الجــــــــــــمهوريـــة الجــــــــزائرية الديمـــــقراطيـة الشـــــعبيـة

PEOPLE'S DEMOCRATIC REPUBLIC OF ALGERIA

وزارة التــــــــــــعليم العـــــالي والبـــــحث العـــلمـــي

Ministry of Higher Education and Scientific Research

جـــــــامعة أبي بكـــر بـلـقـايد – تـلمســــــــان –

Abou-Bakr BELKAID University – Tlemcen –
Faculty of TECHNOLOGY

MASTER'S THESIS

Presented for the obtention of the **MASTER'S degree**

In : Telecommunications

Specialty : Networks and telecommunications

By : KORTI Sidi Mohammed Mortada

MEDIANI Mehdi Nour Eddine

Subject

Intrusion Detection System Using Machine Learning Techniques

Publicly defended, on 29 / 06 / 2022, before the jury composed of :

Mr. D. MOUSSAOUI	MCA	University of Tlemcen	President
Mr. R. BOUABDALLAH	MAA	University of Tlemcen	Examiner
Mr. M. HADJILA	MCA	University of Tlemcen	Supervisor
Mr. A. BOUIDAINE	PhD student	University of Tlemcen	Co-Supervisor

Academic year : 2021 / 2022

Dedication

I have the opportunity and the pride to dedicate this modest work:

To all my family and especially to my parents and my grandmother for their love, their confidence, their advice and their support that allowed me to realize the studies for which I am destined and consequently this thesis.

To my brother and my sister.

To all my classmates who have become friends over the years with whom I have shared the most pleasant moments throughout my university career.

To all my teachers who have contributed to my education from primary, middle and secondary school to higher education.

Without forgetting all those who, by their knowledge and their actions, have helped me and taught me and have made me what I am.

And finally to all those who support me and whom I love.

Mehdi

Dedication

I have the honor and the pleasure to dedicate this modest work:

To my beloved family who have always been there for me and supported me throughout my studies.

To my mother for her endless love, support and sacrifices, whose words of encouragement and prayers still ring in my ears.

To my father, who has been a constant source of support and comfort, as well as his presence and assistance in times when I have always needed him.

To my brother and my sister who helped me enormously in this work by sharing with me all of their valuable experiences and knowledge for which i will always be grateful.

To my wonderful colleague ***Marwa*** *and her kind soul for playing such an important role throughout my university journey. I am deeply indebted to her for her intellectual support, inspiration, warm presence and generosity.*

To my dear friends with whom I have shared the best and most pleasant moments throughout my university journey and all my colleagues of my class in the department of telecommunications.

To my aunt ***Malika BELKADDAR*** *who has meant and continues to mean so much to me, although she is no longer of this world, her memories continue to be part of my life. May Allah grant her his vast paradise*

And finally to my dearest grandmother ***Mansouria BENYELLES*** *for her prayers and douas...*

Mortada

Acknowledgements

After giving thanks to God the Almighty and the Benevolent, we would like to express our heartfelt gratitude to everyone who participated in the realization of this thesis.

*We would like to thank our supervisor, **Mr. M. HADJILA**, for his availability, rigor, and guidance throughout the elaboration of this thesis.*

Our warmest thanks goes to the jury members for their interest in our research and willingness to evaluate this work.

*We would also like to express our deepest appreciation to our internship supervisor, **Mr. A. BLOUD**, for all of his hard work, pedagogy, and guidance throughout the internship.*

Finally, we would like to thank the entire staff of the university's telecommunications department for their constant support and assistance throughout our studies.

Abstract

Over the last decade, the world has experienced an unprecedented movement of innovation due to the significant development of information and communication technologies, particularly the Internet. Unfortunately, this advancement has been accompanied by an increase in cyber-attacks, and the protection of these communication networks appears to be the next challenge of the upcoming decades.

As the primary defense, the intrusion detection systems have been the subject of numerous researches and play a crucial role in network security. This thesis summarizes the research conducted in the area of network anomaly detection with the goal of developing a model capable of detecting and classifying a wide range of attacks while also adapting to a constantly changing threat scenario.

The proposed approach has been tested on the public database CIC-IDS2017. The database will first be pre-processed and normalized and then applied to various classification machine learning algorithms to create models and compare their performance using different evaluation metrics such as (Accuracy, Precision, Recall, F1-score, etc.). The experimental results have shown that the performance of the machine learning algorithms used resulted in a relatively high accuracy score: Random Forest 97.02%, Decision Tree 96.74%, K-Nearest Neighbors 96.24%, MLP 87.57% and SVM 81.12%.

Keywords: Cybersecurity, Cyber-attacks, Intrusion Detection System (IDS), Deep Learning, Machine Learning, Network Anomaly Detection, CIC-IDS2017.

Résumé

Avec l'important développement des technologies de l'information et de la communication et particulièrement Internet, le monde a connu durant la dernière décennie un mouvement d'innovations sans précédent. Malheureusement ces avancées furent accompagnées par la progression des cybers attaques, la protection de ces réseaux de communication apparait comme le prochain défi des futures décennies.

En tant que premier rempart, les systèmes de détection d'intrusion ont fait l'objet de nombreuses recherches et jouent un rôle crucial dans la sécurité des réseaux. Ce mémoire présente les travaux menés dans le cadre du domaine de la détection d'anomalies dans les réseaux avec le but de développer un modèle capable de détecter et de classifier un large éventail d'attaques tout en s'adaptant à un scénario de menace en constante évolution.

L'approche proposée a été testée sur la base de données publique CIC-IDS2017. La base de données sera d'abord prétraitée et normalisée, puis appliquée à divers algorithmes d'apprentissage automatique de classification pour créer des modèles et comparer leurs performances à l'aide de différentes mesures d'évaluation telles que (Accuracy, Precision, Recall, F1-score, etc.). Les résultats expérimentaux ont montré que les performances des algorithmes d'apprentissage automatique utilisés ont atteint un score de précision relativement élevé : Random Forest 97.02%, Decision Tree 96.74%, K-Nearest Neighbors 96.24%, MLP 87.57% et SVM 81.12%.

Mots clés: Cybersécurité, Cyber-attaques, Système de détection d'intrusion (IDS), L'apprentissage profond, L'apprentissage automatique, Détection d'anomalies dans les réseaux, CIC-IDS2017.

Table of contents

List of Figures

List of Tables

Acronyms

AEs AutoEncoders

AI Artificial Intelligence

ANNs Artificial Neural Networks

CIC Canadian Institute for Cybersecurity

CNNs Convolutional Neural Networks

CPU Central Processing Unit

DDoS Distributed Denial of Service

DIDS Distributed Intrusion Detection System

DL Deep Learning

DoS Denial of Service

FN False Negative

FNR False Negative Rate

FP False Positive

FPR False Positive Rate

FTP File Transfer Protocol

GBT Gradient Boosted Tree

GPU Graphics Processing Unit

HIDS Host-based Intrusion Detection System

HTTP Hypertext Transfer Protocol

ICMP	Internet Control Message Protocol
IDS	Intrusion Detection System
INF	Infinity
IP	Internet Protocol
IRC	Internet Relay Chat
KNN	K-Nearest Neighbors
LSTM	Long Short Term Memory
MITM	Man In The Middle
ML	Machine Learning
MLP	Multi-layer Perceptron
NAC	Network Access Control
NaN	Not A Number
NIDS	Network-based Intrusion Detection System
NNs	Neural Networks
OS	Operating System
P2P	Peer-to-Peer
R2L	Remote to Local
R2U	Remote to User
RAM	Random-Access Memory
ReLU	Rectified Linear Unit
RNNs	Recurrent Neural Networks
ROC	Receiver Operating Characteristic
SNMP	Simple Network Management Protocol
SSD	Solid State Drives

SSH	Secure Shell
SVM	Support Vector Machine
Tanh	Hyperbolic tangent
TCP	Transmission Control Protocol
TN	True Negative
TNR	True Negative Rate
TP	True Positive
TPR	True Positive Rate
U2R	User to Root
UDP	User Datagram Protocol
XSS	Cross Site Scripting

General introduction

With the spectacular progress of information and communication technologies, as well as the expansion of computer networks, the world has evolved rapidly over the last decade towards a fully connected lifestyle for all economic and social actors. This evolution has provided us with inescapable facilities in terms of long-distance communication (instant messaging and video calls), file transmission, online shopping (e-commerce and online banking), social networking and many other forms of information exchange. However, the widespread use of these information technology applications has opened the door to new threats and vulnerabilities.

Due to the massive amount of data flowing through the networks and the uncontrolled structure of the Internet, the preservation of sensitive information and communications has emerged as a challenge for cybersecurity. A study by Arkose Labs [1] estimated that in August 2020, approximately 445 million cyberattacks took place worldwide, twice as many as in 2019. This gives rise to an ongoing struggle against these cyber attacks that target vulnerabilities and weaknesses in systems, orchestrated by perpetrators with the intent to either steal sensitive information, commit fraud, espionage, hijacking and many other malicious activities.

Cybersecurity is the set of techniques and practices that allow to maintain the confidentiality, integrity and availability of information. Faced with the daily evolution of attacks and the increasing interactivity of hackers, it is essential to have a precise action plan to identify the elements at risk. Intrusion Detection Systems (IDS) are a solution that meets these requirements, adopted to prevent any imminent threat of violation of security policies, networks and computer systems.

The purpose of an intrusion detection system (IDS) is to monitor network traffic in order to detect any misuse or abnormal behavior. Two types of IDS have been proposed, which are network-based systems (NIDS) and host-based systems (HIDS). The first ones try to detect any attempt to subvert the normal behavior of the system by analyzing the network traffic, while the second ones try to detect intrusions by analyzing the events on the local system where the IDS is running.

Machine Learning techniques have been well adopted as the primary detection algorithm in IDSs owing to their model-free properties and learning capacity. Leveraging the recent development of Machine Learning techniques such as Deep Learning is expected to significantly improve existing IDSs, especially for the detection of impersonation attacks in large-scale networks.

In this study, we explore several Machine Learning and Deep Learning approaches that have proven their reliability in the field of intrusion detection. Our thesis is organized as follows:

- The first chapter is devoted to the presentation of the network security and its various forms, we will also tackle the different types of anomalies as well as the most widespread attacks.
- The second chapter will serve as an introduction to intrusion detection systems, their types, functions and classifications.
- The third chapter is reserved for the presentation of Deep Learning, its history, its recurrent models and their architectures.
- The fourth chapter is related to the discussion of the results obtained while detailing the methodology used and the parameters taken into account.

Chapter 1

Network Security

1.1 Definition

Network security can be defined as the process of designing a defensive strategy and implementing the measures and safeguards required to protect the underlying networking infrastructure from unauthorized access, malicious activity, potential threats or intrusion attempts. As a result, a secure platform is created so that users can perform their tasks in a safe environment.

1.2 Key principles of network security

Computer and network security are built on three pillars (Confidentiality, Integrity, and Availability), commonly referred to by the "CIA triad" [2]. A network can only be considered secure when it has all three elements in play simultaneously.

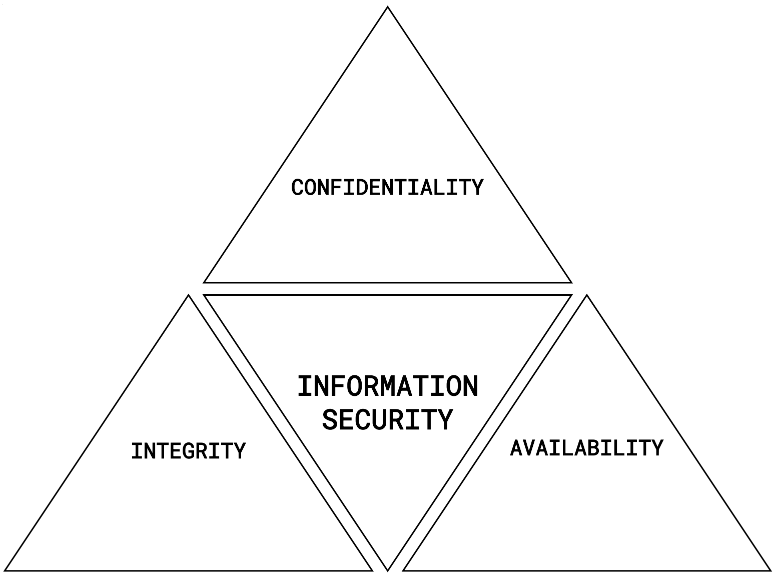


Figure 1.1: CIA triad diagram.

- **Confidentiality:** is about keeping the content of information secret and inaccessible to unauthorized entities, and ensuring that only approved individuals have access to sensitive information.

- **Integrity:** helps maintain the trustworthiness (reliability) of data by having it in its intended state and immune to any improper modifications from unauthorized people or malicious software.

- **Availability:** ensures that authorized parties have unimpeded access to data when needed.

In addition to CIA, another set of protections must be implemented to secure information. These are authentication, authorization, and accounting (AAA) [3]:

- **Authentication:** is the process of verifying a user's identity. A user proves their identity by providing their credentials, which are then matched against a file stored in an authorized user database or data authentication server, thus preventing unauthorized access to a device, system or network.
- **Authorization:** following authentication, authorization is the process of determining what types or qualities of activities, resources or services a user is permitted.
- **Accounting:** consist of monitoring and recording user activity while accessing network resources. This can include the amount of system time or the amount of data sent and received during a session, as well as holding the individual accountable for their actions.

1.3 Different types of network security

There are several types of network security, the most common of which are covered below:

1.3.1 Network Access Control (NAC)

A network design that restricts network resources and infrastructure access to only compliant, authenticated, and trusted endpoint devices, denying unauthorized access and potential threats. This is achieved by deploying a password, unique user ID and authentication process to access the network.

1.3.2 Network segmentation

An architectural technique that divides a network into multiple segments, each of which functions as its own tiny network (see Figure 1.2).
This allows network administrators and organizations to :

- Control the flow of traffic across segments.
- Improve network monitoring and performance.
- Prevent malware from spreading by isolating a network in one area, while keeping another segment of the network protected.

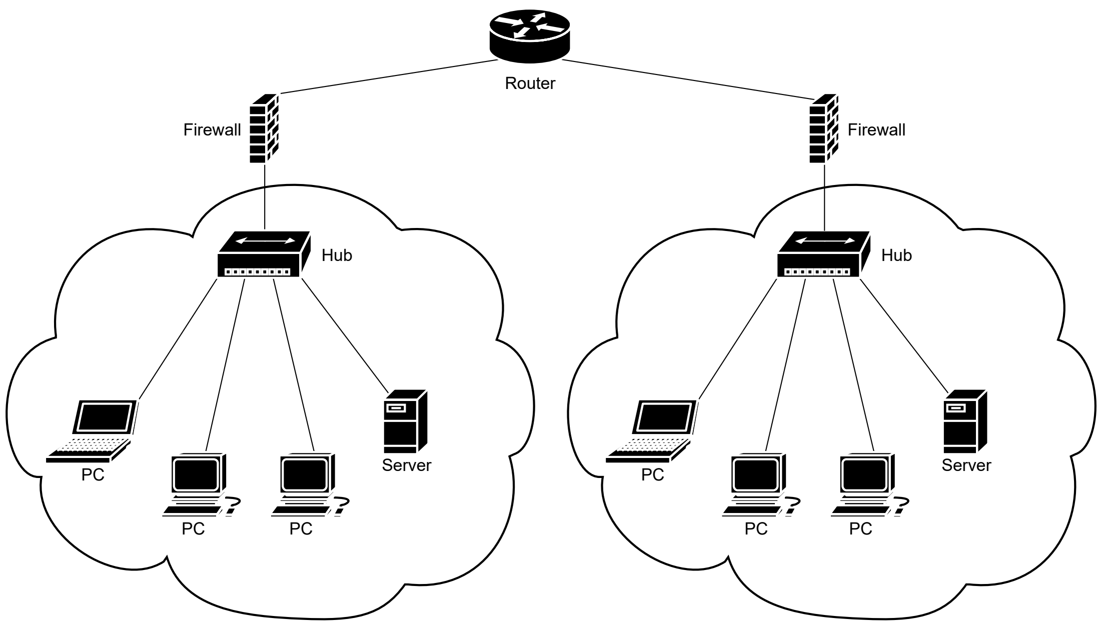


Figure 1.2: Segmented network architecture.

1.3.3 Behavioral analytics

An advanced threat detection technique that compares prior network activity data to current events in order to discover anomalous behavior.

1.3.4 Firewalls

A combination of hardware appliances and software programs that act as a barrier between a trusted internal network and the wider internet. They filter incoming and outgoing traffic based on predefined security rules, preventing threats from gaining access to the network.

1.3.5 Antivirus and Antimalware software

Software used to protect computers and workstations from malware and viruses by identifying and eliminating harmful programs that have infiltrated the system and proactively preventing endpoint devices infection.

1.4 Anomaly detection and attack types

1.4.1 Anomaly types

A sample that does not exhibit the well-defined properties of a normal sample is considered an anomaly (or outlier) [4]. There are three generally accepted categories which all anomalies fall into:

1.4.1.1 Point anomaly

A point anomaly is defined as a single data point that is unusual compared to the rest of the data [5].

1.4.1.2 Contextual anomaly

Also called conditional anomaly, it contains data points that deviate significantly from other data points that exist in the same context [5]. This means that observing the same point across different contexts will not always give us an indication of anomalous behavior.

1.4.1.3 Collective anomaly

A collective anomaly is a collection of similar data points that, when compared to the rest of the data, can be considered anomalous [5]. While each of the individual data instances in a collective anomaly may not be anomalies by themselves, their collective occurrence is anomalous.

1.4.2 Network attack types

Network attacks are attempts to violate the 3 essential features (Confidentiality, Integrity, Availability). The attacks can be summarized under 4 headings.

1.4.2.1 Denial of service/Distributed denial of service

Denial of Service (DoS) is a type of cyberattack in which attackers prevent legitimate users from accessing authorized data. Attackers accomplish this by flooding the network with fake traffic (excessive number of unnecessary packets) that exceed the server's capacity, resulting in a denial of service to additional requests [6].

A Distributed Denial of Service (DDoS) is a type of DoS attack where the traffic used to overwhelm the target is coming from many distributed sources.

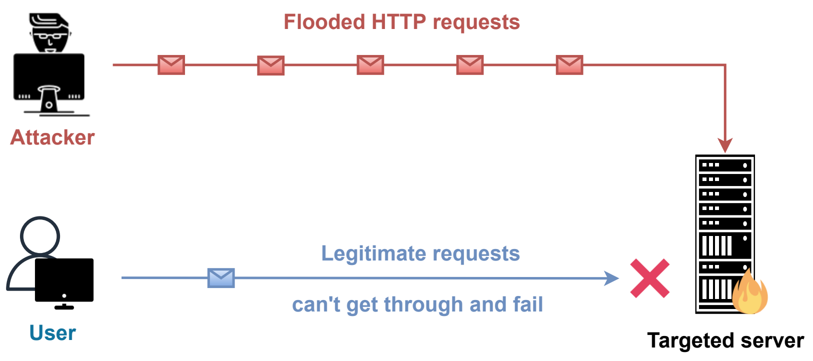


Figure 1.3: DoS attack.

