}}{TP_{os} + FN_{eg}} \tag{3.3}$$

4) **F1-Score**: F1-Score otherwise referred as F1-Measure is the harmonic mean of recall and precision. A high F1-score shows that the ML model performs better (F1 Score ϵ [0, 1]).

$$\text{F\_measure} = \frac{2 \times (precision \times recall)}{precision + recall} \tag{3.4}$$

5) **False Positive Rate (FPR):** The ratio of Normal connection records classified as Attacks to the total number of Normal connection records is calculated by FPR. A low value of FPR is needed to obtain an ML model with good performance (FPR ϵ [0, 1]) as shown in the relationship of Equation 3.5.

$$\text{FPR} = \frac{\text{FP}_{os}}{\text{FP}_{os} + \text{TN}_{eg}} \tag{3.5}$$

6) **ROC Curve:** An ROC curve (Receiver Operating Characteristic curve) is a graphical representation of the performances of a classification model concerning at all classification thresholds. The curve usually takes two parameters which are the TPR and FPR. The evaluation metrics work well only in binary classification problems, hence, presents challenges as not all problems are binary classification. There are some proposed strategies to work in multi-class classification scenarios (LANDGREBE; DUIN, 2008; YANG *et al.*, 2021).

7) **AUC:** Area Under the Curve (AUC) measures the ability of a classifier to distinguish between different classes and is used as a summary of the ROC curve (FERRIS *et al.*, 2015). With higher AUC, an implemented model performs better at distinguishing between the positive and negative classes. The AUC can be calculated for multi-class problems using the Equations 3.6 and 3.7.

$$AUC = \frac{1}{c(c-1)} \sum_{j=1}^{c} \sum_{k>j}^{c} (\text{AUC}(j \mid k) + \text{AUC}(k \mid j)) \tag{3.6}$$

where c is the number of classes and $\text{AUC}(j \mid k)$ is the AUC with class j as the positive class and class k as the negative class. In general, $\text{AUC}(j \mid k) \neq \text{AUC}(k \mid j)$ in the multiclass case (PEDREGOSA *et al.*, 2011).

Equation 3.7 extends Equation 3.6 to be used for calculating roc-auc curve which are weighted by prevalence. This algorithm is used by setting the keyword argument multi-class to 'OneVsOne (ovo) or OneVsRest (OVR)' and average to *'weighted'*. The *'weighted'* option returns a prevalence-weighted average as described in Fawcett (2006).

$$AUC = \frac{1}{c(c-1)} \sum_{j=1}^{c} \sum_{k>j}^{c} p(j \cup k)(\mathrm{AUC}(j \mid k) + \mathrm{AUC}(k \mid j)) \tag{3.7}$$

8) **Matthew's Correlation Coefficient (MCC)**

MCC provided by Sci-kit learn library is available in the package *K.matthews_corrcoef(y_true, y_pred, *, sample_weight=None)* where K represents the sklearn.metrics function is used to measure the quality of binary or multi-class classification tasks. It accounts for the true and false positives and negatives and is generally regarded as a balanced measure which can be used even if the classes have different sizes. The MCC is in essence a correlation coefficient value between -1 and +1. A coefficient of +1 represents a perfect prediction, 0 an average random prediction and -1 an inverse prediction. The statistic is also known as the phi coefficient[10], [11]. Given that tp, tn, fp and fn are the true positive, true negative, false positive and false negative outputs of a classification problem in a confusion matrix, the MCC for a binary classification can be represented with the function in Equation 3.8.

$$MCC = \frac{tp \times tn - fp \times fn}{\sqrt{(tp+fp)(tp+fn)(tn+fp)(tn+fn)}} \tag{3.8}$$

In the multi-class case, the Matthews correlation coefficient can be defined in terms of a confusion_matrix C for K classes. To simplify the definition consider the following intermediate variables:

- $t_k = \sum_i^K C_{ik}$ the number of times class k truly occur.
- $p_k = \sum_i^K C_{ki}$ the number of times class k was predicted.
- $c = \sum_k^K C_{kk}$ the total number of samples correctly predicted.

[10] https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0041882
[11] https://en.wikipedia.org/wiki/Phi_coefficient

– $s = \sum_i^K \sum_j^K C_{ij}$ the total number of samples.

Then, the multi-class MCC can be defined as given in Equation 3.9.

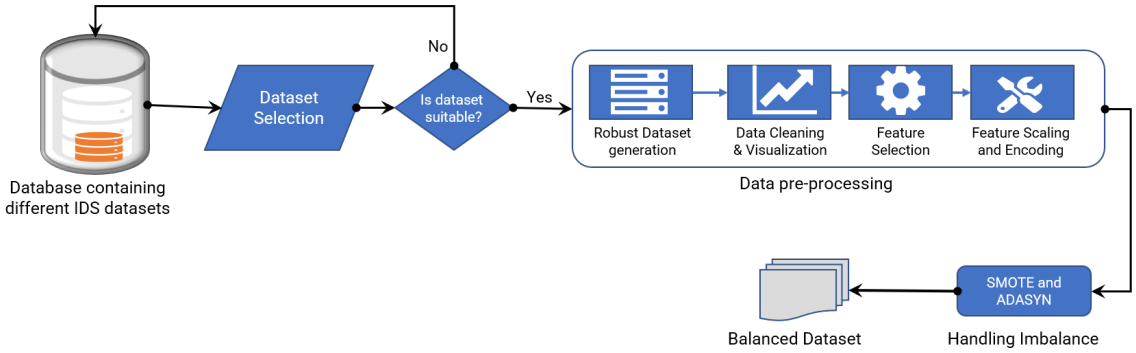
$$MCC = \frac{c \times s - \sum_k^K p_k \times t_k}{\sqrt{\left(s^2 - \sum_k^K p_k^2\right) \times \left(s^2 - \sum_k^K t_k^2\right)}} \tag{3.9}$$

In a situation where more than one labels exist, the value of the MCC will no longer range between -1 and +1. Instead the minimum value will be somewhere between -1 and 0 depending on the number and distribution of ground true labels. The maximum value is always +1 (BUITINCK *et al.*, 2013)

3.3 Method for Dataset Preparation and Preprocessing

Having a suitable dataset is as good as having a well performing model. Data acquisition appears to be the most cumbersome and time consuming phase in data science and machine learning. In this section, we describe the various steps taken to achieve a suitable dataset for the training of our model. Figure 3.1 show the flow of actions in preparing the dataset to be used in the IDS development. The work flow involve the dataset selection phase, generating a novel dataset from the selected datasets, preprocessessing the generated dataset *(data cleaning, balancing, feature encoding, feature selection)*, data data splitting.

Figure 3.1 – Dataset Pre-processing flowchart for the IDS



Source: Author (2022)

3.3.1 Database Selection

This is the most critical aspect of intrusion detection systems model development as performance of a model is to a large extent dependent on it. In the literature, more

than 10 different datasets were presented. To achieve the objectives of this work, we have selected the CIC-IDS2017 and CSE-CICIDS2018 datasets to develop our IDS models. These two datasets contain the most recent attack types as well as highest number of attacks compared to other datasets. Henceforth, the name CIC-IDS2017 and IDS2017 will be used to mean the same thing while CSE-CICIDS2018 and IDS2018 will also be used interchangeably. Table 3.1 and Table 3.2 present the distribution streams of records in the IDS2017 and IDS2018 datasets respectively. The IDS2018 dataset contains 16,232,943 traffic instances among which about 17% of the dataset is attack type while 83% are normal network traffic. Hence, there is serious data imbalance. Also, in the IDS2017 dataset, the total number of instances is 3,119,345 consisting of approximately 81% benign and 19% attack activities. Tables 3.1 and 3.2 show that records and percentage distribution of features in IDS2017 and IDS2018 datasets respectively. Although the Table 3.1 shows that the Benign attacks contains 83.345%, this value was achieved after some irrelevant features such as the Timestamp, Flow ID, Source IP, Destination IP are removed.

Table 3.1 – Distribution of stream records in CICIDS2017 dataset

Label Name	**Value**	**Percentage (%)**
BENIGN	2359289	83.3452
DoS Hulk	231073	8.1630
PortScan	158930	5.6144
DDoS	41835	1.4779
DoS GoldenEye	10293	0.3636
FTP-Patator	7938	0.2804
SSH-Patator	5897	0.2083
DoS slowloris	5796	0.2048
DoS Slowhttptest	5499	0.1943
Bot	1966	0.0695
Web Attack - Brute Force	1507	0.0532
Web Attack - XSS	652	0.0230
Infiltration	36	0.0013
Web Attack - Sql Injection	21	0.0007
Heartbleed	11	0.0004

Source: Author (2022)

Table 3.2 – Distribution of stream records in CSE-CIC-IDS2018 dataset

Label Name	**Value**	**Percentage (%)**
Benign	13484708	83.07001
DDOS attack-HOIC	686012	4.22605
DDoS attacks-LOIC-HTTP	576191	3.54952
DoS attacks-Hulk	461912	2.84552
Bot	286191	1.76303
FTP-BruteForce	193360	1.19116
SSH-Bruteforce	187589	1.15561
Infiltration	161934	0.99756
DoS attacks-SlowHTTPTest	139890	0.86177
DoS attacks-GoldenEye	41508	0.25570
DoS attacks-Slowloris	10990	0.06770
DDOS attack-LOIC-UDP	1730	0.01066
Brute Force -Web	611	0.00376
Brute Force -XSS	230	0.00142
SQL Injection	87	0.00054

Source: Author (2022)

3.3.2 Data Preprocessing

In this stage of developing our model, data cleaning, selection of important features, dimensionality reduction, encoding and standardization of the data are performed. We use this stage to understand and prepare the dataset to meet our need. The two datasets used in this work contains over 80 features each. First, we generate a new dataset containing all the attacks and benign activities in one CSV file. This is necessary because the CICIDS2017 and CSE-CIC-IDS2018 contains 7 and 10 different CSV files respectively with each file containing both benign and attacks. For better performance of our model using a comprehensive dataset, it is important to merge these files into a new dataset. Pandas library was used to perform this task of generating a robust dataset. It is noteworthy that to the best of our knowledge, this work is the first to use all the network files in the IDS2017 and IDS2018 datasets to model an IDS for computer networks.

3.3.2.1 Data Cleaning and Visualization

To avoid bias in our model, we need to clean the data to ensure there are no duplicate values, missing values in the form of Not a Number (NAN), -nan, naf. We also drop all columns that does not contribute to the performance of our model. These are columns whose mean is zero. This means that the features do not have correlation with other features. This

activity is generally called Exploratory Data Analysis (EDA). It involves the use of statistical tools to gain more insight about the dataset. Python provides the matplotlib, plotly, seaborn, pandas framework and pandas-profiling function for visualization and manipulation of Dataframes. From the EDA, we observed that the features of the two datasets are provided in both forward and backward directions of network flow and packets. CICFlowMeter[12] *(software for extracting network traffics)* generates Bidirectional Flows (BiFlow), where the first packet determines the forward (source to destination) and backward (destination to source) directions, hence the 83 statistical features such as Duration, Number of packets, Number of bytes, Length of packets, etc. are also calculated separately in the forward and reverse direction. The distribution of the streams of data points in the two datasets are shown in Tables 3.1 and 3.2. About 288,602 data instances are incorrect incomplete in the IDS2017 dataset, so these were removed resulting to the features consisting of almost 83% benign and 17% attack in both datasets.

Also, an error was observed in the columns of the dataset. Such properties as Flow ID, Source IP, Source Port are not necessary for the model training as they are not part of the packets transmitted from the attack devices but are usually assigned by the network capturing and extraction software called CICFlowMeter. In both datasets, the "Flow Bytes/s", "Flow Packets/s"features include the values "Infinity"and "NaN"in addition to the numerical values, which can be modified to -1 and 0 respectively to make them suitable for machine learning algorithms.

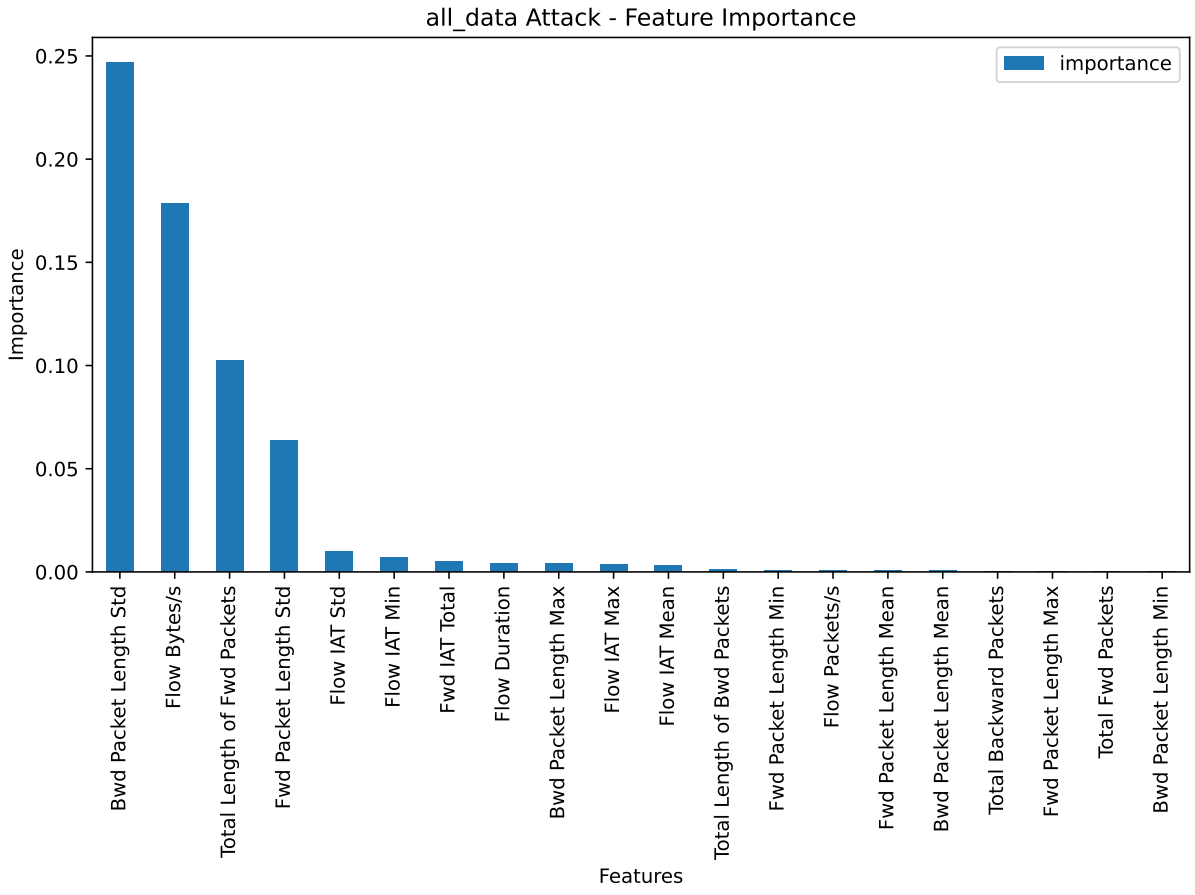
3.3.2.2 Feature Selection

In a network traffic, every packet play important role in deciding the generalized property of such flow defining it as an attack or not. Notwithstanding, not all these features help in the IDS performance, This is why feature selection is important as it helps to reduce model overloading. In our work, the Random Forest Regressor (RFR) was used for the feature selection. In all, a total of 64 out of the 80 features with higher significance were extracted for each of the datasets and for each attack type. During the selection phase, we calculate the importance using the standard deviation of the total values in a particular network traffic which then determines to what extent the feature contributes to the characteristic of that flow. The features were used in the both ML/DL based implementation of the

[12] https://www.unb.ca/cic/research/applications.htmlCICFlowMeter

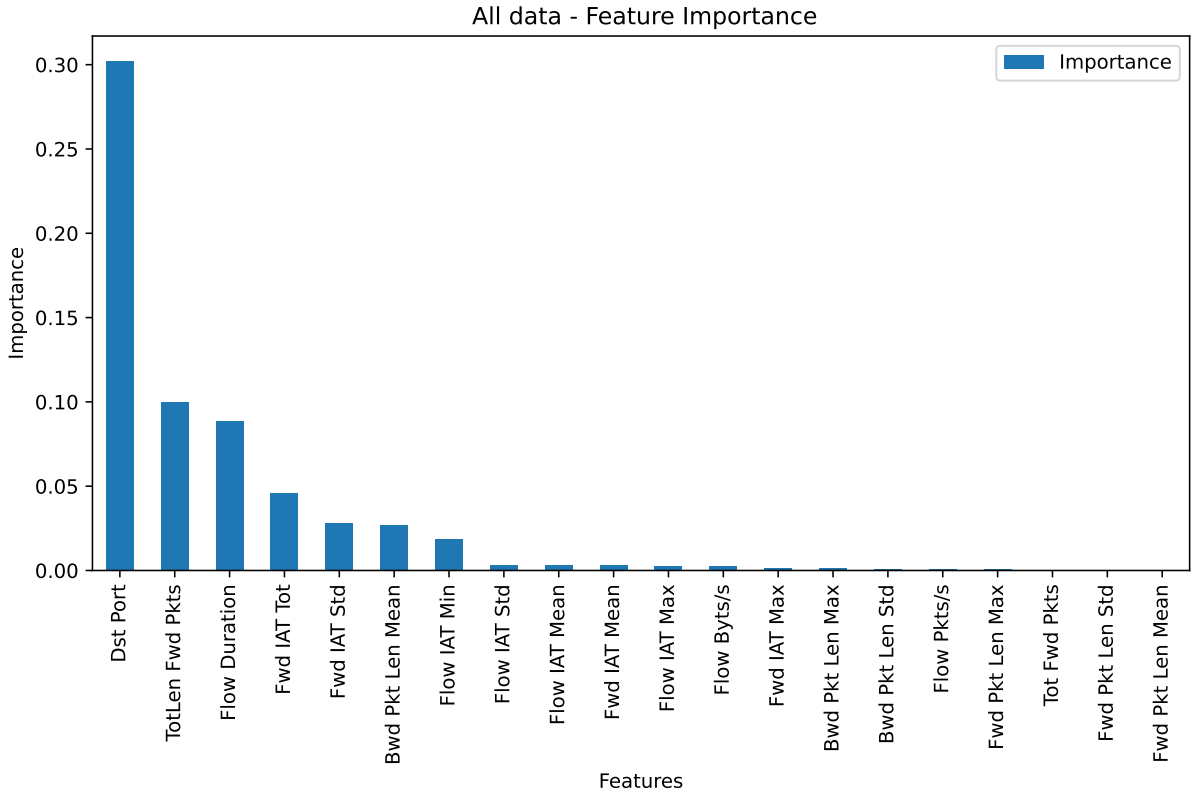
proposed models. However, Deep Learning algorithms such as CNN are incorporated with the capacity to extract the most important features necessary for obtaining a better performance automatically. We use this characteristics of CNN to select the most important feature of the datasets necessary for obtaining more precise results and increasing the model performance. Figures 3.2 and 3.3 show the first 20 selected most important features for the entire dataset consisting of all attack types and benign activities for CICIDS2017 and CSE-CICIDS2018 datasets respectively. We observe that Bwd Packet Length Std with a standard deviation of 0.51486000 is the strongest feature that determine the nature of network flows in the CICIDS2017 dataset but the Dst Port having the standard deviation of 0.301898 mostly defines the behaviour of all features of the CSE-CICIDS2018 dataset.

Figure 3.2 – Selected Important features of all attacks and benign activities for IDS2017 dataset.



Source: Author (2022)

Figure 3.3 – Selected Important features of all attacks and benign activities for IDS2018 dataset.



Source: Author (2022)

Regarding the classification of the network traffic as attacks or benign, we observe that different features the nature of each packet class, hence, we calculate the feature importance for describing each of the network flows as attack or not. Tables 3.3 and 3.4 show the first seven most important features for each attack type selected for the ML while Table 3.5 represents selected features for all the labels in the CIC-IDS2017 dataset . Similarly, Tables 3.6 and 3.7 show the most important features selected for each attack and Table 3.8 demonstrates the selected feature for the entire dataset with their corresponding importance for the CICIDS2018 dataset, respectively.

Table 3.3 – Distribution of Seven most important features of each attack types in CICIDS2017

Attack & Feature Name	**Weight Importance**		**Attack & Feature Name**	**Weight Importance**
Bot			**DoS Hulk**	
Bwd Packet Length Mean	0.351579		Bwd Packet Length Std	0.51486
Flow IAT Max	0.012488		Fwd Packet Length Std	0.06757
Flow IAT Min	0.011538		Fwd Packet Length Max	0.00521
Flow IAT Std	0.00978		Flow IAT Min	0.00189
Flow Duration	0.009112		Flow Duration	0.00143
Flow IAT Mean	0.001649		Total Backward Packets	0.00038
Flow Bytes/s	0.001461		Flow IAT Max	0.00027
DDoS			**DoS Slowhttptest**	
Bwd Packet Length Std	0.470994		Flow IAT Mean	0.644977
Total Backward Packets	0.093082		Fwd Packet Length Min	0.07578
Fwd IAT Total	0.009132		Bwd Packet Length Mean	0.024823
Flow Duration	0.006553		Fwd Packet Length Std	0.021984
Total Length of Fwd Packets	0.006271		Fwd Packet Length Mean	0.017339
Flow IAT Min	0.005916		Bwd Packet Length Std	0.012957
Flow IAT Std	0.005452		Total Length of Bwd Packets	0.00781
DoS GoldenEye			**DoS Slowloris**	
Flow IAT Max	0.512499		Flow IAT Mean	0.46975
Total Backward Packets	0.0335		Total Length of Bwd Packets	0.075171
Flow IAT Min	0.02252		Bwd Packet Length Mean	0.031522
Bwd Packet Length Std	0.018482		Fwd IAT Total	0.012912
Fwd Packet Length Min	0.007583		Total Fwd Packets	0.010462
Fwd Packet Length Max	0.001629		Fwd Packet Length Max	0.000881
Flow IAT Mean	0.001367		Flow Bytes/s	0.000766

Source: Author (2022)

Table 3.4 – Distribution of Seven most important features of each attack types in CICIDS2017 (cont'd)

Attack & Feature Name	Weight Importance		Attack & Feature Name	Weight Importance
FTP-Patator			**Heartbleed**	
Fwd Packet Length Max	0.201079		Bwd Packet Length Mean	0.052
Fwd Packet Length Std	0.02709		Total Backward Packets	0.044
Fwd Packet Length Mean	0.010558		Bwd Packet Length Max	0.04
Bwd Packet Length Mean	0.000773		Total Length of Bwd Packets	0.04
Total Length of Bwd Packets	0.00048		Flow IAT Min	0.04
Flow IAT Min	0.000214		Total Fwd Packets	0.036
Total Fwd Packets	0.000204		Total Length of Fwd Packets	0.032
Infiltration			**PortScan**	
Total Length of Fwd Packets	0.167736		Flow Bytes/s	0.3124
Fwd Packet Length Mean	0.110285		Total Length of Fwd Packets	0.3047
Total Backward Packets	0.030409		Flow IAT Max	0.0006
Bwd Packet Length Mean	0.023994		Flow Duration	0.0003
Bwd Packet Length Std	0.019637		Fwd IAT Total	0.0002
Flow Duration	0.013585		Flow IAT Mean	0.0002
Flow IAT Min	0.009838		Fwd Packet Length Max	0.0001
SSH-Patator			**Web Attack**	
Fwd Packet Length Max	0.001212		Bwd Packet Length Std	0.006545
Flow Duration	0.000652		Total Length of Fwd Packets	0.002554
Flow IAT Mean	0.00055		Flow Bytes/s	0.002129
Flow IAT Max	0.00054		Bwd Packet Length Max	0.00183
Flow IAT Std	0.00043		Fwd Packet Length Std	0.001519
Flow Packets/s	0.000359		Flow IAT Min	0.001285
Total Length of Fwd Packets	0.000274		Fwd Packet Length Max	0.001051

Source: Author (2022)

Table 3.5 – Selected important features for all data in the CICIDS2017 Dataset

Features	Importance		Features	Importance
Bwd Packet Length Std	0.51486000		Flow IAT Mean	0.00012480
Fwd Packet Length Std	0.06757244		Bwd Packet Length Mean	0.00003376
Fwd Packet Length Max	0.00521211		Flow IAT Std	0.00003127
Flow IAT Min	0.00189296		Flow Bytes/s	0.00003034
Flow Duration	0.00142572		Bwd Packet Length Max	0.00001560
Total Backward Packets	0.00038006		Total Length of Fwd Packets	0.00000868
Flow IAT Max	0.00027001		Bwd Packet Length Min	0.00000814
Flow Packets/s	0.00019481		Total Fwd Packets	0.00000711
Total Length of Bwd Packets	0.00017732		Fwd Packet Length Mean	0.00000675
Fwd IAT Total	0.00016574		Fwd Packet Length Min	0.00000017

Source: Author (2022)

Table 3.6 – Important Features for each Attack Selected for Training ML Models in CICIDS2018

Attack & Feature Name	Weight Importance		Attack & Feature Name	Weight Importance
Bot			**DoS attacks-Hulk**	
Dst Port	0.9778002		Dst Port	0.1217184
Bwd Pkt Len Mean	0.0016399		Fwd Pkt Len Std	0.0005904
Fwd Pkt Len Mean	0.0001549		Bwd Pkt Len Std	0.0004640
Tot Fwd Pkts	0.0000391		Bwd Pkt Len Max	0.0004525
Flow Duration	0.0000346		Fwd Pkt Len Max	0.0004123
Flow Pkts/s	0.0000320		Fwd Pkt Len Mean	0.0001294
Flow IAT Mean	0.0000291		TotLen Fwd Pkts	0.0001158
DDOS attack-HOIC			**DoS attacks-SlowHTTPTest**	
Dst Port	0.1860614		Flow Duration	0.001858
Flow Duration	0.0006483		Flow IAT Max	0.001823
Flow IAT Max	0.0001887		Flow IAT Mean	0.001689
Flow IAT Std	0.0000182		Flow Pkts/s	0.001408
Fwd Pkt Len Mean	0.0000175		Dst Port	0.00068
TotLen Bwd Pkts	0.0000164		TotLen Fwd Pkts	0.000000
Bwd Pkt Len Max	0.0000138		TotLen Bwd Pkts	0.000000
DDOS attack-LOIC-UDP			**DoS attacks-Slowloris**	
Tot Fwd Pkts	0.204000		Dst Port	0.1244648
TotLen Fwd Pkts	0.152000		Flow IAT Mean	0.0000197
Dst Port	0.000019		Flow Pkts/s	0.0000181
Bwd Pkt Len Max	0.000000		Flow IAT Max	0.0000151
Flow IAT Std	0.000000		Flow Duration	0.0000125
Flow IAT Mean	0.000000		Flow Byts/s	0.0000057
Flow Pkts/s	0.000000		TotLen Fwd Pkts	0.0000051

Source: Author (2022)

Table 3.7 – Important Features for each Attack Selected for Training ML Models in CICIDS2018 (Cont'd)

Attack & Feature Name	Weight Importance		Attack & Feature Name	Weight Importance
DDoS attacks-LOIC-HTTP			**FTP-BruteForce**	
Flow IAT Max	0.30239180		Dst Port	0.00521824
TotLen Fwd Pkts	0.24173650		Flow Duration	0.00193359
Flow Duration	0.12394880		Flow IAT Max	0.00162984
Dst Port	0.03648575		Flow IAT Mean	0.00146722
Bwd Pkt Len Std	0.00399764		Tot Fwd Pkts	0.00000012
Flow IAT Mean	0.00137322		Bwd Pkt Len Min	0.00000000
Fwd Pkt Len Std	0.00100608		Flow IAT Std	0.00000000
DoS attacks-GoldenEye			**Infiltration**	
Dst Port	0.11784600		Dst Port	0.113557
Flow IAT Max	0.00000344		Flow Byts/s	0.096748
Flow IAT Mean	0.00000344		Flow IAT Max	0.053656
Flow Duration	0.00000330		Flow IAT Mean	0.034847
Tot Bwd Pkts	0.00000234		Flow Duration	0.032149
Flow Pkts/s	0.00000151		Flow Pkts/s	0.02261
Tot Fwd Pkts	0.00000041		Flow IAT Std	0.018963
SSH-Bruteforce			**Web Attack**	
Dst Port	0.9410732		Dst Port	0.231259
Flow Pkts/s	0.0002628		Fwd Pkt Len Mean	0.142589
Flow IAT Mean	0.0002277		Flow IAT Max	0.073328
Flow IAT Max	0.0000304		Flow Duration	0.009586
Flow IAT Std	0.0000169		Flow IAT Mean	0.008623
Flow Duration	0.0000045		Fwd Pkt Len Min	0.005125
Protocol	0.0000000		TotLen Fwd Pkts	0.002925

Source: Author (2022)

Table 3.8 – Selected important features for all data in the CICIDS2018 Dataset

Features	Importance		Features	Importance
Dst Port	0.301898		Flow IAT Max	0.002816
TotLen Fwd Pkts	0.099690		Flow Byts/s	0.002406
Flow Duration	0.088873		Fwd IAT Max	0.001474
Fwd IAT Tot	0.046164		Bwd Pkt Len Max	0.001353
Fwd IAT Std	0.028198		Bwd Pkt Len Std	0.001102
Bwd Pkt Len Mean	0.027077		Flow Pkts/s	0.000941
Flow IAT Min	0.018644		Fwd Pkt Len Max	0.000934
Flow IAT Std	0.003375		Tot Fwd Pkts	0.000456
Flow IAT Mean	0.003282		Fwd Pkt Len Std	0.000307
Fwd IAT Mean	0.002989		Fwd Pkt Len Mean	0.000246

Source: Author (2022)

3.3.2.3 Feature Scaling and Label Encoding

Usually, it is important to have the various labels encoded into numerical values so that the algorithm can present accurate interpretation of them. Also, there are high

differences in the values of the data points. Some are in thousands while some are in tens. This can create a high bias in the model performance, hence, there is need to scale the values to lie between 0 and 1. The *LabelEncoder, StandardScaler, MaxAbsScaler and MinMaxScaler* methods of sklearn were tested. In the end, we observed the MaxAbsScaler method achieved better results as it scales and translates each feature individually such that the maximal absolute value of each feature in the training set will be 1.0. It does not shift/center the data, and thus does not destroy any sparsity.

3.3.2.4 Handling Data Imbalance

One of the factors that affects the performances of ML/DL models is the issue of data imbalance which occurs when there is an uneven distribution of the instances in the dataset. In our case, the datasets were observed to contain data imbalance since it contains about 83% benign features and only 17% attack features. According to Gosain e Sardana (2017) there are many approaches which have been used to handle data imbalance which are categorized into Over-sampling or Under-sampling techniques.

In ML, there are several method of handling data imbalance which are grouped as either oversampling or undersampling technique. In oversampling, samples of the existing data points are generated based on the characteristics of the existing data. In this work, we used the method proposed in Chawla *et al.* (2002) called SMOTE and Borderline_Smote proposed by Han, Wang e Mao (2005). Unlike the Undersampling techniques which randomly removes some of the majority class, the SMOTE and Borderline_Smote generates synthetic instances of the minority class based on the existing data points, thereby ensuring no information loss. Assuming that T, N and k defined by SMOTE(T, N, k) are inputs to the SMOTE algorithm where T is the number of minority class samples; N is the amount of SMOTE and k is the number of nearest neighbours, the output of the operation can be define as $(N/100) * T$ synthetic minority class samples (CHAWLA *et al.*, 2002).

These techniques are used to generate artificial minority examples along the line segments joining the minority samples and its 'k' minority class nearest neighbors. Based on the rate of oversampling required, the neighbors from the 'k' nearest neighbors are randomly chosen. Hence, this technique implements the kNN algorithm in its core. we have chosen this technique due to its ability to overcome overfitting and the challenge of removing important information from the dataset as experienced in other methods.

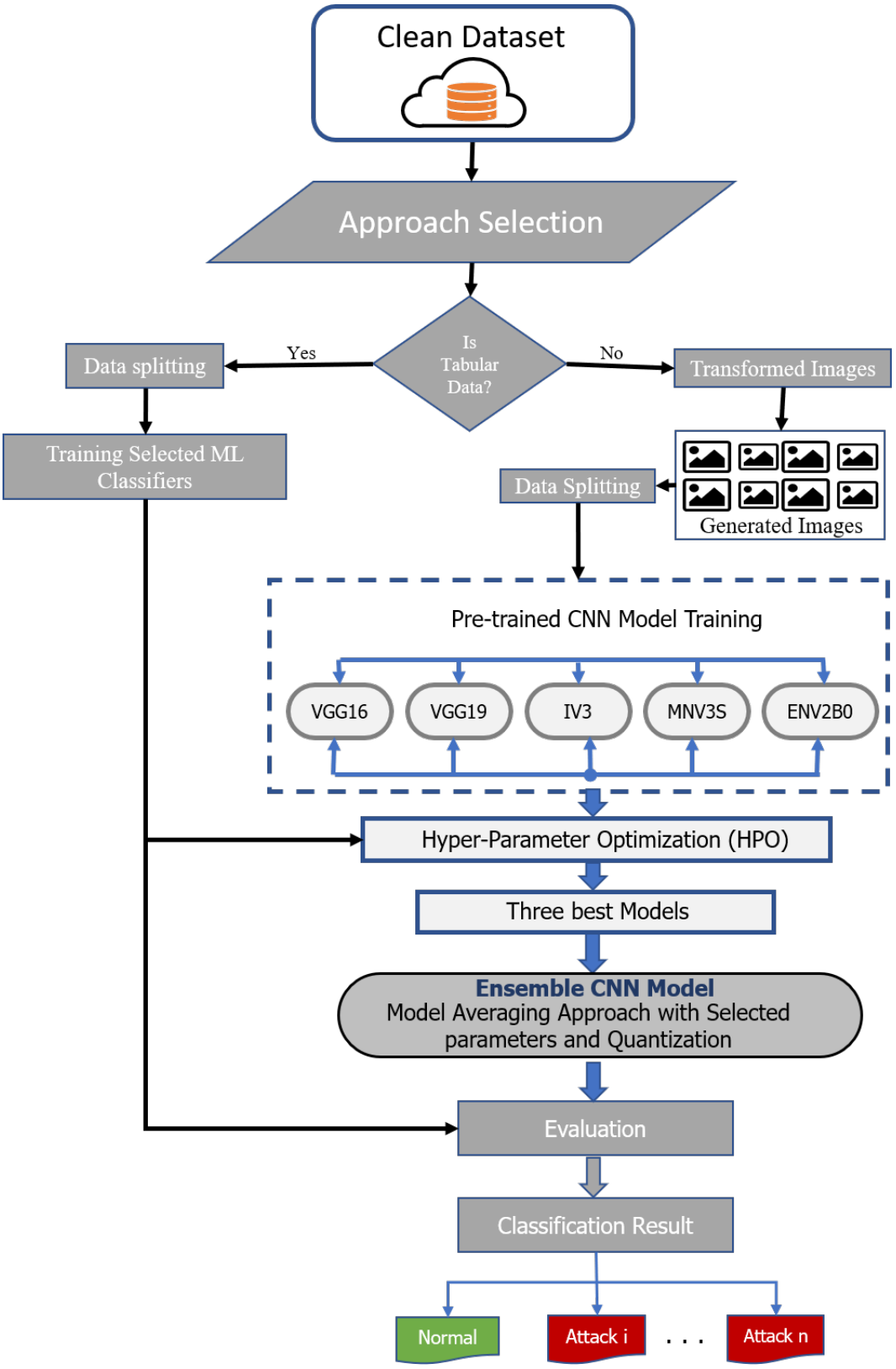
3.3.3 Creation of Training and Testing Data

In order to obtain our model, it needs to be trained, validated and tested on different samples of the datasets. The behaviour of the model on the test dataset determines the performance of our model. The two datasets used in this project has only one unbundled CSV file created during the preprocessing stage. Therefore, we split each of the datasets into training, validation and testing data. Usually, ML operations used 80:20 percent ratio for the train-test sets. This approach over time has proven to be inefficient as the model fails to generalize well on the dataset. This paved the way for the use of cross-validation while training the model. Sklearn library provides different cross-validation split mechanisms such as K-Fold, StratifiedKFold etc. In our work, the K-Fold and StratifiedKFold methods were tested and at the end, StratifiedKfold of 10 splits was adopted as it yielded better performance because it ensures that each set contains approximately the same percentage of samples of each target class as the complete set.

3.4 Proposed IDS Model Implementation

The general implementation approach for the proposed model architecture is shown in Figure 3.4. Considering our dataset that is originally in tabular form, we develop two different models where one is based on tabular data and the other on image data according to the flowchart shown in Figure 3.4. The first model is trained using classical ML algorithms *(Random forest, Decision Tree, AdaBoost, Extra tree, LightGBM and XGBoost)* discussed earlier in Chapter 2 and One-Dimensional CNN (1D-CNN) on the tabular data. The second model is achieved using two dimensional (2D) CNN architecture based on pre-trained models implemented on image data as discussed later in Section 3.4 for transfer learning.

Figure 3.4 – Flowchart for the Implementation of the Model Development



Source: Author (2022)

CNN is widely used for its special properties of automatically learning and extracting features from images for image recognition and segmentation tasks (TEODORO *et al.*, 2021a).

Images can easily be supplied as inputs into the CNN without additional feature description, extraction and reconstruction process. A typical CNN comprises of convolutional layers, pooling layers and fully connected layers.

$$\text{Correlation}(f*g)(i,j) = \sum_a \sum_b f(a,b)g(i+a,j+b) \tag{3.10}$$

$$\text{Convolution}(f*g)(i,j) = \sum_a \sum_b f(a,b)g(i-a,j-b). \tag{3.11}$$

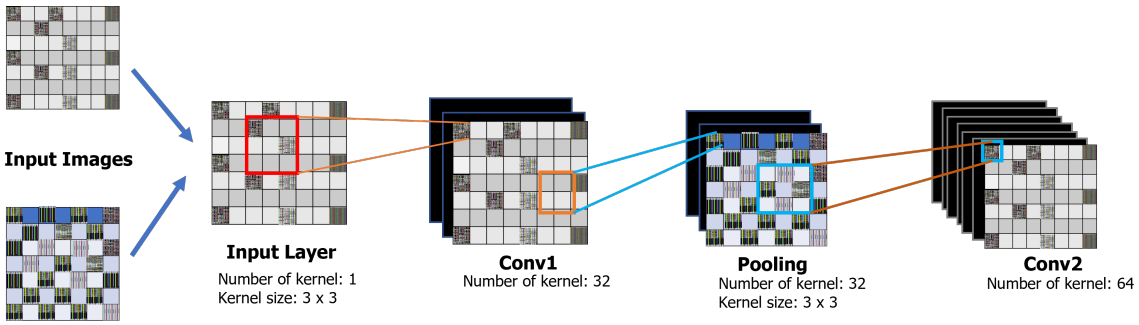
In convolutional layers, the feature patterns of images are automatically extracted by convolution operations using filters of different sizes which can be 3 X 3, 4 X 4 etc but generally an nxn matrix of weights. Through back propagation, the filter values get updated iteratively as the model learns the patterns in the images going by the epochs. For this reasons, many authors and ML engineers usually employ the cross-correlation operation that is computationally cost effective because inverting the filter at each operation is unnecessary. According to Zhang *et al.* (2020), the functions used for determining the cross-correlation and spatial features of convolution operation are quiet alike as shown in Equations 3.10 and 3.11, respectively.

In our research, we have used the 3 X 3 filter to perform the convolution operation leading the feature learning and extraction in the base CNN architecture. In pooling layers, the data complexity can be reduced without losing important information through local correlations to avoid overfitting. Also, Pooling layer sub-samples the output of the layer above which reduces the dimensionality resulting in fewer learned parameters. When a pooling layer is used in conjunction with a convolutional layer it adds translation invariance to the network.

Optionally, Dropout and Batch Normalization layers can also be added to improve the performance of the learning process depending on the input parameters and model architecture. A dropout layer works by omitting data with a fixed probability to help prevent overfitting. It also reduces bias towards weights from the layer before the dropout layer. The effects of a dropout layer are larger when the dataset is small. Batch normalization is a technique that normalizes the values with respect to the current batch, it can be applied to either the inputs directly or the activation of the previous layer. Batch normalization speeds up training of the network, adds regularization and reduces the generalization error in the network.

The output of the input layer and hidden layers after applying all the necessary parameters are Flatten into a one dimensional array and passed into the Dense (Fully Connected Layers). Fully-connected layer consists of neurons that are connected to all neurons of the layer before, these are the layers that are used to build an ANN. The fully-connected layer is sometimes referred to as a dense or linear layer and they serve as a channel to connect all feature maps and generate the output. Figure 3.5 shows a typical convolution operation in a CNN.

Figure 3.5 – A typical 2 layer convolution operation in a CNN



Source: Author (2022)

Transfer Learning (TL) is a system in DL where the weights of existing models trained on large amount of data are reused in another dataset (TEODORO *et al.*, 2021a). Sometimes, TL may be the best option for image classification tasks where datasets are not enough as CNN requires a large amount of dataset for good performance. The successful application of TL in image tasks is because the feature patterns learned by the bottom layers of CNN models are usually general patterns that are applicable to many different tasks, and only the features learned by the top layers are specific features for a particular dataset. Therefore, the bottom layers of CNN models can be directly transferred to different tasks. To improve the effectiveness of TL, fine-tuning can be used in the TL process of IDS models design. In fine-tuning, most of the layers of the pre-trained model are frozen (i.e., their weights are retained), while a few of the top layers are unfrozen to re-train the model on a new dataset. Fine-tuning enables the learning model to update the higher-order features in the pre-trained model to better fit the target task or dataset (LI *et al.*, 2021). One of the most important issues in the TL methods is unifying distribution of the source and target samples (WEISS; KHOSHGOFTAAR; WANG, 2016). Maximum Mean Discrepancy (MMD) (BORGWARDT *et al.*, 2006) is one of the successful distribution distance estimators that is

used frequently in TL methods. MMD estimates distance between two distributions based on Reproducing Kernel Hilbert Space (RKHS) (BORGWARDT *et al.*, 2006).

In the proposed architecture, we have selected VGG16, VGG19, MobileNetV3 (MNV3), EfficientNetV2 (ENV2B0) and InceptionV3 (IV3) as the base learners due to their success in image classification tasks (PETROV; HOSPEDALES, 2019; TEODORO *et al.*, 2021a). These are pre-trained CNN models on the ImageNet dataset which is a benchmark dataset for image classification tasks consisting of over 14 million images of 1, 000 classes (PETROV; HOSPEDALES, 2019). The VGG architecture is composed of convolutional layers (CLs) and rectified linear unit (ReLU) activation function. VGG come in two different variations of VGG16 and VGG19 with the VGG16 having a total of 16 layers and VGG19 having 19 layers; and both implementing the 3 X 3 filter dimensions for the CLs (ALOM *et al.*, 2018). Due to the wide adoption (PETROV; HOSPEDALES, 2019; ALOM *et al.*, 2018) and non-complexity of implementation of the VGG architecture with both the 16 and 19 layers variations, we have used both in this work to demonstrate its applicability in IDS systems. Another CNN based pre-trained model architecture that has high relevance in application is the Inception model that was published by GoogLeNet having the V1, V2 and V3 variations (IOFFE; SZEGEDY, 2015). With V2 and V3 being the improved versions of the original V1 architecture, the V2 modified the V1 with the inclusion of batch normalization layers (IOFFE; SZEGEDY, 2015) for training streamlining and improved performance while V3 that included larger spatial features and factoring convolutions for improved computational efficiency. We have used InceptionV3 (IV3) architecture in our work as it is more flexible and lightweight in comparison to the earlier versions in terms of memory requirement and trainable parameters. IV3 has a file size of 92 MB and 23.9 million trainable parameters (SZEGEDY *et al.*, 2016).

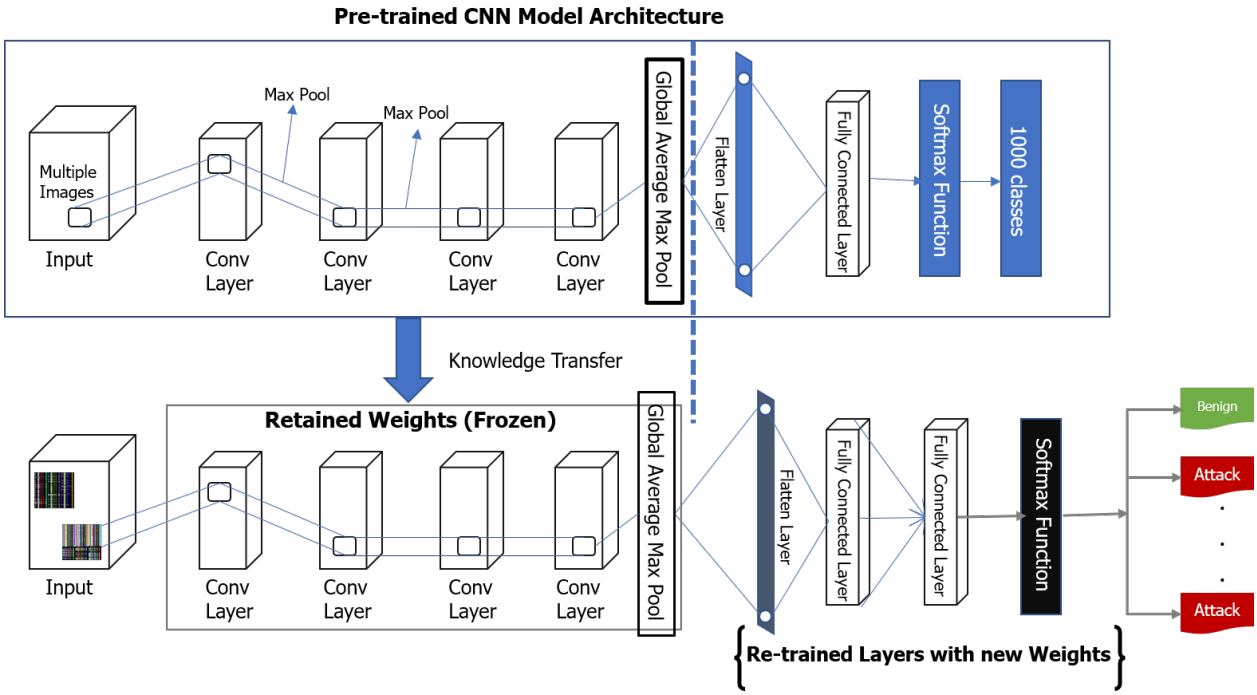
As the depth, width and resolutions of the network architecture increases, more computational cost is required for training. To overcome this, the study by Tan e Le (2019) proposed the EfficientNet CNN architecture that dynamically grows all the dimensions efficiently using a simple composite coefficient. Based on the mobile inverted bottleneck convolution (MBConv), the various versions of the EfficientNets ranging from EfficientNetB0 to EfficientNetB7 were developed. The EfficientNetV2 comes with higher performance advantages to include faster training speed and better parameter efficiency than previous architectures (TAN; LE, 2021). This architecture family has variations of EfficientNetV2B0-B4

and EfficientNetV2S, EfficientNetV2M and EfficientNetV2L that were developed using a combination of training-aware neural architecture search and scaling, to jointly optimize training speed and parameter efficiency. The neurons and other parameters were searched from the search space enriched with new ops such as Fused-MBConv to obtain a algorithm that trained faster than state-of-the-art algorithms while being up to 6.8x smaller. Based on the literature, we selected the EfficientNetV2B0 (ENV2B0) with 79MB memory capacity and 7.2 million training parameters (TAN; LE, 2021; TAN; LE, 2019).

In Howard *et al.* (2017) another class of efficient architecture called MobileNets that target applications on mobile and embedded devices was proposed. Using a streamlined architecture that operates on Depth-Wise Separable Convolutions (DWSC), authors developed the light weight CNN pre-trained models with better performances on edge devices. Two simple global hyper-parameters that permits sufficient trade off between accuracy and latency were implemented; thus, grants model builder privilege to define and choose the best model sizes for their projects. Just like other pre-trained models, the MobileNet comes in different forms to include the MobileNet, MobileNetV2 and MobileNetV3 (Small and Large) (HOWARD *et al.*, 2017; HOWARD *et al.*, 2019; SANDLER *et al.*, 2018). For the purpose of this work, we selected the MobileNetV3Small (MNv3S) that has 2.9 million trainable parameters and 14 MB memory consumption.

In all, for the TL part of the proposed model training process, the VGG16, VGG19, MNV3S, ENV2B0 and IV3 architecture were re-trained on the datasets together with base CNN and were used to implement the proposed IDS. Top three (top-3) performing models are selected to construct the ensemble model proposed in this work. The flowchart for implementing the proposed model is shown in Figure 3.4 and the overall architecture of the TL approach is shown in Figure 3.6.

Figure 3.6 – Transfer Learning Model Implementation Architecture



Source: Author (2022)

3.4.1 Image Generation and Formatting

After the preprocessing stage, we need to prepare the data to be suitable as an input to the CNN architecture. AS earlier stated, the CNN achieves better performance when working on image data. Since our data is a tabular data in the CSV file format, we need to carry out transformation to obtain image samples according to (LI *et al.*, 2021; HUSSAIN *et al.*, 2020a). There are mainly three basic transformation algorithms which are frequently being used for data transformation. They are Normalization with Min Max Scaler (MMS), Quantile Transformation (QT) and Power Transformation (PT). MMS defined by Equation 3.12 is known to be the most used normalization technique but it's drawback is hinged on its inability to completely handle outliers, thus, may lead to some values been extremely small. For this reason, we adopted the QT method proposed in (LOKMAN *et al.*, 2019).

$$X' = \frac{X - Min(X)}{Max(X) - Min(X)} \times 255 \tag{3.12}$$

The QT normalization method transforms the feature distribution to a normal distribution and re-calculates all the features values based on the normal distribution. Therefore, the majority of variable values are close to the median values, which is effective in handling

outliers. Based on the formula, QT distributes all features according to the same specified pattern as shown in Equation 3.13.

$$Q = G^{-1}(F(x)) \tag{3.13}$$

where F is the cumulative distribution function of the feature and G^{-1} is the quantile function of the desired output distribution G[13] (BUITINCK *et al.*, 2013). Power transforms are a family of parametric, monotonic transformations that aim to map data from any distribution to as close to a Gaussian distribution as possible in order to stabilize variance and minimize skewness.

During the image generation phase, based on the timestamp and feature size of the network traffic dataset, the data samples are converted in chunks of various sizes. In the case of our dataset, each of them has 64 important features which needs to be converted into 3 channel image representing the RGB color. Therefore, each color image generated is transformed to 64 x 64 x 3 total feature values, meaning that each channel of the RGB has 64 features. This implies that, the first 64 samples of each chunk were converted into image matrix of channel 1, next 64 samples of each chunk were converted into image matrix of channel 2 and the last 64 samples of each chunk were converted into image matrix of channel 3, then, all are generally mapped into the RGB channel of the image. This means that each chunk of the dataset consists of 64 X 3 = 192 consecutive data samples. This step is repeated until all the labels in both datasets are correctly transformed. It is important to note here that the time-series correlation of the original network traffic data can be retained since the images generated are based on the timestamps of the sample data, hence, obtained results are assured of correctness.

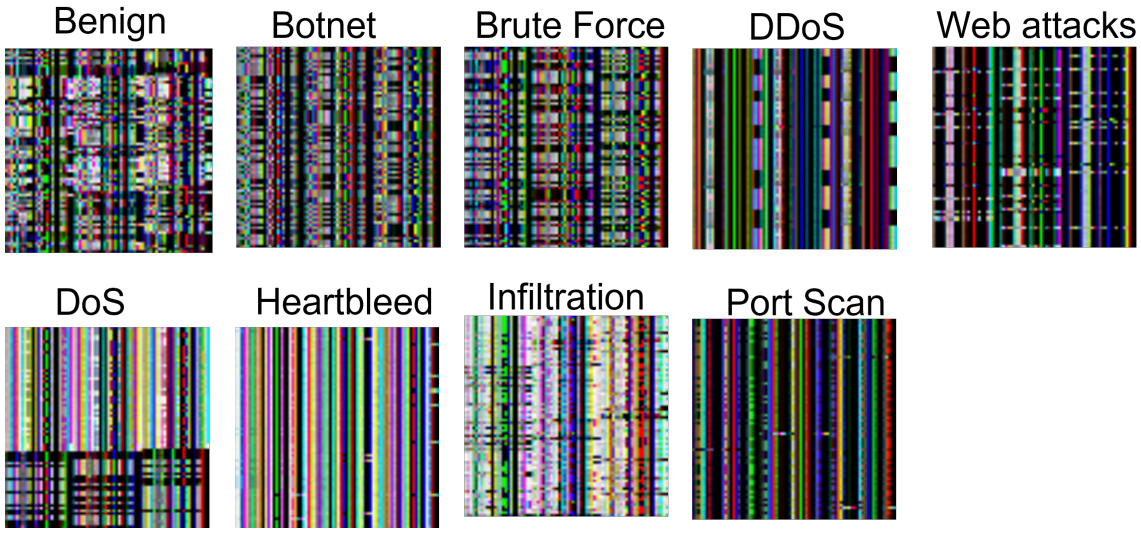
Following the transformation, we label the generated images accordingly based on the attack pattern in the chunks. For instance, image labelled "Normal"if the sample in the chunk/image are normal. Also, if the sample in the chunk/image contains attack samples, the image is labelled with the corresponding frequent attack type in the chunk to either of the attack types in the datasets including "DDoS Attack", "Web Attack", "Infiltration", "DoS", "Botnet", "PortScan", "Heartbleed"as the case may be.