1.4.2.2 Probe (Reconnaissance)

Probe is usually a preliminary step toward a further attack seeking to compromise a host or network. Its primary goal is to gather all possible information about the target, such as the network structure, the operating system in use, the types of software installed and/or applications used. It is commonly used by attackers to identify breaches or weaknesses that will eventually allow them to bypass an organization's security [5].

1.4.2.3 User to Root (User to Root (U2R))

In this type of attack, the attacker attempts to gain unauthorized access to an administrative account in order to access or alter valuable resources. The attacker gains access to a normal user account before progressing to the root account by exploiting system vulnerabilities [7].

1.4.2.4 Remote to User (R2U) / Remote to Local (R2L) (Remote to User / Remote to Local)

The attacker gains local access as a user of a targeted machine to have the privilege of sending packets over its network. To acquire this privilege, the attacker relies on system vulnerabilities or brute-force attacks [5].

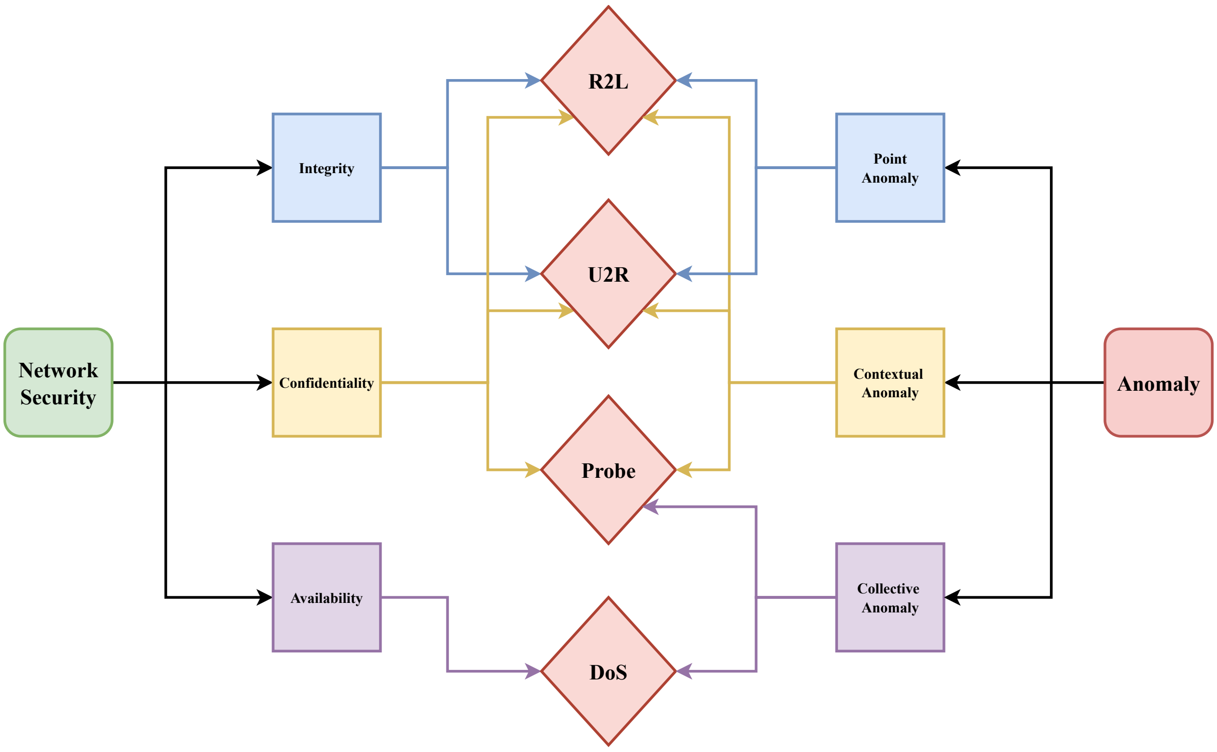


Figure 1.4: The relationship between network anomalies and network attacks [8].

Classifying network attacks based on the anomalies they cause can be helpful in detecting them. DoS attacks, for example, have been found to increase the amount of data flowing and the number of packets in the network. They are therefore classified as collective anomalies. Conversely, it would be more suitable to classify U2R and R2U as contextual and point anomalies since the attack is targeted at a particular user, a certain port, and has a specific purpose [8]. The relationship between network anomalies and network attacks is summarized in Figure 1.4.

1.5 Most common attacks

1.5.1 DoS HULK

HTTP unbearable load king is a DoS attack that is designed to repeatedly generate numerous TCP SYN flood and multi- threaded HTTP GET flood requests that will create a load on a web server, thus exhausting the web server's resources [9].

1.5.1.1 TCP-SYN flood

The TCP-SYN flood attack attempts to exploit the three-way handshake method by flooding the server with SYN requests. The concept of the three-way handshake is explained as follows:
A TCP connection is established via a three-way handshake, in which a client and server communicate via TCP/IP, with flags set on the TCP layer of a packet. A TCP flag is a series of bits that indicate how a packet should be handled by the server. This communication takes place in the following steps [10]:

1. The client initiates the connection to the server by sending a packet with the SYN (synchronize) flag.

2. The server responds to the client by sending a TCP packet with the SYN and ACK (acknowledge) flags set, as if to acknowledge the client's connection request.

3. If the connection is refused due to a closed port, the server will respond with a TCP packet with the RST (reset) flag set.

4. In case the port is open, the client will respond to the server with an ACK packet, completing the TCP connection.

In the TCP-SYN flood attack, the attacker sends the SYN packet to the target (server) and receives the SYN-ACK packet in response. The attacker, however, fails to send the required ACK packet, resulting in an incomplete connection (see Figure 1.5). The server then reserves a resource by storing these requests in a log queue. As the number of requests increases, the server becomes inaccessible [8].

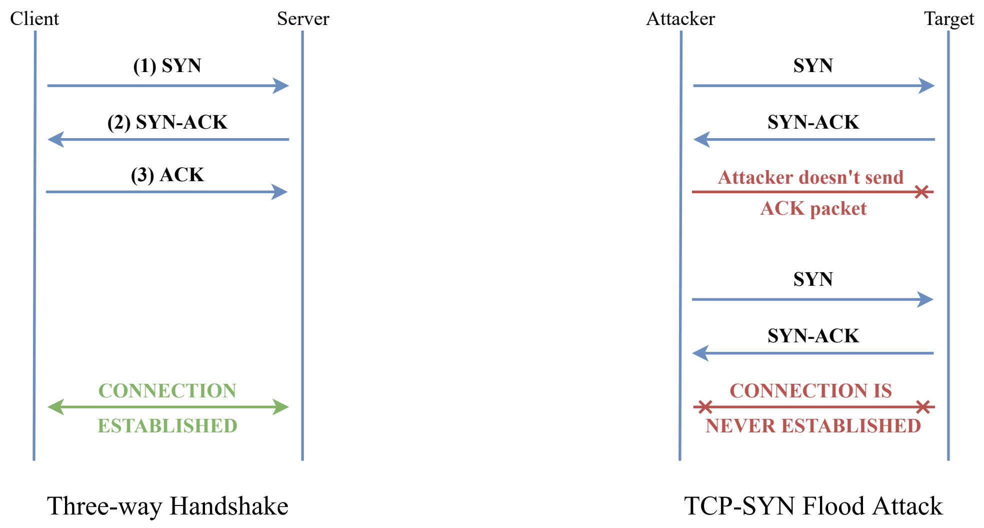


Figure 1.5: A comparison of TCP-SYN Flood Attack with a successful Three-Way Handshake.

1.5.1.2 HTTP-GET flood

An HTTP flood is a type of DDoS attack used by hackers to attack web servers and applications. It involves flooding the server with HTTP GET or POST requests that are specifically designed to consume a large portion of the server's resources, resulting in a denial of service, meaning that the server no longer responds to legitimate HTTP GET requests [11].

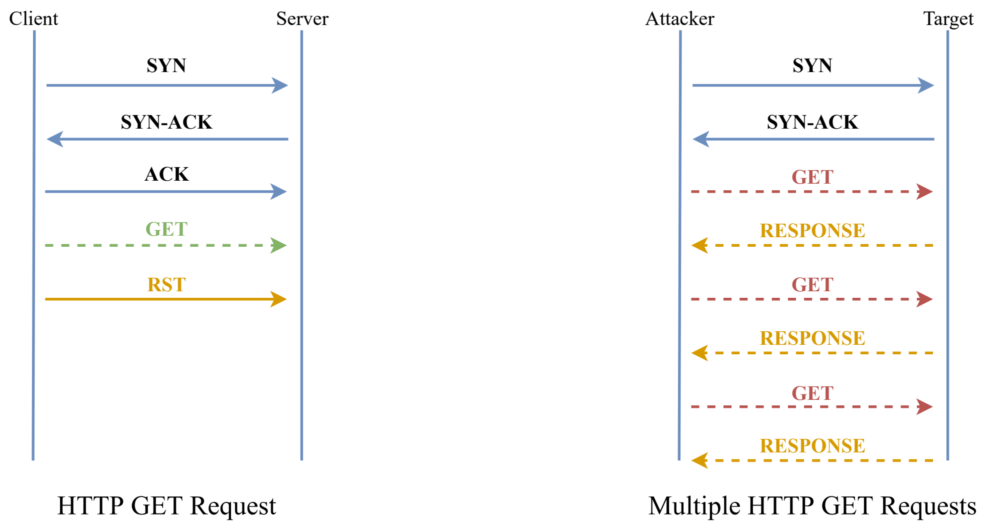


Figure 1.6: The comparison of singular and multiple HTTP GET requests in HTTP GET Flood attack.

1.5.2 DoS Slowloris

Slowloris is an application layer DoS attack that connects a single computer to a targeted web server using incomplete HTTP requests, then keeps these connections active for as long as possible. Furthermore, by leaving these connections open while waiting for the completion of each attack request, the affected servers will eventually fill their maximum simultaneous connection pool, denying any further connection attempts from clients. This type of DoS attack requires minimal bandwidth to launch and only affects the target web server, leaving other services and ports unaffected [12].

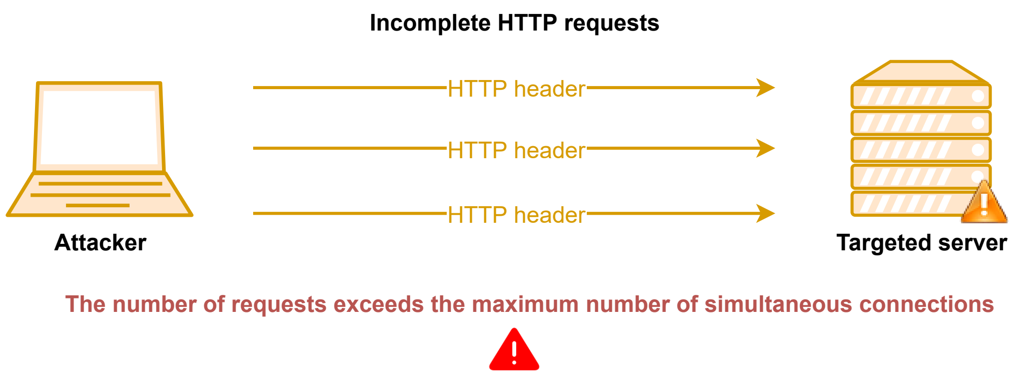


Figure 1.7: Slowloris DoS attack.

1.5.3 Botnet

Botnet is a large group of internet-connected computers, often referred to as ”zombies”, that have been infected with malicious program(s) and are controlled remotely by botnet owners, also known as ”herders”. The herder commands the botnet through a command-and-control server that communicates via protocols such as Internet Relay Chat (IRC) or peer-to-peer (P2P) networking. These compromised devices are used by attackers to launch large-scale attacks without their owners' knowledge. Botnet attacks typically involve sending spam, data theft, exploiting sensitive information, or launching vicious DDoS attacks [13].

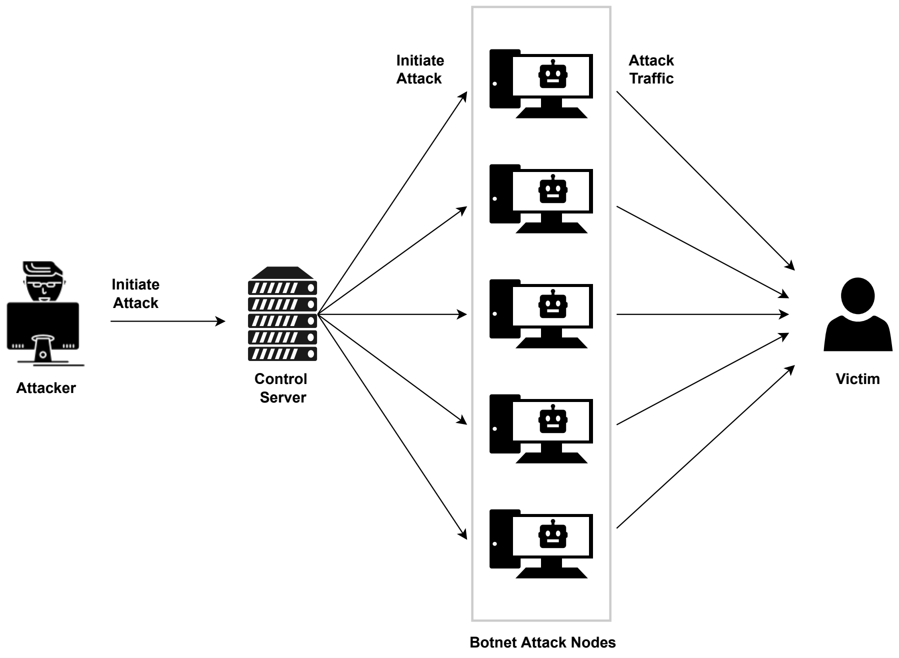


Figure 1.8: A typical Botnet attack structure.

1.5.4 Malware

Malware refers to any malicious software that is created with the intent of harming or exploiting any programmable device, service, or network. It is most commonly used by cybercriminals to extract data from computer systems without the user's knowledge or consent. This data can include anything from financial information to medical records to personal emails and passwords, etc. Examples of common malware includes viruses, worms, Trojan viruses, spyware, adware, and ransomware [14].

1.5.5 Port scanning

Port scanning is often the first reconnaissance step used by hackers when attempting to infiltrate a network. It involves sending packets to specific ports on a host and using the responses to determine which applications and services the target device is running. The hacker can then begin testing for vulnerabilities and plan an attack [15].
The port scanners are mainly of two types:

- **Brute force scanners** establish a full connection to the target machine after scanning each port one by one for the specific range and determining whether or not the port is open. Therefore, their presence can be easily detected due to the scanner's numerous attempts to establish a connection to many ports on a target host in a short period of time [16].

- **Stealth scanners** perform scans in a stealthy manner without establishing a full connection with the target. They send a single packet with a specific flag set to the target, and based on the response, they can determine whether the ports are open or not [16].

Port scanning types can be listed as follows:

1.5.5.1 SYN scan

In this attack, the attacker tries to establish a TCP/IP connection with a target server by sending a large number of packets with only the SYN flag to every possible port on the server. If the server responds with a SYN/ACK packet from a particular port, it indicates that the port is open. The attacker then sends an RST packet, tearing down the connection and tricking the server into believing there was a communication failure and the client did not establish a connection, when in fact the port remains open and vulnerable to exploitation [17].

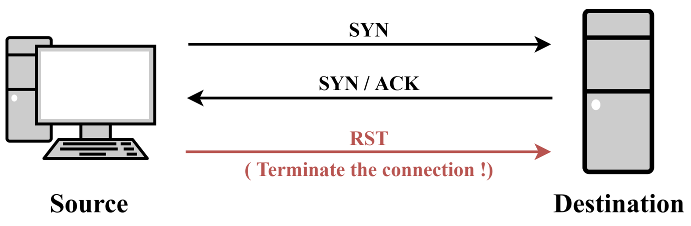


Figure 1.9: SYN scan.

1.5.5.2 TCP Connect scan

In this scan, the first two steps (SYN and SYN/ACK) are exactly the same as for a SYN scan. Before terminating the connection with a RST packet, the attacker first establishes a connection with the target by acknowledging the SYN/ACK with his own ACK packet. A TCP Connect scan can be inferred if a large number of connections are established from a single host at multiple ports in a very short period of time [16].

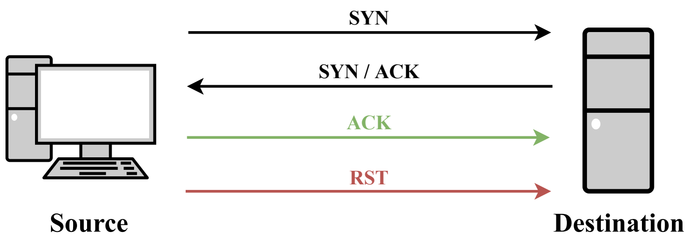


Figure 1.10: TCP Connect scan.

1.5.5.3 ACK scan

This scan is used to detect the presence of a firewall on the target host or between the target machine and the scan machine. Indeed, contrary to other scans, the ACK scan will not seek to find out which port is open on the target machine, but rather to know if a filtering system is active by replying for each port with "filtered" or "unfiltered" [18].

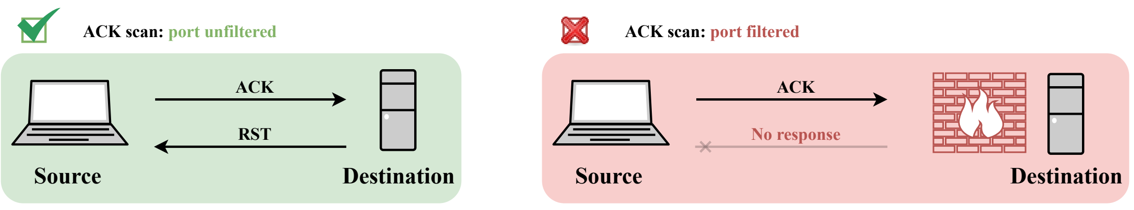


Figure 1.11: ACK scan.

1.5.5.4 FIN scan

During this scan, the attacker sends a large number of packets with only the FIN flag set to different ports on the target host. Sending a FIN packet without any prior exchange between the two hosts confuses the target since it indicates that the attacker wants to terminate a connection that has never been established. If the port is open, the target ignores the packet; otherwise, the target sends a RST packet [16].

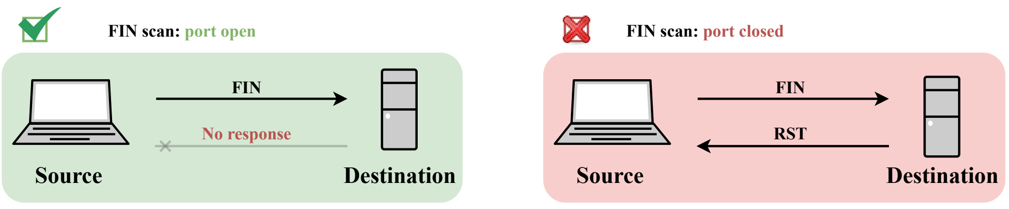


Figure 1.12: FIN scan.