The last pre-processing procedure is re-scaling the image to be in suitable input format for the CNN models. The images generated are in the range of 64 X 64 X 3 but the

[13] https://en.wikipedia.org/wiki/Power_transform

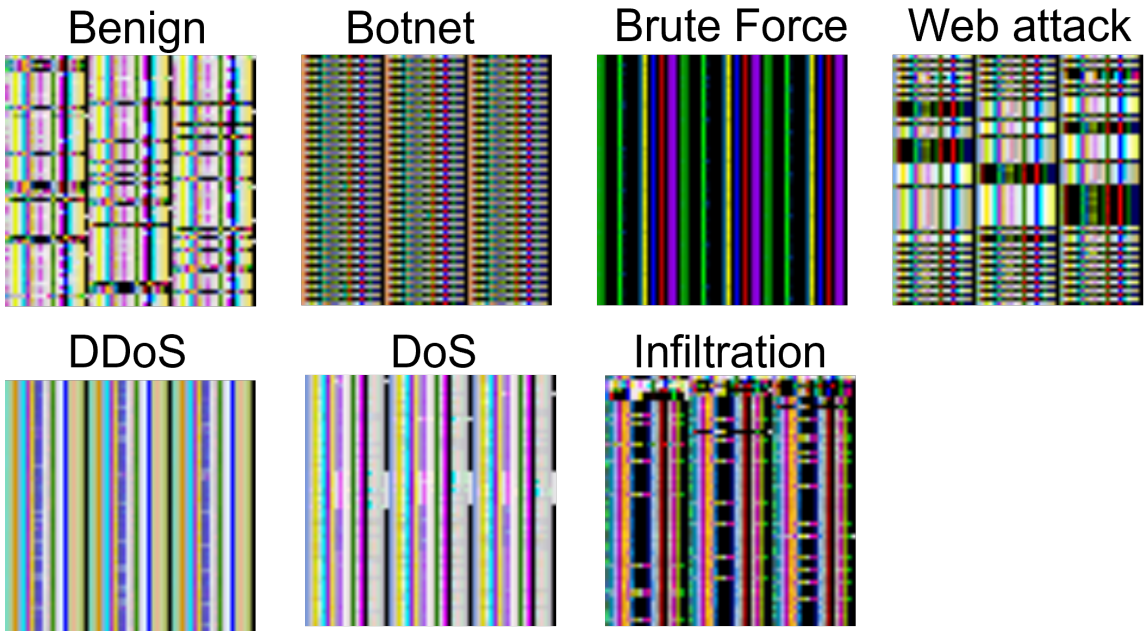
pre-trained models such as VGG16, MobileNetV3 and others expect images in the range of 224 X 224 X 3. So, we resized the images in the range of 256 X 256 X 3. This will allow the CNN to learn easily all the patterns in the image and therefore improve the learning speed as the filters convolve on the images. Representative sample of the images in the datasets are shown in Figure 3.8 and 3.7 respectively.

Figure 3.7 – Image Samples of the CSE-CIC-IDS2017 dataset after conversion



Source: Author (2022)

Figure 3.8 – Image Samples of the CSE-CIC-IDS2018 dataset after conversion



Source: Author

From both datasets, it can be seen that there are large differences in the feature patterns between the normal samples and different types of attacks. The feature patterns of Web attacks images in Figure 3.8 are more random concentrated at the top and bottom of the images and sparse at the center while in Figure 3.7, are some what more dense through out the image. The difference in the patterns of the images results in the frequency and techniques of each attack strategy employed by the attacker. Sequel to this variations, the CNN model is able to learn more features there making it more robust for attack detection and classification.

3.4.2 ELETL-IDS: Ensemble Transfer Learning Model

In this section, we discuss the proposed ensemble model. Ensemble learning is a technique that combines various base learning models to create an enhanced single model. Because an aggregation of several learners performs better than a single learner in detection rate and specificity (YANG; MOUBAYED; SHAMI, 2021; DAS *et al.*, 2021).

Model averaging is an ensemble technique where multiple sub-models contribute equally to a combined prediction. This may lead to overfitting which is a deep challenge in ML tasks. A variation of this approach, called a weighted average ensemble, weighs the contribution of each ensemble member by the trust or expected performance of the model on a holdout dataset. This allows well-performing models to contribute more and less-well-performing models to contribute less. The weighted average ensemble provides an improvement over the model average ensemble, hence, it has been used in this work and compared with concatenation method. With Softmax activation function which returns the predicted probability for each class in a classification task, we obtain the confidence of each class. The model averaging method calculates the average classification probability of base learners for each class, and then returns the class label with the highest average confidence which are output of the softmax function as discussed in (LARGE; LINES; BAGNALL, 2019).

$$\mathrm{Softmax}(\mathbf{z})_i = \frac{e^{\mathbf{z}_i}}{\sum_{j=1}^{C} e^{\mathbf{z}_j}} \tag{3.14}$$

where Z is the input vector, C is the number of classes in the dataset, e^{z_i} and e^{z_j} are the standard exponential functions for the input and output vectors, respectively. The predicted class label obtained by the model confidence averaging method can be defined by

Equation 3.15. This is obtained by taking the $argmax$ of the predicted labels in the selected class

$$\hat{y} = \underset{i \in \{1,\cdots,c\}}{\operatorname{argmax}} \frac{\sum_{j=1}^{k} p_j \left(y = i \mid B_j, \boldsymbol{x}\right)}{k} \tag{3.15}$$

where B_j is the j_{th} base learner, k is the number of selected base CNN learners, and k = 3 in the proposed IDS; $p_j(y = i|Bj, x)$ indicates the prediction confidence of a class value i in a data sample x using B_j.

Given that N is the number of instances, K is the number of base CNN models, and C is the number of classes, we can calculate the computational complexity of the ELETL-IDS and that of the time for the model averaging method as $O(NKC)$. With small values of K and C, the speed of execution of the averaging method is usually high.

3.4.3 Best Model through Hyper-Parameter Optimization (HPO)

One of the most important steps in ML/DL tasks is handling hyper-parameters (HPs). While the model itself adjusts the model parameters during the learning process to achieve optimal results, we can fine tune the hyper-parameters such that it aids the model to achieve best performance at optimized time. Such hyper-parameters as learning rate, epochs, Batch Normalization, Dropout, weights are usually tuned for the model. In the proposed TL framework, the dropout rate, learning rate and the percentage of frozen layers are defined as model-design parameters while batch size, epochs and early stop patience level are model-training hyper-parameters aimed at increasing the model training speed and overall performance. These parameters directly influence the architecture, efficiency and effectiveness of the CNN models.

HPO is an automated process of selecting the best parameters for model performance in ML or DL using optimization techniques. Particle Swam optimization (PSO), Genetic Algorithm (GA), Random Search (RS), Bayesian Optimization - Tree Parzen Estimator (BO-TPE) (YANG; SHAMI, 2020) are some of the many search algorithms for performing HPO. TPE algorithm is implemented on Hyperopt (a library for hyperparams tuning with bayesian optimization in Python). We have adopted the BO-TPE (BERGSTRA; YAMINS; COX, 2013) technique in thus research due to its ability to keep conditional dependencies; yet providing an optimal performance results with efficiency with all types of HPs.

3.4.4 Deep Learning Model Quantization

Generally, ML/DL models are usually heavy due to the number of parameters which were trained to get a highly accurate model. Where these models are to be deployed on cloud servers, there is no challenge as the servers have the capacity to host these models. Notwithstanding, the issue of latency still exist. But when we are meant to use these models on edge devices such as mobile phones, tablets, embedded device and other devices, this becomes a huge problem. This challenge is solved using the quantization mechanism.

Model Quantization according to Guo (2018), Krishnamoorthi (2018), Wang *et al.* (2019), Banner, Nahshan e Soudry (2019) involves replacing data types with reduced width data types. In the IDS datasets, the features are encodes and Float64 and Int64. These require too large memory space for the dataset to be loaded into memory which also results to a heavy model. In this approach, quantization helps to transform the data types of the features of the trained model with little or no loss of accuracy. For example, replacing 32-bit Floating Point (FP32) with 8-bit Integers (INT8). The values can often be encoded to preserve more information than simple conversion. Quantization can be post-training Quantization (PTQ) or Training-Aware Quantization (TAQ). In post-training quantization, the chosen quantization method is applied to the resulting model while the TAQ is applied during the model training. Both methods yields good accuracy with reduced model size as described Jacob *et al.* (2018) and Krishnamoorthi (2018).

In PTQ, model size can be reduced to a desirable size by compressing the wights and/or both weight and activation for better inference without retraining the model (KRISHNAMOORTHI, 2018). This method is simpler to use but may lead to loss of accuracy. Through weight only quantization, we reduce the precision of the weights of the network to 8-bits float format. This is a simple approach as validation of the result may not be required; thus somehow inefficient. On the other hand, we can choose to quantize both the weights and activation to achieve better precision, hence, calibration data is required along with calculated dynamic ranges of activation.

The latter approach, TAQ deals with building the model and training it from scratch. This approach can lead to improved accuracy in comparison to the corresponding floating point 32bit (FP32) results (NOGAMI *et al.*, 2019), but with high compression rate. However, to build, train and tune the new model can be a lot demanding and time consuming (FUKETA *et al.*, 2018; LIN; TALATHI; ANNAPUREDDY, 2016). In this work, we utilized the Model

Optimization Toolkit[14] provided by TensorFlow called *TensorFlow Lite or TFLite*. This library was used to perform both TAQ and PTQ on our model such that it can be applied to devices which : (i) have low memory capacity and therefore can only run optimized models (ii) has low energy resources, hence, need to optimize energy usage in order to save battery life (iii) requires low latency thereby presenting faster inference time to the users making them assume that the process happens instantaneously.

Particularly designed for mobile, edge, or Internet of Things (IoT) devices, TFLite optimizes for performance, model size, and power consumption. Additionally, it supports Graphic Processing Units (GPU) representatives for model inference on the GPU. Through their Application Peripheral Interfaces (APIs), these delegates can interact with the native libraries for GPU acceleration. For instance, the GPU delegate leverages the Metal API for iOS devices and the Android Neural Network API for Android devices to perform hardware-accelerated inference operations.

In our work, we implemented both the TAQ and PTQ methods on the 1D-CNN model while on the ensmble model, we only used the PTQ method. This is because the ensemble architecture does not support Train Aware Quantization due to the heterogeneous nature of the layers consisting the ensemble network.

[14] https://www.tensorflow.org/model_optimization/guide/quantization/training

4 RESULTS AND DISCUSSION

In this chapter, we present and discuss the main results obtained in this work ranging from the models trained on tabular data and the transfer learning models implemented on image data. The results are present in different segments. The entire work is carried out on multi-class classification of the network traffic using the method described earlier. The results presented are based on the various metrics defined in Chapter 3. First, we merged the similar attack types such as *DoS GoldenEye, DoS Slowloris, DoS Hulk, DoS Slowhttptest* were combined into a single DoS attack. Similarly, the *FTP-Patator, SSH-Patator* combined into Brute Force and *Web Attack Sql Injection, Web attack XSS, Web attack Brute Force* combined into the Web Attack class for the IDS2017 dataset. This procedure was carried out on IDS2018 dataset in which *DDOS attack-HOIC, DDoS attacks-LOIC-HTTP, DDOS attack-LOIC-UDP*: **DDoS**, *DoS attacks-Hulk, DoS attacks-SlowHTTPTest, DoS attacks-GoldenEye, DoS attacks-Slowloris*: **DoS**, *FTP-BruteForce, SSH-Bruteforce*: **Brute Force** and *Brute Force -Web, Brute Force -XSS, SQL Injection*: **Web Attacks** classes. We encoded the categorical feature (label) into numeric data using the $LabelEncoder$ function. The result of the merged dataset is shown in Table 4.1.

Table 4.1 – Distribution of samples in the Data sets after merging similar attacks to obtain the 7 classes for IDS2018 and 9 classes for IDS2017

CIC-IDS2017			CSE-CICIDS2018		
Attack	Distribution	Encoded Labels	Attack	Distribution	Encoded Labels
Benign	2359289	0	Benign	13484708	0
DoS	252661	4	DDoS	1263933	3
PortScan	158930	7	DoS	654300	4
DDoS	41835	3	BruteForce	380940	2
BruteForce	13835	2	Botnet	286191	1
WebAttacks	2180	8	Infiltration	161934	5
Botnet	1966	1	WebAttacks	923	6
Infiltration	36	6	Heartbleed	Nil	Nil
Heartbleed	11	5	PortScan	Nil	Nil

Source: Author (2022)

4.1 Models Developed on Tabular Data

Different machine learning classifiers including the Random Forest (RF), Adaboost (AD), Decision Tree (DT), LightGBM (LGBM), Extra Tree (ET) , XBGoost (XGB) and 1D-CNN

were trained on the datasets. For each algorithm, we first used the default parameters as shown in Table 4.2 to train and evaluate the performance of the resulting IDS model; then, performed tuning to using *GridSearchCV*. The obtained parameters from the grid was used to fine-tune the model and compared the results in terms of train time, test time, error and other metrics.

Table 4.2 – Default parameters for the ML Algorithms

Algorithm	Default Parameters
Random Forest Classifier	n_estimators=100, criterion='gini', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0,max_features='auto', max_leaf_nodes=None, min_impurity_decrease=0.0, bootstrap=True,oob_score=False,n_jobs=None, random_state=None,verbose=0,warm_start=False, class_weight=None,ccp_alpha=0.0,max_samples=None
Extra Tree Classifier	n_estimators=100,criterion='gini',max_depth=None, min_samples_split=2,min_samples_leaf=1, min_weight_fraction_leaf=0.0,max_features='auto', max_leaf_nodes=None,min_impurity_decrease=0.0, bootstrap=False,oob_score=False,n_jobs=None, random_state=None,verbose=0,warm_start=False, class_weight=None,ccp_alpha=0.0,max_samples=None
Decision Tree Classifier	criterion='gini',splitter='best',max_depth=None, min_samples_split=2,min_samples_leaf=1, min_weight_fraction_leaf=0.0,max_features=None, random_state=None,max_leaf_nodes=None, min_impurity_decrease=0.0,class_weight=None, ccp_alpha=0.0
AdaBoost	base_estimator=None,n_estimators=50,learning_rate=1.0, algorithm='SAMME.R',random_state=None,
LGBMClassifier	boosting_type='gbdt',num_leaves=31,max_depth=-1, learning_rate=0.1,n_estimators=100, subsample_for_bin=200000,objective=None, class_weight=None,min_split_gain=0.0, min_child_weight=0.001,min_child_samples=20, subsample=1.0,subsample_freq=0,colsample_bytree=1.0, reg_alpha=0.0,reg_lambda=0.0,random_state=None, n_jobs=-1,silent=True,importance_type='split',

4.1.1 Classical Machine Learning Model Performance

The results obtained for the different IDS models is presented in Table 4.3 for CSE-CICIDS2018 dataset. From Table 4.3, it can be observed that XGB has the highest accuracy,

precision, recall, F-score and AUC, thus making it the best performing model on the dataset. This is expected as it has shown very high accuracy in previous works outperforming some ML models in same datasets (SHARAFALDIN; LASHKARI; GHORBANI, 2018). The LGBM model follows the XGB having obtained an accuracy, precision, recall, F-score and AUC of 98.8%, 98.83%, 98.83%, 98.83% and 99.96% respectively. A close look at the evaluation metrics shows proximate values for each of the models for each metric. For instance, DT reached almost 99% for all the metrics, RF having approximately 98% for all the metrics as well as ET classifier. Since the ML IDS models has relatively similar metrics performance, we measured the train and test time for each of the models which helps to understand more and better differentiate each model accurately with respect to their performance. As shown in Table 4.3, more test time of about 15.1 seconds is required for the ET classifier to predict all the classes present in the test set as a result of higher number of trees in its architecture. DT has the lowest detection time of 0.25sec while LGBM has a memory size of 2.4MB, hence, the lowest in file size. Therefore, based on the file size and detection time for LGBM, IDS model based on LGBM would be preferred for implementation in the domains where low memory and latency is highly needed while IDS model based on XGB would be most suitable for high larger memory device where accuracy of performance is paramount.

Table 4.3 – Performance Evaluation of the trained models on CSE-CICIDS2018 dataset showing the time for prediction and model size

Model / Metrics	Accuracy	Precision	Recall	F-Score	AUC	File Size	Test Time (s)
DT	98.7	98.67	98.67	98.67	99.25	10MB	0.25
RF	98.4	98.43	98.43	98.43	99.93	1200MB	9.98
ET	**98.3**	98.35	98.35	98.35	99.85	5500MB	15.1
AD	97.8	97.74	97.65	97.8	98.8	350MB	14.2
LGBM	98.8	98.83	98.83	98.83	99.96	**2.4MB**	3.4
XGB	**98.9**	**98.97**	**98.98**	**98.97**	99.9	1500MB	4.25

Source: Author (2022)

Similarly, performance results obtained for the CIC-IDS2017 dataset using the various metrics is shown in Table 4.4. All the classifiers used in this case achieved an overall performance of 99% in terms of all the evaluation metrics expect AD which only achieved an accuracy of 66%. ET in this case achieved the best performance in all metrics although having the highest detection or prediction time and memory requirement. On the other hand, DT has the prediction time of 0.18sec in comparison with other classifiers. LGBM

being a lightweight model, has the lowest memory requirement of about 3.1MB with an accuracy of 99.16%.

Table 4.4 – Performance Evaluation of the trained models on CIC-IDS2017 dataset showing the time for prediction and model size

Model / Metrics	Accuracy	Precision	Recall	F-Score	AUC	File Size	Test Time (s)
DT	99.59	99.59	99.59	99.59	99.76	5.7MB	**0.18**
RF	99.49	99.48	99.47	99.47	99.98	319MB	6.83
ET	**99.68**	99.68	99.67	99.67	99.97	1630MB	11.09
AD	69.67	66.79	66.78	66.68	67.9	400MB	12
LGBM	99.16	96.96	96.43	96.43	96.81	**3.1MB**	5.49
XGB	**99.51**	**99.52**	**99.51**	**99.51**	99.97	3.76MB	3.37

Source: Author (2022)

An IDS model can be described as reliable, efficient and accurate when the performance of such model in classifying each of the classes in the dataset is closely related to other metrics. From Table 4.5 which show the class report on each class, we can observe that not much differences exist between the precision, recall and F1-score demonstrated by the IDS model in identifying the class to which the detected network traffic belongs. For instance, in classifying benign class using DT, the model achieved a precision of 95.67%, recall 95.03 and F1-score 95.35%. This shows that when the IDS alerts that the traffic is benign, it does so with a confidence level of 95.67%. Similarly, with the XGB classifier, Brute Force attack is identified with 100% confidence level applying to precision, recall and F1-score. This shows how robust the models are in performance.

Table 4.5 – Report on the model Performances in classifying each label in the IDS2018 dataset

Decision Tree Classifier				**Random Forest Classifier**			**AdaBoost Classifier**		
Labels	**Performance Metrics**			**Performance Metrics**			**Performance Metrics**		
	Precision	**Recall**	**F-Score**	**Precision**	**Recall**	**F-Score**	**Precision**	**Recall**	**F-Score**
Benign	95.67	95.03	95.35	94.16	94.97	94.56	96.53	96.22	96.37
Bot	100.00	99.99	100.00	100.00	99.99	100.00	97.80	97.88	97.77
Brute Force	100.00	100.00	100.00	100.00	100.00	100.00	96.70	96.60	96.66
DDoS	100.00	100.00	100.00	100.00	100.00	100.00	97.23	97.60	98.00
DoS	100.00	100.00	100.00	100.00	100.00	100.00	98.60	97.50	98.80
Infiltration	95.08	95.70	95.39	95.37	94.09	94.72	98.80	98.60	99.00
Web Attacks	99.99	100.00	99.99	99.52	1.00	99.76	96.70	98.70	97.00
LightGBM Classifier				**Extra Tree Classifier**			**XGBoost Classifier**		
Benign	98.22	93.33	95.71	94.47	93.97	94.22	93.50	85.52	89.33
Bot	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00
BruteForce	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00
DDoS	100.00	100.00	100.00	99.99	99.99	99.99	99.97	99.99	99.98
DoS	100.00	100.00	100.00	100.00	100.00	100.00	99.98	100.00	99.99
Infiltration	93.67	98.31	95.94	94.02	94.51	94.26	86.94	93.98	90.32
Web Attacks	99.98	100.00	99.99	99.99	100.00	99.99	99.56	99.95	99.75

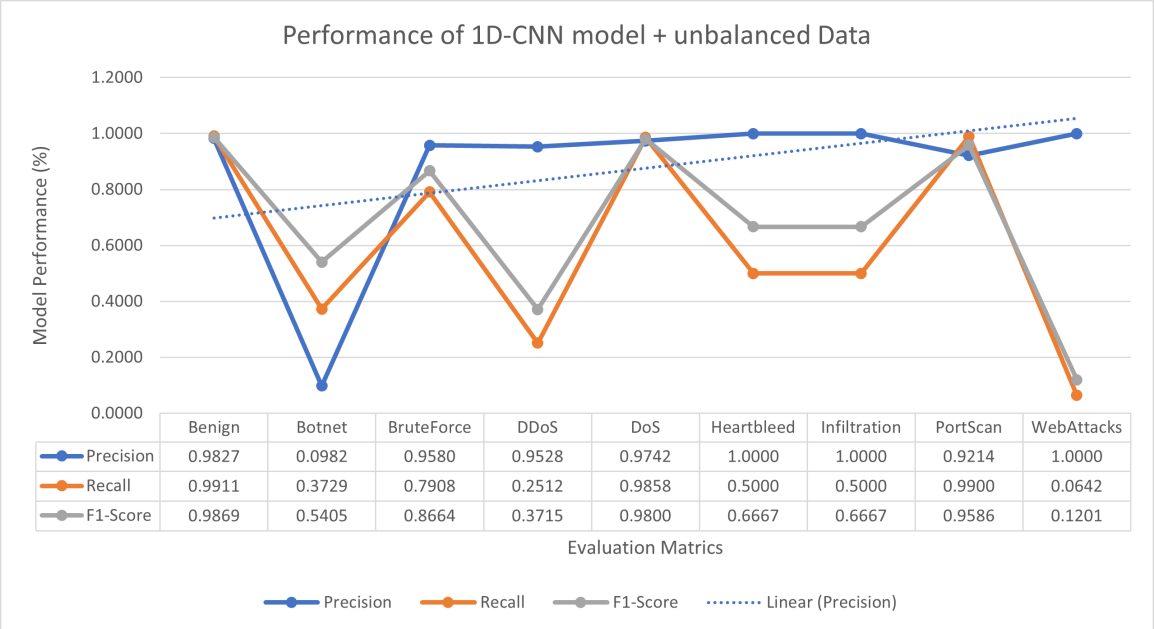
Source: Author (2022)

4.1.2 1D-CNN Model Evaluation

We used three different approaches in developing the IDS based

a) 1D-CNN based on imbalanced dataset. In this scenario, we evaluated the performance of the model on unbalanced dataset achieved an accuracy, precision, recall, f1-score of 0.9776, 0.9827, 0.9911, 0.9867, respectively and MCC of 0.9240. Figure4.1 demonstrates the performance of the model on the CSE-CICIDS2018 dataset in terms of the precision, recall and F-Score in detecting each of the network traffic classes. The classification report in Table 4.6, Evaluation metric plot in Figure 4.1 and Confusion matrix in Figure 4.2 (a) show that the model was able to detect with high accuracy the classes with higher instances but was unable to accurately classify those with less number of instances like Infiltration and Heartbleed having only 6 and 2 instances in the test set respectively.

Figure 4.1 – Precision, Recall and F1-Score showing the model performance on balanced (SMOTE) data



	Benign	Botnet	BruteForce	DDoS	DoS	Heartbleed	Infiltration	PortScan	WebAttacks
Precision	0.9827	0.0982	0.9580	0.9528	0.9742	1.0000	1.0000	0.9214	1.0000
Recall	0.9911	0.3729	0.7908	0.2512	0.9858	0.5000	0.5000	0.9900	0.0642
F1-Score	0.9869	0.5405	0.8664	0.3715	0.9800	0.6667	0.6667	0.9586	0.1201

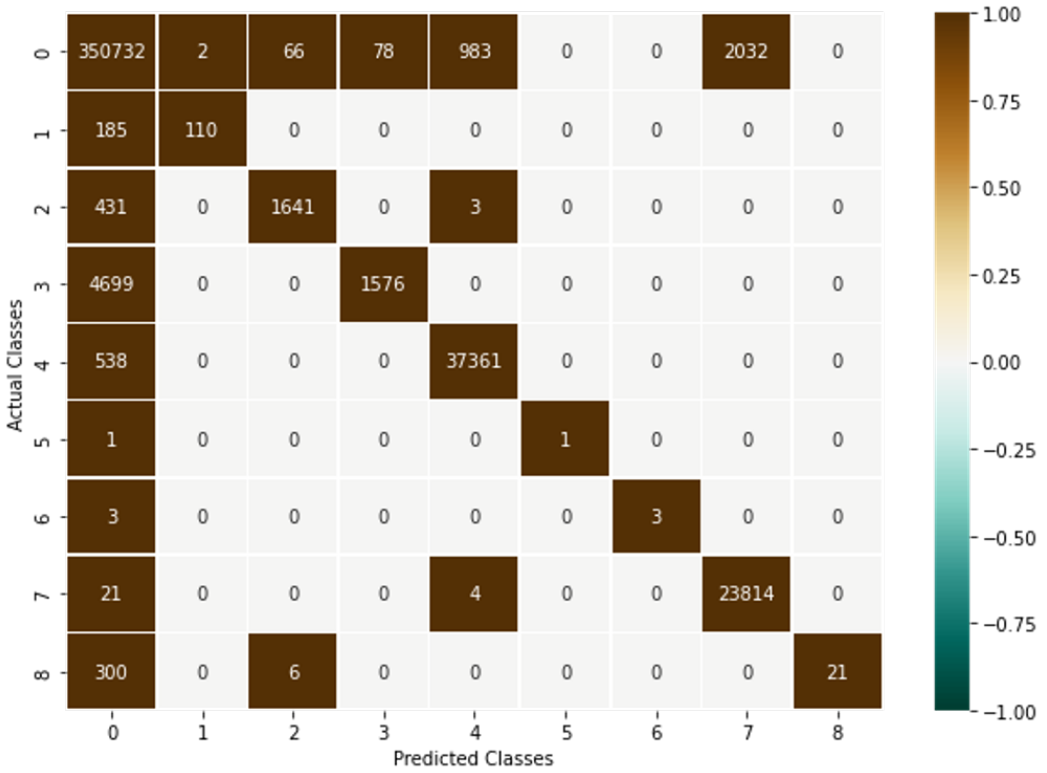
Source: Author (2022)

Table 4.6 – Comparing the effect of imbalance on the detection rate of 1D-CNN in classifying each class in the IDS2017 dataset

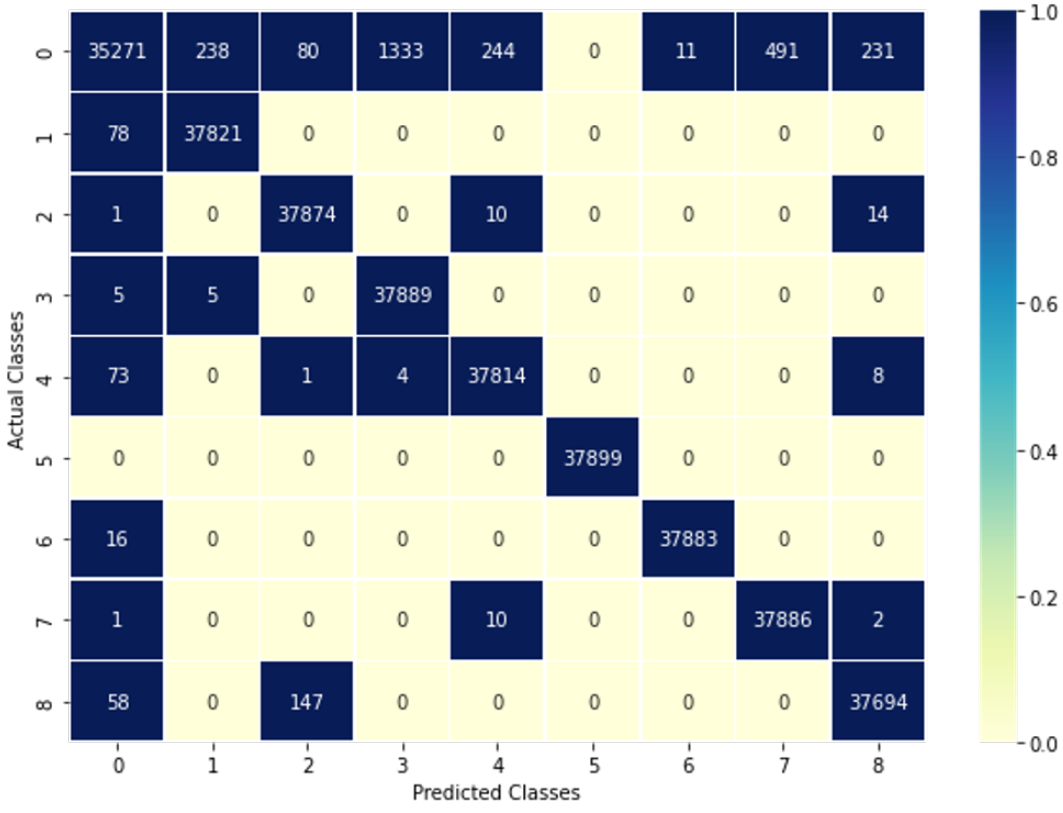
1D CNN Imbalance					1D CNN + SMOTE				
Attack / Metrics	Precision	Recall	F1-Score	Support	Attack / Metrics	Precision	Recall	F1-Score	Support
Benign	**0.9827**	**0.9911**	**0.9869**	**353893**	Benign	0.9935	0.9307	0.9610	37899
Botnet	0.0982	0.3729	0.5405	293	Botnet	0.9936	0.9970	0.9959	37899
BruteForce	0.9580	0.7908	0.8664	2075	BruteForce	0.9940	0.9993	0.9967	37899
DDoS	0.9528	0.2512	0.3715	6275	DDoS	0.9659	0.9997	0.9825	37899
DoS	0.9742	0.9858	0.9800	37899	DoS	0.9931	0.9977	0.9954	37899
Heartbleed	1.0000	0.5000	0.6667	2	Heartbleed	1.0000	1.0000	1.0000	37899
Infiltration	1.0000	0.5000	0.6667	6	Infiltration	0.9997	0.9996	0.9996	37899
PortScan	0.9214	0.9900	0.9586	23839	PortScan	0.9872	0.9997	0.9934	37899
WebAttacks	1.0000	0.0642	0.1201	327	WebAttacks	0.9933	0.9946	0.9939	37899

Source: Author (2022)

Figure 4.2 – Confusion Matrix showing the three implementation approaches used for the 1D-CNN model (a) 1D-CNN based on unbalanced data. (b) 1D-CNN based on balanced (SMOTE) data



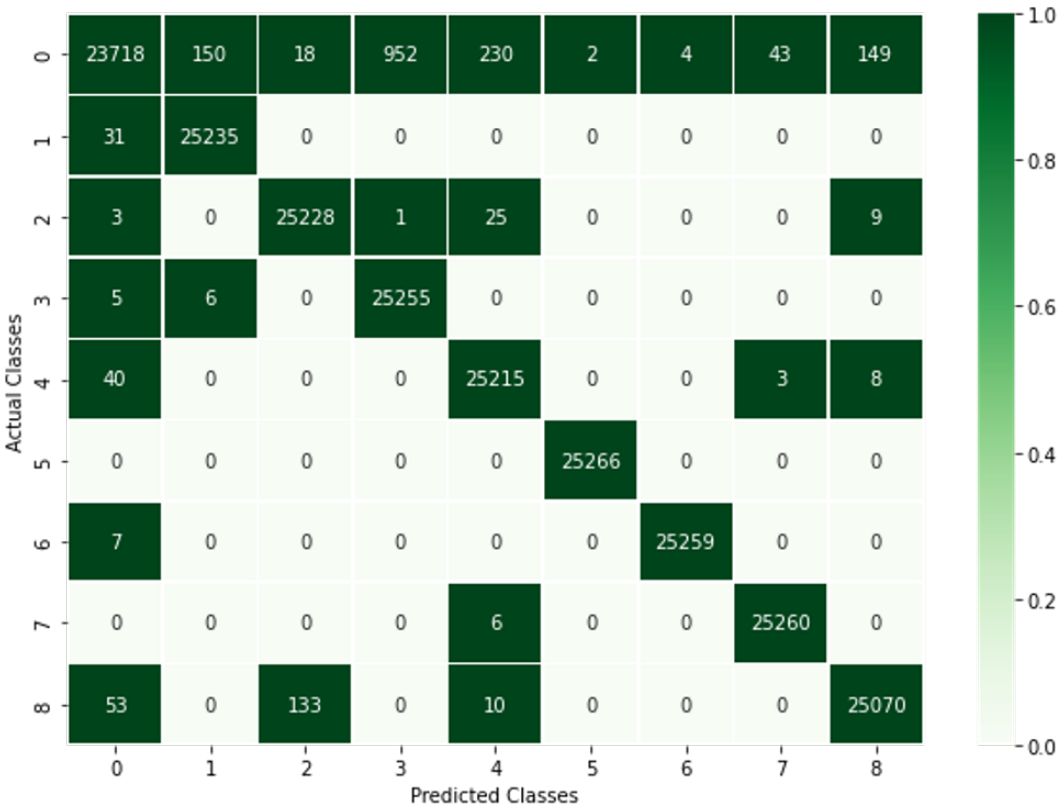
a). 1D-CNN Confusion Matrix on unbalanced Dataset with train_test_splt



b). 1D-CNN Confusion Matrix on balanced Dataset with train_test_splt

Source: Author (2022)

Figure 4.3 – Confusion Matrix showing the three implementation approaches used for the 1D-CNN model (c) 1D-CNN based on SMOTE + StratifiedKFold

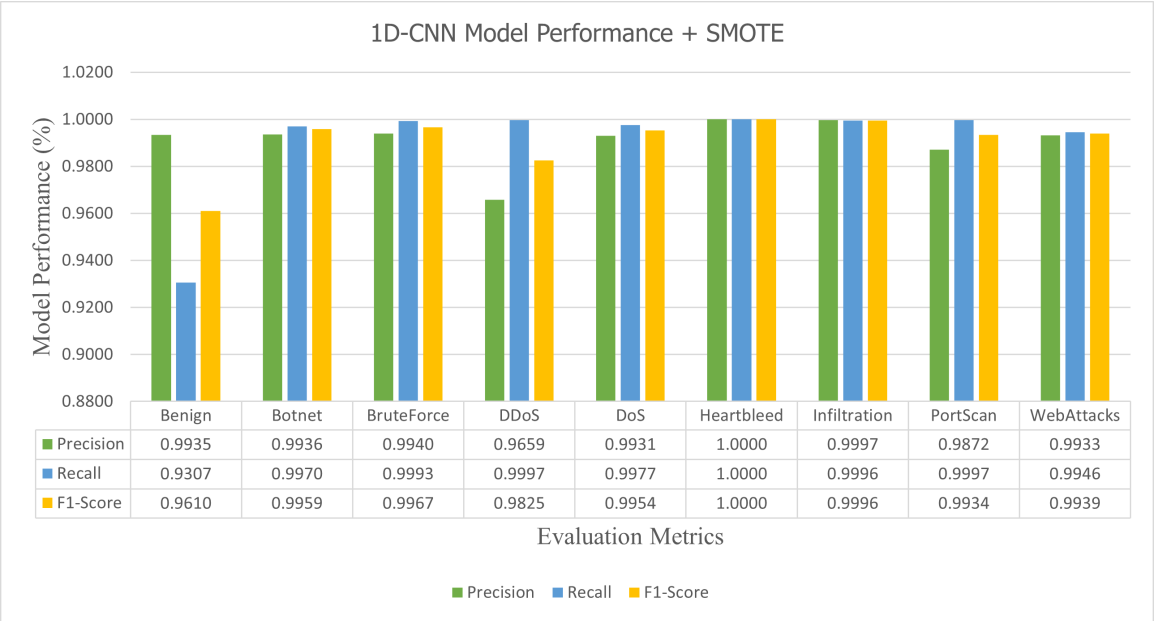


c). 1D-CNN Confusion Matrix on balanced Dataset with Stratified Kfold Split

Source: Author (2022)

b) 1D-CNN based on Balanced Dataset (SMOTE) On the balanced data, the model performed better than the case of unbalanced data as it was able to classify the samples which were mis-classified previously. In contrast to the 1D-CNN unbalanced that classified the Heartbleed and Infiltration with 0.5000 precision, this approach identified both attack classes with precision of 1.0000 and 0.9997 respectively showing that it is a more effective way to model IDS systems. Figure 4.4 shows the performance of the model with respect to the precision, recall and f1-score while Figure4.2 (b) show the confusion matrix.

Figure 4.4 – Classification Report showing the model performance on balanced (SMOTE) data

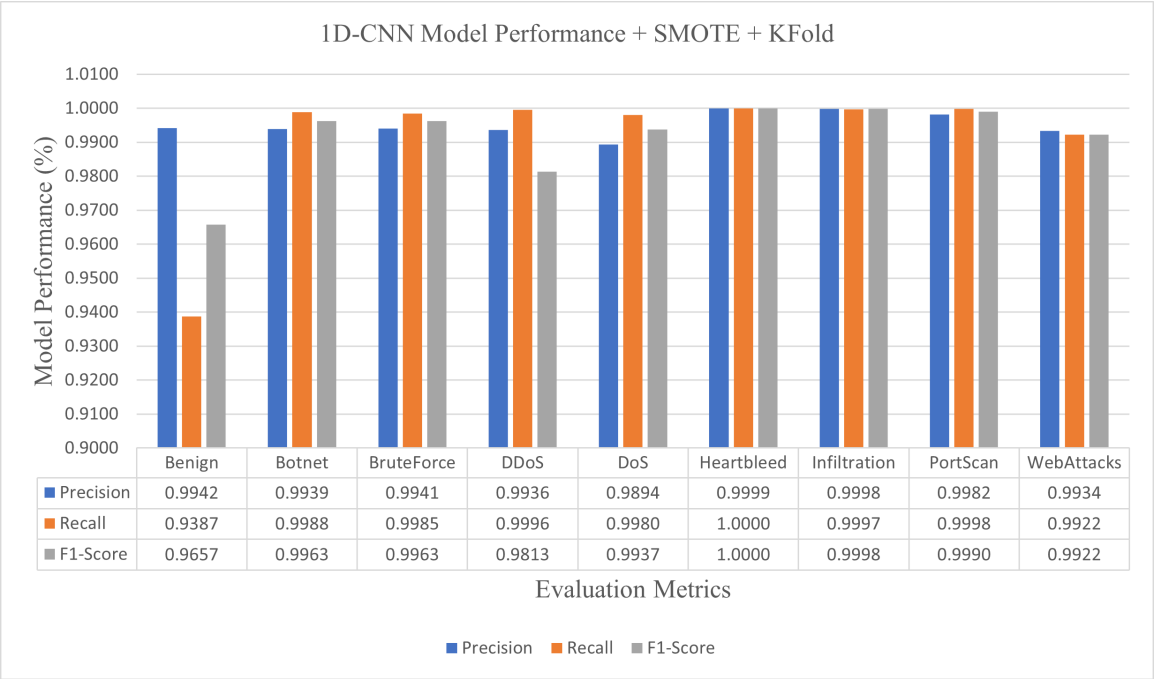


	Benign	Botnet	BruteForce	DDoS	DoS	Heartbleed	Infiltration	PortScan	WebAttacks
Precision	0.9935	0.9936	0.9940	0.9659	0.9931	1.0000	0.9997	0.9872	0.9933
Recall	0.9307	0.9970	0.9993	0.9997	0.9977	1.0000	0.9996	0.9997	0.9946
F1-Score	0.9610	0.9959	0.9967	0.9825	0.9954	1.0000	0.9996	0.9934	0.9939

Source: Author (2022)

c) 1D-CNN based on SMOTE and Stratified Split In this scenario, we use 10-fold cross validation split on the balanced dataset during the training process to ensure that the same amount of each class is present in the splits, thereby reducing bias and data leakage. This approach showed the best performance as can be seen in Figures 4.5 and 4.3 (c). This therefore is the best approach to develop an IDS using 1D-CNN architecture as it gives the best result considering all the evaluation metrics reaching an accuracy of 99.17%, 99.98% precision, 99.97% recall, 99.97% f-score and 99.06 MCC on the test set as shown in Figure 4.6.

Figure 4.5 – Classification Report showing the model performance SMOTE + StratifiedKFold

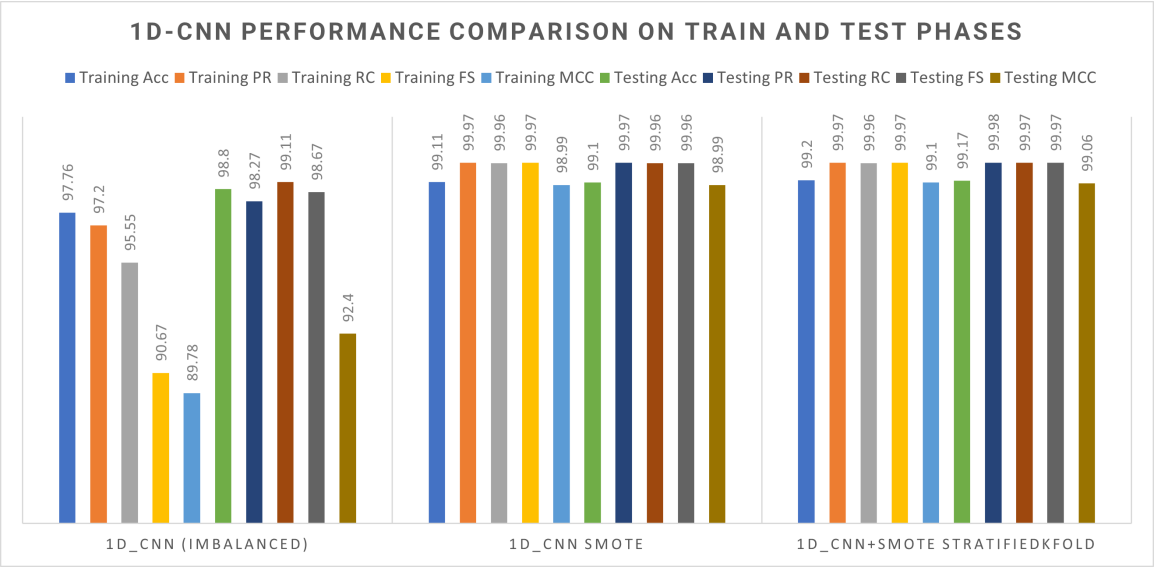


	Benign	Botnet	BruteForce	DDoS	DoS	Heartbleed	Infiltration	PortScan	WebAttacks
Precision	0.9942	0.9939	0.9941	0.9936	0.9894	0.9999	0.9998	0.9982	0.9934
Recall	0.9387	0.9988	0.9985	0.9996	0.9980	1.0000	0.9997	0.9998	0.9922
F1-Score	0.9657	0.9963	0.9963	0.9813	0.9937	1.0000	0.9998	0.9990	0.9922

Source: Author (2022)

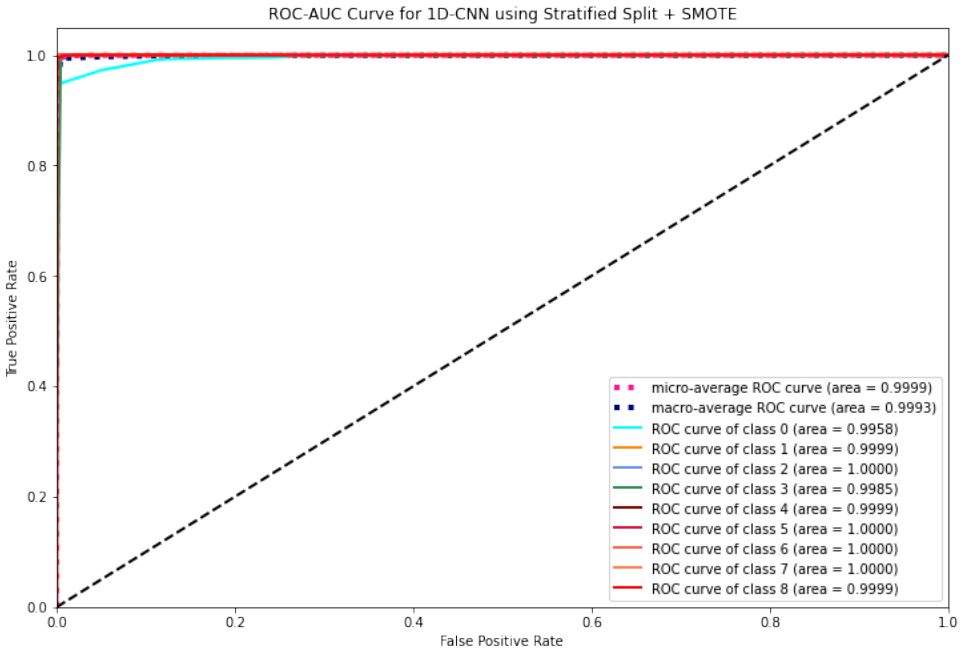
In conclusion, as can be seen in Figure 4.6, with StratifiedKFold (10 splits), a more robust model is achieved. This is because the model trains and learns better on the dataset so that it can easily adapt to unseen data. In all the cases, the performance of the model is within same range in all the metrics achieving almost 100%. For instance, the MCC improved from 89.78% to 99.10% on training **(+9.32%)** and FS improved from 98.67% to 99.97% on testing **(+1.3%)**. In each case, a batch size of 128, filter size of 3, softmax function and categorical cross-entropy were used. Each model was trained for 20 epochs with a learning rate of 0.01 and decay momentum of 0.9. The ROC curve for the 1D-CNN is shown in Figure 4.7, indicating that the 1D-CNN achieves a micro-average ROC curve of 99.99% and macro-average ROC curve of 99.99%. implying that the FPR is almost zero (0) while the TPR is approximately 1% for all the classes in the dataset.

Figure 4.6 – Training and Testing performance comparison of the three scenarios implemented with the 1D-CNN method. Where Acc = Accuracy, PR = Precision, RC = Recall and FS = F1-Score.



Source: Author (2022)

Figure 4.7 – Receiver Operating curve for the `1D-CNN` for the IDS2017



Source: Author (2022)

4.2 Models Based on Image Data

The proposed architecture is evaluated on two benchmark datasets CIC-IDS2017 and CSE-CIC-IDS2018 as discussed in Subsection 2.4.9 and 2.4.10 respectively. $StratifiedKFold$

split with 10-folds was used to split the dataset during training which ensures that there is even distribution of each class in the folds, hence overcoming model overfit. Usually, network traffic dataset are composed of high volume of normal traffic and less number of attacks as in the case of our datasets, so we handled data imbalance before converting the samples to image. Table 4.7 shows the total number of images generated for each dataset. A total of 19,055 images was generated for the CIC-IDS2017 dataset consisting of 9 classes and 21, 230 images comprising of 7 classes for the CSE-CIC-IDS2018 dataset.

Table 4.7 – Distribution of generating images of the datasets used in the model training and evaluation.

Classes	**CIC-IDS2017**	**CSE-CICIDS2018**
Benign	2575	4470
Bot	2062	2241
Brute Force	2057	3004
DDoS	2062	4189
DoS	2060	3614
Heartbleed	2057	-
Infiltration	2060	2036
Port Scan	2059	-
Web Attack	2063	1676
Total	**19055**	**21230**

Source: Author (2022)

The transformed images are used in the model training. We first trained the selected six models with the default parameters to evaluate the behaviour of the IDS models in classifying the threats in a computer network. Then, we perform an algorithmic search to determine the most suitable parameters to be used during the training process in order to reduce the training time while achieving similar or improved results. After performing the HPO with the BS-TPE algorithm, the best hyper-Parameter were used to train the model. During the search, the parameters used are shown in Table 4.8. Number of epochs, batch size, learning rate, dropout rate, early stopping patience and number of frozen layers were selected as the search space to be optimized as shown in the Table 4.8. The Early stopping patience is used to save training time as the algorithm saves the best model performances during the training and stops training when the validation accuracy does not increase between two consecutive epochs. This is because, the validation accuracy is monitored during the training as it helps to determine if the model overfits or not. Other metrics such

as the accuracy, precision, loss or recall can be monitored but we chose to monitor the validation accuracy.

Table 4.8 – Hyper-Parameters obtained after BS-TPE optimization for the Model Configuration

Hyper-Parameter	Model	Search Range	Optimal Value
Number of Epochs	All CNN pre-trained Models	[5,30]	20
Batch size		[32,256]	128
Early stop patience		[1,4]	2
Learning rate		[0.001,01]	0.003
Dropout rate		[0.2, 0.8]	0.5
Number of layers frozen from the pre-trained layers	MobileNetV3Small	[100,232]	232
	EfficientNetV2B0	[200,323]	236
	InceptionV3	[80,159]	148
	VGG16	[8,16]	15
	VGG19	[10,19]	19

Source: Author (2022)

4.2.1 IDS Evaluation Using Transfer Learning

Each of the selected CNN based network was trained on both datasets and the results obtained in each case is shown in Table 4.9 and 4.10 for the CIC-IDS2017 and CSE-CIC-IDS2018 respectively. With the search parameters defined in Table 4.8, we performed both the IDS modelling for optimized and non-optimized training. The best parameters obtained with the BS-TPE were used in the HPO models while for the non-optimized case, we randomly selected some values for the learning-rate=0.01, batch-size=256, number of frozen layers in VGG16=14. On the CIC-IDS2017 dataset shown in Table 4.9, the base CNN architecture with 14 layers and 184,007 trainable parameters achieved an accuracy, recall, precision, F-score of 0.9976. AUC and MCC were 0.9980 and 0.9984 respectively. AUC and MCC are within the same range which show that the performance of the model is consistent. Notably, this was the lowest performance obtained on the dataset as compared to the pre-trained architectures. IV3 outperformed all other models achieving an overall performance in all metrics of 100% in both optimized and non-optimized scenarios. Considering the AUC and MCC, the IV3-HPO also performed better than all other models. Again, MobileNetV3Small ranks second in both scenarios of optimization and non-optimization with AUC and MCC of almost 100% *(0.9999)*. As defined in Equation3.8, the MCC value is used to validate the AUC value for the models which best shows the performance of each model. From the forgoing,

each of the models has a very high DR and low FAR showing that they can detect each attack at a fast rate.

Table 4.9 – Performance Evaluation of Optimized and non-Optimized trained Models on CIC-IDS2017

Models	**Accuracy**	**Precision**	**Recall**	**F-Score**	AUC	**MCC**
Base CNN	0.9976	0.9976	0.9976	0.9976	0.9980	0.9984
VGG16	0.9993	0.9993	0.9993	0.9993	0.9995	0.9998
VGG19	0.9981	0.9981	0.9981	0.9981	0.9982	0.9988
IV3	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**0.9993**	**0.9992**
MNV3S	**0.9999**	**0.9999**	**0.9999**	**0.9999**	**0.9998**	**0.9995**
ENV2B0	**0.9982**	**0.9982**	**0.9982**	**0.9982**	**0.9986**	**0.9990**
Base CNN - HPO	0.9978	0.9978	0.9978	0.9978	0.9979	0.998
VGG16 - HPO	0.9996	0.9996	0.9996	0.9996	0.9995	0.9997
VGG19 - HPO	0.9994	0.9994	0.9994	0.9994	0.9996	0.9992
IV3 - HPO	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**0.9999**	**0.9999**
MNV3S - HPO	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**0.9999**	**0.9998**
ENV2B0 - HPO	**0.9999**	**0.9999**	**0.9999**	**0.9999**	**0.9989**	**0.9988**

Source: Author (2022)

Table 4.10 shows the IDS model performance on the CSE-CICIDS2018 dataset. In this case, the base CNN model also achieved the least performance showing that pre-trained models tends to have the possibility of performing better than CNN architecture with respect to time and accuracy. This is due to the fact that pre-trained models already learned from a large volume of dataset and can easily learn new patterns from new input data. Notwithstanding, the architectural design of CNN networks developed from scratch may affect its performance compared to the pre-trained networks. VGG19-HPO and ENV2B0-HPO are the best performing model on the dataset reaching the accuracy of 0.9900 and 0.9910 respectively against the base CNN with 0.9797 accuracy.

With the BS-TPE optimization algorithm used for the Hyper-Parameter search in the search space, we obtain models with better performances. In Tables 4.9 and 4.10, the models trained with selected Hyper-Parameters tend to have better performances in terms of evaluation metrics and the time cost function. The learning curves for the models on the datasets is shown in Figures 4.8 and 4.9 for the IDS2017 dataset and 4.10 and 4.11 for the CSE-CICIDS2018 dataset. The learning curves show the performance rate of learning and loss during the model training phase. Cross-entropy loss function was used to control the loss during the training. This helps to determine if a model overfits or under-fits. A good indicator of this is the validation loss and validation accuracy. Where the validation

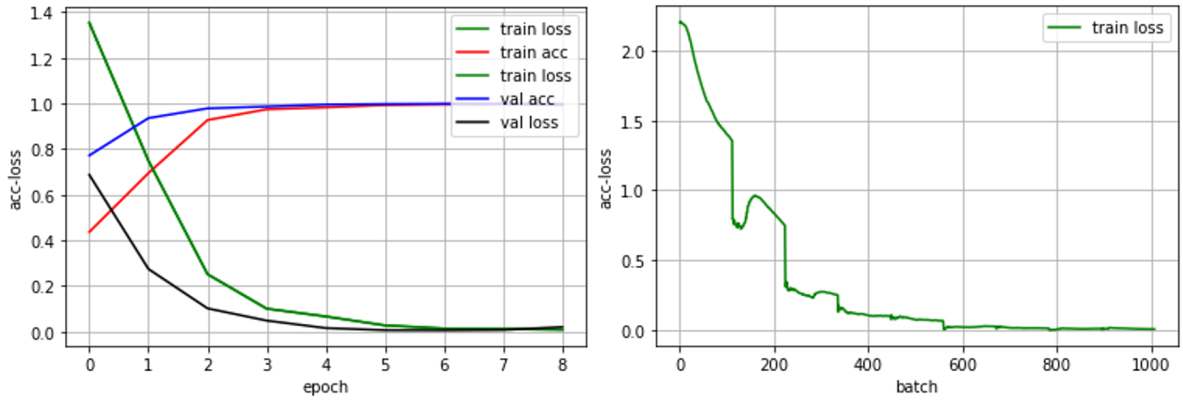
accuracy is above or these same with the training accuracy, we obtain a normal IDS model. In Figures 4.8 and 4.10, we can observe that the validation accuracy in each of the IDS models are in the same range with the training accuracy which validates that the model did not overfit during the training; hence confirms the specificity and reliability of the results.