1.5.5.5 NULL scan

In this attack, the attacker sends many TCP packets with no flags to different ports of the victim machine. The open ports ignore these packets whereas closed ports reply back with a RST packet. This scan can be easily detected because sending a packet without a flag is a behavior that will never be seen in a normal machine-to-machine exchange. Some firewalls or filtering modules may malfunction as a result of this scan, allowing packets to pass through [18].

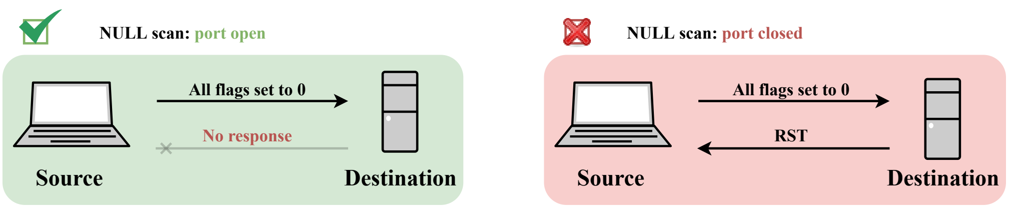


Figure 1.13: NULL scan.

1.5.5.6 XMAS scan

Unlike the FIN scan, the XMAS scan sends TCP packets with the URG, PUSH and FIN flags set to 1 in order to bypass certain firewalls or filtering systems. It's important to know that when sending a packet with these three flags set, an active service behind the targeted port won't send back any packets. However, if the port is closed, we will receive a RST/ACK packet [18].

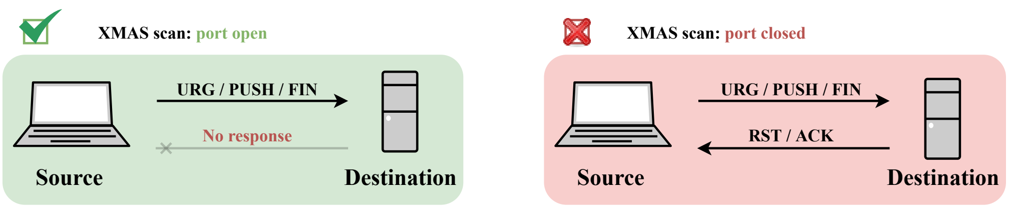


Figure 1.14: XMAS scan.

1.5.5.7 UDP scan

Scanning UDP ports is not difficult to implement due to the simplicity of exchanges between machines using the UDP transport protocol. In this scan a large number of UDP packets arrive at the destination. If the port is open, the targeted server will provide no feedback to our scanning machine. If no application is ready to receive our packet on the targeted UDP port, an ICMP packet with the corresponding error code, i.e. the "ICMP_Port_Unreach_Error" message, is sent back [19].

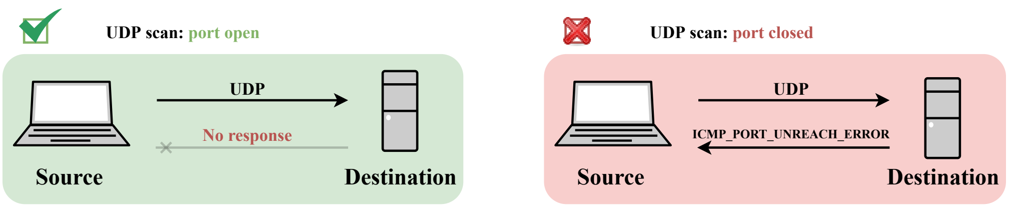


Figure 1.15: UDP scan.

1.5.5.8 IP fragmentation attack

IP fragmentation is the process of splitting a datagram into smaller pieces of information called packets. These must be of a specific size in order for the receiving parties to be able to process them and transfer the data successfully. There are many forms of IP fragmentation attacks. They generally involve sending datagrams that cannot be reassembled upon delivery. The objective is to exploit the server's resources and prevent it from performing its intended functions [20].

1.5.6 FTP-Patator

FTP (File Transfer Protocol) is a network protocol for transferring files between computers over (TCP/IP) connections. To transfer files via FTP, users must have permission by providing credentials (a valid username and password) to the FTP server [21]. The FTP-Patator attack is a brute force attack that focuses on stealing a user's credentials through the FTP protocol.

A brute-force attack is a trial-and-error method that hackers use to decode login information and encryption keys in order to gain unauthorized access to systems. Some attackers use automated tools to guess all possible passwords until the correct combination is found [22].

1.5.7 SSH-Patator

SSH, also known as Secure Shell or Secure Socket Shell, is a cryptographic network protocol that allows users to have secure communication over an insecure network (e.g., The internet) by providing strong password and public key authentication, as well as an encrypted session for transferring files and executing server programs. The protocol is primarily used by network administrators to manage systems and applications remotely, enabling them to log in to another computer over a network, execute commands and transfer files from one computer to another [23]. Similar to FTP-Patator, by trying many combinations of usernames and passwords, a successful brute force SSH attack allows hackers to gain remote access to target systems. The goal of these attacks is to obtain personal information from the user that can be used to access their online accounts and network resources [24].

1.5.8 Web attacks

Every website on the internet is somewhat vulnerable to security attacks. Threats range from human error to sophisticated attacks by coordinated cybercriminals. The most frequent web attacks are the following:

1.5.8.1 XSS

Cross-site scripting (XSS) is a type of injection attack in which an attacker injects malicious code (usually in the form of a browser-side script) into otherwise legitimate and secure websites. The actual attack takes place when the victim visits the affected web page, which delivers the malicious script to the victim's browser, which in turn executes it since it has no way of knowing that the script is untrustworthy. Instead of targeting the web application itself, XSS attacks typically target the application's users directly by stealing cookies, session tokens or other sensitive information stored by the victim's browser, allowing cybercriminals to impersonate real users and use their accounts [25].

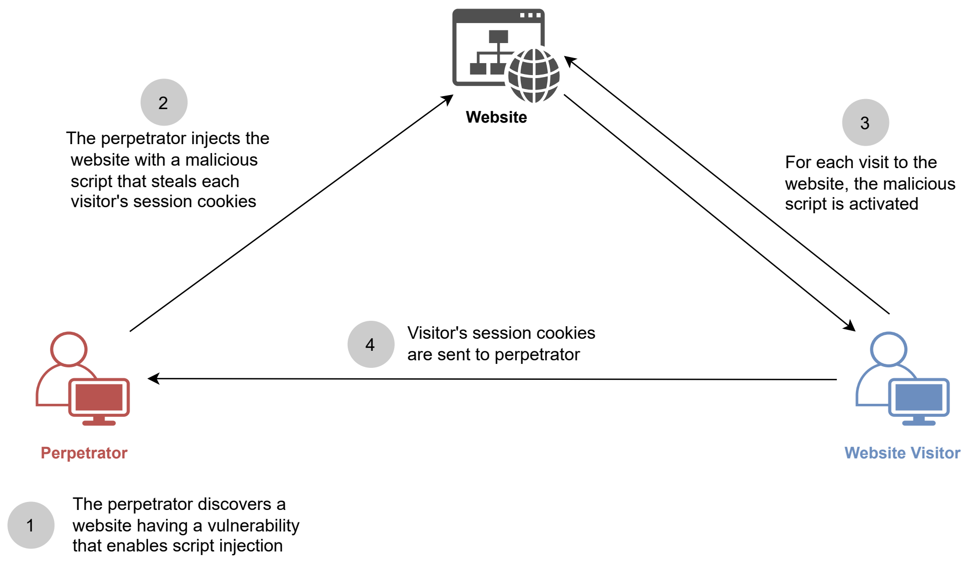

Website
2
The perpetrator injects the website with a malicious script that steals each visitor's session cookies
3
For each visit to the website, the malicious script is activated
4
Visitor's session cookies are sent to perpetrator
Perpetrator
Website Visitor
1
The perpetrator discovers a website having a vulnerability that enables script injection

Figure 1.16: XSS attack.

1.5.8.2 SQL Injection

Structured Query Language (SQL) is a standardized programming language designed to handle relational databases and manipulate various operations on the data they store [26]. An SQL injection vulnerability can affect any website or web application that uses an SQL database such as MySQL, Oracle, SQL Server, or others. Attackers use the SQL injection attack to target these databases using SQL statements specifically designed to trick systems and bypass application security measures, allowing them to retrieve the contents of the entire SQL database, such as passwords, credit card details, or user personal information, as well as to add, modify, and delete records in the database [27].

1.5.8.3 Man in the middle (MITM)

A common type of cyberattack in which attackers insert themselves between two parties in order to eavesdrop or impersonate one of the parties, allowing them to intercept information and data from either party while making it appear as if a regular exchange of information is taking place.

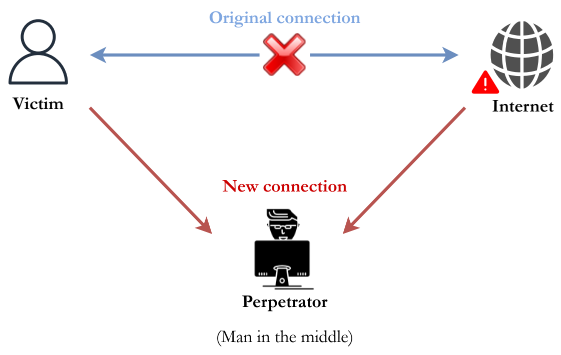


Figure 1.17: Man in the middle attack.

1.5.9 IP spoofing

An intruder uses IP spoofing to trick a system into thinking it is communicating with a known, trusted entity by sending a packet to a target host with the IP source of a known, trusted host rather than it's own IP source address for the purpose of gaining access to the system [28].

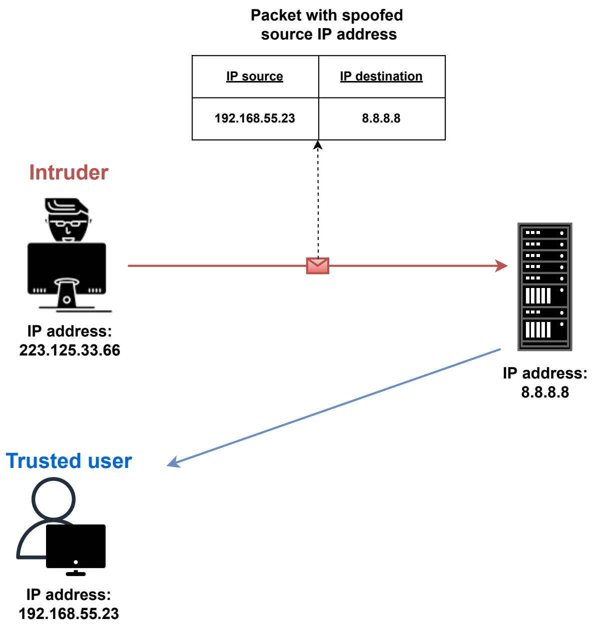

Packet with spoofed source IP address
IP source
IP destination
192.168.55.23
8.8.8.8
Intruder
IP address: 223.125.33.66
IP address: 8.8.8.8
Trusted user
IP address: 192.168.55.23

Figure 1.18: IP spoofing.

1.6 Conclusion

In this first chapter we have defined what network security is, its fundamentals which are confidentiality, integrity and availability and its different types. We have also presented some of the most common attacks that a network can face. Despite the effectiveness of these various network security techniques, a network is never completely safe from intrusion. In the next chapter, we will focus on a solution to this problem by presenting intrusion detection systems.

Chapter 2

Intrusion Detection System

2.1 Introduction

With the emergence of a limitless communication paradigm and the growing number of networked digital devices, cybersecurity, which aims to protect information or communication technologies, is of increasing concern due to the extraordinary growth in the volume and variety of security threats to these systems [29].
Any intrusion can have disastrous consequences. For example, personal data can be damaged, altered or illegally accessed. Therefore, detection and prevention of network abuses is becoming more and more strategic to ensure an adequate level of protection against external and internal threats. In this context, many techniques are emerging to monitor network traffic and distinguish abnormal from normal behavior to detect undesired or suspicious activities. One promising tool for detecting attacks is the Intrusion Detection System (IDS) [30].

2.2 Definition

IDS is a monitoring system that listens to network traffic in a stealthy manner in order to alert the administrator to security breaches, abnormal or suspicious activities or other problems that could compromise the computer network, thus enabling preventive action to be taken against intrusion threats.

2.3 Components of intrusion detection systems

In general, an IDS contain three main components as depicted in Figure 2.1:

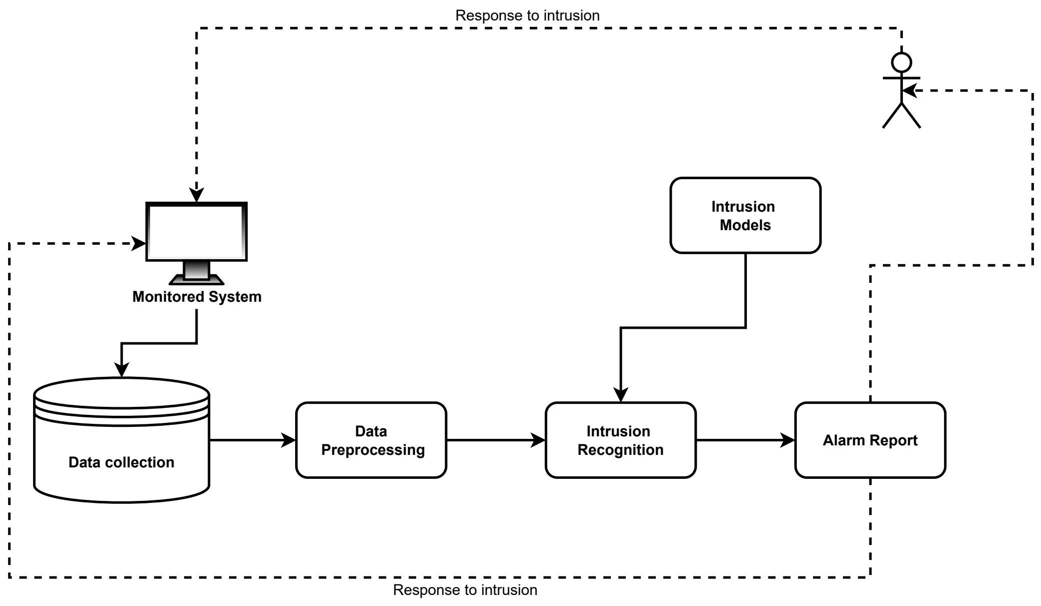


Figure 2.1: The main function of an IDS.

2.3.1 Data collection

The sensor observes the activity of the network through the information they receive from a data source, usually a pre-processing is done before sending a sequence of events to be analyzed, these sequences inform on the evolution of the state of the system and its activity.

2.3.2 Data pre-processor

The data is processed by a data pre-processor to extract activity records that are important for security analysis.

2.3.3 Intrusion recognition

Its objective is to analyze the activity data using intrusion models that have already been established for the IDS. If the detection rule determines that there is an intrusion, the IDS produces an alert and the decision engine then decides the appropriate action according to the decision table. This can be a response that automatically blocks a network connection or a report sent to the security manager.

2.4 IDS functions

The main functions of an IDS can be summarized in the following points:

- Compares the collected data with baseline data to detect intrusions.
- Logs attacks and save alert details (e.g. @IP of the intruder). This allows for remediation of security breaches to prevent recorded attacks from occurring again.
- Sends an alert in SNMP format to a third-party hypervisor in case of an attack.
- Sends an email or visual notification to one or more supervisors to notify of a serious attack.
- Applies corrective measures when an intrusion is detected.

2.5 IDS Classification

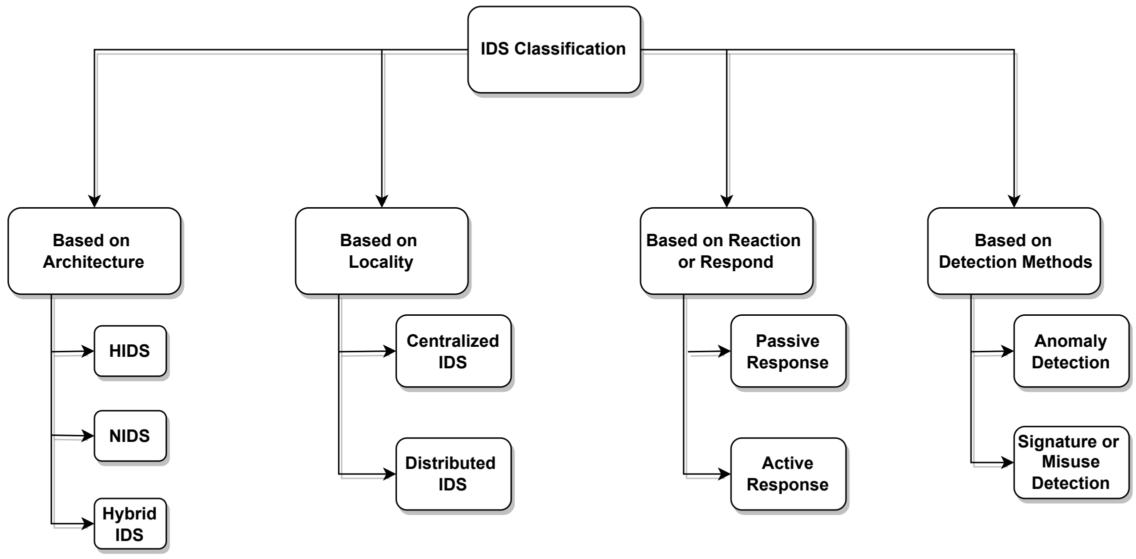


Figure 2.2: IDS Classification.

2.5.1 Types of IDS

Based on the location and methodology of the IDS module deployed in the network, we can distinguish three classes of IDS [6]:

2.5.1.1 Network-based IDS

The Network-based Intrusion Detection System (NIDS) is located on a separate machine that continuously checks for malicious activities by inspecting and analyzing the stream of packets flowing across the network, blocking attacks if necessary, and creating reports [31].

2.5.1.2 Host-based IDS

A Host-based Intrusion Detection System (HIDS) runs on a single machine and monitors the computer infrastructure on which it is installed, analyzing all inbound and outbound traffics of the corresponding client and tracking changes made to registry settings and critical system configuration, log and content files [6].

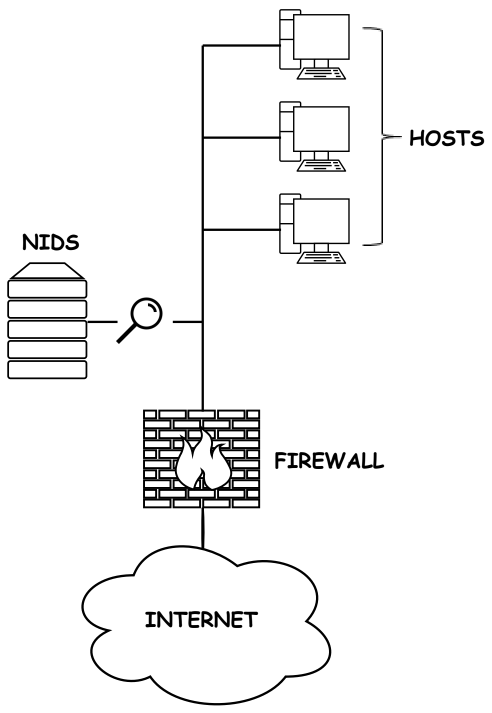


Figure 2.3: Network-based IDS.

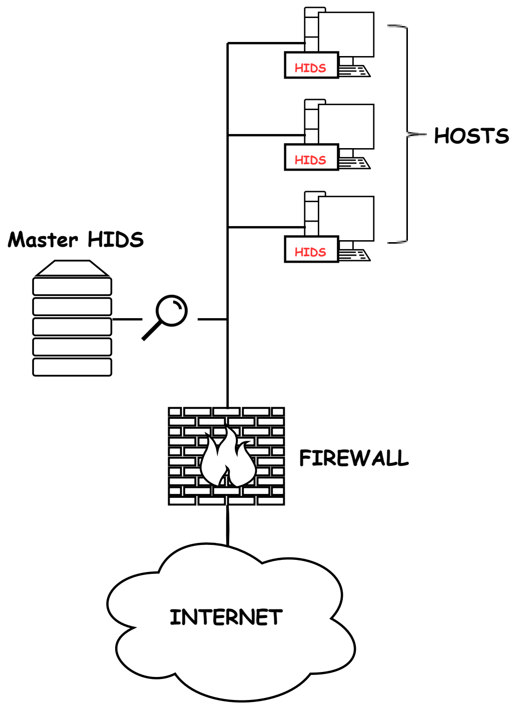


Figure 2.4: Host-based IDS.

2.5.1.3 Hybrid IDS

The network-based IDS may burden the workload and thus miss some malicious activities, whereas the host-based IDS does not monitor all network traffic and therefore has a lesser workload than the network-based IDS. Therefore, the hybrid IDS, as shown in Figure 2.5, deploys IDS modules in the network as well as on clients to simultaneously monitor specific client and network activities [6].

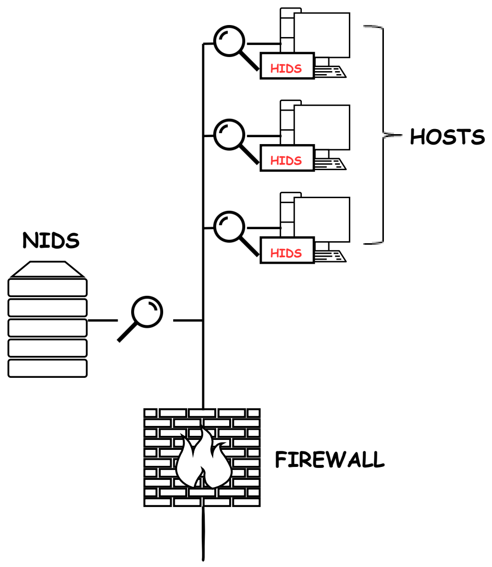


Figure 2.5: Hybrid IDS.

2.5.1.4 Performance comparison of Host-based IDS and Network-based IDS

Performance in terms of:	**Host-Based IDS**	**Network-Based IDS**
Intruder deterrence	Strong deterrence for inside intruders	Strong deterrence for outside intruders
Threat response time	Weak real time response but performs better for a long term attack	Strong response time against outside intruders
Assessing damage	Excellent in determining extent of damage	Very weak in determining extent of damage
Intruder prevention	Good at preventing inside intruders	Good at preventing outside intruders
Threat anticipation	Good at trending and detecting suspicious behavior patterns	Good at trending and detecting suspicious behavior patterns

Table 2.1: Performance comparison of HIDS and NIDS [32].

2.5.2 Detection methods

There exist two main types of intrusion detection systems:

2.5.2.1 Misuse-based IDS

A misuse-based IDS, known as a signature-based IDS rely on pattern matching techniques. It looks for events that match a predefined pattern of events that describe a known attack (signatures). An alarm is triggered when an event matches the signature of an attack in the database. This method is suitable for detecting known attacks, but new or unknown attacks (also called zero-day exploits) are difficult to detect [33].

2.5.2.2 Anomaly-based IDS

Anomaly, or behavior-based, intrusion detection involves establishing profiles of normal user/network behavior and comparing actual behavior to those profiles then raising an alarm if there is any deviation. This detection method can be very effective in identifying unknown attacks.

2.5.2.3 Comparison of IDS types based on the methodology

	Misuse-based	**Anomaly-based**
Method	Identify known attack patterns	Identify unusual activity patterns
Detection Rate	High	Low
False alarm rate	Low	High
Unknown attack detection	Incapable	Capable

Table 2.2: Comparison of IDS types based on the methodology [6].

2.5.3 Location of data analysis

Intrusion detection systems can also be classified based on the actual location of the data analysis:

2.5.3.1 Centralized IDS

Consist of multiple monitors that monitor the behavior of their respective hosts or network traffic passing by, as well as a single central analyzer that analyzes the data received from each monitor. The primary goal of this architecture is to make event correlation easier. The central analyzer, on the other hand, provides a single point of failure and a single target for an attack. That is, if the central analyzer fails or is attacked, the entire system is jeopardized. Furthermore, communication with the central component can cause parts of the network to become overloaded [34].

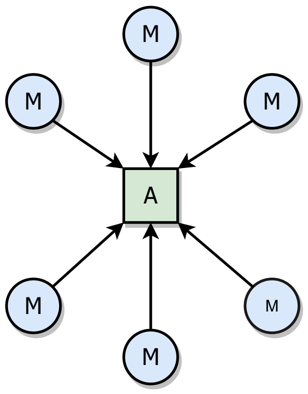


Figure 2.6: Overview of centralized IDS architecture that consist of monitors (M) and analysis unit (A) [35].

2.5.3.2 Distributed Intrusion Detection System (DIDS)

Is designed to operate in a non-homogeneous environment, which means that DIDS shares the task of the central analysis unit equally among all monitors and provides the ability to aggregate information from different sources to detect attacks against the network system. This architecture has some advantages over the centralized approach. Mainly, distributed architectures do not have a single point of failure. In addition, instead of having a central monitoring station to which all data must be transmitted, independent entities perform the data collection and analysis. This allows for better scalability of the system [34].

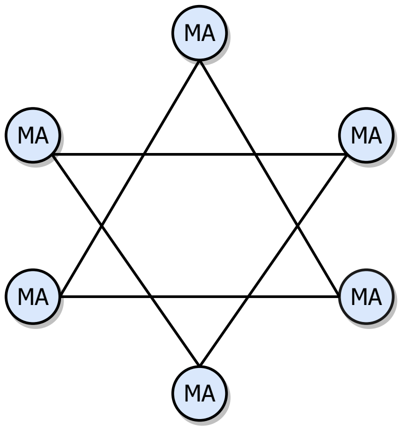


Figure 2.7: Architecture of Distributed IDS [35].

2.5.3.3 Distributed and Centralized IDS advantages and disadvantages

IDS	Advantages	Disadvantages
Distributed IDS	-Flexibility and scalability -Reduce computational costs -Monitoring, analysis, and processing of attack data is easier	-The data stream among the host and the agent may produce high network traffic overheads -Can generate diverse outputs from different IDs.
Centralized IDS	-The maintenance and administration cost lower compared to the case of a distributed system -All of the IDS activities are controlled directly by a central console.	-Not able to detect malicious events occurring at different places at the same time. -A hacker can incapacitate the programs running on a system, making the IDS unusable or unreliable

Table 2.3: Distributed and Centralized IDS advantages and disadvantages [36].

2.5.4 Behavior in case of detection of an attack

Each IDS will respond differently after detecting an attack, depending on its configuration and function, and the responses are divided into two categories:

2.5.4.1 Passive response

Consists primarily of logging and notifying personnel, it is incapable to perform any protective or corrective function on its own. Notification can take many forms, including an email, text message, pop-up window, or notification on a central monitor.

2.5.4.2 Active response

Aims to minimize the damage caused by the attack by providing real-time corrective action and automatically blocking suspicious attacks without any intervention required by an operator. Actions may include gathering information regarding the nature of the attack, blocking the source address, closing connections, restarting a server and so on.

Passive	**Active**
Administrator notification: generate alarm *(through email, online/pager notification, etc.)* generate report *(can contain information about an intrusion such as attack target, criticality, time, source IP/user account, description of suspicious packets, etc. as well as intrusion statistics for some period of time such as number of alarms from each IDS, attack targets grouped by IP, etc.)* *Other responses:* enable additional IDS enable local/remote/network activity logging enable intrusion analysis tools backup tampered with files trace connection for information gathering purposes	*Host-based response actions:* deny full/selective access to file delete tampered with file allow to operate on fake file restore tampered with file from backup restrict user activity disable user account shutdown compromised service/host restart suspicious process terminate suspicious process disable compromised services abort suspicious system calls delay suspicious system calls *Network-based response actions:* enable/disable additional firewall rules restart targeted system block suspicious incoming/outgoing network connection block ports/IP addresses trace connection to perform attacker isolation/quarantine create remote decoy

Table 2.4: List of common passive and active intrusion responses [37].

2.6 Conclusion

This chapter allowed us to familiarize ourselves with the notion of IDS. We defined its architecture and its three main components and summarized the different types of IDS while comparing their performances, we also discussed detection methods and the IDS's behavior in the event of an attack. In the following chapter we will introduce the concept of Deep Learning.

Chapter 3

Machine Learning & Deep Learning

3.1 Introduction

Deep Learning (DL) is a subset of Machine Learning (ML), which is itself a subset of Artificial Intelligence (AI) (see Figure 3.1). The term "Artificial Intelligence" was coined in 1955 by the American mathematician "John MacCarthy". The real challenge of artificial intelligence turned out to be solving tasks that are normally performed by humans intuitively, such as visual perception, speech recognition, decision making and translation between languages [38]. Deep Learning has revolutionized technology industries and many aspects of modern society are all powered by DL technology: from web searches to content filtering on social networks to video recommendations, and it is increasingly present in consumer products such as cameras and smartphones [39].

In what follows, we will review the concepts of ML. We then present the Neural Network model that underlies DL. Next, we outline the different Deep Neural Network architectures and discuss their uses.

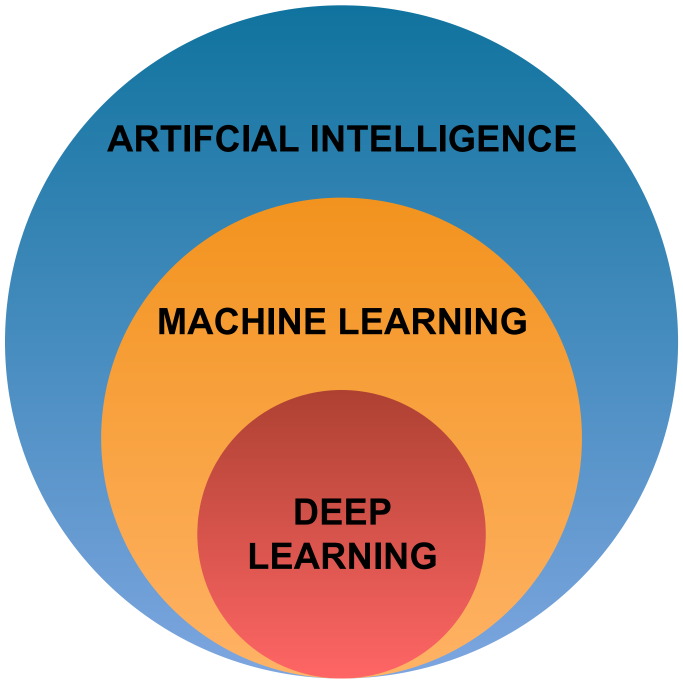


Figure 3.1: Venn diagram representing the relationships between AI, ML and DL.

3.2 Brief overview of Machine Learning

ML consists of various algorithms and approaches to solve different types of problems. Arthur Samuel in his seminal work defined machine learning as, "a field of study that gives computers the ability to learn without being explicitly programmed"[40]. Another American by the name of Tom Mitchell gave in 1998 a more modern definition of Machine Learning by stating that a machine learns when its performance in a certain task improves with new experiences [41]. In the field of machine learning, the most common learning technique (supervised learning) is directly inspired by the way we humans learn to do things. A machine is given a set of examples that the machine must study to develop what is called a model and these examples are generally grouped in what is called a dataset.

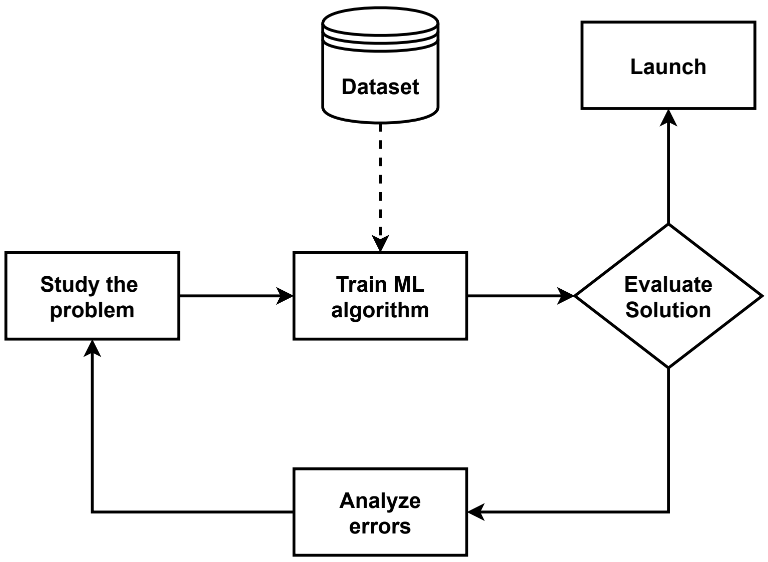


Figure 3.2: Machine Learning approach [42].

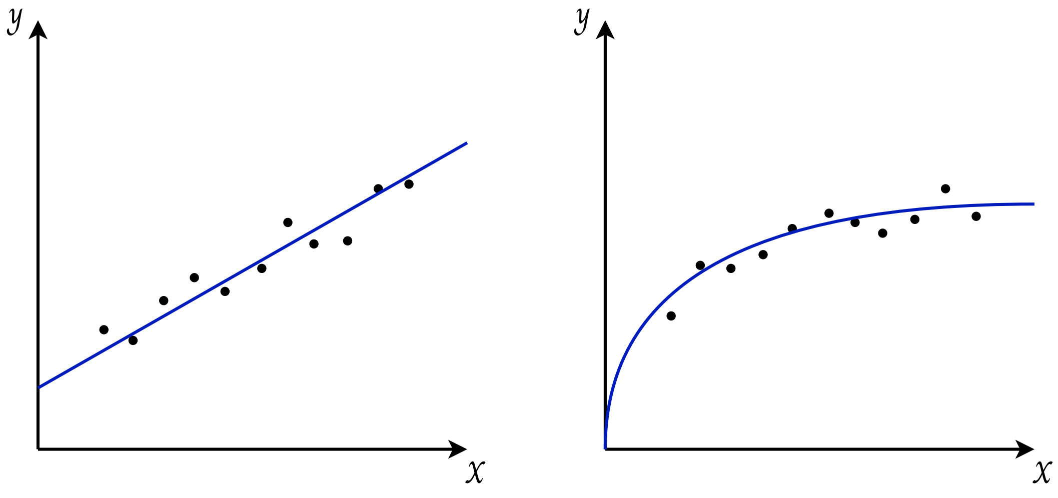
To better understand supervised learning, it is essential to have a good understanding of the following 4 concepts:

1. **Dataset:**
   Supervised learning is when a machine is given many examples (x, y) in order to learn the relationship between x and y. These examples (x, y) are compiled into an array called a dataset :

   - The variable y is named target, it is the value we are trying to predict.
   - The variable x is called a feature, it influences the value of y and we usually have many features $(x_1, x_2, ...)$ in our dataset that we group into a matrix X.

2. **Model and its parameters:**
   In ML, we develop a model from this dataset. It can be a linear model as you can see on the left (see Figure 3.3a), or a non-linear model as you can see on the right (see Figure 3.3b). We define a, b, c etc. as the parameters of a model.

3. **Cost function:**
   Another thing to note is that a model returns errors with respect to our dataset. We call the set of these errors the cost function (see Figure 3.14).



(a) Linear model **(b)** Non-linear model

Figure 3.3: Linear and non-linear models.

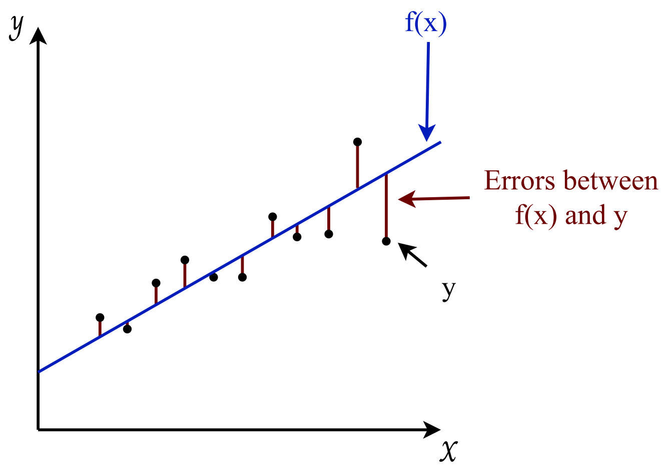
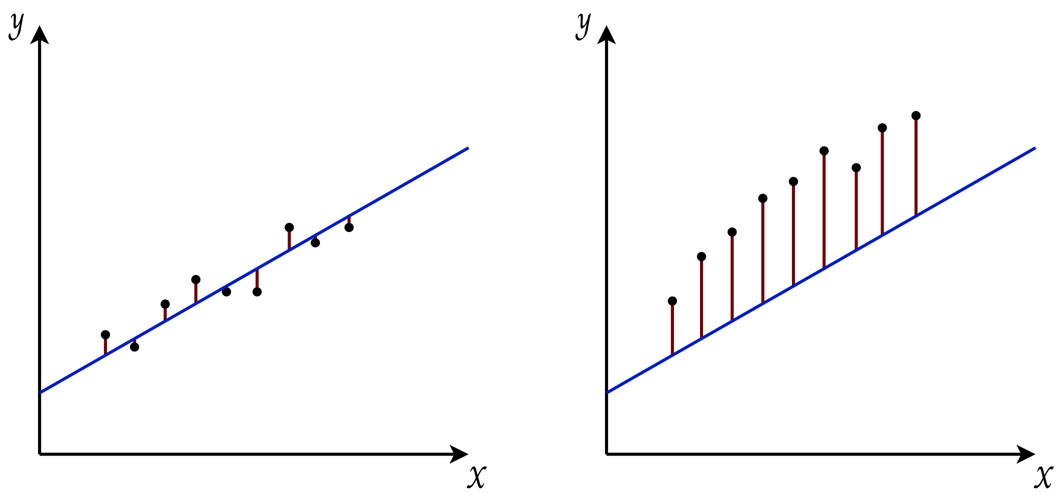


Figure 3.4: Set of errors.

Having a good model means having a model that produces small errors (see Figure 3.5).