Table 4.10 – Performance Evaluation of Optimized and non-Optimized trained Models on CSE-CIC-IDS2018

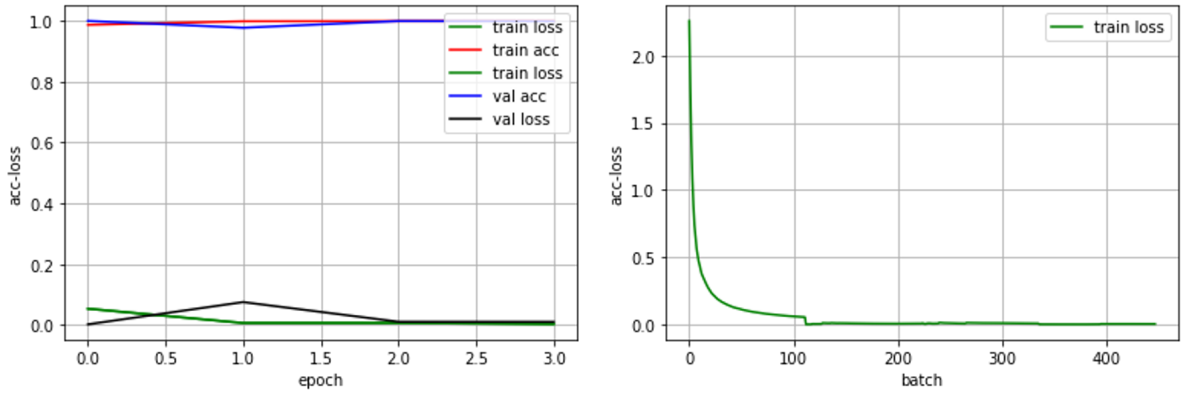
Models	**Accuracy**	**Precision**	**Recall**	**F-Score**	**AUC**	**MCC**
Base CNN	97.97	0.9806	0.9797	0.9799	0.9798	0.9800
VGG16	0.9880	0.9897	0.9897	0.9897	0.988	0.9886
VGG19	0.9887	0.9887	0.9887	0.9887	0.9877	0.9890
IV3	**0.9769**	**0.9777**	**0.9769**	**0.9770**	**0.9796**	**0.9775**
MNV3S	**0.9832**	**0.9833**	**0.9832**	**0.9831**	**0.9852**	**0.9854**
ENV2B0	**0.9890**	**0.9889**	**0.9890**	**0.9890**	**0.9870**	**0.9895**
Base CNN - HPO	0.9890	0.9890	0.9890	0.9890	0.9880	0.9892
VGG16 - HPO	0.9899	0.9899	0.9899	0.9899	0.99	0.9912
VGG19 - HPO	0.9900	0.9900	0.9900	0.9900	0.9890	0.9932
IV3 - HPO	**0.9870**	**0.9870**	**0.9870**	**0.9870**	**0.9880**	**0.9897**
MNV3S - HPO	**0.9888**	**0.9888**	**0.9888**	**0.9888**	**0.9890**	**0.9899**
ENV2B0 - HPO	**0.9910**	**0.9910**	**0.9910**	**0.9910**	**0.9916**	**0.9920**

Source: Author (2022)

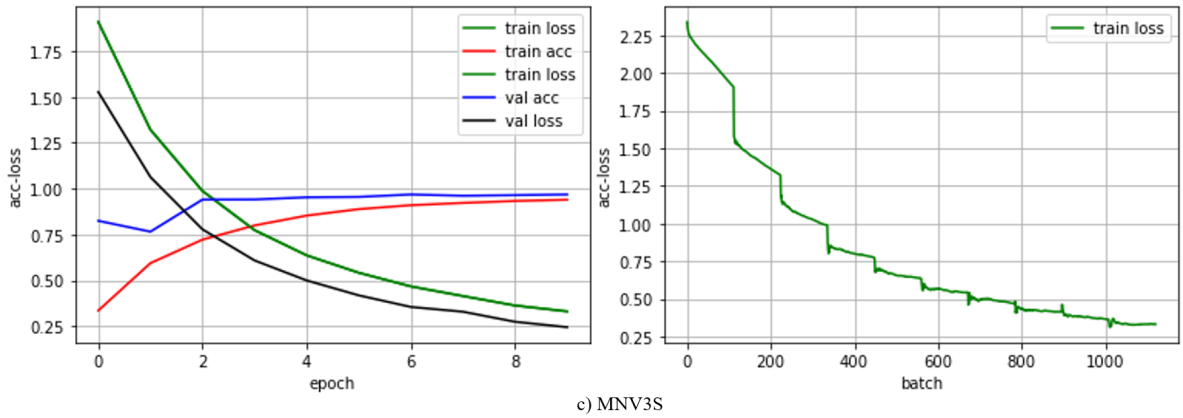
Figure 4.8 – Learning Curves for each of the models (a) Base CNN, (B)ENV2B0, (C)MNV3S considering the number of epochs and batch size in each case on IDS2017 dataset



a) Base CNN



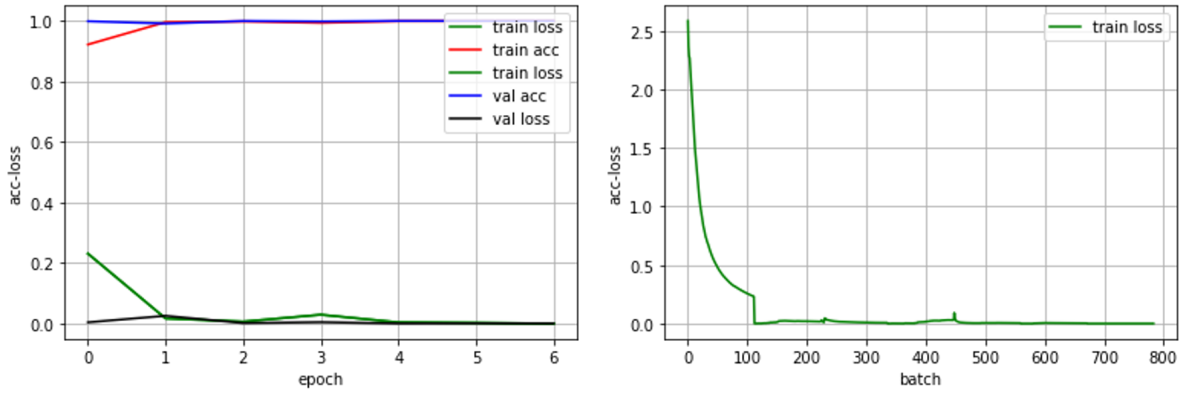
b) ENV2B0



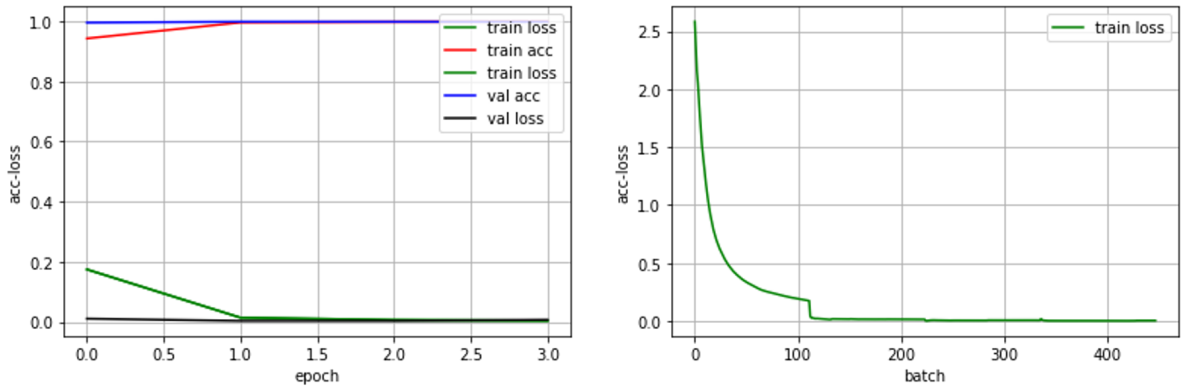
c) MNV3S

Source: Author (2022)

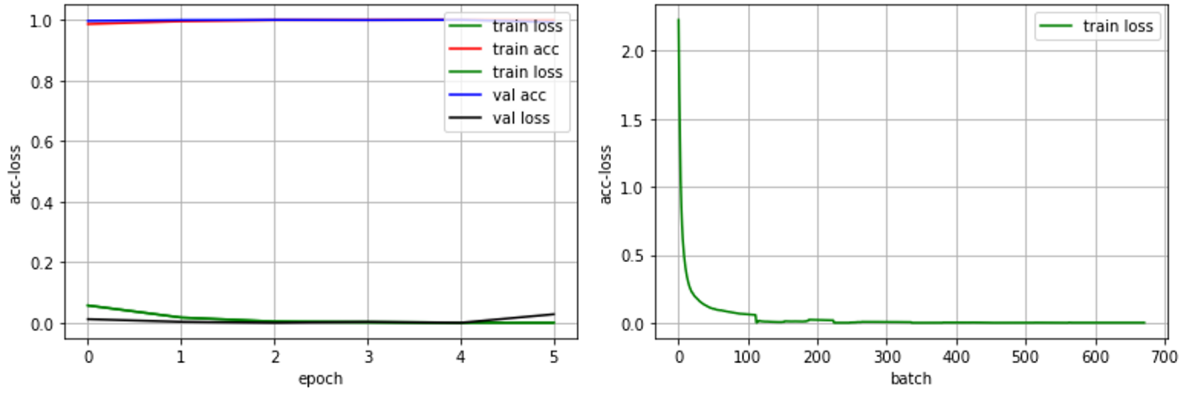
Figure 4.9 – Learning Curves for each of the models (D) VGG16, (E) VGG19 and (F) IV3 considering the number of epochs and batch size in each case on IDS2017 dataset



d) VGG16



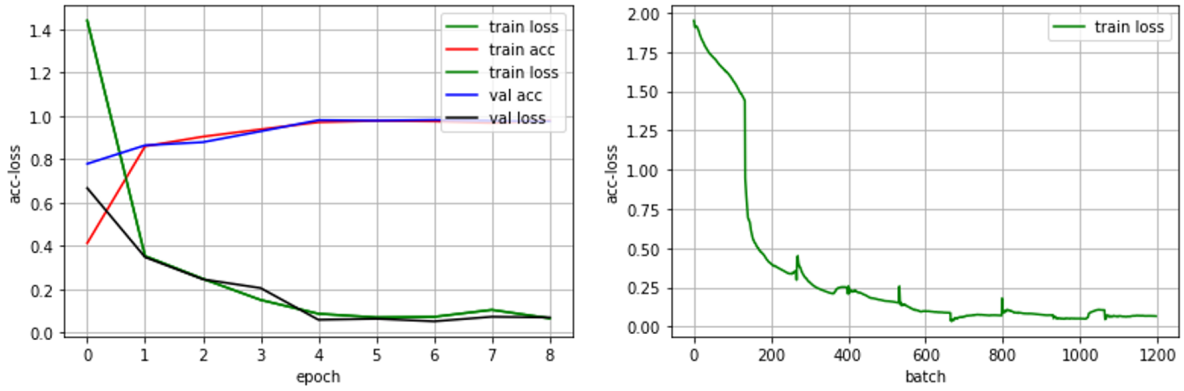
e) VGG19



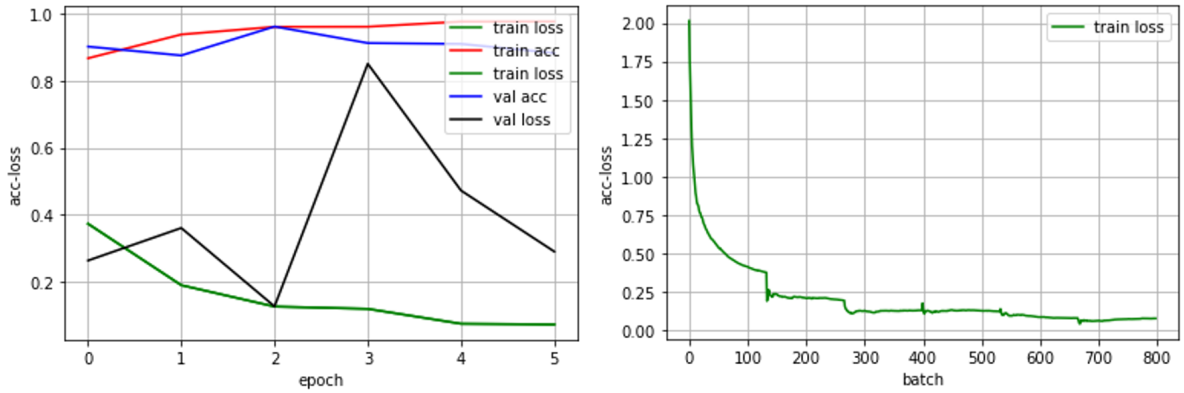
f) IV3

Source: Author (2022)

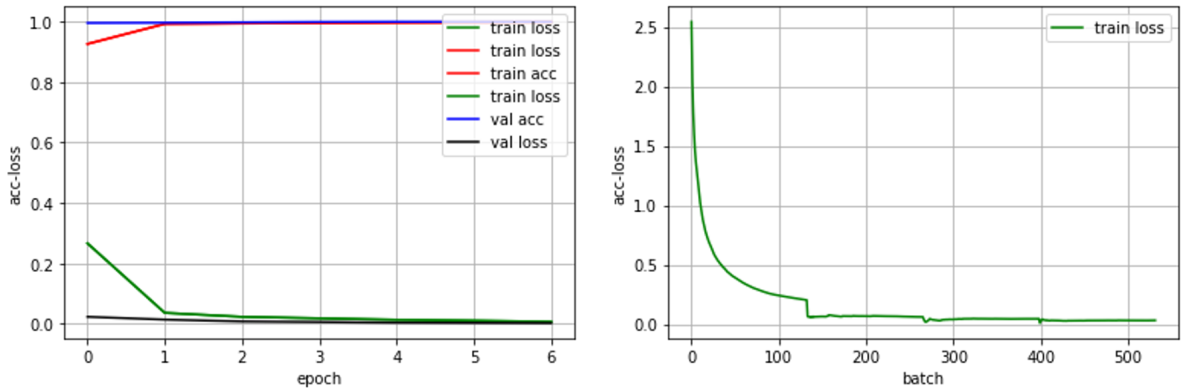
Figure 4.10 – Learning Curves for each of the models (a) Base CNN, (B)ENV2B0, (C)MNV3S considering the number of epochs and batch size in each case on IDS2018 dataset



a) Base CNN



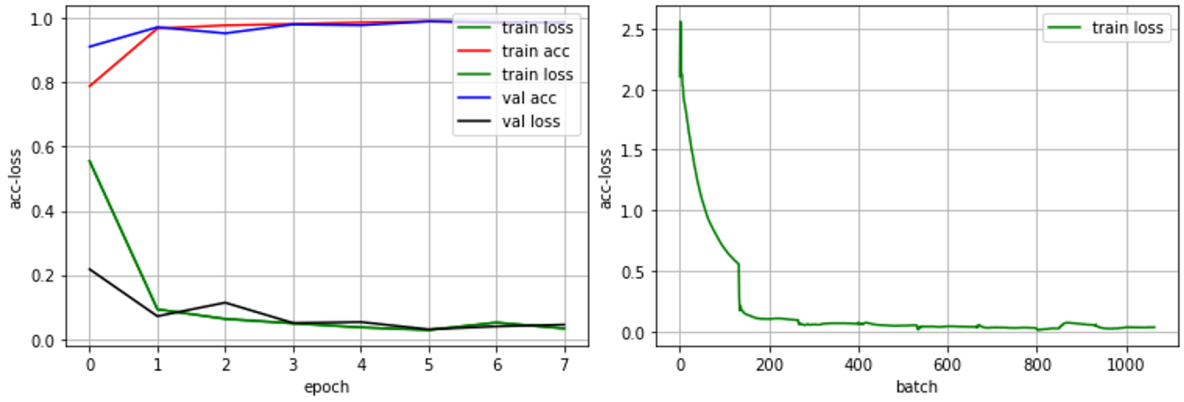
b) ENV2B0



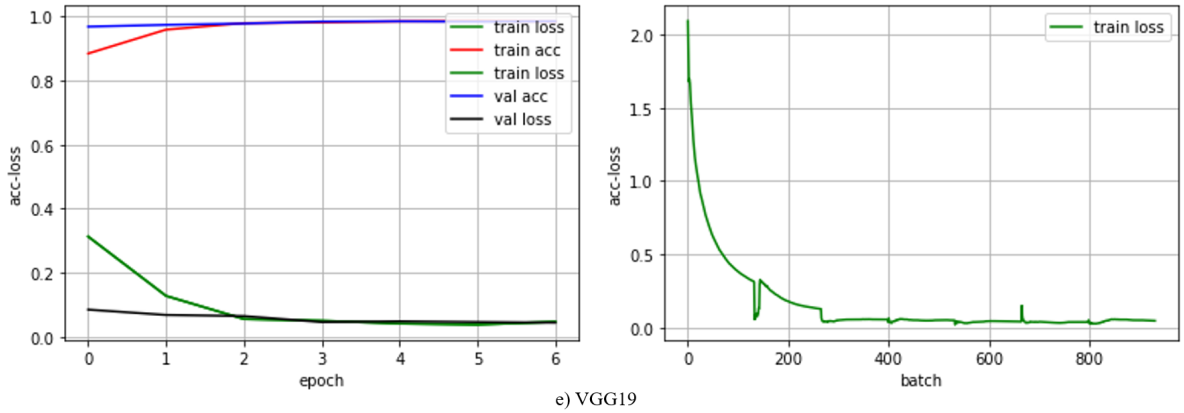
c) MNV3S

Source: Author (2022)

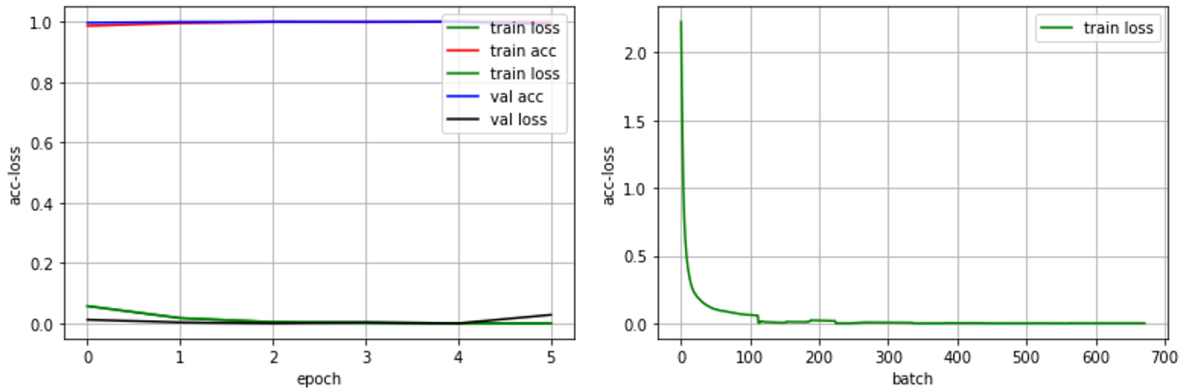
Figure 4.11 – Learning Curves for each of the models (D) VGG16, (E) VGG19 and (F) IV3 considering the number of epochs and batch size in each case on IDS2018 dataset



d) VGG16



e) VGG19



f) IV3

Source: Author (2022)

4.2.2 Ensemble Model based on Transfer Learning

In constructing the ensemble model, some key parameters used in selecting the best three models include: the training time Tt(s) (total time taken for the model to be trained on the dataset), test time tt(s) (i.e the total time taken to predict all the images in the test set), the overall MCC and AUC values, and the F-Score in additions to all the regular evaluation metrics. Therefore, the Tt(s) and tt(s) including test time per packet tt/p(s) were measured

during the training and evaluation phases. Test time per packet is the average time taken to detect a packet of data. This is equivalent to the total test time divided by the total number of images in the sample. The results obtained for is shown in Tables 4.13.

From Table 4.13, we observe that the training and testing time for the IV3, MNV3S and ENV2B0 are the lowest. Since the accuracy and other metrics are good, compared to other IDS models that has higher parameters and more training time, we considered them to be best fit for the construction of the ELETL-IDS model. This is aimed at reducing model complexity, memory consumption, latency while maintaining good accuracy. On CICIDS2017, IV3-HPO was trained for about 5272 seconds and it took 859 second to complete test on all the data in the test set. The same IDS model was trained in 6793.55 seconds on IDS2018 and evaluation time took 793 seconds. Similarly, for MNV3S-HPO, 1025 seconds was the training time on CIC-IDS2017 and 831 seconds to train on the CSE-CIC-IDS2018. In both scenarios of optimization and non-optimization, the IV3, MNV3S and ENV2B0 had the least time and resource requirement compared to other pre-trained models.

The performance evaluation of the model in terms of the metrics is shown in Tables 4.11 and 4.12 for the IDS2017 and IDS2018 respectively. The reason for the lower training and testing times as seen is due to some of the neurons in the network were frozen and only selected layers were trained.

Table 4.11 – Performance Evaluation ELETL-IDS Model on CIC-IDS2017

Models	**Accuracy**	**Precision**	**Recall**	**F-Score**	AUC	**MCC**
ELETL-IDS (Train)	1.00	1.00	1.00	1.00	0.9995	0.9994
ELETL-IDS (Valid)	1.00	0.9998	0.9986	1.00	-	-
ELETL-IDS (Test)	1.00	1.00	1.00	1.00	0.9993	0.9992

Source: Author (2022)

Table 4.12 – Performance Evaluation of ELETL-IDS Model on CSE-CIC-IDS2018

Models	**Accuracy**	**Precision**	**Recall**	**F-Score**	**AUC**	**MCC**
ELETL-IDS (Train)	0.9999	0.9999	0.9999	0.9999	0.9986	0.9985
ELETL-IDS (Valid)	0.9997	0.9998	0.9996	0.9995	-	-
ELETL-IDS (Test)	0.9999	0.9999	0.9999	0.9999	0.9993	0.9990

Source: Author (2022)

Comparing the base CNN trained from scratch with the pre-trained, we can observe that more training and testing time is required to get optimum performance. This means high computational power is also required and thus; transfer learning presents a more

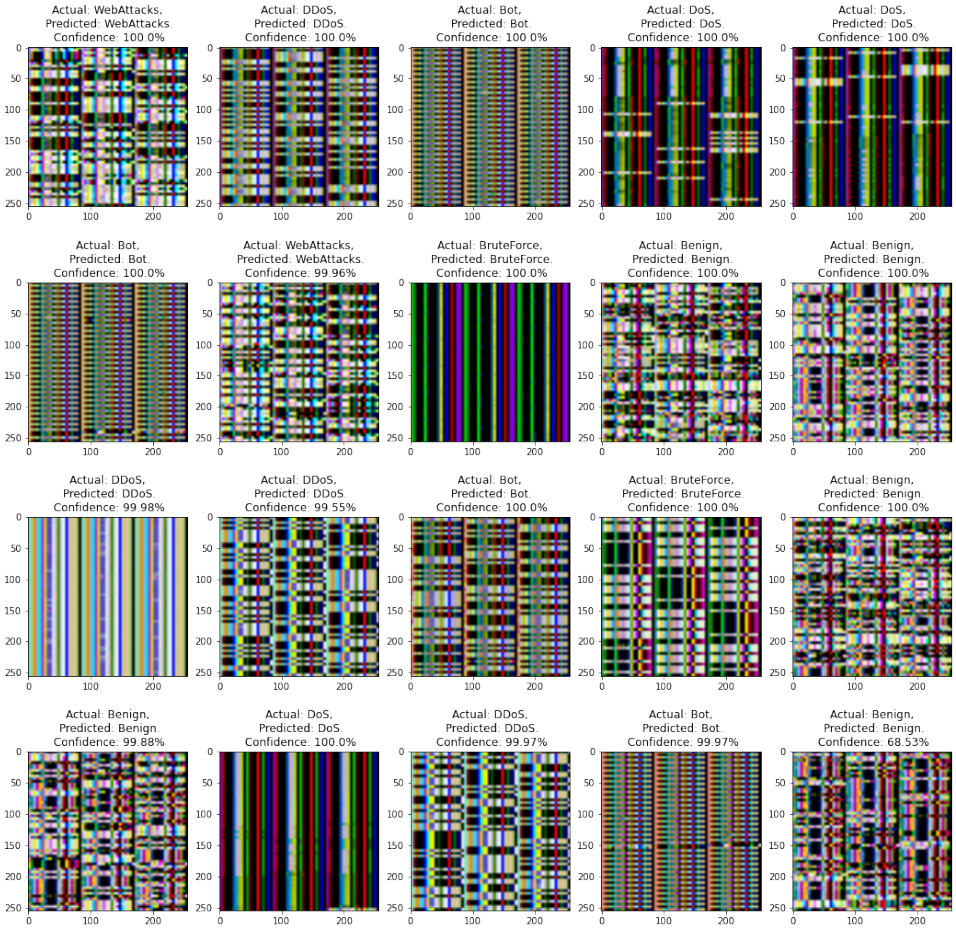
convenient approach to training CNN based IDS models. While 43649.54 seconds was needed to train base CNN on CSE-CICIDS2018 dataset, only 934.22 seconds was required to train a better performing model on the same dataset using ENV2B0-HPO approach showing a time variation of over 191.62 seconds. In this case, models trained using TL show better lower latency in the detection of network traffics and taking a decision on it as appropriate.

Table 4.13 – Time-base model Evaluation

CIC-IDS2017			
Models	**Train Time (Tt(s))**	**Test Time (tt(s))**	**Test time per packet (tt/p(s))**
Base CNN	39772	1341	0.2837
VGG16	16669	1638	0.3466
VGG19	10990	1759	0.3722
IV3	**6578**	**1057**	**0.2237**
MNV3S	**1170**	**857**	**0.1813**
ENV2B0	**4172**	**930**	**0.1967**
Base CNN - HPO	30932	1120	0.237
VGG16 - HPO	13679	1268	0.2683
VGG19 - HPO	8980	1420	0.3005
IV3 - HPO	**5272**	**859**	**0.1817**
MNV3S - HPO	**1025**	**748**	**0.1582**
ENV2B0 - HPO	**3282**	**856**	**0.1811**
CSE-CICIDS2018			
Models	**Train Time (Tt(s))**	**Test Time (tt(s))**	**Test time per packet (tt/p(s))**
Base CNN	43649.54	1048	0.2468
VGG16	21229.61	1499	0.3530
VGG19	21316.19	1792	0.4220
IV3	**7848.12**	**890**	**0.2096**
MNV3S	**975.67**	**838**	**0.1974**
ENV2B0	**10032.56**	**875**	**0.2483**
Base CNN - HPO	36987.56	912	0.2588
VGG16 - HPO	18964.59	988	0.2804
VGG19 - HPO	19121.76	1021	0.2897
IV3 - HPO	**6793.55**	**793**	**0.225**
MNV3S - HPO	**831.23**	**739**	**0.2097**
ENV2B0 - HPO	**934.22**	**781**	**0.2216**

Source: Author (2022)

Figure 4.12 – Prediction Result of the proposed ELETL-IDS model on IDS2017

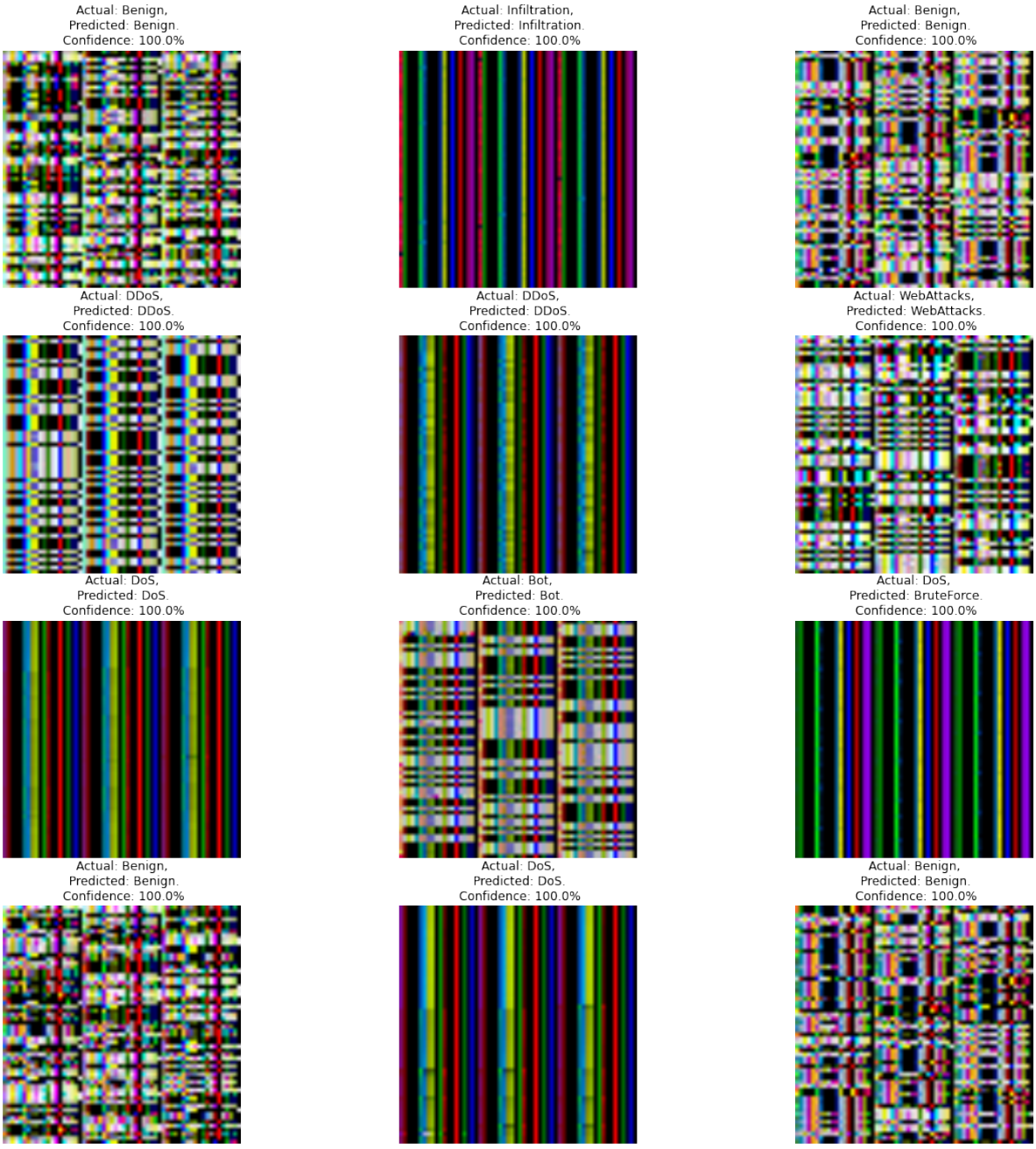


Source: Author (2022)

Confusion matrix helps us to understand the performance of the models with respect to the various tasks being classified by the model. In Figure 4.14, we present the confusion metrics for ELETL-IDS obtained on the CIC-IDS2017 and CSE-CIC-IDS2018 datasets respectively. We can observe that ELETL-IDS correctly classified all the attacks and benign features in each case with an accuracy of 100% since there is no mis-classification of the attack signatures. Our model only failed to classify all the classes in IDS2018 dataset, hence, it mis-classified 18 benign classes as infiltration and 27 infiltration attacks as benign, with an accuracy of 99.76% and an error rate of 0.24%. This shows that even though there were mis-classification of events, the FAR and FPR are very small, therefore the IDS model predic-

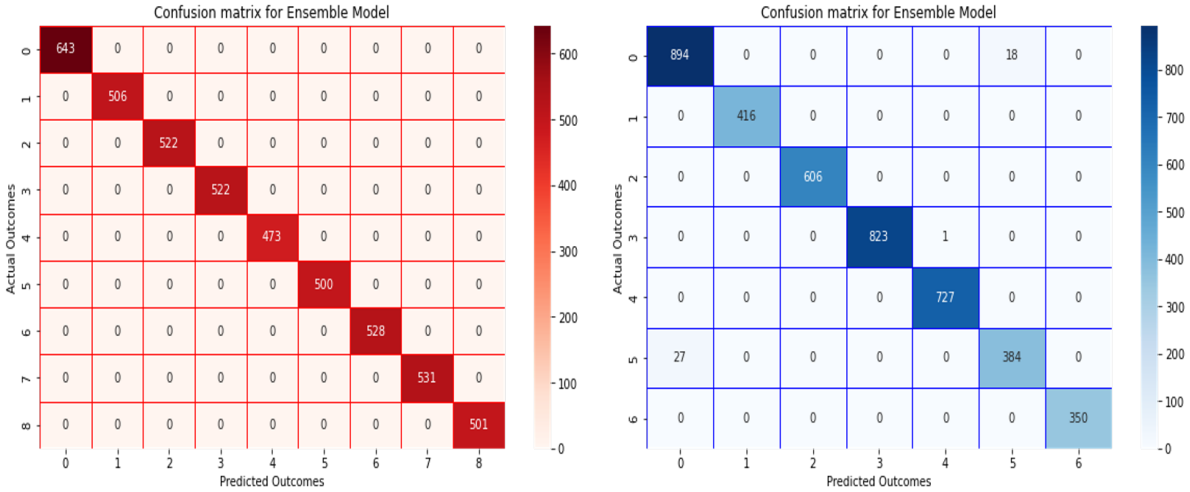
tions are reliable. The predicted results are shown in Figures 4.12 and 4.13 for IDS2017 and IDS2018 respectively.

Figure 4.13 – Prediction Result of the proposed ELETL-IDS model on IDS2018



Source: Author (2022)

Figure 4.14 – Confusion Matrix showing the performance of the proposed ELETL-IDS model on selected dataset (a) CIC-IDS2017 and (b) CSE-CIC-IDS2018

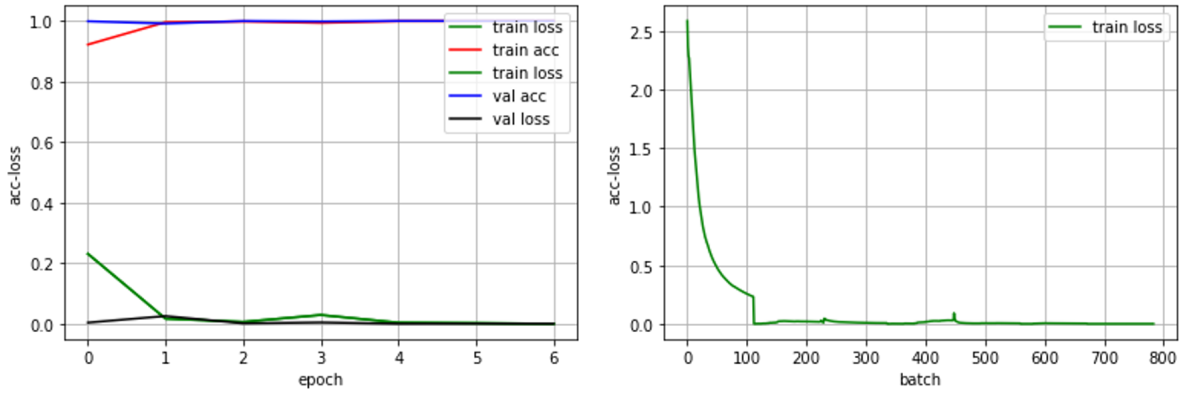


a. CIC-IDS2017 b. CSE-CIC-IDS2018

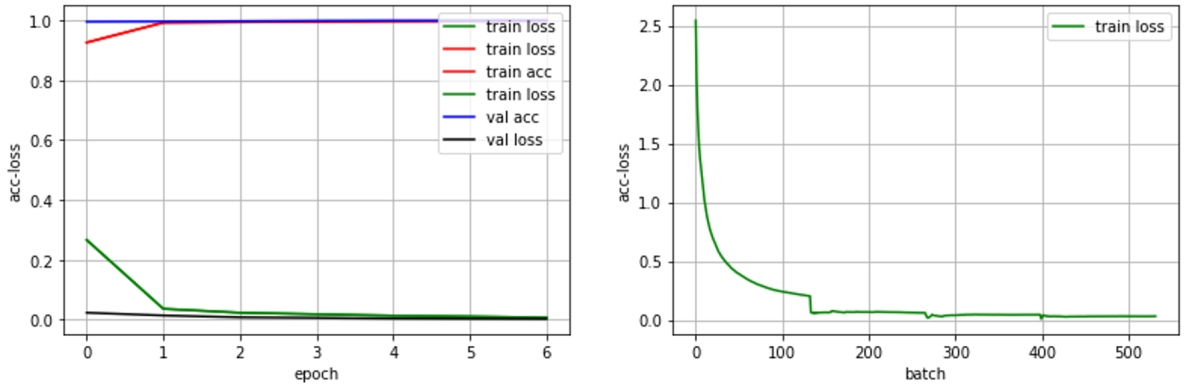
Source: Author (2022)

In Figure 4.15, we present the learning curves for the ELETL-IDS model showing the reconstruction error and training loss of the model while extracting and learning the features of the input images. On both datasets, our model during training maintained both training and validation accuracy of almost 100% from the first epoch with 100 batch size using the categorical cross-entropy loss function.

Figure 4.15 – Learning Curves for the ELETL-IDS Model on the selected Data sets



a. Training Curve for ELETL-IDS on CIC-IDS2017



b. Training Curve for ELETL-IDS on CSE-CIC-IDS2018

Source: Author (2022)

We compare the performance of ELETL-IDS and 1D-CNN IDS models with other state-of-the-art. In terms of accuracy, our proposed models achieved better performances compared to the presented models in Table 4.14. The CNN used in Zhang *et al.* (2020b) showed an accuracy of 99.85% on the CICIDS2017 dataset while our IDS model showed an accuracy of 100.00% on the same dataset. In a similar situation, the work of Farhana *et al.* (2020) can classify the traffic features with an accuracy of 98.18% on the IDS2018 dataset and 97.38% on the IDS2017 dataset using a method similar to our work.

Table 4.14 – Comparison of the IDS models with Related works

IDS Model	Eval. Metrics	Feature Extraction	Dataset	Acc (%)
Ensemble Kim (2017a)	PR, RC, FS	-	CICIDS2017	99.02
Hybrid (CNN+LSTM) Sun *et al.* (2020)	PR, RC, FS	-	CICIDS2017	98.67
Ensemble Varanasi e Razia (2021)	PR, RC, FS	CNN	CICIDS2017	99.00
Ensemble Kim, Shin e Choi (2019)	-	CNN	CICIDS2018	99.77
Ensemble Parvat *et al.* (2017)	PR, RC, FS	Gaussian naïve bayes	NSL-KDD	81.27
Ensemble Hu *et al.* (2020)	ACC, DR, FPR	CNN	NSL-KDD	83.83
CNN Riyaz e Ganapathy (2020)	ACC	Linear Correlation Coefficient	NSL-KDD	99.88
Ensemble Wu, Chen e Li (2018)	ACC, DR, FPR	CNN	NSL-KDD	79
HCRNNIDS Khan (2021)	PR, RC, FS	-	CICIDS2018	97.75
CNN Ferrag *et al.* (2020)	FPR, FNR, ACC	CNN	CICIDS2017, CICIDS2018	IDS2017 = 97.38, IDS2018 = 98.18
CNN Zhang *et al.* (2020b)	ACC, DR, FPR, FS	Conditional Random Field and LCC	CICIDS2017	99.85
SVM Preethi e Khare (2021)	RC, PR, ACC, FS	Principal Component Analysis/Auto Encoder	NSL-KDD	97
LSTM-DL Lin, Ye e Xu (2019)	RC, PR, ACC, FS	CNN	IDS2018	96.2
ResNet50 Masum e Shahriar (2021)	ACC	CNN	NSL-KDD	81.15
VGG-19 + DNN Masum e Shahriar (2021)	ACC	VGG19	NSL-KDD	86.6
VGG-16 + DNN Masum e Shahriar (2021)	ACC	VGG16	NSL-KDD	89.3
RBM Mayuranathan, Murugan e Dhanakoti (2021)	FPR, FNR, ACC	Random Harmony Search	KDD Cup'99	99.79
Ensemble ELETL-IDS [Proposed]	**FPR, FNR, ACC**	**CNN/RFR**	**CICIDS2017, CICIDS2018**	**IDS2017 = 100.0, IDS2018 = 99.99**
1D-CNN [Proposed]	**FPR, FNR, ACC, MCC**	**CNN/RFR**	**CICIDS2017, CICIDS2018**	**IDS2017 = 99.30, IDS2018 = 99.20**
ELETL-IDS [Quantized]	**FPR, FNR, ACC, MCC**	**CNN/RFR**	**CICIDS2017, CICIDS2018**	**IDS2017 = 98.9, IDS2018 = 98.9**
1D-CNN [Quantized]	**FPR, FNR, ACC, MCC**	**CNN/RFR**	**CICIDS2017, CICIDS2018**	**IDS2017 = 98.20, IDS2018 = 98.20**

Source: Author (2022)

4.3 Evaluation of Models Optimized using Quantization

With quantization, we reduced the memory requirement capacity of the model such that it can be deployed on edge devices thereby achieving a robust model on the low memory capacity devices. Original model expects an input of FP32 bits. After apply half-integer quantization, the input data type was converted to INT32. With this, the resulting model size was reduced by approximately 67% for the Base CNN model from 21.2MB to 7.1MB and 77% for the proposed ELETL-IDS model from 338.199 MB to 77.976 MB as shown in Table 4.15. Similarly, the accuracy of the models dropped to about 0.7% for the Base CNN-PTQ, 0.81% for Base CNN-TAQ and 1.1% for the ELETL-IDS model using the PTQ weight only quantization as shown in Table 4.16. The quantization scheme discussed here is as described in TensorFlow Lite[1].

Table 4.15 – Comparison of Original Model size and Model sizes after quantization

Models	**Original Model Size**	**Model Size After Quantization**	**Reduction**
Base CNN	21.2MB	7.1MB	67%
ELETL-IDS	338.199MB	77.976MB	77%

Source: Author (2022)

Table 4.16 – Comparison of Accuracy of Original and Quantize Models

Models	**Original Accuracy %**	**Quantize Model Acc %**	Reduction
Base CNN-ATQ	98.90	98.20	0.707
Base CNN-PTQ	97.7	96.90	0.81%
ELETL-IDS-PTQ	100	98.9	1.1%

Source: Author (2022)

[1] https://www.tensorflow.org/model_optimization/guide/quantization/post_training

5 CONCLUSION AND FUTURE WORK

The field of Computing and computer Networks and related domains that use Internet connection is largely threatened by so many intrusive activities. Several approaches have been used including knowledge-based expert systems to monitor and deter the attacks from occurring such as the firewalls, anti-virus software and even network monitoring agents like humans. Due to the inefficiencies experienced in these approaches, more suitable way is the implementation of IDS developed with intelligence based on ML/DL algorithms. In this work, we have studied and evaluated different algorithms including classical ML and DL to develop suitable IDS model that can be efficiently used in different platforms in mitigating network intrusion.

A typical IDS functions in four difference mechanisms: packet decoding, packet Preprocessing, decision system and detection mechanisms. Using this architecture, we first obtained two datasets comprising high volume of network traffic flow of up to 16 million generated in both forward and backward directions over a long period of time using different attack strategies. We used several methods to preprocess the dataset such that they are suitable for the IDS model development. Feature engineering was done to select important features; hence, 64 features of the datasets were used for the model development. To overcome data imbalance so that a robust model was developed, we over-sampled the minority classes in the datasets using SMOTE. During the model training, reconstruction error and training loss were controlled using cross-entropy loss function.

Two approaches were adopted in this work in order to achieve a robust IDS model. We first implemented ML and 1D-CNN algorithms on Tabular data and then, converted the dataset to images having 64 x 64 x 3 image size. Six different ML models including Decision Tree (DT), Random Forest (RF), Extra Tree (ET), AdaBoost (AD), XGBoost (XGB) and LightGBM (LGBM) were trained on the Tabular data which was first scaled using MaxAbsScaler. Going on, we implemented 1D-CNN using three approaches on the dataset including imbalanced data, balanced with train-test-split and balanced with StratifiedKFold split. The results in each scenario showed improvement in previous method. Specifically, the training accuracy of the three 1D-CNN models are 97.76% for imbalanced dataset, 99.11% for balanced dataset and 99.20% for the balanced dataset with stratifiedKFold split. This presents an improvement in IDS model in comparison with the work of Ferrag *et al.* (2020)

which showed an accuracy of 97.38% on CIC-IDS2017 and 98.18% on CSE-CICIDS2018 dataset.

On the image dataset, to make the image suitable for feature extraction and learning process, we resized each image to 256 x 256 x 3 and used transfer learning with VGG16, VGG19, IV3, MNV3S and ENV2B0 to develop the proposed model. In the end, we constructed an ensemble model (ELETL-IDS) based on IV3, MNV3S and ENV2B0 which were found to be the best performing models. The ELETL-IDS achieved an accuracy of 100% on the IDS2017 dataset and 99.99% on the IDS2018 dataset. To enhance the usage coverage of the model, we employed quantization technique to obtain a low memory capacity model which can be deployed on any device. The results obtained show that our developed models are efficient, reliable and compatible to operate on any device with memory requirement of about 77MB (4x smaller) than original model size. 1D-CNN and ELETL-IDS can classify up to 9 different network traffic with average accuracy of 99.99%. The proposed models can be implemented either as NIDS or HIDS. In general, the results obtained demonstrated an advancement in IDS model design using pre-trained models compared to the work of Masum e Shahriar (2021).

5.1 Challenges and Recommendations

Several challenges were encountered during the research but were all overcome except the deployment of the model on a network interface to test using flow packets generated by us in real-time scenario. Therefore, in the future, we hope to test the model developed in a test-bed while exploring other available algorithms that can improve the performance of the IDS in terms of latency and complexity.

REFERENCES

ACM. **Data Mining and Knowledge Discovery - SIGKDD - KDD Cup**. 2016. <https://kdd.org/kdd-cup>. (Accessed on 03/03/2022).

AHMAD, T.; ANWAR, M. A.; HAQUE, M. Machine learning techniques for intrusion detection. In: **Handbook of Research on Intrusion Detection Systems**. [S.l.]: IGI Global, 2020. p. 47–65.

AL-GARADI, M. A. *et al.* A survey of machine and deep learning methods for internet of things (iot) security. **IEEE Communications Surveys & Tutorials**, IEEE, v. 22, n. 3, p. 1646–1685, 2020.

AL-TASHI, Q. *et al.* Approaches to multi-objective feature selection: A systematic literature review. **IEEE Access**, IEEE, v. 8, p. 125076–125096, 2020.

ALASADI, S. A. Anomaly detection system for internet traffic based on tf-idf and bfr clustering algorithms. **International Journal of Engineering & Technology**, v. 8, n. 1.5, p. 131–137, 2019.

ALBAWI, S.; MOHAMMED, T. A.; AL-ZAWI, S. Understanding of a convolutional neural network. In: IEEE. **2017 International Conference on Engineering and Technology (ICET)**. [S.l.], 2017. p. 1–6.

ALDWAIRI, T.; PERERA, D.; NOVOTNY, M. A. An evaluation of the performance of restricted boltzmann machines as a model for anomaly network intrusion detection. **Computer Networks**, Elsevier, v. 144, p. 111–119, 2018.

ALJAWARNEH, S.; YASSEIN, M. B.; ALJUNDI, M. An enhanced j48 classification algorithm for the anomaly intrusion detection systems. **Cluster Computing**, Springer, v. 22, n. 5, p. 10549–10565, 2019.

ALMSEIDIN, M. *et al.* Evaluation of machine learning algorithms for intrusion detection system. In: IEEE. **2017 IEEE 15th International Symposium on Intelligent Systems and Informatics (SISY)**. [S.l.], 2017. p. 000277–000282.

ALOM, M. Z. *et al.* The history began from alexnet: A comprehensive survey on deep learning approaches. **arXiv preprint arXiv:1803.01164**, 2018.

ALRAWASHDEH, K.; PURDY, C. Toward an online anomaly intrusion detection system based on deep learning. In: IEEE. **2016 15th IEEE international conference on machine learning and applications (ICMLA)**. [S.l.], 2016. p. 195–200.

AMINANTO, E.; KIM, K. Deep learning in intrusion detection system: An overview. In: HIGHER EDUCATION FORUM. **2016 International Research Conference on Engineering and Technology (2016 IRCET)**. [S.l.], 2016.

ANDERSON, J. P. Computer security threat monitoring and surveillance. **Technical Report, James P. Anderson Company**, 1980.

APRUZZESE, G. *et al.* On the effectiveness of machine and deep learning for cyber security. In: IEEE. **2018 10th international conference on cyber Conflict (CyCon)**. [S.l.], 2018. p. 371–390.

ATAWODI, I. S. A machine learning approach to network intrusion detection system using k nearest neighbor and random forest. 2019.

AXELSSON, S. Intrusion detection systems: A survey and taxonomy. Citeseer, 2000.

BALRAM, S.; WISCY, M. Detection of tcp syn scanning using packet counts and neural network. In: IEEE. **2008 IEEE International Conference on Signal Image Technology and Internet Based Systems**. [S.l.], 2008. p. 646–649.

BANNER, R.; NAHSHAN, Y.; SOUDRY, D. Post training 4-bit quantization of convolutional networks for rapid-deployment. **Advances in Neural Information Processing Systems**, v. 32, 2019.

BASUMALLIK, S.; MA, R.; EFTEKHARNEJAD, S. Packet-data anomaly detection in pmu-based state estimator using convolutional neural network. **International Journal of Electrical Power & Energy Systems**, Elsevier, v. 107, p. 690–702, 2019.

BERGSTRA, J.; YAMINS, D.; COX, D. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In: PMLR. **International conference on machine learning**. [S.l.], 2013. p. 115–123.

BHATI, B. S.; RAI, C. Analysis of support vector machine-based intrusion detection techniques. **Arabian Journal for Science and Engineering**, Springer, v. 45, n. 4, p. 2371–2383, 2020.

BHATI, B. S.; RAI, C. S. Ensemble based approach for intrusion detection using extra tree classifier. In: SOLANKI, V. K. *et al.* (Ed.). **Intelligent Computing in Engineering**. Singapore: Springer Singapore, 2020. p. 213–220.

BHUYAN, M. H.; BHATTACHARYYA, D. K.; KALITA, J. K. Network anomaly detection: methods, systems and tools. **Ieee communications surveys & tutorials**, IEEE, v. 16, n. 1, p. 303–336, 2013.

BISWAS, S. K. *et al.* Intrusion detection using machine learning: A comparison study. **International Journal of pure and applied mathematics**, v. 118, n. 19, p. 101–114, 2018.

BORGWARDT, K. M. *et al.* Integrating structured biological data by kernel maximum mean discrepancy. **Bioinformatics**, Oxford University Press, v. 22, n. 14, p. e49–e57, 2006.

BORKAR, A.; DONODE, A.; KUMARI, A. A survey on intrusion detection system (ids) and internal intrusion detection and protection system (iidps). In: IEEE. **2017 International conference on inventive computing and informatics (ICICI)**. [S.l.], 2017. p. 949–953.

BRAEI, M.; WAGNER, S. Anomaly detection in univariate time-series: A survey on the state-of-the-art. **arXiv preprint arXiv:2004.00433**, 2020.

BRASPENNING, P. J.; THUIJSMAN, F.; WEIJTERS, A. J. M. M. **Artificial neural networks: an introduction to ANN theory and practice**. [S.l.]: Springer Science & Business Media, 1995. v. 931.

BREIMAN, L. Bagging predictors. **Machine learning**, Springer, v. 24, n. 2, p. 123–140, 1996.

BUITINCK, L. *et al.* API design for machine learning software: experiences from the scikit-learn project. In: **ECML PKDD Workshop: Languages for Data Mining and Machine Learning**. [S.l.: s.n.], 2013. p. 108–122.

BYVATOV, E.; SCHNEIDER, G. Support vector machine applications in bioinformatics. **Applied bioinformatics**, v. 2, n. 2, p. 67–77, 2003.

CAI, J. *et al.* Feature selection in machine learning: A new perspective. **Neurocomputing**, Elsevier, v. 300, p. 70–79, 2018.

CENKERAMADDI, L. R. *et al.* A survey on sensors for autonomous systems. In: IEEE. **2020 15th IEEE Conference on Industrial Electronics and Applications (ICIEA)**. [S.l.], 2020. p. 1182–1187.

CHANDOLA, V.; BANERJEE, A.; KUMAR, V. Anomaly detection: A survey. **ACM computing surveys (CSUR)**, ACM New York, NY, USA, v. 41, n. 3, p. 1–58, 2009.

CHANDRASHEKHAR, S.; TAMANE, M.; BHARATI. **Patent on Deep Learning Based Intrusion Prediction Model - MGM University Aurangabad**. 2021. <https://mgmu.ac.in/deep-learning-based-intrusion-prediction-model/>. (Accessed on 09/08/2022).

CHAUHAN, M.; AGARWAL, M. Study of various intrusion detection systems: A survey. **Smart and Sustainable Intelligent Systems**, Wiley Online Library, p. 355–372, 2021.

CHAWLA, N. V. *et al.* Smote: synthetic minority over-sampling technique. **Journal of artificial intelligence research**, v. 16, p. 321–357, 2002.

CHEN, T.; GUESTRIN, C. Xgboost: A scalable tree boosting system. In: **Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining**. [S.l.: s.n.], 2016. p. 785–794.

CHEN, W. *et al.* Distributed resilient filtering for power systems subject to denial-of-service attacks. **IEEE Transactions on Systems, Man, and Cybernetics: Systems**, IEEE, v. 49, n. 8, p. 1688–1697, 2019.

CHEN, Y.-K. Challenges and opportunities of internet of things. In: IEEE. **17th Asia and South Pacific design automation conference**. [S.l.], 2012. p. 383–388.

CHOI, H. *et al.* Unsupervised learning approach for network intrusion detection system using autoencoders. **The Journal of Supercomputing**, Springer, v. 75, n. 9, p. 5597–5621, 2019.

CHOI, J.-W. Web server hacking and security risk using dns spoofing and pharming combined attack. **Journal of the Korea Institute of Information and Communication Engineering**, The Korea Institute of Information and Commucation Engineering, v. 23, n. 11, p. 1451–1461, 2019.

CIC. **DDoS 2019 | Datasets | Research | Canadian Institute for Cybersecurity | UNB**. 2019. <https://www.unb.ca/cic/datasets/ddos-2019.html>. (Accessed on 02/27/2022).

CRESWELL, A. *et al.* Generative adversarial networks: An overview. **IEEE Signal Processing Magazine**, IEEE, v. 35, n. 1, p. 53–65, 2018.

DAS, S. *et al.* Network intrusion detection and comparative analysis using ensemble machine learning and feature selection. **IEEE Transactions on Network and Service Management**, p. 1–1, 2021.

DENNING, D. E. An intrusion-detection model. **IEEE Transactions on software engineering**, IEEE, n. 2, p. 222–232, 1987.

DHALIWAL, S. S.; NAHID, A.-A.; ABBAS, R. Effective intrusion detection system using xgboost. **Information**, MDPI, v. 9, n. 7, p. 149, 2018.

DOULIGERIS, C.; MITROKOTSA, A. Ddos attacks and defense mechanisms: classification and state-of-the-art. **Computer networks**, Elsevier, v. 44, n. 5, p. 643–666, 2004.

DOWELL, C. The computerwatch data reduction tool. In: **13th National Computer Security Conference**. [S.l.: s.n.], 1990. p. 99–108.

ELFWING, S.; UCHIBE, E.; DOYA, K. Sigmoid-weighted linear units for neural network function approximation in reinforcement learning. **Neural Networks**, Elsevier, v. 107, p. 3–11, 2018.

FARHANA, K. *et al.* An intrusion detection system for packet and flow based networks using deep neural network approach. **International Journal of Electrical & Computer Engineering (2088-8708)**, v. 10, n. 5, 2020.

FARNAAZ, N.; JABBAR, M. Random forest modeling for network intrusion detection system. **Procedia Computer Science**, Elsevier, v. 89, p. 213–217, 2016.