(a) Low loss **(b)** High loss

Figure 3.5: Good model (on the left) vs bad model (on the right).

4. **Learning algorithm:**
   The main objective in supervised learning is to find the model parameters $(a, b, c, etc.)$ that minimize the Cost Function. To achieve this, we employ a learning algorithm, the most common of which is the Gradient Descent algorithm.

3.3 Review of Artificial Neural Networks

In Deep Learning, instead of developing a model, we develop what are known as Artificial Neural Networks (ANNs). The concept remains the same as in machine learning, but this time our model is not a simple function such as $f(x) = ax + b$, but rather a network of functions connected to each other (a Neural Network). The more functions these networks contain, the more the machine can learn to perform complex tasks like object recognition, identifying people in a picture, driving a car, and so on.

To understand how artificial Neural Networks work, it is necessary to go back to the origin of their history in order to learn how they were invented and how they evolved over time to reach the technology we know today.

3.3.1 History of Artificial Neural Networks

The first Neural Networks (NNs) were invented in 1943 by two mathematicians and neuroscientists named Warren McCulloch and Walter Pitts. In their scientific article entitled: "A logical calculus of the ideas immanent in nervous activity"

[43], they explain how they were able to program artificial neurons inspired by the functioning of biological neurons.

In biology, neurons are excitable cells connected to each other and whose role is to transmit information in our nervous system. Each neuron is composed of a cell body, and many branching extensions called dendrites, plus one very long extension called the axon [42]. (as represented in Figure 3.6).

Dendrites are the neuron's entry point. The neuron receives signals from previous neurons via synapse. When the sum of these signals exceeds a certain threshold, the neuron is activated and sends an electrical signal along the axon to the endings, where it is forwarded to other neurons in our nervous system [42].

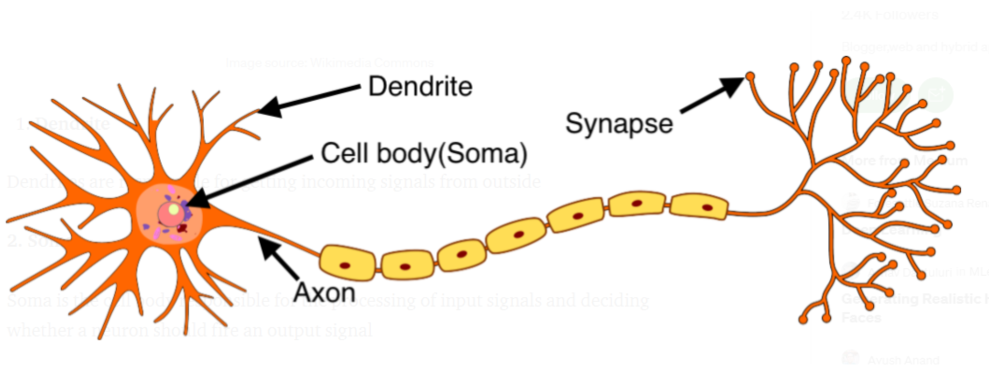


Figure 3.6: Biological neuron.

3.3.2 Modeling an artificial neuron

What Warren McCulloh and Walter Pitts tried to do was to model this operation, considering that a neuron could be represented by a transfer function, which takes as input signals x and returns an output y (see Figure 3.7).

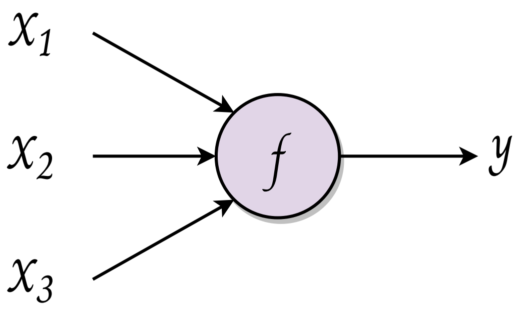


Figure 3.7: Transfer function f with inputs (x_1, x_2, x_3) and output y.

Within this function, there are 2 main phases:

3.3.2.1 Aggregation phase

We sum up all of the neuron's inputs, multiplying each one by a coefficient w. This coefficient represents synaptic activity, i.e. whether the signal is excitatory ($w = +1$) or inhibitory ($w = -1$). In this aggregation phase, we obtain an expression of the form:

$$Aggregation : f = w_1 x_1 + w_2 x_2 + w_3 x_3 \tag{3.1}$$

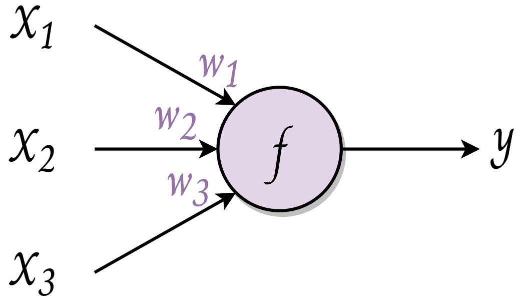


Figure 3.8: Aggregation phase.

3.3.2.2 Activation phase

We look at the result of the previous calculation, and if it exceeds a certain threshold (usually 0), then the neuron is activated and returns an output $y = 1$. If not, it remains at 0.

$$Activation = \begin{cases} y = 1 \quad if \quad f \geq 0 \\ y = 0 \quad else \end{cases} \tag{3.2}$$

This is how Warren McCulloch and Walter Pitts managed to develop the first artificial neurons later renamed "Threshold Logic Unit". This name comes from the fact that their model was originally designed to handle only logic inputs of 0 or 1. They were able to demonstrate that certain logic functions, such as the AND gate and the OR gate, could be reproduced using this model. They also demonstrated that by connecting several of these functions to each other, similar to the neurons in our brain, any Boolean logic problem could be solved [43].

However, even if this model lays the basis of what Deep Learning is today, it contains a certain number of flaws, notably the fact that it does not have a learning algorithm.

A solution was found 15 years later in 1957 by Franck Rosenblatt the inventor of the Perceptron by proposing the first learning algorithm in the history of Deep Learning.

3.3.3 The Perceptron

The Perceptron is the basic unit of Neural Networks. It is a binary classification model, capable of linearly separating two classes of points.

We provide variables $(x_1, x_2, ...x_n)$ to a neuron and multiply each input of the neuron by a weight $(w_1, w_2, ...w_n)$, in addition we will pass a complementary coefficient b that we call the bias which gives us a function:

$$z = w_1x_1 + w_2x_2 + b \tag{3.3}$$

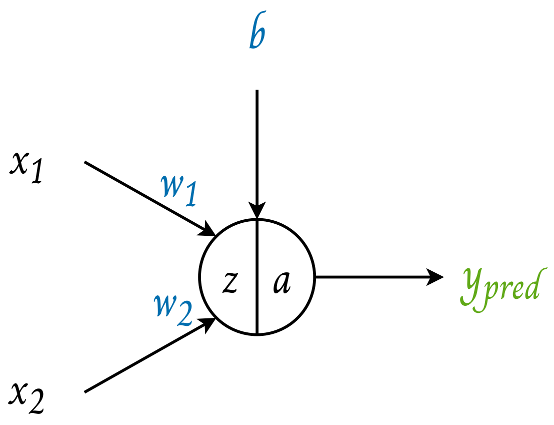


Figure 3.9: Perceptron model.

The Perceptron is defined by the following characteristics:

- The input data vector $(x_1, x_2, ..., x_n)$.
- The weights $(w_1, w_2, ..., w_n)$ accorded to each input for neuron activation.
- The aggregation function: sums the products of the information by their associated weights and processes this data: $S = \sum_{i=1}^{n} x_i w_i + b$.
- The bias b allows the shift of the activation function by a constant amount.

- Activation function a(z) which associates to the aggregated value a unique value, the neuron compares this value to a threshold and decides the output either active or inactive. This function can take several forms.

The Perceptron also has a learning algorithm that allows it to find the values of its parameters W in order to obtain suitable outputs y [44].

To develop this algorithm, Frank Rosenblatt was inspired by Hebb's theory, which suggests that when two biological neurons are jointly excited, they strengthen their synaptic links [45]. In neuroscience, this phenomenon is called synaptic plasticity, and it is what allows our brain to build its memory, learn new things or make new associations.

Based on this concept, Frank Rosenblatt has developed a learning algorithm, which consists in training an artificial neuron on reference data (X, y) so that it reinforces its parameters W each time an input X is activated at the same time as the output y present in these data. He devised the following formula (3.4), in which the parameters W are updated by calculating the difference between the reference output and the output produced by the neuron, and by multiplying this difference by the value of each input X, and by a positive learning rate α.

$$W = W + \alpha(y_{\text{true}} - y)X \tag{3.4}$$

- y_{true} : reference output.
- y : output produced by the neuron.
- X : neuron input.
- α : learning rate.

If our neuron produces an output different from the one it is supposed to produce, ($y = 0$, $y_{\text{true}} = 1$), then our formula returns :

$$W = W + \alpha(1 - 0)X$$

$$W = W + \alpha X$$

Therefore, for inputs x that are equal to 1, the coefficient w will be increased by a small α step. This leads to an increase of the function f and brings our neuron closer to the activation threshold.

$$\begin{cases} w_1 = w_1 + \alpha & (x_1 = 1) \\ w_2 = w_2 & (x_2 = 0) \end{cases}$$

As long as the neuron produces a bad output ($y \neq y_{\text{true}}$) , the coefficient W will continue to increase (see Figure 3.10) , until y_{true} equals y and at that moment our formula returns ($W = W + 0$), which means that our parameters will stop adjusting.

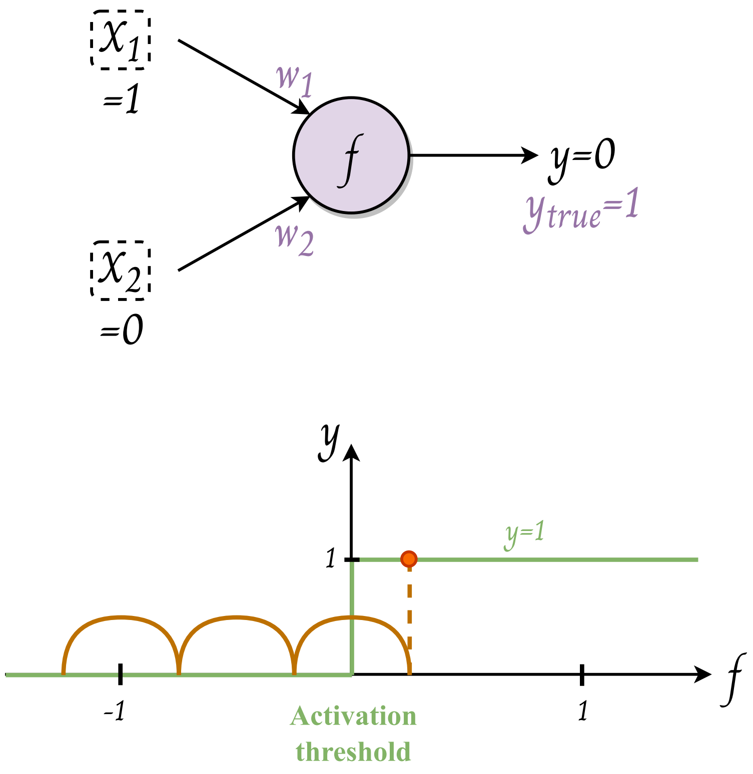


Figure 3.10: The first learning algorithm in Deep Learning history.

3.3.4 Common activation functions

Activation functions decide whether a neuron should be activated or not by taking the weighted sum of all the inputs of the previous layer, and then generate and transmit an output value (usually non-linear) to the next layer. They also help to normalize the output of each neuron to a range between 1 and 0 or between -1 and 1. In the following we present some examples of activation functions:

3.3.4.1 Sigmoid function

The following sigmoid activation function (see Figure 3.11) converts the weighted sum to a value between 0 and 1. It is especially used for models where we have to predict the probability as an output (since probability of anything exists only between the range of 0 and 1).

$$a(z) = \frac{1}{1 + e^{-z}} \tag{3.5}$$

3.3.4.2 ReLU function

The following Rectified Linear Unit (ReLU) activation function often works a little better than a smooth function like the sigmoid, while also being significantly easier to compute (see Figure 3.12).

$$a(z) = max(0, z) = \begin{cases} 0 \quad if \quad z < 0 \\ z \quad if \quad z \geq 0 \end{cases} \tag{3.6}$$

The negative side of the graph makes the gradient value zero. Due to this reason, during the back propagation process, the weights and biases for some neurons are not updated. This can create dead neurons which never get activated. All the negative input values become zero immediately, which decreases the model's ability to fit or train from the data properly [46].

3.3.4.3 Tanh function

Hyperbolic tangent (Tanh) function is very similar to the sigmoid activation function, and even has the same S-shape with the difference in output range of -1 to 1 (see Figure 3.13).

$$a(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} \tag{3.7}$$

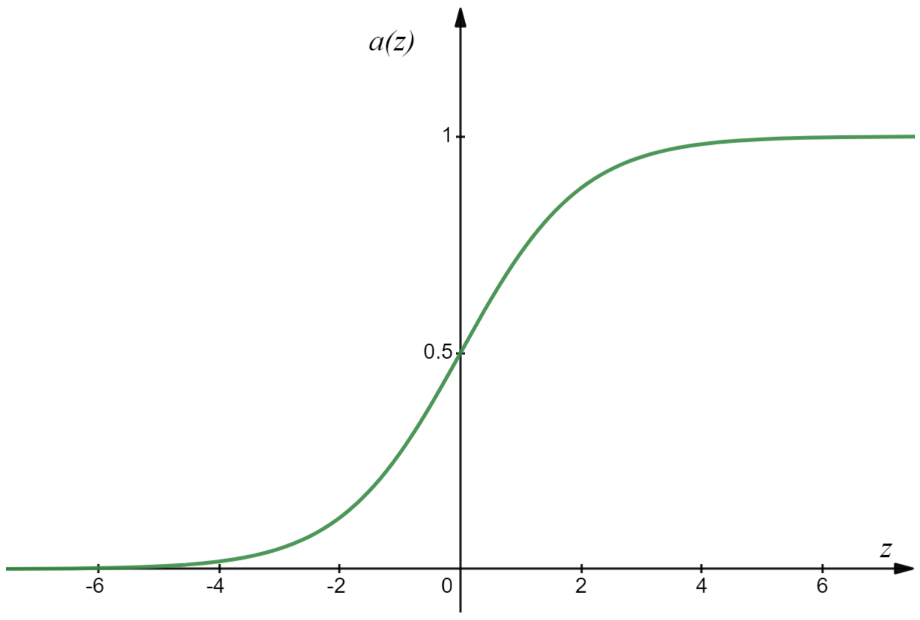


Figure 3.11: Sigmoid function.

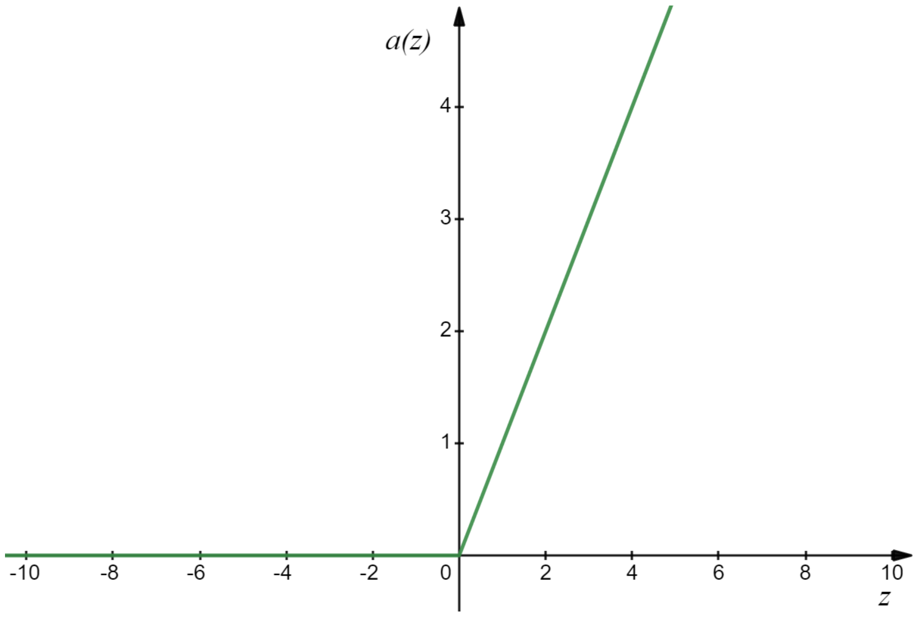


Figure 3.12: ReLU function.

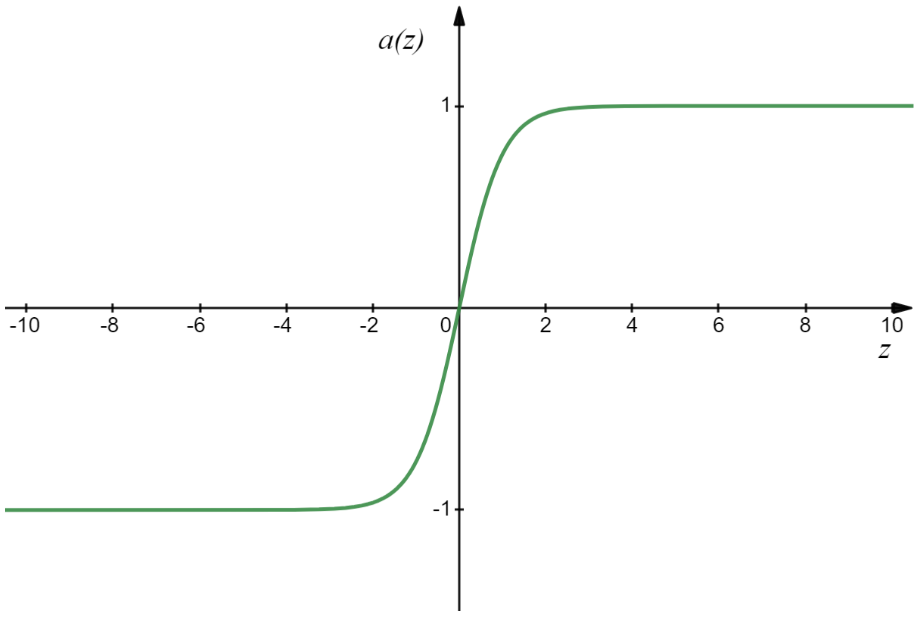


Figure 3.13: Tanh function.

Our goal is to set the parameters w and b in order to obtain the best possible model that is to say the model that makes the smallest errors between the outputs $a(z)$ and the real data y and for that we will start by defining a **loss function** that will allow to measure its errors.

3.3.5 Loss function

A loss function is a function that allows us to quantify the errors made by a model and to measure the distances between the outputs $a(z)$ and the data y that we have (see Figure 3.14).

There are several loss functions. One of them, widely used in regression problems, is the logarithmic loss function defined by the following formula:

$$L = \frac{-1}{m} \sum_{i=1}^{m} y_i log(a_i) + (1 - y_i) log(1 - a_i) \tag{3.8}$$

- m : amount of data.
- y_i : data number i.
- a_i : output number i.