FAWCETT, T. An introduction to roc analysis. **Pattern Recognition Letters**, v. 27, n. 8, p. 861–874, 2006. ISSN 0167-8655. ROC Analysis in Pattern Recognition. Disponível em: <https://www.sciencedirect.com/science/article/pii/S016786550500303X>.

FERRAG, M. A. *et al.* Deep learning techniques for cyber security intrusion detection: A detailed analysis. In: **6th International Symposium for ICS & SCADA Cyber Security Research 2019 6**. [S.l.: s.n.], 2019. p. 126–136.

FERRAG, M. A. *et al.* Deep learning for cyber security intrusion detection: Approaches, datasets, and comparative study. **Journal of Information Security and Applications**, Elsevier, v. 50, p. 102419, 2020.

FERREIRA, A. J.; FIGUEIREDO, M. A. Efficient feature selection filters for high-dimensional data. **Pattern recognition letters**, Elsevier, v. 33, n. 13, p. 1794–1804, 2012.

FERRIS, M. H. *et al.* Using roc curves and auc to evaluate performance of no-reference image fusion metrics. In: **2015 National Aerospace and Electronics Conference (NAECON)**. [S.l.: s.n.], 2015. p. 27–34.

FIORE, U. *et al.* Network anomaly detection with the restricted boltzmann machine. **Neurocomputing**, Elsevier, v. 122, p. 13–23, 2013.

FIORESE, T.; MONTINO, P. Learning-based intrusion detection system for on-board vehicle communication. In: **ITASEC**. [S.l.: s.n.], 2021. p. 180–192.

FRIEDMAN, J. H. Greedy function approximation: a gradient boosting machine. **Annals of statistics**, JSTOR, p. 1189–1232, 2001.

FU, K. *et al.* Credit card fraud detection using convolutional neural networks. In: SPRINGER. **International conference on neural information processing**. [S.l.], 2016. p. 483–490.

FUKETA, H. *et al.* Image-classifier deep convolutional neural network training by 9-bit dedicated hardware to realize validation accuracy and energy efficiency superior to the half precision floating point format. In: IEEE. **2018 IEEE International Symposium on Circuits and Systems (ISCAS)**. [S.l.], 2018. p. 1–5.

GAO, N. *et al.* An intrusion detection model based on deep belief networks. In: IEEE. **2014 Second International Conference on Advanced Cloud and Big Data**. [S.l.], 2014. p. 247–252.

GASTI, P. *et al.* Dos and ddos in named data networking. In: IEEE. **2013 22nd International Conference on Computer Communication and Networks (ICCCN)**. [S.l.], 2013. p. 1–7.

GENTLEMAN, R.; CAREY, V. J. Unsupervised machine learning. In: **Bioconductor case studies**. [S.l.]: Springer, 2008. p. 137–157.

GHARIB, A. *et al.* An evaluation framework for intrusion detection dataset. In: **2016 International Conference on Information Science and Security (ICISS)**. [S.l.: s.n.], 2016. p. 1–6.

GHARIB, M. *et al.* Autoids: auto-encoder based method for intrusion detection system. **arXiv preprint arXiv:1911.03306**, 2019.

GOSAIN, A.; SARDANA, S. Handling class imbalance problem using oversampling techniques: A review. In: IEEE. **2017 international conference on advances in computing, communications and informatics (ICACCI)**. [S.l.], 2017. p. 79–85.

GUO, Y. A survey on methods and theories of quantized neural networks. **arXiv preprint arXiv:1808.04752**, 2018.

HALIMAA, A.; SUNDARAKANTHAM, K. Machine learning based intrusion detection system. In: IEEE. **2019 3rd International conference on trends in electronics and informatics (ICOEI)**. [S.l.], 2019. p. 916–920.

HAN, E.-H. S.; KARYPIS, G.; KUMAR, V. Text categorization using weight adjusted k-nearest neighbor classification. In: SPRINGER. **Pacific-asia conference on knowledge discovery and data mining**. [S.l.], 2001. p. 53–65.

HAN, H.; WANG, W.-Y.; MAO, B.-H. Borderline-smote: a new over-sampling method in imbalanced data sets learning. In: SPRINGER. **International conference on intelligent computing**. [S.l.], 2005. p. 878–887.

HEBA, F. E. *et al.* Principle components analysis and support vector machine based intrusion detection system. In: IEEE. **2010 10th international conference on intelligent systems design and applications**. [S.l.], 2010. p. 363–367.

HEBERLEIN, L. T. *et al.* **A network security monitor**. [S.l.], 1989.

HINTON, G.; SRIVASTAVA, N.; SWERSKY, K. Neural networks for machine learning lecture 6a overview of mini-batch gradient descent. **Cited on**, v. 14, n. 8, p. 2, 2012.

HO, S. *et al.* A novel intrusion detection model for detecting known and innovative cyberattacks using convolutional neural network. **IEEE Open Journal of the Computer Society**, v. 2, p. 14–25, 2021.

HONDA, S. *et al.* Topase: Detection of brute force attacks used disciplined ips from ids log. In: IEEE. **2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)**. [S.l.], 2015. p. 1361–1364.

HOWARD, A. *et al.* Searching for mobilenetv3. In: **Proceedings of the IEEE/CVF international conference on computer vision**. [S.l.: s.n.], 2019. p. 1314–1324.

HOWARD, A. G. *et al.* Mobilenets: Efficient convolutional neural networks for mobile vision applications. **arXiv preprint arXiv:1704.04861**, 2017.

HU, Z. *et al.* A novel wireless network intrusion detection method based on adaptive synthetic sampling and an improved convolutional neural network. **IEEE Access**, IEEE, v. 8, p. 195741–195751, 2020.

HUANG, S.; LEI, K. Igan-ids: An imbalanced generative adversarial network towards intrusion detection system in ad-hoc networks. **Ad Hoc Networks**, Elsevier, v. 105, p. 102177, 2020.

HUSSAIN, F. *et al.* Iot dos and ddos attack detection using resnet. In: **2020 IEEE 23rd International Multitopic Conference (INMIC)**. [S.l.: s.n.], 2020. p. 1–6.

HUSSAIN, F. *et al.* Machine learning in iot security: Current solutions and future challenges. **IEEE Communications Surveys Tutorials**, v. 22, n. 3, p. 1686–1721, 2020.

INGRE, B.; YADAV, A. Performance analysis of nsl-kdd dataset using ann. In: IEEE. **2015 international conference on signal processing and communication engineering systems**. [S.l.], 2015. p. 92–96.

IOFFE, S.; SZEGEDY, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In: PMLR. **International conference on machine learning**. [S.l.], 2015. p. 448–456.

JACOB, B. *et al.* Quantization and training of neural networks for efficient integer-arithmetic-only inference. In: **Proceedings of the IEEE conference on computer vision and pattern recognition**. [S.l.: s.n.], 2018. p. 2704–2713.

JO, W. *et al.* Packet preprocessing in cnn-based network intrusion detection system. **Electronics**, Multidisciplinary Digital Publishing Institute, v. 9, n. 7, p. 1151, 2020.

JOHNSON, D. **Unsupervised Machine Learning: Algorithms, Types with Example**. 2022. <https://www.guru99.com/unsupervised-machine-learning.html#applications-of-unsupervised-machine-learning>. (Accessed on 03/13/2022).

JOSE, S. *et al.* A survey on anomaly based host intrusion detection system. In: IOP PUBLISHING. **Journal of Physics: Conference Series**. [S.l.], 2018. v. 1000, n. 1, p. 012049.

KALCHBRENNER, N.; GREFENSTETTE, E.; BLUNSOM, P. A convolutional neural network for modelling sentences. **arXiv preprint arXiv:1404.2188**, 2014.

KANG, M.-J.; KANG, J.-W. Intrusion detection system using deep neural network for in-vehicle network security. **PloS one**, Public Library of Science San Francisco, CA USA, v. 11, n. 6, p. e0155781, 2016.

KANNARI, P. R.; SHARIFF, N. C.; BIRADAR, R. L. Network intrusion detection using sparse autoencoder with swish-prelu activation model. **Journal of Ambient Intelligence and Humanized Computing**, Springer, p. 1–13, 2021.

KARATAS, G.; DEMIR, O.; SAHINGOZ, O. K. Deep learning in intrusion detection systems. In: IEEE. **2018 International Congress on Big Data, Deep Learning and Fighting Cyber Terrorism (IBIGDELFT)**. [S.l.], 2018. p. 113–116.

KAREEM, F. Q. *et al.* Sql injection attacks prevention system technology. **Asian Journal of Research in Computer Science**, v. 13, p. 32, 2021.

KARLIK, B.; OLGAC, A. V. Performance analysis of various activation functions in generalized mlp architectures of neural networks. **International Journal of Artificial Intelligence and Expert Systems**, Citeseer, v. 1, n. 4, p. 111–122, 2011.

KAUR, G.; KAUR, N. Penetration testing–reconnaissance with nmap tool. **International Journal of Advanced Research in Computer Science**, v. 8, n. 3, 2017.

KAUR, S.; SINGH, M. Hybrid intrusion detection and signature generation using deep recurrent neural networks. **Neural Computing and Applications**, Springer, v. 32, n. 12, p. 7859–7877, 2020.

KAYACIK, H. G.; ZINCIR-HEYWOOD, N. Analysis of three intrusion detection system benchmark datasets using machine learning algorithms. In: SPRINGER. **International Conference on Intelligence and Security Informatics**. [S.l.], 2005. p. 362–367.

KE, G. *et al.* Lightgbm: A highly efficient gradient boosting decision tree. **Advances in neural information processing systems**, v. 30, 2017.

KHAN, F. A. *et al.* A novel two-stage deep learning model for efficient network intrusion detection. **IEEE Access**, IEEE, v. 7, p. 30373–30385, 2019.

KHAN, L. U. *et al.* Federated learning for internet of things: Recent advances, taxonomy, and open challenges. **IEEE Communications Surveys Tutorials**, v. 23, n. 3, p. 1759–1799, 2021.

KHAN, M. A. Hcrnnids: Hybrid convolutional recurrent neural network-based network intrusion detection system. **Processes**, v. 9, n. 5, 2021. ISSN 2227-9717. Disponível em: <https://www.mdpi.com/2227-9717/9/5/834>.

KHRAISAT, A.; ALAZAB, A. A critical review of intrusion detection systems in the internet of things: techniques, deployment strategy, validation strategy, attacks, public datasets and challenges. **Cybersecurity**, Springer, v. 4, n. 1, p. 1–27, 2021.

KHRAISAT, A. *et al.* A novel ensemble of hybrid intrusion detection system for detecting internet of things attacks. **Electronics**, Multidisciplinary Digital Publishing Institute, v. 8, n. 11, p. 1210, 2019.

KHRAISAT, A. *et al.* **Survey of intrusion detection systems: techniques, datasets and challenges. Cybersecurity 2, 20 (2019)**. 2019.

KIM, J. *et al.* Method of intrusion detection using deep neural network. In: IEEE. **2017 IEEE international conference on big data and smart computing (BigComp)**. [S.l.], 2017. p. 313–316.

KIM, J.; SHIN, Y.; CHOI, E. An intrusion detection model based on a convolutional neural network. **Journal of Multimedia Information System**, Korea Multimedia Society, v. 6, n. 4, p. 165–172, 2019.

KIM, J.-Y.; BU, S.-J.; CHO, S.-B. Zero-day malware detection using transferred generative adversarial networks based on deep autoencoders. **Information Sciences**, Elsevier, v. 460, p. 83–102, 2018.

KIM, P. Convolutional neural network. In: **MATLAB deep learning**. [S.l.]: Springer, 2017. p. 121–147.

KIM, P. Deep learning. In: **MATLAB Deep Learning**. [S.l.]: Springer, 2017. p. 103–120.

KINGMA, D. P.; BA, J. Adam: A method for stochastic optimization. **arXiv preprint arXiv:1412.6980**, 2014.

KOSTAS, K. **Anomaly Detection in Networks Using Machine Learning**. Dissertação (Mestrado) — University of Essex, Colchester, UK, 2018.

KRISHNAMOORTHI, R. Quantizing deep convolutional networks for efficient inference: A whitepaper. **arXiv preprint arXiv:1806.08342**, 2018.

LAKSHMINARAYANA, D. H.; PHILIPS, J.; TABRIZI, N. A survey of intrusion detection techniques. In: **2019 18th IEEE International Conference On Machine Learning And Applications (ICMLA)**. [S.l.: s.n.], 2019. p. 1122–1129.

LANDGREBE, T. C.; DUIN, R. P. Efficient multiclass roc approximation by decomposition via confusion matrix perturbation analysis. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, v. 30, n. 5, p. 810–822, 2008.

LANGE, S.; RIEDMILLER, M. Deep auto-encoder neural networks in reinforcement learning. In: IEEE. **The 2010 International Joint Conference on Neural Networks (IJCNN)**. [S.l.], 2010. p. 1–8.

LARGE, J.; LINES, J.; BAGNALL, A. A probabilistic classifier ensemble weighting scheme based on cross-validated accuracy estimates. **Data mining and knowledge discovery**, Springer, v. 33, n. 6, p. 1674–1709, 2019.

LASHKARI, A. H.; KAUR, G.; RAHALI, A. Didarknet: A contemporary approach to detect and characterize the darknet traffic using deep image learning. In: **2020 the 10th International Conference on Communication and Network Security**. [S.l.: s.n.], 2020. p. 1–13.

LEE, C. *et al.* Automatic disease annotation from radiology reports using artificial intelligence implemented by a recurrent neural network. **American Journal of Roentgenology**, Am Roentgen Ray Soc, v. 212, n. 4, p. 734–740, 2019.

LEE, S.-W. *et al.* Towards secure intrusion detection systems using deep learning techniques: Comprehensive analysis and review. **Journal of Network and Computer Applications**, Elsevier, p. 103111, 2021.

LEEVY, J. L.; KHOSHGOFTAAR, T. M. A survey and analysis of intrusion detection models based on cse-cic-ids2018 big data. **Journal of Big Data**, SpringerOpen, v. 7, n. 1, p. 1–19, 2020.

LI, X. *et al.* Transfer learning based intrusion detection scheme for internet of vehicles. **Information Sciences**, Elsevier, v. 547, p. 119–135, 2021.

LIN, D.; TALATHI, S.; ANNAPUREDDY, S. Fixed point quantization of deep convolutional networks. In: PMLR. **International conference on machine learning**. [S.l.], 2016. p. 2849–2858.

LIN, P.; YE, K.; XU, C.-Z. Dynamic network anomaly detection system by using deep learning techniques. In: SPRINGER. **International conference on cloud computing**. [S.l.], 2019. p. 161–176.

LIN, W.-C.; KE, S.-W.; TSAI, C.-F. Cann: An intrusion detection system based on combining cluster centers and nearest neighbors. **Knowledge-based systems**, Elsevier, v. 78, p. 13–21, 2015.

LIN, W.-H. *et al.* Using convolutional neural networks to network intrusion detection for cyber threats. In: **2018 IEEE International Conference on Applied System Invention (ICASI)**. [S.l.: s.n.], 2018. p. 1107–1110.

LIPPMANN, R. *et al.* Results of the darpa 1998 offline intrusion detection evaluation. In: . [S.l.: s.n.], 1999.

LIU, H.; LANG, B. Machine learning and deep learning methods for intrusion detection systems: A survey. **applied sciences**, mdpi, v. 9, n. 20, p. 4396, 2019.

LIU, J.; GAO, Y.; HU, F. A fast network intrusion detection system using adaptive synthetic oversampling and lightgbm. **Computers & Security**, Elsevier, v. 106, p. 102289, 2021.

LIU, J. *et al.* Adversarial machine learning: A multi-layer review of the state-of-the-art and challenges for wireless and mobile systems. **IEEE Communications Surveys Tutorials**, p. 1–1, 2021.

LIU, K.; ZHANG, L. M.; SUN, Y. W. Deep boltzmann machines aided design based on genetic algorithms. In: TRANS TECH PUBL. **Applied mechanics and materials**. [S.l.], 2014. v. 568, p. 848–851.

LIU, L. Research on logistic regression algorithm of breast cancer diagnose data by machine learning. In: **2018 International Conference on Robots Intelligent System (ICRIS)**. [S.l.: s.n.], 2018. p. 157–160.

LIU, M. *et al.* Host-based intrusion detection system with system calls: Review and future trends. **ACM Computing Surveys (CSUR)**, ACM New York, NY, USA, v. 51, n. 5, p. 1–36, 2018.

LIU, Y. *et al.* Deep learning based encryption policy intrusion detection using commodity wifi. In: **2019 IEEE 5th International Conference on Computer and Communications (ICCC)**. [S.l.: s.n.], 2019. p. 2129–2135.

LOKMAN, S.-F. *et al.* The impact of different feature scaling methods on intrusion detection for in-vehicle controller area network (can). In: SPRINGER. **International Conference on Advances in Cyber Security**. [S.l.], 2019. p. 195–205.

LOPEZ-MARTIN, M.; CARRO, B.; SANCHEZ-ESGUEVILLAS, A. Application of deep reinforcement learning to intrusion detection for supervised problems. **Expert Systems with Applications**, Elsevier, v. 141, p. 112963, 2020.

LUNT, T. Detecting intruders in computer systems. In: **Proceedings of the 1993 conference on auditing and computer technology**. [S.l.: s.n.], 1993. v. 61.

LUNT, T. F. Ides: An intelligent system for detecting intruders. In: ROME, ITALY. **Proceedings of the symposium: computer security, threat and countermeasures**. [S.l.], 1990. p. 30–45.

LUNT, T. F. *et al.* Knowledge based intrusion detection. In: **Proceedings of the Annual AI Systems in Government Conference, Washington, DC**. [S.l.: s.n.], 1989.

LYON, G. Nmap security scanner. **línea] URL: http://nmap. org/[Consulta: 8 de junio de 2012]**, 2014.

LYU, M. R.; LAU, L. K. Firewall security: Policies, testing and performance evaluation. In: IEEE. **Proceedings 24th Annual International Computer Software and Applications Conference. COMPSAC2000**. [S.l.], 2000. p. 116–121.

MAMUN, M. S. I. *et al.* Detecting malicious urls using lexical analysis. In: SPRINGER. **International Conference on Network and System Security**. [S.l.], 2016. p. 467–482.

MASDARI, M.; KHEZRI, H. A survey and taxonomy of the fuzzy signature-based intrusion detection systems. **Applied Soft Computing**, Elsevier, v. 92, p. 106301, 2020.

MASEER, Z. K. *et al.* Benchmarking of machine learning for anomaly based intrusion detection systems in the cicids2017 dataset. **IEEE Access**, v. 9, p. 22351–22370, 2021.

MASUM, M.; SHAHRIAR, H. A transfer learning with deep neural network approach for network intrusion detection. **International journal of intellligent computing research**, v. 12, n. 1, 2021.

MAYURANATHAN, M.; MURUGAN, M.; DHANAKOTI, V. Best features based intrusion detection system by rbm model for detecting ddos in cloud environment. **Journal of Ambient Intelligence and Humanized Computing**, Springer, v. 12, n. 3, p. 3609–3619, 2021.

MCHUGH, J. Testing intrusion detection systems: a critique of the 1998 and 1999 darpa intrusion detection system evaluations as performed by lincoln laboratory. **ACM Transactions on Information and System Security (TISSEC)**, ACM New York, NY, USA, v. 3, n. 4, p. 262–294, 2000.

MEHROTRA, L.; SAXENA, P. S. An assessment report on: Statistics-based and signature-based intrusion detection techniques. In: **Information and Communication Technology**. [S.l.]: Springer, 2018. p. 321–327.

MENDONçA, R. V. *et al.* Intrusion detection system based on fast hierarchical deep convolutional neural network. **IEEE Access**, v. 9, p. 61024–61034, 2021.

MERYEM, A.; OUAHIDI, B. E. Hybrid intrusion detection system using machine learning. **Network Security**, Elsevier, v. 2020, n. 5, p. 8–19, 2020.

MEYER, D.; WIEN, F. T. Support vector machines. **The Interface to libsvm in package e1071**, v. 28, 2015.

MILENKOSKI, A. *et al.* Evaluating computer intrusion detection systems: A survey of common practices. **ACM Computing Surveys (CSUR)**, ACM New York, NY, USA, v. 48, n. 1, p. 1–41, 2015.

MISHRA, M.; SRIVASTAVA, M. A view of artificial neural network. In: IEEE. **2014 International Conference on Advances in Engineering & Technology Research (ICAETR-2014)**. [S.l.], 2014. p. 1–3.

MISHRA, P. *et al.* A detailed investigation and analysis of using machine learning techniques for intrusion detection. **IEEE Communications Surveys & Tutorials**, IEEE, v. 21, n. 1, p. 686–728, 2018.

MORE, S. *et al.* A knowledge-based approach to intrusion detection modeling. In: IEEE. **2012 IEEE Symposium on Security and Privacy Workshops**. [S.l.], 2012. p. 75–81.

MOUSTAFA, N.; SLAY, J. The significant features of the unsw-nb15 and the kdd99 data sets for network intrusion detection systems. In: IEEE. **2015 4th international workshop on building analysis datasets and gathering experience returns for security (BADGERS)**. [S.l.], 2015. p. 25–31.

MOUSTAFA, N.; SLAY, J. Unsw-nb15: a comprehensive data set for network intrusion detection systems (unsw-nb15 network data set). In: IEEE. **2015 military communications and information systems conference (MilCIS)**. [S.l.], 2015. p. 1–6.

MUKKAMALA, S.; SUNG, A. H.; ABRAHAM, A. Intrusion detection using an ensemble of intelligent paradigms. **Journal of network and computer applications**, Elsevier, v. 28, n. 2, p. 167–182, 2005.

MUSA, U. S. *et al.* Intrusion detection system using machine learning techniques: A review. In: IEEE. **2020 international conference on smart electronics and communication (ICOSEC)**. [S.l.], 2020. p. 149–155.

MUTZ, D. *et al.* Anomalous system call detection. **ACM Transactions on Information and System Security (TISSEC)**, ACM New York, NY, USA, v. 9, n. 1, p. 61–93, 2006.

NAIR, V.; HINTON, G. E. Rectified linear units improve restricted boltzmann machines. In: **Icml**. [S.l.: s.n.], 2010.

NASR, M.; BAHRAMALI, A.; HOUMANSADR, A. Deepcorr: Strong flow correlation attacks on tor using deep learning. In: **Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security**. [S.l.: s.n.], 2018. p. 1962–1976.

NEAL, R. M. Connectionist learning of belief networks. **Artificial intelligence**, Elsevier, v. 56, n. 1, p. 71–113, 1992.

NEHINBE, J. O. A critical evaluation of datasets for investigating idss and ipss researches. In: IEEE. **2011 IEEE 10th International Conference on Cybernetic Intelligent Systems (CIS)**. [S.l.], 2011. p. 92–97.

NGUYEN, D. C. *et al.* Federated learning for internet of things: A comprehensive survey. **IEEE Communications Surveys Tutorials**, v. 23, n. 3, p. 1622–1658, 2021.

NOBLE, C. C.; COOK, D. J. Graph-based anomaly detection. In: **Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining**. [S.l.: s.n.], 2003. p. 631–636.

NOGAMI, W. *et al.* Optimizing weight value quantization for cnn inference. In: IEEE. **2019 International Joint Conference on Neural Networks (IJCNN)**. [S.l.], 2019. p. 1–8.

NWANKPA, C. *et al.* Activation functions: Comparison of trends in practice and research for deep learning. **arXiv preprint arXiv:1811.03378**, 2018.

PANDA, M.; ABRAHAM, A.; PATRA, M. R. A hybrid intelligent approach for network intrusion detection. **Procedia Engineering**, Elsevier, v. 30, p. 1–9, 2012.

PANIGRAHI, R.; BORAH, S. A detailed analysis of cicids2017 dataset for designing intrusion detection systems. **International Journal of Engineering & Technology**, v. 7, n. 3.24, p. 479–482, 2018.

PAPAMARTZIVANOS, D.; MÁRMOL, F. G.; KAMBOURAKIS, G. Introducing deep learning self-adaptive misuse network intrusion detection systems. **IEEE Access**, IEEE, v. 7, p. 13546–13560, 2019.

PARK, J. *et al.* Network log-based ssh brute-force attack detection model. **CMC-COMPUTERS MATERIALS & CONTINUA**, TECH SCIENCE PRESS 871 CORONADO CENTER DR, SUTE 200, HENDERSON, NV 89052 USA, v. 68, n. 1, p. 887–901, 2021.

PARK, N.; KANG, N. Mutual authentication scheme in secure internet of things technology for comfortable lifestyle. **Sensors**, Multidisciplinary Digital Publishing Institute, v. 16, n. 1, p. 20, 2016.

PARK, N. *et al.* Wipi mobile platform with secure service for mobile rfid network environment. In: SPRINGER. **Asia-Pacific Web Conference**. [S.l.], 2006. p. 741–748.

PARVAT, A. *et al.* Network intrusion detection system using ensemble of binary deep learning classifiers. In: SPRINGER. **International Conference on Smart Trends for Information Technology and Computer Communications**. [S.l.], 2017. p. 3–10.

PEDREGOSA, F. *et al.* Scikit-learn: Machine learning in Python. **Journal of Machine Learning Research**, v. 12, p. 2825–2830, 2011.

PELING, I. B. A. *et al.* Implementation of data mining to predict period of students study using naive bayes algorithm. **Int. J. Eng. Emerg. Technol**, v. 2, n. 1, p. 53, 2017.

PENG, T.; LECKIE, C.; RAMAMOHANARAO, K. Survey of network-based defense mechanisms countering the dos and ddos problems. **ACM Computing Surveys (CSUR)**, ACM New York, NY, USA, v. 39, n. 1, p. 3–es, 2007.

PETERSON, L. E. K-nearest neighbor. **Scholarpedia**, v. 4, n. 2, p. 1883, 2009.

PETROV, D.; HOSPEDALES, T. M. Measuring the transferability of adversarial examples. **arXiv preprint arXiv:1907.06291**, 2019.

PHARATE, A. *et al.* Classification of intrusion detection system. **International Journal of Computer Applications**, Citeseer, v. 118, n. 7, 2015.

PIJPKER, J.; VRANKEN, H. The role of internet service providers in botnet mitigation. In: IEEE. **2016 European Intelligence and Security Informatics Conference (EISIC)**. [S.l.], 2016. p. 24–31.

POSTEL, J. **RFC0768: User Datagram Protocol**. [S.l.]: RFC Editor, 1980.

POSTEL, J. *et al.* Transmission control protocol. STD 7, RFC 793, September, 1981.

POTLURI, S.; DIEDRICH, C. Accelerated deep neural networks for enhanced intrusion detection system. In: IEEE. **2016 IEEE 21st international conference on emerging technologies and factory automation (ETFA)**. [S.l.], 2016. p. 1–8.

PREETHI, D.; KHARE, N. Sparse auto encoder driven support vector regression based deep learning model for predicting network intrusions. **Peer-to-Peer Networking and Applications**, Springer, v. 14, n. 4, p. 2419–2429, 2021.

PROTIĆ, D. D. Review of kdd cup'99, nsl-kdd and kyoto 2006+ datasets. **Vojnotehnički glasnik**, v. 66, n. 3, p. 580–596, 2018.

QUINLAN, J. R. Induction of decision trees. **Machine learning**, Springer, v. 1, n. 1, p. 81–106, 1986.

QUINLAN, R. **C4-5 programs for machine learning**. [S.l.]: Elsevier, 2014. v. 1.

RADOGLOU-GRAMMATIKIS, P. I.; SARIGIANNIDIS, P. G. Securing the smart grid: A comprehensive compilation of intrusion detection and prevention systems. **IEEE Access**, IEEE, v. 7, p. 46595–46620, 2019.