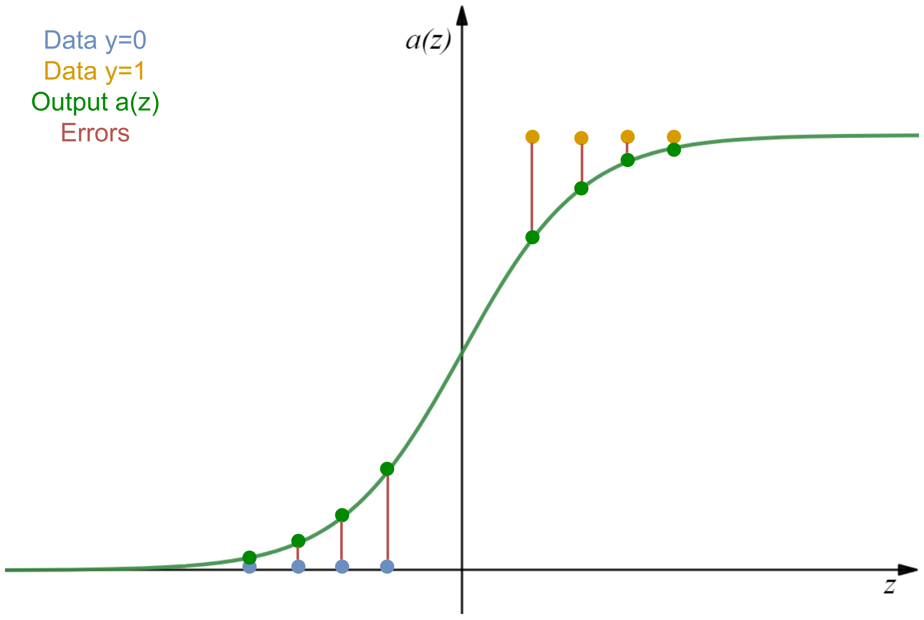


Figure 3.14: Distances between outputs $a(z)$ and the data y.

Once we dispose of our loss function we can use it to minimize the errors of our model and for that we will use the **gradient descent algorithm**.

3.3.6 Gradient descent algorithm

Gradient descent is one of the most widely used learning algorithms in DL. It consists in adjusting the parameters w and b in order to reduce the model's errors. To do so, we must first determine how this function varies according to the different parameters.

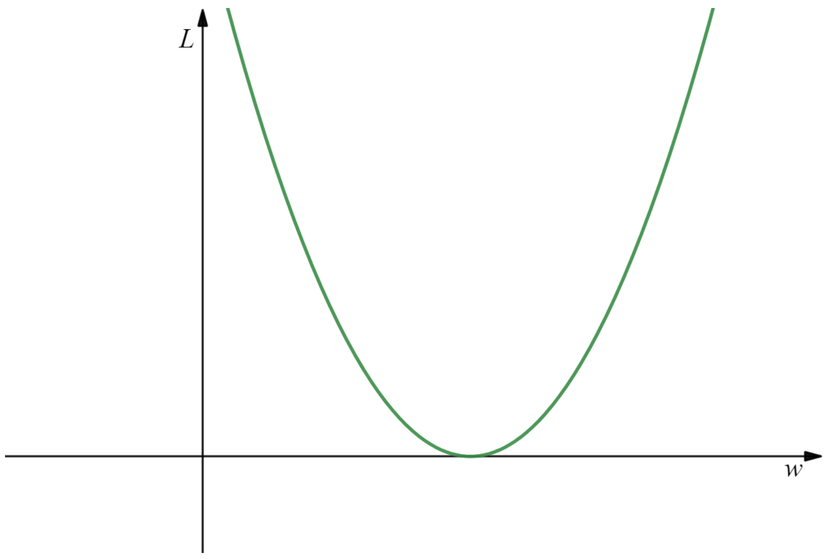


Figure 3.15: Log loss curve.

By calculating the derivative of the function, we can learn how the function varies. If the derivative is negative, it indicates that the function decreases as w increases. Conversely, if the derivative is positive, it indicates that the loss function increases when w increases (see Figure 3.16).

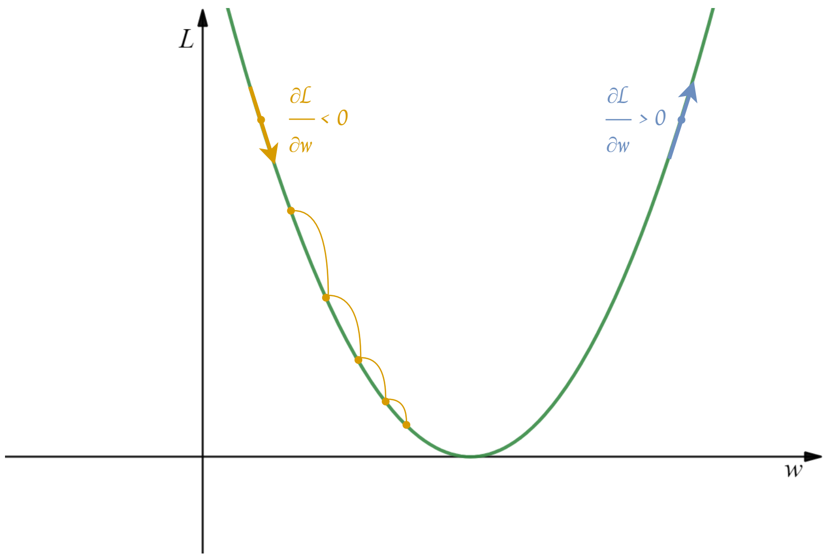

L
$\frac{\partial \mathcal{L}}{\partial w} < 0$
$\frac{\partial \mathcal{L}}{\partial w} > 0$
w

Figure 3.16: Gradient descent algorithm.

The formula for gradient descent is as follows:

$$W_{\mathrm{t+1}} = W_{\mathrm{t}} - \alpha \, \frac{\partial L}{\partial W_{\mathrm{t}}} \tag{3.9}$$

$W_{\mathrm{t+1}}$: Parameter W at instant $t+1$

W_{t} : Parameter W at instant t

α : Positive learning rate

$\frac{\partial L}{\partial W_{\mathrm{t}}}$: Gradient at instant t

By repeating this formula in a loop we are able to reach the minimum of the loss function by progressively descending its curve, hence the gradient descent term.

The Perceptron being a linear model made it a not very useful model since a large part of the phenomena of our universe are not linear. Artificial intelligence had its first winter, from 1974 to 1980, during which there were almost no investors to fund AI research.

AI was on the verge of death until 1986, when Geoffrey Hinton, one of the fathers of Deep Learning, developed the Multi-Layer Perceptron (MLP), the first true artificial Neural Network.

3.3.7 The Multi-Layer Perceptron

This model proposes to connect together several neurons which makes the resolution of more complex problems possible (see Figure 3.17).

For example we connect three Perceptrons; the first two receive inputs x_1 and x_2, performing calculations based on their parameters, and return two outputs y_1 and y_2, which they in turn send to the third Perceptron which performs calculations to produce the final output y_3.

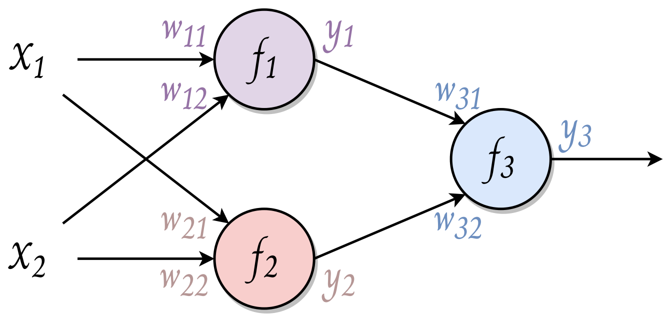


Figure 3.17: Example of a Multi-Layer Perceptron (MLP).

$$\begin{cases} f_1 = w_{11}x_1 + w_{12}x_2 + b_1 \\ f_2 = w_{21}x_1 + w_{22}x_2 + b_2 \\ f_3 = w_{31}y_1 + w_{32}y_2 + b_3 \end{cases} \tag{3.10}$$

We can see from the graphs below (see Figure 3.18) that the most accurate model is the Multi-Layer Perceptron.

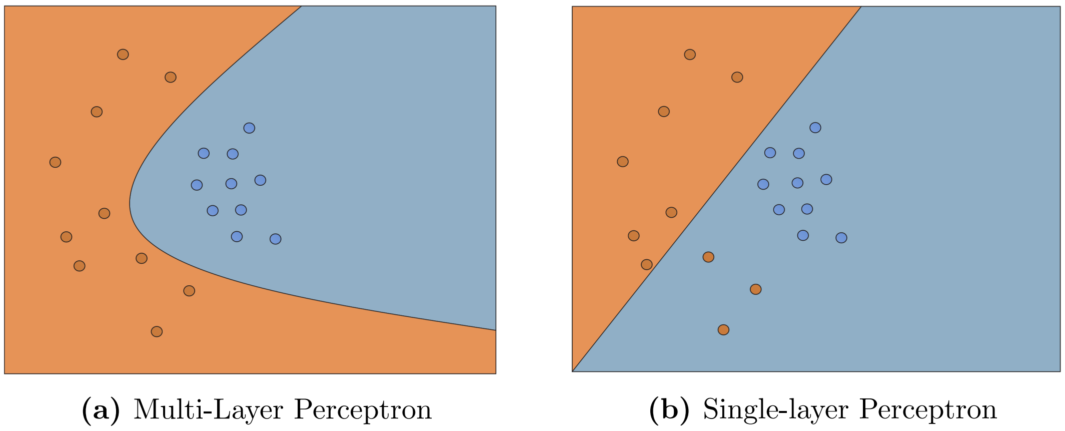

(a) Multi-Layer Perceptron **(b)** Single-layer Perceptron

Figure 3.18: Performance comparison between the single layer Perceptron and the Multi-Layer Perceptron.

Now the question that arises is how to train such a Neural Network so that it does what we ask it to do? in other words, how to find the values of all the parameters w and b in order to obtain a good model.

The solution is to use the "Back Propagation" method, which consists in determining how the network's output varies according to the parameters (w, b) present in each layer. To do so, we calculate a sequence of gradients indicating how the output varies according to the last layer, then how the last layer varies according to the second last layer, and so on until we reach the very first layer of our network (see Figure 3.19). Using the gradient, we can then update the parameters (w, b) of each layer so that they minimize the error between the output of the model and the expected output (y_{true}).

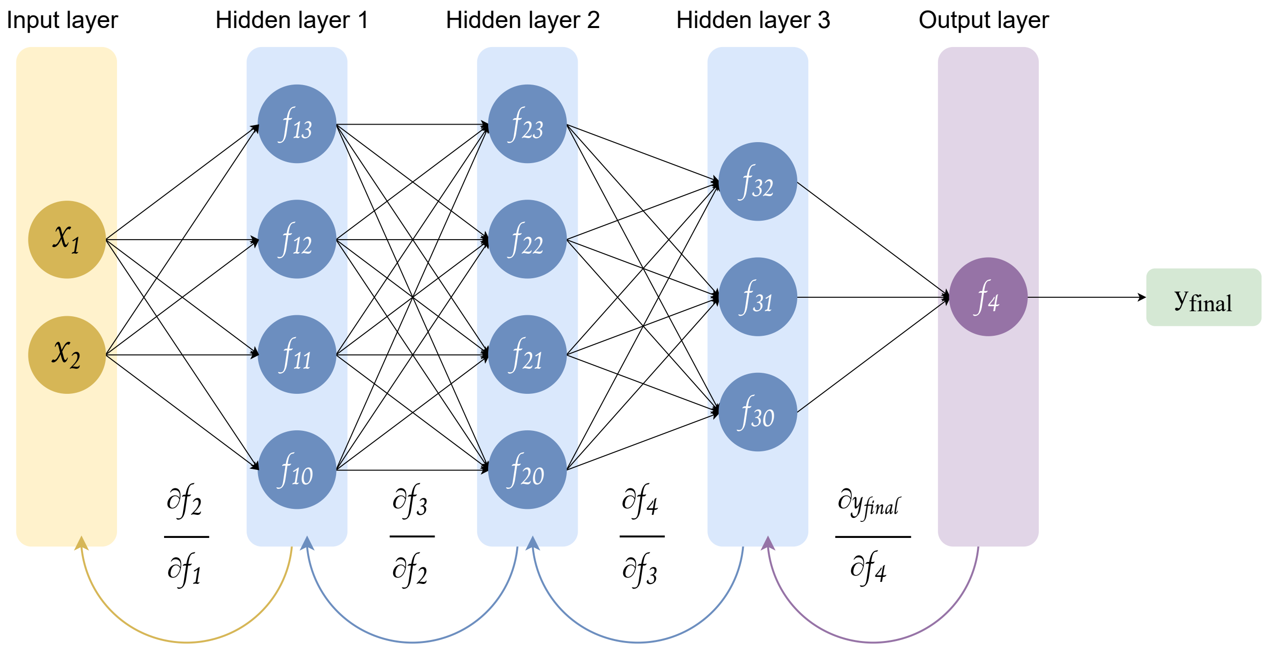


Figure 3.19: Back propagation method.

3.4 Convolution Neural Networks

The first Multi-Layer Perceptron variants were developed in 1989. The famous Yann LeCun invented the first Convolutional Neural Networks (CNNs) [47], which are inspired by the organization of an animal's visual cortex [48], [49] and are designed to automatically and adaptively learn spatial hierarchies of features, from low-level to high-level patterns, and capable of recognizing and processing images by introducing mathematical filters at the beginning of these networks.

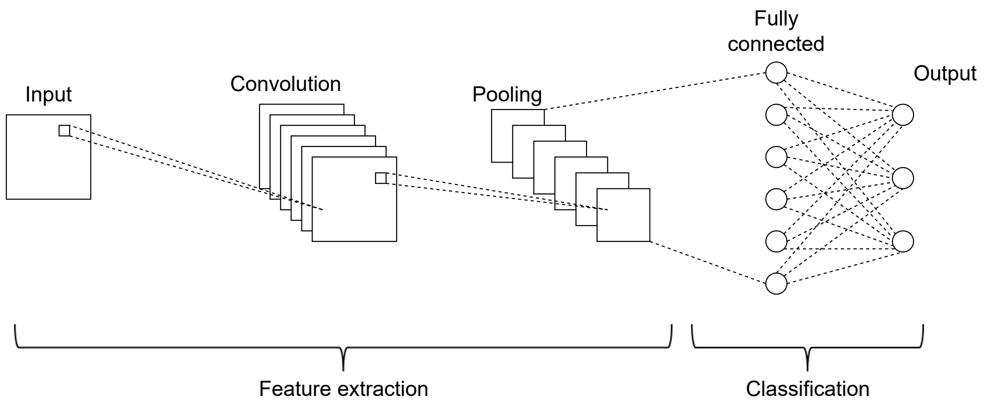


Figure 3.20: Outline of CNN [50].

CNN is a mathematical construct that is typically composed of three types of layers (or building blocks): convolution, pooling, and fully connected layers [51].

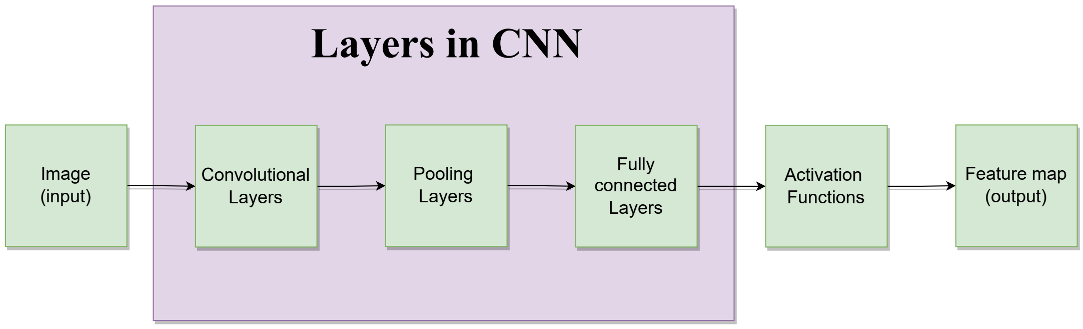
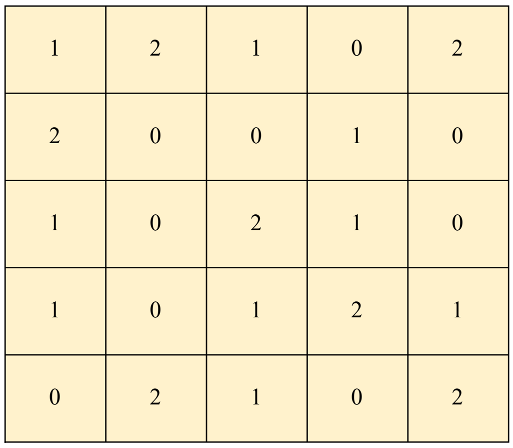


Figure 3.21: Layers in CNN.

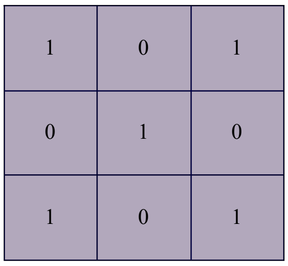
3.4.0.1 Convolutional layer

A convolution layer is a fundamental component of the CNN architecture that performs feature extraction [51]. It contains a set of filters (or kernels) that perform convolution operations on the input image before passing the result to the next layer. These convolution operations pass the input images through a set of convolutional filters, each of which activates certain features in the images [52]. The resulting output is called a feature map or activation map. A filter can technically be thought of as a relatively small matrix (typically 3x3 or 5x5) that can detect patterns such as edges, corners, circles, etc. The deeper the network, the more sophisticated these filters become. Thus, in later layers, instead of detecting edges in simple shapes, our filter may be able to detect specific objects such as eyes, ears or even different types of animals.

Let's take an example where we have an input image of size 5x5 and we want our convolutional layer to contain a filter of size 3x3 (see Figure 3.22).



(a) Input image

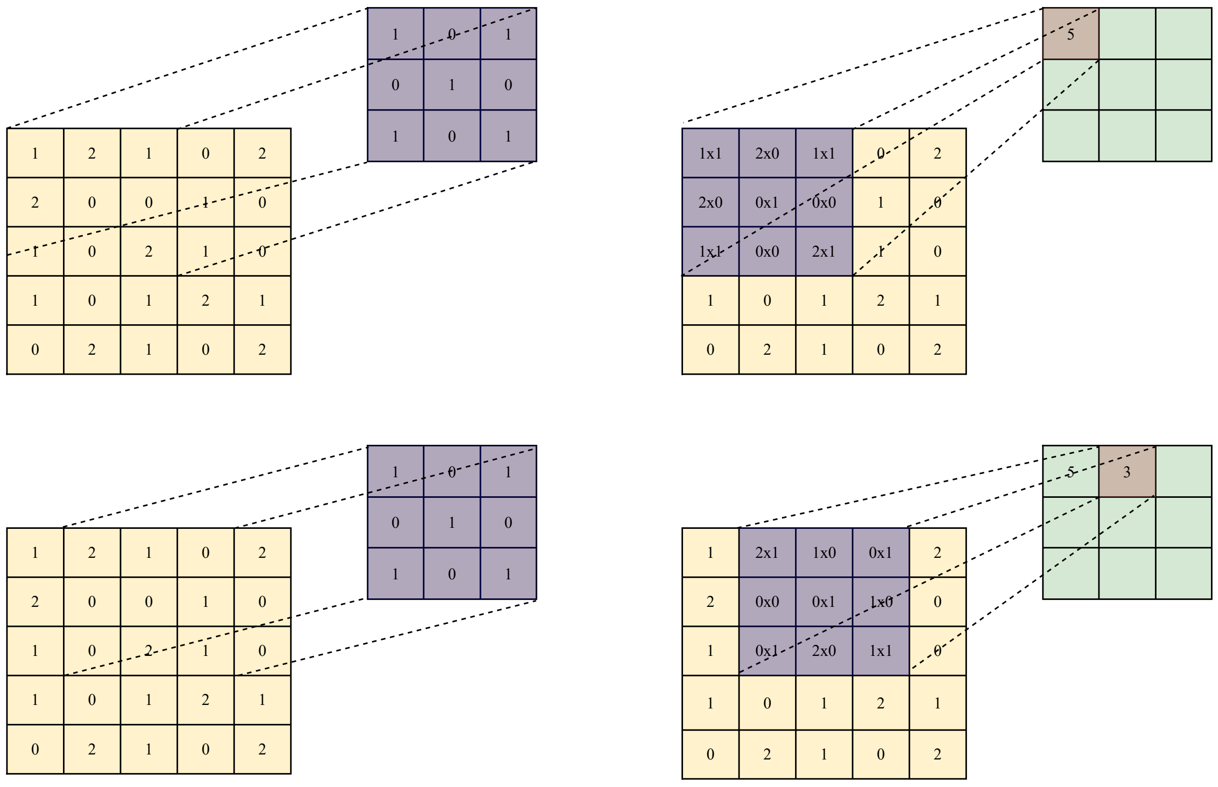


(b) Convolutional filter

Figure 3.22: Example of a (5x5) input image and a (3x3) convolutional filter.

When the convolutional layer receives the input, it slides over each 3x3 set of pixels from the input itself until it has covered every 3x3 block of pixels from the entire image.

Now let's apply convolution operations to this input using this filter and a step size of 1 pixel (see Figure 3.23).



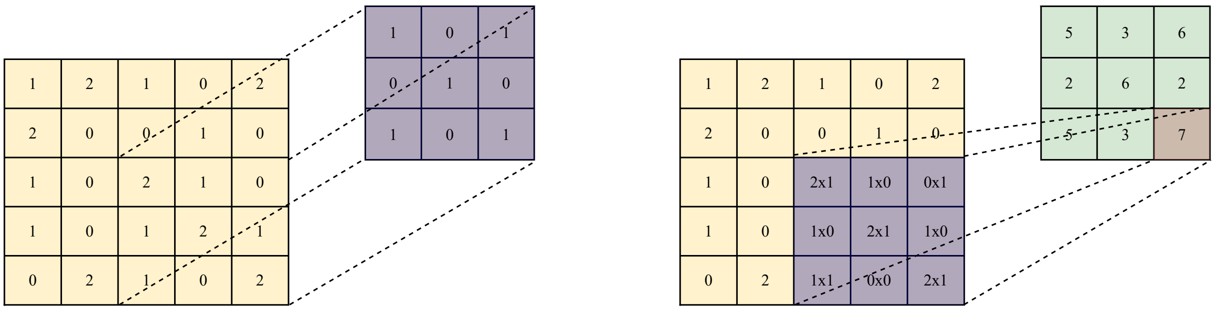


Figure 3.23: An example of convolution operation with a kernel size of 3x3 no padding, and a stride of 1.

By dragging the filter to all possible positions, we created a 3x3 output called a feature map or activation map (see Figure 3.24).

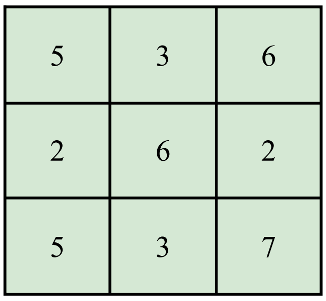


Figure 3.24: Feature map.

1. The **number of filters** affects the depth of the output. For example, three separate filters would produce three different feature maps, creating a depth of three.

2. **Stride** is the distance, or number of pixels (usually 1), over which the kernel moves on the input matrix. A larger stride results in a smaller output.

3. **Zero-padding** is usually used when the filters do not fit the input image. This sets all elements that fall outside of the input matrix to zero, resulting in a larger or equally sized output. There are three types of padding [53]:

   - **No padding:** in this case, the last convolution is dropped if the dimensions are not matched.

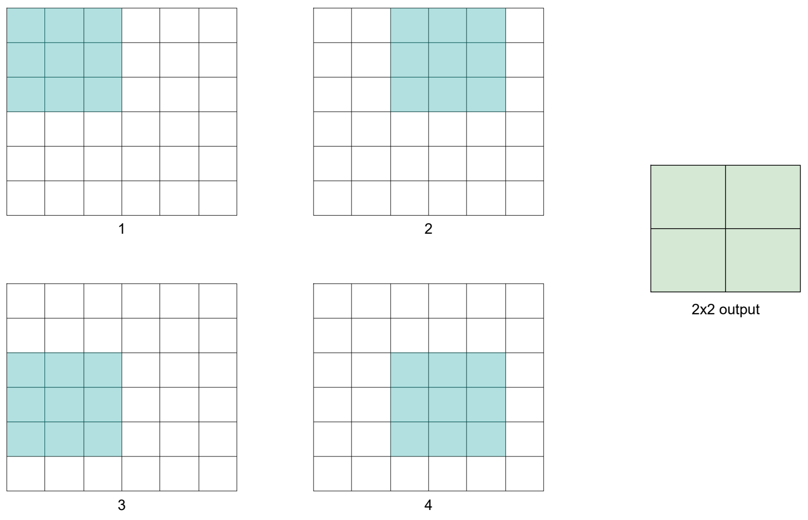


Figure 3.25: No padding.

- **Same padding:** ensures that the output layer has the same size as the input layer.

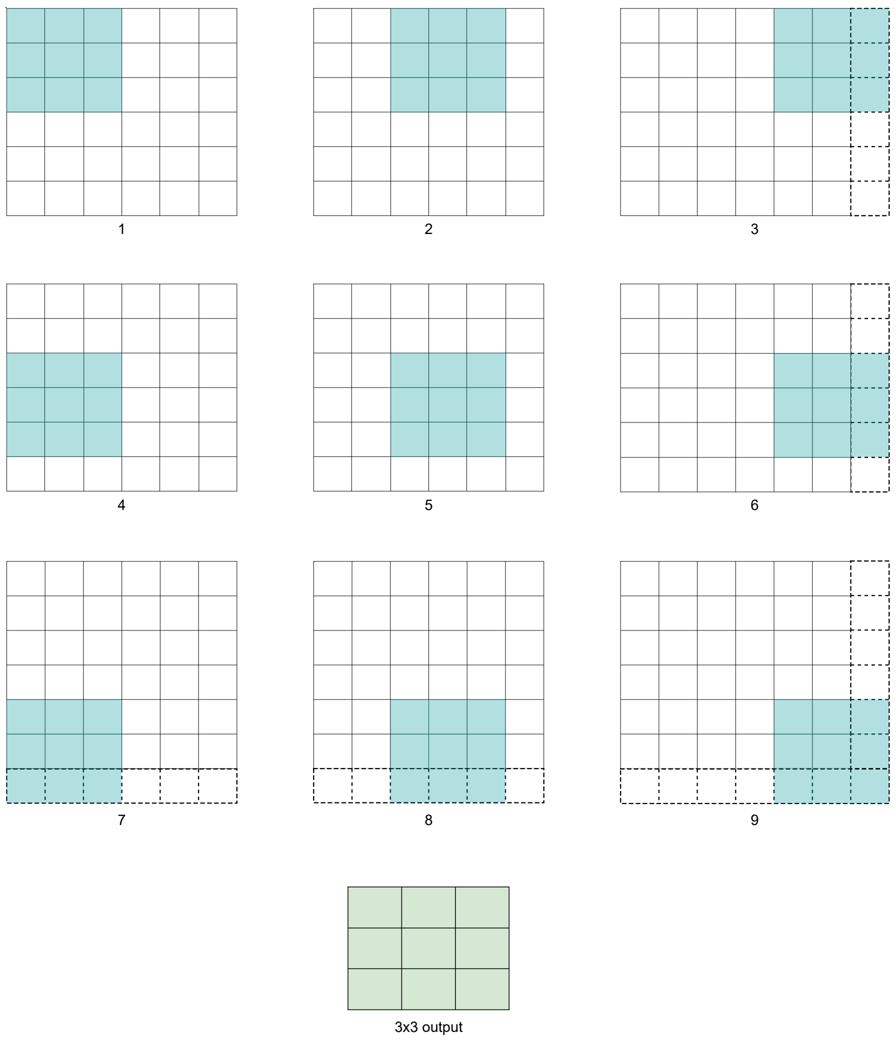


Figure 3.26: Same padding.

- **Full padding:** increases the size of the output by adding zeros to the border of the input.

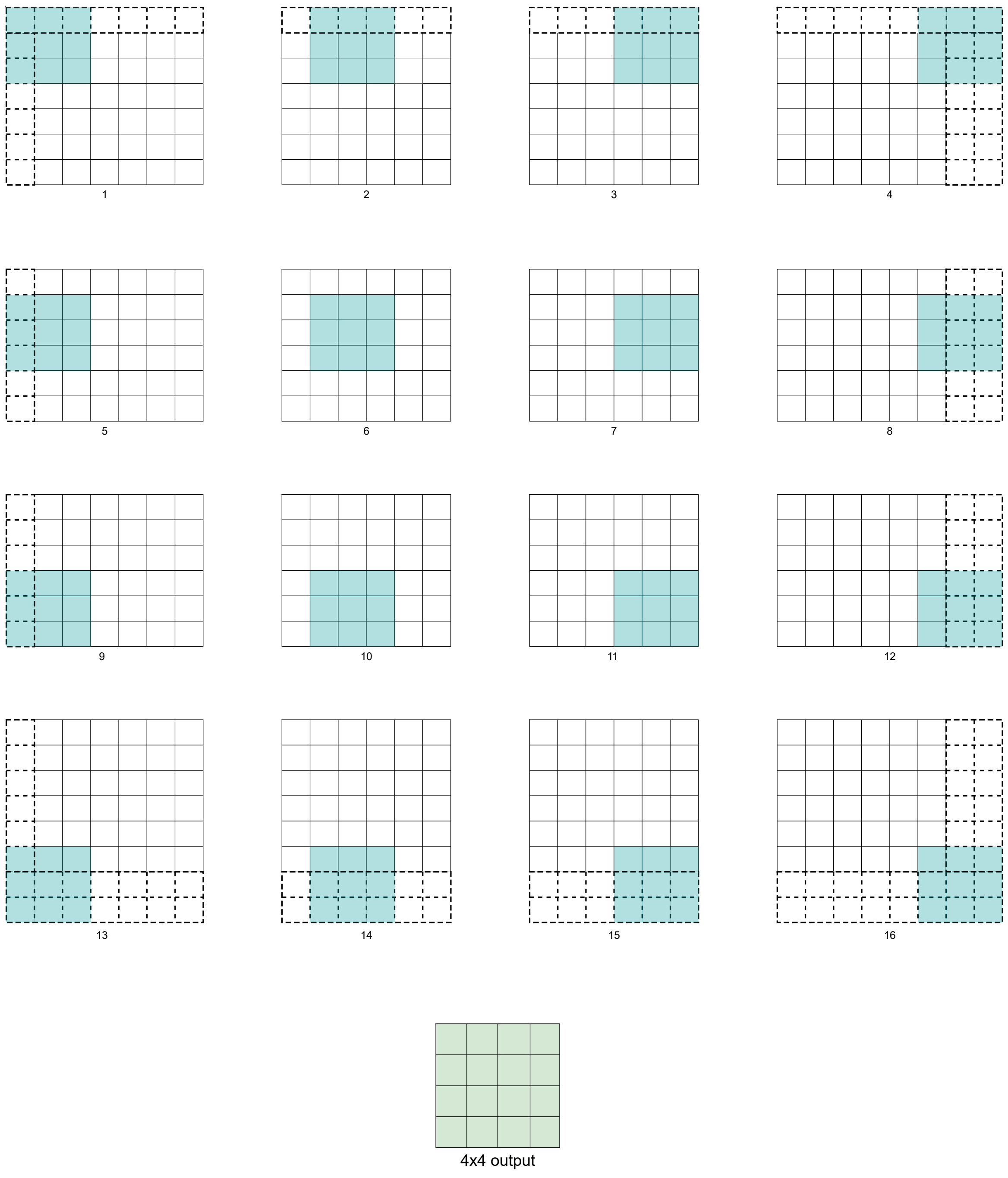


Figure 3.27: Full padding.

3.4.0.2 Pooling layer

The pooling layer simplifies the output by performing a nonlinear downsampling operation that aims to reduce the size of the convolutional feature maps, allowing for much faster computation due to the reduced amount of learnable parameters in the network [51]. It is therefore common to periodically insert a pooling layer between two successive convolutional layers of a CNN architecture. There are several approaches to pooling. The most commonly used approaches are max-pooling and average pooling.

- **Max pooling:** returns the maximum value of the part of the image covered by the kernel. It also works as a noise suppressor by completely eliminating noisy activations and also performs denoising and dimensionality reduction [54].

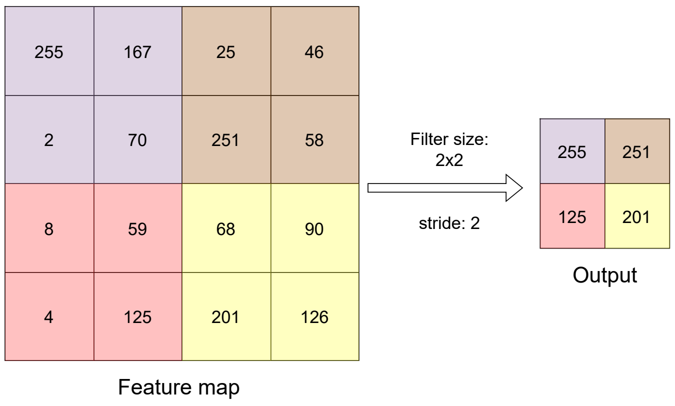


Figure 3.28: Max pooling.

- **Average pooling:** calculates the average value of the elements present in the region of the feature map covered by the filter.

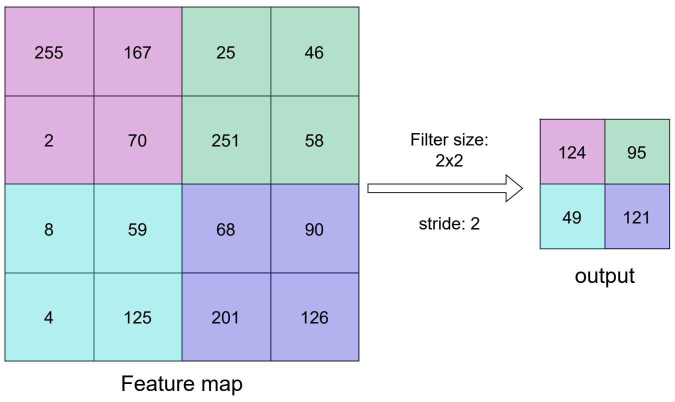


Figure 3.29: Average pooling.

3.4.0.3 Fully connected layer

The output of the final pooling or convolution layer is flattened into a one-dimensional (1D) array of numbers (or vector) (see Figure 3.30) and fed to one or more fully connected layers in which each input is connected to every output by a learnable weight. The final fully connected layer has the same number of output nodes as the number of classes, while each fully connected layer is followed by a nonlinear activation function [51].

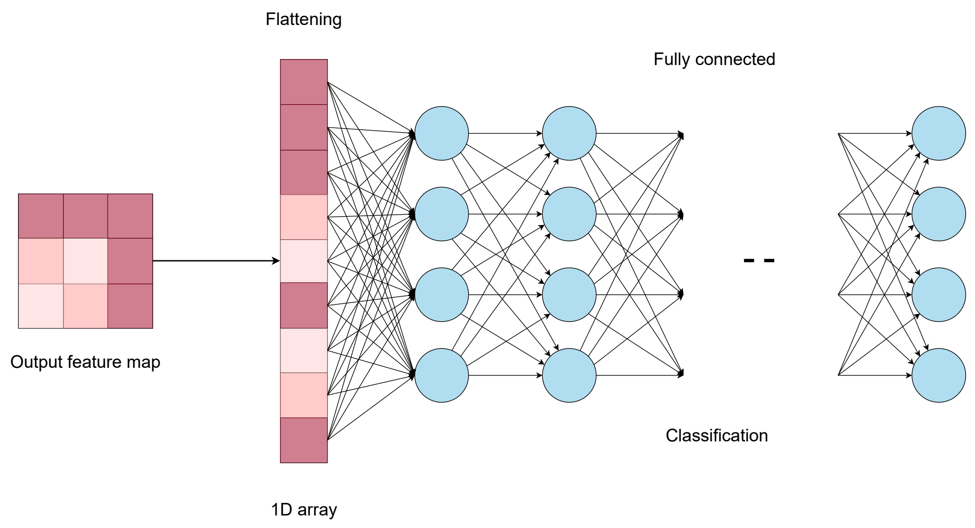


Figure 3.30: The output feature map is flattened before being passed to the fully connected layers.