RAMESH, G.; MENEN, A. Automated dynamic approach for detecting ransomware using finite-state machine. **Decision Support Systems**, Elsevier, v. 138, p. 113400, 2020.

RIYAZ, B.; GANAPATHY, S. A deep learning approach for effective intrusion detection in wireless networks using cnn. **Soft Computing**, Springer, v. 24, p. 17265–17278, 2020.

RODRÍGUEZ, G. E. *et al.* Cross-site scripting (xss) attacks and mitigation: A survey. **Computer Networks**, Elsevier, v. 166, p. 106960, 2020.

ROY, S. S. *et al.* A deep learning based artificial neural network approach for intrusion detection. In: SPRINGER. **International Conference on Mathematics and Computing**. [S.l.], 2017. p. 44–53.

RUDER, S. An overview of gradient descent optimization algorithms. arxiv 2016. **arXiv preprint arXiv:1609.04747**, 2016.

RUSSAKOVSKY, O. *et al.* Imagenet large scale visual recognition challenge. **International journal of computer vision**, Springer, v. 115, n. 3, p. 211–252, 2015.

SAHU, S.; MEHTRE, B. M. Network intrusion detection system using j48 decision tree. In: IEEE. **2015 International Conference on Advances in Computing, Communications and Informatics (ICACCI)**. [S.l.], 2015. p. 2023–2026.

SALAKHUTDINOV, R.; HINTON, G. Deep boltzmann machines. In: PMLR. **Artificial intelligence and statistics**. [S.l.], 2009. p. 448–455.

SALAMA, M. A. *et al.* Hybrid intelligent intrusion detection scheme. In: **Soft computing in industrial applications**. [S.l.]: Springer, 2011. p. 293–303.

SALLOUM, S. A. *et al.* Machine learning and deep learning techniques for cybersecurity: A review. In: **AICV**. [S.l.: s.n.], 2020. p. 50–57.

SANDLER, M. *et al.* Mobilenetv2: Inverted residuals and linear bottlenecks. In: **Proceedings of the IEEE conference on computer vision and pattern recognition**. [S.l.: s.n.], 2018. p. 4510–4520.

SARAVANAN, R.; SUJATHA, P. A state of art techniques on machine learning algorithms: a perspective of supervised learning approaches in data classification. In: IEEE. **2018 Second International Conference on Intelligent Computing and Control Systems (ICICCS)**. [S.l.], 2018. p. 945–949.

SARIKAYA, R.; HINTON, G. E.; DEORAS, A. Application of deep belief networks for natural language understanding. **IEEE/ACM Transactions on Audio, Speech, and Language Processing**, IEEE, v. 22, n. 4, p. 778–784, 2014.

SARITAS, M. M.; YASAR, A. Performance analysis of ann and naive bayes classification algorithm for data classification. **International Journal of Intelligent Systems and Applications in Engineering**, v. 7, n. 2, p. 88–91, 2019.

SARKER, I. H. Deep cybersecurity: a comprehensive overview from neural network and deep learning perspective. **SN Computer Science**, Springer, v. 2, n. 3, p. 1–16, 2021.

SCHAPIRE, R. E. The boosting approach to machine learning: An overview. **Nonlinear estimation and classification**, Springer, p. 149–171, 2003.

SERVIN, A.; KUDENKO, D. Multi-agent reinforcement learning for intrusion detection. In: **Adaptive Agents and Multi-Agent Systems III. Adaptation and Multi-Agent Learning**. [S.l.]: Springer, 2005. p. 211–223.

SETHI, K. *et al.* Deep reinforcement learning based intrusion detection system for cloud infrastructure. In: **2020 International Conference on COMmunication Systems NETworkS (COMSNETS)**. [S.l.: s.n.], 2020. p. 1–6.

SHAHRIAR, M. H. *et al.* G-ids: Generative adversarial networks assisted intrusion detection system. In: **2020 IEEE 44th Annual Computers, Software, and Applications Conference (COMPSAC)**. [S.l.: s.n.], 2020. p. 376–385.

SHARAFALDIN, I.; LASHKARI, A. H.; GHORBANI, A. A. Toward generating a new intrusion detection dataset and intrusion traffic characterization. **ICISSp**, v. 1, p. 108–116, 2018.

SHARAFALDIN, I. *et al.* Developing realistic distributed denial of service (ddos) attack dataset and taxonomy. In: IEEE. **2019 International Carnahan Conference on Security Technology (ICCST)**. [S.l.], 2019. p. 1–8.

SHINDE, R. *et al.* An intelligent heart disease prediction system using k-means clustering and naïve bayes algorithm. **International Journal of Computer Science and Information Technologies**, Citeseer, v. 6, n. 1, p. 637–639, 2015.

SHIRAVI, A. *et al.* Toward developing a systematic approach to generate benchmark datasets for intrusion detection. **computers & security**, Elsevier, v. 31, n. 3, p. 357–374, 2012.

SHONE, N. *et al.* A deep learning approach to network intrusion detection. **IEEE transactions on emerging topics in computational intelligence**, IEEE, v. 2, n. 1, p. 41–50, 2018.

SHU, J. *et al.* Collaborative intrusion detection for vanets: a deep learning-based distributed sdn approach. **IEEE Transactions on Intelligent Transportation Systems**, IEEE, v. 22, n. 7, p. 4519–4530, 2020.

SINAGA, K. P.; YANG, M.-S. Unsupervised k-means clustering algorithm. **IEEE Access**, v. 8, p. 80716–80727, 2020.

SMAHA, S. E. *et al.* Haystack: An intrusion detection system. In: ORLANDO, FL, USA. **Fourth Aerospace Computer Security Applications Conference**. [S.l.], 1988. v. 44.

STAŃCZYK, U.; JAIN, L. C. Feature selection for data and pattern recognition: an introduction. In: **Feature Selection for Data and Pattern Recognition**. [S.l.]: Springer, 2015. p. 1–7.

STIAWAN, D. *et al.* Investigating brute force attack patterns in iot network. **Journal of Electrical and Computer Engineering**, Hindawi, v. 2019, 2019.

SU, K.; LI, J.; FU, H. Smart city and the applications. In: IEEE. **2011 international conference on electronics, communications and control (ICECC)**. [S.l.], 2011. p. 1028–1031.

SUN, P. *et al.* Dl-ids: Extracting features using cnn-lstm hybrid network for intrusion detection system. **Security and Communication Networks**, Hindawi, v. 2020, 2020.

SZEGEDY, C. *et al.* Rethinking the inception architecture for computer vision. In: **Proceedings of the IEEE conference on computer vision and pattern recognition**. [S.l.: s.n.], 2016. p. 2818–2826.

TAHERI, L.; KADIR, A. F. A.; LASHKARI, A. H. Extensible android malware detection and family classification using network-flows and api-calls. In: IEEE. **2019 International Carnahan Conference on Security Technology (ICCST)**. [S.l.], 2019. p. 1–8.

TAN, M.; LE, Q. Efficientnet: Rethinking model scaling for convolutional neural networks. In: PMLR. **International conference on machine learning**. [S.l.], 2019. p. 6105–6114.

TAN, M.; LE, Q. Efficientnetv2: Smaller models and faster training. In: PMLR. **International Conference on Machine Learning**. [S.l.], 2021. p. 10096–10106.

TAN, Z. *et al.* A system for denial-of-service attack detection based on multivariate correlation analysis. **IEEE transactions on parallel and distributed systems**, IEEE, v. 25, n. 2, p. 447–456, 2013.

TANDON, G.; CHAN, P. K. Learning useful system call attributes for anomaly detection. In: **FLAIRS Conference**. [S.l.: s.n.], 2005. p. 405–411.

TANG, T. A. *et al.* Intrusion detection in sdn-based networks: Deep recurrent neural network approach. In: **Deep Learning Applications for Cyber Security**. [S.l.]: Springer, 2019. p. 175–195.

TANG, T. A. *et al.* Deep learning approach for network intrusion detection in software defined networking. In: IEEE. **2016 international conference on wireless networks and mobile communications (WINCOM)**. [S.l.], 2016. p. 258–263.

TAVALLAEE, M. *et al.* A detailed analysis of the kdd cup 99 data set. In: IEEE. **2009 IEEE symposium on computational intelligence for security and defense applications**. [S.l.], 2009. p. 1–6.

TEODORO, A. A. *et al.* An analysis of image features extracted by cnns to design classification models for covid-19 and non-covid-19. **Journal of signal processing systems**, Springer, p. 1–13, 2021.

TEODORO, A. A. M. *et al.* An fpga-based performance evaluation of artificial neural network architecture algorithm for iot. **Wireless Personal Communications**, v. 1, p. 1, 2021.

THAMILARASU, G.; CHAWLA, S. Towards deep-learning-driven intrusion detection for the internet of things. **Sensors**, Multidisciplinary Digital Publishing Institute, v. 19, n. 9, p. 1977, 2019.

TIAN, Q. *et al.* An intrusion detection approach based on improved deep belief network. **Applied Intelligence**, Springer, v. 50, n. 10, p. 3162–3178, 2020.

VAN, N. T.; THINH, T. N. *et al.* An anomaly-based network intrusion detection system using deep learning. In: IEEE. **2017 international conference on system science and engineering (ICSSE)**. [S.l.], 2017. p. 210–214.

VARANASI, V. R.; RAZIA, S. Cnn implementation for ids. In: **2021 3rd International Conference on Advances in Computing, Communication Control and Networking (ICAC3N)**. [S.l.: s.n.], 2021. p. 970–975.

VASHIST, A. *et al.* Securing a wireless network-on-chip against jamming-based denial-of-service and eavesdropping attacks. **IEEE Transactions on Very Large Scale Integration (VLSI) Systems**, IEEE, v. 27, n. 12, p. 2781–2791, 2019.

VEMBANDASAMY, K.; SASIPRIYA, R.; DEEPA, E. Heart diseases detection using naive bayes algorithm. **International Journal of Innovative Science, Engineering & Technology**, v. 2, n. 9, p. 441–444, 2015.

VENKATESH, B.; ANURADHA, J. A review of feature selection and its methods. **Cybernetics and information technologies**, v. 19, n. 1, p. 3–26, 2019.

VERMA, A.; RANGA, V. Machine learning based intrusion detection systems for iot applications. **Wireless Personal Communications**, Springer, v. 111, n. 4, p. 2287–2310, 2020.

VIEGAS, E. *et al.* Towards an energy-efficient anomaly-based intrusion detection engine for embedded systems. **IEEE Transactions on Computers**, IEEE, v. 66, n. 1, p. 163–177, 2016.

VIGNA, G.; KEMMERER, R. A. Netstat: A network-based intrusion detection system. **Journal of computer security**, IOS Press, v. 7, n. 1, p. 37–71, 1999.

VOKOROKOS, L.; BALÁŽ, A. Host-based intrusion detection system. In: IEEE. **2010 IEEE 14th International Conference on Intelligent Engineering Systems**. [S.l.], 2010. p. 43–47.

WALL, D. **Cybercrime: The transformation of crime in the information age**. [S.l.]: Polity, 2007. v. 4.

WAN, S. *et al.* A knowledge based machine tool maintenance planning system using case-based reasoning techniques. **Robotics and Computer-Integrated Manufacturing**, Elsevier, v. 58, p. 80–96, 2019.

WANG, J.; SU, X. An improved k-means clustering algorithm. In: **IEEE 3rd International Conference on Communication Software and Networks**. [S.l.: s.n.], 2011. p. 44–46.

WANG, K. *et al.* Haq: Hardware-aware automated quantization with mixed precision. In: **Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition**. [S.l.: s.n.], 2019. p. 8612–8620.

WANG, R. *et al.* Deep learning for anomaly detection. In: **Proceedings of the 13th International Conference on Web Search and Data Mining**. [S.l.: s.n.], 2020. p. 894–896.

WANG, X. *et al.* A novel hybrid mobile malware detection system integrating anomaly detection with misuse detection. In: **Proceedings of the 6th International Workshop on Mobile Cloud Computing and Services**. [S.l.: s.n.], 2015. p. 15–22.

WEISS, K.; KHOSHGOFTAAR, T. M.; WANG, D. A survey of transfer learning. **Journal of Big data**, SpringerOpen, v. 3, n. 1, p. 1–40, 2016.

WINKLER, J. A unix prototype for intrusion and anomaly detection in secure networks. In: **Proceedings of the 13th National Computer Security Conference**. [S.l.: s.n.], 1990. p. 115–124.

WU, K.; CHEN, Z.; LI, W. A novel intrusion detection model for a massive network using convolutional neural networks. **IEEE Access**, IEEE, v. 6, p. 50850–50859, 2018.

WU, T. *et al.* Intrusion detection system combined enhanced random forest with smote algorithm. **EURASIP Journal on Advances in Signal Processing**, SpringerOpen, v. 2022, n. 1, p. 1–20, 2022.

WUNDERLICH, S. *et al.* Comparison of system call representations for intrusion detection. In: SPRINGER. **International Joint Conference: 12th International Conference on Computational Intelligence in Security for Information Systems (CISIS 2019) and 10th International Conference on EUropean Transnational Education (ICEUTE 2019)**. [S.l.], 2019. p. 14–24.

XIN, Y. *et al.* Machine learning and deep learning methods for cybersecurity. **Ieee access**, IEEE, v. 6, p. 35365–35381, 2018.

YANG, L.; MOUBAYED, A.; SHAMI, A. Mth-ids: a multitiered hybrid intrusion detection system for internet of vehicles. **IEEE Internet of Things Journal**, IEEE, v. 9, n. 1, p. 616–632, 2021.

YANG, L.; SHAMI, A. On hyperparameter optimization of machine learning algorithms: Theory and practice. **Neurocomputing**, Elsevier, v. 415, p. 295–316, 2020.

YANG, Y. *et al.* Improving the classification effectiveness of intrusion detection by using improved conditional variational autoencoder and deep neural network. **Sensors**, Multidisciplinary Digital Publishing Institute, v. 19, n. 11, p. 2528, 2019.

YANG, Z.; LI, D. Application of logistic regression with filter in data classification. In: **2019 Chinese Control Conference (CCC)**. [S.l.: s.n.], 2019. p. 3755–3759.

YANG, Z. *et al.* Learning with multiclass auc: Theory and algorithms. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, p. 1–1, 2021.

YAO, R. *et al.* Intrusion detection system in the smart distribution network: A feature engineering based ae-lightgbm approach. **Energy Reports**, Elsevier, v. 7, p. 353–361, 2021.

YIN, C. *et al.* A deep learning approach for intrusion detection using recurrent neural networks. **Ieee Access**, IEEE, v. 5, p. 21954–21961, 2017.

ZHANG, A. *et al.* **Dive Into Deep Learning: Release 0.16.1**. The authors, 2020. Disponível em: <https://books.google.com.br/books?id=TNMTzgEACAAJ>.

ZHANG, C.; PATRAS, P.; HADDADI, H. Deep learning in mobile and wireless networking: A survey. **IEEE Communications Surveys Tutorials**, v. 21, n. 3, p. 2224–2287, 2019.

ZHANG, H. *et al.* An effective convolutional neural network based on smote and gaussian mixture model for intrusion detection in imbalanced dataset. **Computer Networks**, v. 177, p. 107315, 2020. ISSN 1389-1286. Disponível em: <https://www.sciencedirect.com/science/article/pii/S1389128620300712>.

ZHANG, H. *et al.* An effective convolutional neural network based on smote and gaussian mixture model for intrusion detection in imbalanced dataset. **Computer Networks**, Elsevier, v. 177, p. 107315, 2020.

ZHANG, J. *et al.* Intrusion detection system using deep learning for in-vehicle security. **Ad Hoc Networks**, Elsevier, v. 95, p. 101974, 2019.

ZHANG, J. *et al.* Deep learning based attack detection for cyber-physical system cybersecurity: A survey. **IEEE/CAA Journal of Automatica Sinica**, IEEE, v. 9, n. 3, p. 377–391, 2021.

ZHANG, Y. *et al.* Network intrusion detection: Based on deep hierarchical network and original flow data. **IEEE Access**, IEEE, v. 7, p. 37004–37016, 2019.

ZHANG, Z. *et al.* A model based on convolutional neural network for online transaction fraud detection. **Security and Communication Networks**, Hindawi, v. 2018, 2018.

ZHAO, G.; ZHANG, C.; ZHENG, L. Intrusion detection using deep belief network and probabilistic neural network. In: IEEE. **2017 IEEE international conference on computational science and engineering (CSE) and IEEE international conference on embedded and ubiquitous computing (EUC)**. [S.l.], 2017. v. 1, p. 639–642.

ZHOU, L. *et al.* Cyber-attack classification in smart grid via deep neural network. In: **Proceedings of the 2nd International Conference on Computer Science and Application Engineering**. [S.l.: s.n.], 2018. p. 1–5.

ZHOU, Y. *et al.* Soft-root-sign activation function. **ArXiv**, abs/2003.00547, 2020.

ZOU, X. *et al.* Logistic regression model optimization and case analysis. In: **IEEE 7th International Conference on Computer Science and Network Technology (ICCSNT)**. [S.l.: s.n.], 2019. p. 135–139.

APPENDIX A – Dataset Feature Description

The features of the datasets used in this research are network flows captured in both forward and reverse directions. In Table A.1, a detailed description of the features is presented for better understanding. The table contains information about the total packets in both direction, active and idle time, size of packets, statistical features such as the mean, standard deviation and more. These information determine the characteristics of the network flow and defines the class to which it belongs.

Table A.1 – Comprehensive description of the feature names in the datasets used for the IDS model development taken from www.unb.ca

S/No	Feature Name	Description
1	Flow ID	Flow Identifier
2	Source IP	Source IP Address
3	Source Port	The port where the traffic orignates
4	Destination IP	Detsination IP
5	Destination Port	Detsination Port
6	Protocol	Protocol
7	Timestamp	Timestamp
8	Fl Dur	Flow duration
9	Tot Fw Pk	Total packets in the forward direction
10	Tot Bw Pk	Total packets in the backward direction
11	Tot L Fw Pkt	Total size of packet in forward direction
12	Fw Pkt L Max	Maximum size of packet in forward direction
13	Fw Pkt L Min	Minimum size of packet in forward direction
14	Fw Pkt L Avg	Average size of packet in forward direction
15	Fw Pkt L Std	Standard deviation size of packet in forward direction
16	Bw Pkt L Max	Maximum size of packet in backward direction
17	Bw Pkt L Min	Minimum size of packet in backward direction
18	Bw Pkt L Avg	Mean size of packet in backward direction
19	Bw Pkt L Std	Standard deviation size of packet in backward direction
20	Fl Byt S	flow byte rate that is number of packets transferred per second
21	Fl Pkt S	flow packets rate that is number of packets transferred per second

Table A.1 continued from previous page

S/No	Feature Name	Description
22	Fl Iat Avg	Average time between two flows
23	Fl Iat Std	Standard deviation time two flows
24	Fl Iat Max	Maximum time between two flows
25	Fl Iat Min	Minimum time between two flows
26	Fw Iat Tot	Total time between two packets sent in the forward direction
27	Fw Iat Avg	Mean time between two packets sent in the forward direction
28	Fw Iat Std	Standard deviation time between two packets sent in the forward direction
29	Fw Iat Max	Maximum time between two packets sent in the forward direction
30	Fw Iat Min	Minimum time between two packets sent in the forward direction
31	Bw Iat Tot	Total time between two packets sent in the backward direction
32	Bw Iat Avg	Mean time between two packets sent in the backward direction
33	Bw Iat Std	Standard deviation time between two packets sent in the backward direction
34	Bw Iat Max	Maximum time between two packets sent in the backward direction
35	Bw Iat Min	Minimum time between two packets sent in the backward direction
36	Fw Psh Flag	of times the PSH flag was set in packets travelling in the forward direction (0 for UDP)
37	Bw Psh Flag	of times the PSH flag was set in packets travelling in the backward direction (0 for UDP)
38	Fw Urg Flag	of times the URG flag was set in packets travelling in the forward direction (0 for UDP)
39	Bw Urg Flag	of times the URG flag was set in packets travelling in the backward direction (0 for UDP)
40	Fw Hdr Len	Total bytes used for headers in the forward direction
41	Bw Hdr Len	Total bytes used for headers in the forward direction
42	Fw Pkt S	Number of forward packets per second
43	Bw Pkt S	Number of backward packets per second
44	Pkt Len Min	Minimum length of a flow

Table A.1 continued from previous page

S/No	Feature Name	Description
45	Pkt Len Max	Maximum length of a flow
46	Pkt Len Avg	Mean length of a flow
47	Pkt Len Std	Standard deviation length of a flow
48	Pkt Len Va	Minimum inter-arrival time of packet
49	Fin Cnt	Number of packets with FIN
50	Syn Cnt	Number of packets with SYN
51	Rst Cnt	Number of packets with RST
52	Pst Cnt	Number of packets with PUSH
53	Ack Cnt	Number of packets with ACK
54	Urg Cnt	Number of packets with URG
55	Cwe Cnt	Number of packets with CWE
56	Ece Cnt	Number of packets with ECE
57	Down Up Ratio	Download and upload ratio
58	Pkt Size Avg	Average size of packet
59	Fw Seg Avg	Average size observed in the forward direction
60	Bw Seg Avg	Average size observed in the backward direction
61	Fw Byt Blk Avg	Average number of bytes bulk rate in the forward direction
62	Fw Pkt Blk Avg	Average number of packets bulk rate in the forward direction
63	Fw Blk Rate Avg	Average number of bulk rate in the forward direction
64	Bw Byt Blk Avg	Average number of bytes bulk rate in the backward direction
65	Bw Pkt Blk Avg	Average number of packets bulk rate in the backward direction
66	Bw Blk Rate Avg	Average number of bulk rate in the backward direction
67	Subfl Fw Pk	The average number of packets in a sub flow in the forward direction
68	Subfl Fw Byt	The average number of bytes in a sub flow in the forward direction
69	Subfl Bw Pkt	The average number of packets in a sub flow in the backward direction
70	Subfl Bw Byt	The average number of bytes in a sub flow in the backward direction
71	Fw Win Byt	Number of bytes sent in initial window in the forward direction
72	Bw Win Byt	# of bytes sent in initial window in the backward direction

Table A.1 continued from previous page

S/No	Feature Name	Description
73	Fw Act Pkt	# of packets with at least 1 byte of TCP data payload in the forward direction
74	Fw Seg Min	Minimum segment size observed in the forward direction
75	Atv Avg	Mean time a flow was active before becoming idle
76	Atv Std	Standard deviation time a flow was active before becoming idle
77	Atv Max	Maximum time a flow was active before becoming idle
78	Atv Min	Minimum time a flow was active before becoming idle
79	Idl Avg	Mean time a flow was idle before becoming active
80	Idl Std	Standard deviation time a flow was idle before becoming active
81	Idl Max	Maximum time a flow was idle before becoming active
82	Idl Min	Minimum time a flow was idle before becoming active

APPENDIX B – Pre-trained Model Classification Reports

The five pre-trained models used in this research achieved high performance of in classifying the classes in both datasets used as shown in Table A.2. During the data pre-processing, we experimented with two feature transformation techniques *MinMaxScaler* and *QuartileTransformer*. We found that the results obtained with QT outperformed those obtained with MMS. This accounts for the highly performing models. Also, during the model training, the reconstruction error or loss function was greatly minimized with the help of the categorical cross-entropy loss function. In all areas, we obtain a highly efficient lightweight IDS model.

Table A.2 – Performance of the TL models in classifying each class in the dataset

Performance of Each Algorithm on the IDS2018 Image Data in classifying each class												
	Base CNN				**VGG16**				**VGG19**			
Classes	**Acc**	**PR**	**RC**	**FS**	**Acc**	**PR**	**RC**	**FS**	**Acc**	**PR**	**RC**	**FS**
Benign	0.99	0.97	0.98	0.98	0.99	0.96	0.99	0.97	0.99	0.96	0.99	0.97
Bot		1.00	1.00	1.00		1.00	1.00	1.00		1.00	1.00	1.00
Brute Force		1.00	1.00	1.00		1.00	1.00	1.00		1.00	1.00	1.00
DDoS		1.00	1.00	1.00		1.00	1.00	1.00		1.00	1.00	1.00
DoS		1.00	1.00	1.00		1.00	1.00	1.00		1.00	1.00	1.00
Infiltration		0.96	0.93	0.84		0.96	0.91	0.94		0.98	0.90	0.94
Web Attack		1.00	1.00	1.00		1.00	1.00	1.00		1.00	1.00	1.00
	INV3				**MNV3S**				**ENV2B0**			
Classes	**Acc**	**PR**	**RC**	**FS**	**Acc**	**PR**	**RC**	**FS**	**Acc**	**PR**	**RC**	**FS**
Benign	1.00	1.00	1.00	1.00	0.98	0.95	0.98	0.96	0.99	0.97	0.98	0.98
Bot		1.00	1.00	1.00		1.00	1.00	1.00		1.00	1.00	1.00
Brute Force		1.00	1.00	1.00		1.00	1.00	1.00		1.00	1.00	1.00
DDoS		1.00	1.00	1.00		1.00	1.00	1.00		1.00	1.00	1.00
DoS		1.00	1.00	1.00		1.00	1.00	1.00		1.00	1.00	1.00
Infiltration		1.00	1.00	1.00		0.95	0.88	0.91		0.96	0.93	0.94
Web Attack		1.00	1.00	1.00		1.00	1.00	1.00		1.00	1.00	1.00
Performance of Each Algorithm on the IDS2017 Image Data in classifying each class												
	Base CNN				**IV3**				**VGG16**			
Classes	**ACC**	**PR**	**RC**	**FS**	**Acc**	**PR**	**RC**	**FS**	**Acc**	**PR**	**RC**	**FS**
Benign	1.00	0.99	0.99	0.99	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Bot		1.00	1.00	1.00		1.00	1.00	1.00		1.00	1.00	1.00
Brute Force		1.00	1.00	1.00		1.00	1.00	1.00		1.00	1.00	1.00
DDoS		0.99	0.99	0.99		1.00	1.00	1.00		1.00	1.00	1.00
DoS		1.00	1.00	1.00		1.00	1.00	1.00		1.00	1.00	1.00
Heartbleed		1.00	1.00	1.00		1.00	1.00	1.00		1.00	1.00	1.00
Infiltration		1.00	1.00	1.00		1.00	1.00	1.00		1.00	1.00	1.00
PortScan		1.00	1.00	1.00		1.00	1.00	1.00		1.00	1.00	1.00
Web Attacks		1.00	1.00	1.00		1.00	1.00	1.00		1.00	1.00	1.00
	VGG19				**MNV3B0**				**ENV2B0**			
Classes	**Acc**	**PR**	**RC**	**FS**	**Acc**	**PR**	**RC**	**FS**	**Acc**	**PR**	**RC**	**FS**
Benign	1.00	1.00	1.00	1.00	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99
Bot		1.00	1.00	1.00		1.00	1.00	1.00		1.00	1.00	1.00
Brute Force		1.00	1.00	1.00		1.00	1.00	1.00		1.00	1.00	1.00
DDoS		1.00	0.99	0.99		1.00	1.00	1.00		1.00	1.00	1.00
DoS		1.00	1.00	1.00		1.00	1.00	1.00		1.00	1.00	1.00
Heartbleed		1.00	1.00	1.00		1.00	1.00	1.00		1.00	1.00	1.00
Infiltration		1.00	1.00	1.00		1.00	1.00	1.00		1.00	1.00	1.00
PortScan		1.00	1.00	1.00		1.00	1.00	1.00		1.00	1.00	1.00
Web Attacks		1.00	1.00	1.00		0.99	0.99	0.99		0.99	0.99	0.99

Source: Author (2022)