3.5 Recurrent Neural Networks

Recurrent Neural Networks (RNNs) were first developed in the 1980s based on the work of David Rumelhart [55]. But their recent popularity is mainly due to the work of Juergen Schmidhuber, Sepp Hochreiter and Alex Graves [56], [57]. Their applications are extremely diverse, ranging from speech recognition to driverless cars. All the networks we have seen so far were feedforward Neural Networks. In a feedforward Neural Network, signals flow in a single direction, from input to output, one layer at a time. In a recurrent network, the output of one layer is added to the next input and fed back to the same layer. Unlike feedforward networks, a recurrent network can receive a sequence of values as input and can also produce a sequence of values as output. RNNs have a ”memory” that holds all the information about what was calculated. It uses the same parameters for each input because it performs the same task on all inputs or hidden layers to produce the output. This reduces the complexity of the parameters, unlike other Neural Networks [58].
The mathematical description of the memory transfer process is as follows:

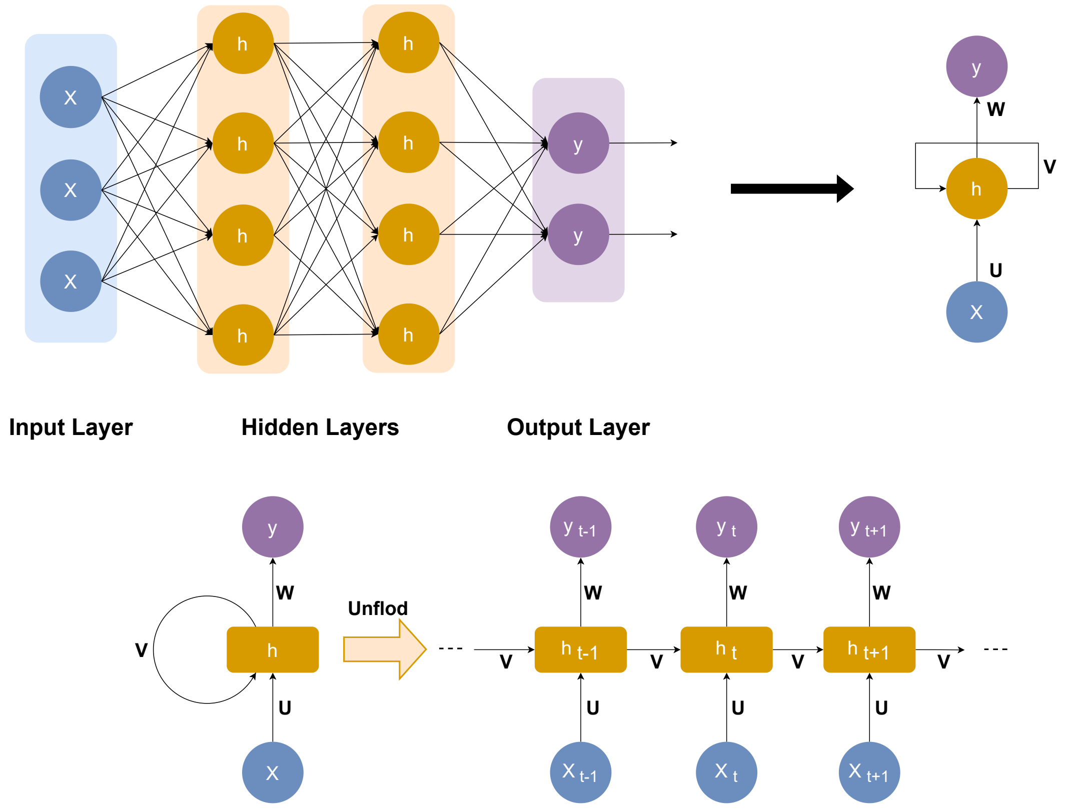


Figure 3.31: RNN architecture.

$$h_t = \varphi(Wx_t + Uh_{\text{t-1}}) \tag{3.11}$$

where:

- $\boldsymbol{h_t}$ is the hidden state at time t.
- $\boldsymbol{W}$ is the weight matrix.
- $\boldsymbol{x_t}$ is the input at the same time t.
- $\boldsymbol{h_{t\text{-}1}}$ is the hidden state at time t-1.
- $\boldsymbol{U}$ is a transition matrix.
- φ is an activation function.

In general, an RNN is an extremely difficult network to train. Since these networks use backpropagation, we run into the problem of the vanishing gradient (as there are more layers in the network, the value of the product of derivatives decreases until at some point the partial derivative of the loss function approaches a value close to zero, and the partial derivative vanishes). Unfortunately, the vanishing gradient is exponentially worse for an RNN. The reason is that each time step is equivalent to an entire layer in a feedforward network. Thus, training an RNN for 100 time steps is equivalent to training a 100-layer feedforward network, which leads to exponentially small gradients and decreasing information over time.

There are several ways to solve this problem, the most popular of which is gating, a technique that helps the network decide when to forget the current input and when to retain it for future steps. One of the most common types of gating today is the Long Short Term Memory (LSTM).

3.6 Long Short Term Memory

A common LSTM unit consists of different memory blocks called cells, an input gate, an output gate and a forget gate. The information is retained by the cells and the memory manipulations are performed by the gates (see Figure 3.32).

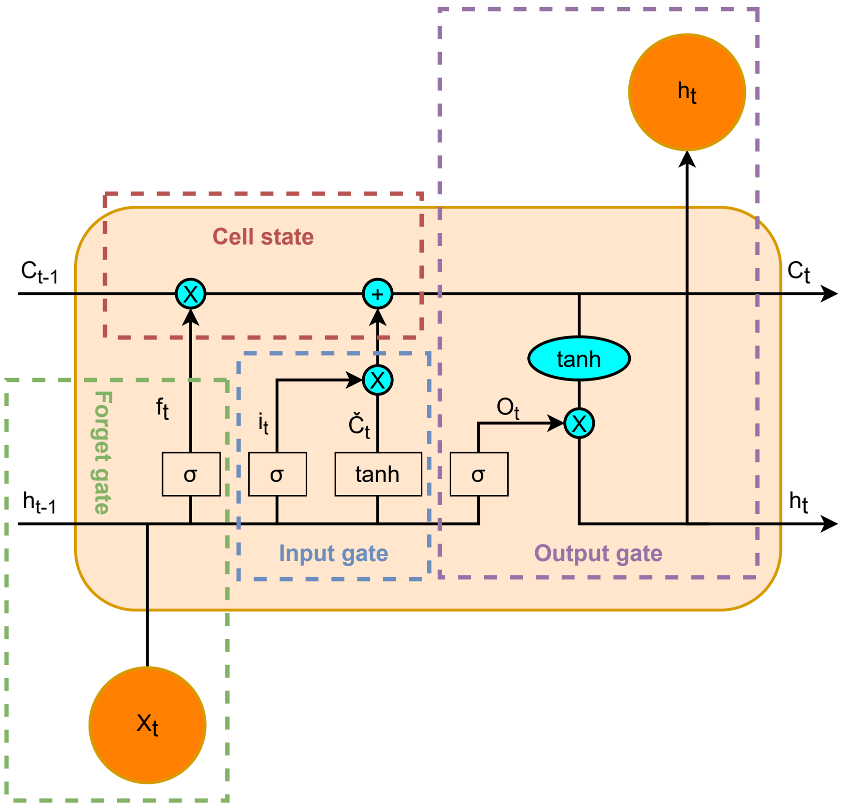


Figure 3.32: Structure of LSTM.

1. **Forget gate:** decides which information can be ignored and which should be retained for future use. Information from the current input X_t and the hidden state $h_{t\text{-}1}$ is passed to the gate and multiplied by weight matrices W_f followed by the addition of a bias b_f. The result is passed to an activation function σ which gives a binary output. If for a particular state of the cell the output is

0, the information is forgotten and for output 1, the information is kept for future use.

$$f_t = \sigma(W_f \cdot [h_{\text{t-1}}, X_t] + b_f) \tag{3.12}$$

2. **Input gate:** is responsible for adding useful information to the cell state. First, the information is regulated using the sigmoid function σ and filters the values to be stored in a similar way to the forget gate using the inputs $h_{\text{t-1}}$ and x_t. Then, a vector is created $\tilde{C}$ using the *tanh* function that gives an output from -1 to +1, which contains all possible values of $h_{\text{t-1}}$ and x_t. The output values generated by the activation functions are ready for point-to-point multiplication to obtain the useful information.

$$i_t = \sigma(W_i \cdot [h_{\text{t-1}}, X_t] + b_i) \tag{3.13}$$

$$\tilde{C} = tanh(W_c \cdot [h_{\text{t-1}}, X_t] + b_c) \tag{3.14}$$

3. **Output gate:** is responsible for extracting useful information from the current state of the cell and presenting it as an output. First, the values of the current state x_t and previous hidden state $h_{\text{t-1}}$ are passed into the third sigmoid function σ. Then the new cell state generated from the cell state is passed through the *tanh* function. Both these outputs are multiplied point-by-point. Based upon the final value, the network decides which information the hidden state should carry. This hidden state is used for prediction [59].

$$o_t = \sigma(W_o \cdot [h_{\text{t-1}}, X_t] + b_o) \tag{3.15}$$

$$h_t = o_t \times tanh(C_t) \tag{3.16}$$

3.7 AutoEncoders

The idea of AutoEncoders (AEs) has been part of the historical landscape of Neural Networks for decades (LeCun [60], 1987; Bourlard and Kamp [61], 1988; Hinton and Zemel [62], 1994). Basically, they are end-to-end networks that are used to compress input data. They transform data from a higher dimensional space to a lower dimensional space, essentially performing compression which is a way to reduce dimensionality and extract meaningful information.

AutoEncoders consist of 3 parts:

- **Encoder:** compresses the data into a lower dimensional representation (also called the latent space).
- **Bottleneck:** ensures a lower dimensional representation of the original data.
- **Decoder:** decompresses the representation back to the original domain.

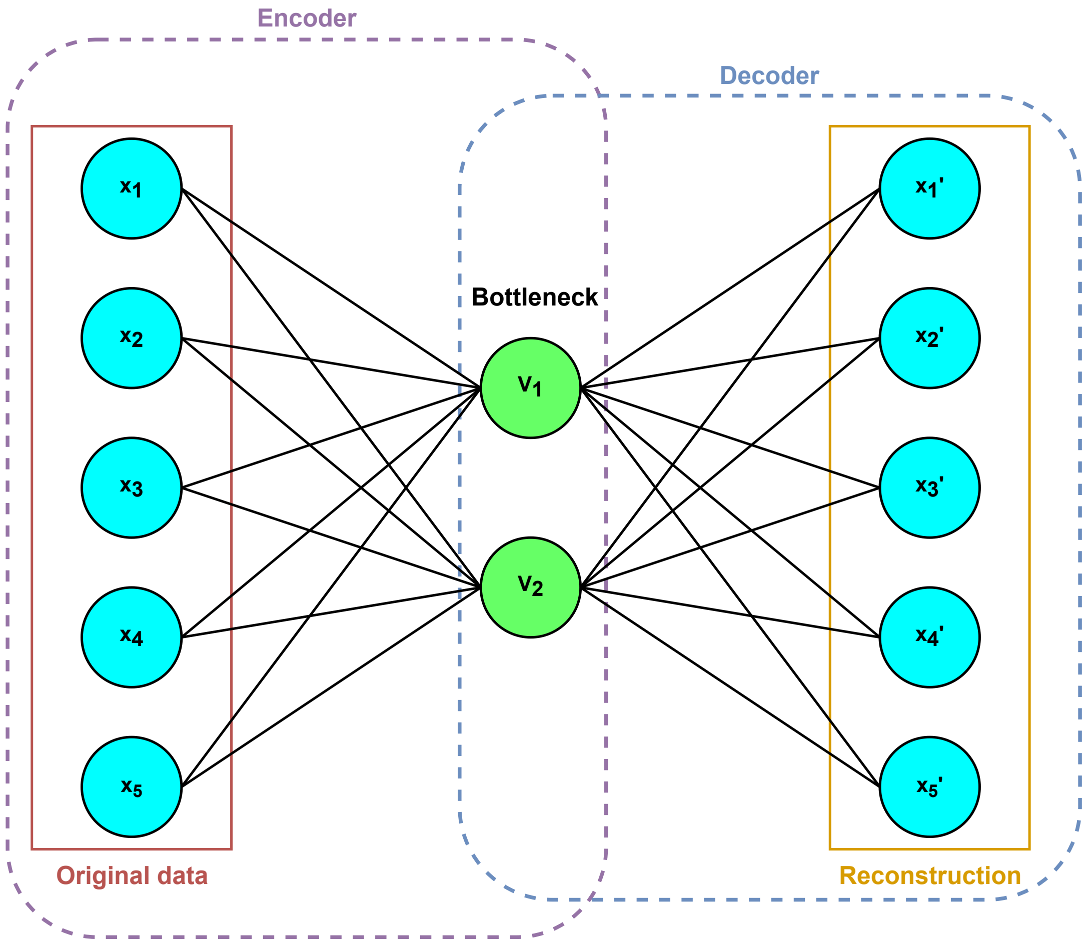


Figure 3.33: Architecture of an AE.

3.8 Metrics for evaluating the model's performance

In this section, we'll define the primary building blocks of the metrics we'll use to evaluate classification models.

3.8.1 Confusion matrix

An NxN matrix that groups a classification model's correct and incorrect guesses. A confusion matrix has one axis for the label predicted by the model and another for the ground truth. N represents the number of classes. For example, N=2 for a binary classification model (see Figure 3.34).

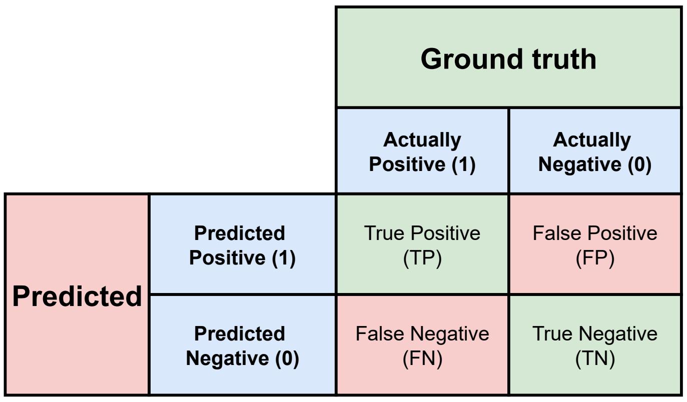


Figure 3.34: Confusion matrix.

Each cell in the confusion matrix represents an evaluation factor. Let's understand these factors one by one:

- **True Positive (TP):** an outcome in which the model predicts that an observation belongs to a class and the observation actually belongs to that class.
- **True Negative (TN):** an outcome in which the model predicts that an observation does not belong to a class and it actually does not belong to that class.
- **False Positive (FP):** an outcome in which the model predicts that an observation belongs to a class when it actually does not.
- **False Negative (FN):** an outcome in which the model predicts that an observation does not belong to a class when it actually does.

3.8.2 Accuracy

The percentage of correct predictions made by a classification model. Formally, accuracy is defined as follows:

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}}$$

For binary classification, accuracy can also be calculated in terms of positives and negatives as follows:

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}$$

3.8.3 Precision

Identifies how often a model was correct when predicting the positive class.

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

3.8.4 Recall or Sensitivity or TPR

It is the number of correct positive results divided by the number of all relevant samples (all samples that should have been identified as positive).

$$\text{Recall} = \text{True Positive Rate (TPR)} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

3.8.5 Specificity or TNR

Percentage of negative instances out of the total actual negative instances. It is similar to recall but the shift is on the negative instances.

$$\text{Specificity} = \text{True Negative Rate (TNR)} = \frac{\text{TN}}{\text{TN} + \text{FP}}$$

3.8.6 FPR

Corresponds to the proportion of negative data points that are mistakenly considered as positive, with respect to all negative data points.

$$\text{False Positive Rate (FPR)} = \frac{\text{FP}}{\text{FP} + \text{TN}}$$

3.8.7 FNR

Corresponds to the proportion of positive data points that are mistakenly considered as negative, with respect to all positive data points.

$$\text{False Negative Rate (FNR)} = \frac{\text{FN}}{\text{FN} + \text{TP}}$$

3.8.8 F1 score

The harmonic mean of the precision and recall, where an F1 score reaches its best value at 1 (perfect precision and recall) and worst at 0.

$$\text{F1 score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

3.8.9 ROC curve

Receiver Operating Characteristic (ROC) curves summarize the trade-off between the true positive rate and false positive rate for a predictive model using different probability thresholds.

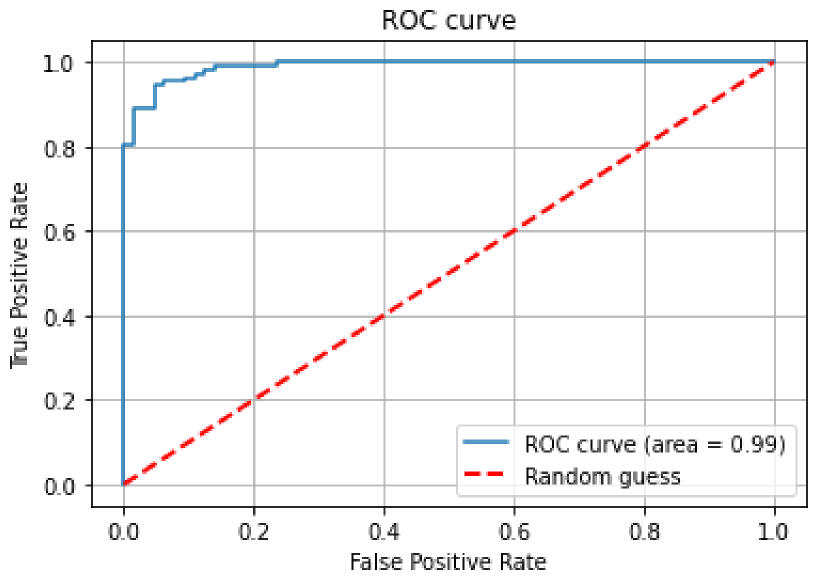


Figure 3.35: ROC curve.

3.9 Conclusion

In this third chapter, we briefly defined machine learning and supervised learning, which served as a basis for a better understanding of Deep Learning. Today, DL has proven to be very effective and has led to significant advances in many domains. Although it was first introduced back in the 1940s, it is only very recently that we have been able to fully exploit this field with the help of powerful computing resources, as well as the appearance of large databases.
This chapter will give us the opportunity to put into practice what we have learned in the theoretical part. The next chapter consists of creating a model capable of identifying and classifying a wide range of attacks using ML and DL algorithms.

Chapter 4

Network anomaly detection, tests and results

4.1 Introduction

The performance of Machine Learning-based intrusion detection systems is highly dependent on the dataset used for model development. In this work, we used CIC-IDS2017, a dataset provided by the Canadian Cybersecurity Institute (CIC), to detect and classify a wide range of attacks such as Denial of Service, PortScan, Web Attacks and many other widespread attacks. We implemented 4 ML algorithms (Random Forest (RF), Decision Tree (DT), Support Vector Machine (SVM), and K-Nearest Neighbors (KNN)) and one DL algorithm (Multi-Layer Perceptron (MLP)) and compared their performance using various evaluation measures (Accuracy, Precision, Recall, F1 score).

4.2 Execution environment

Machine Learning is a field that requires the availability of hardware resources (especially GPUs) capable of performing intense calculations. To carry out our work, we have chosen to work in an experimental environment with the following characteristics:

- **OS:** 64-bit Windows 10.
- **CPU(Intel):** Core i5-8300H @2.30GHz.
- **GPU(NVIDIA):** GeForce GTX 1050.
- **RAM:** 8GB.
- **SSD:** 512GB.

On the software side, we started by setting up a local Python development environment using the **Anaconda** [63] platform, which is a distribution of the Python and R programming languages for scientific computing, which aims to simplify package management and deployment.
For the programming language we chose **Python** [64] (3.9.12), a high-level, interpreted, open source programming language that offers an excellent approach to object-oriented programming. It is the most common and popular language for ML, used by data scientists for various data science projects and applications, thanks to its flexibility and the large number of open source software libraries it supports, such as Numpy, Pandas, Matplotlib, Scikit-Learn, etc.
We also used the **Jupyter Notebook** [65] development environment (6.4.11) to write and execute python codes.

Library	**Description**	**Version**
Numpy [66]	A Python library that provides mathematical function to handle large dimension array. It provides various method/function for Array, Metrics, and linear algebra.	1.21.5
Pandas [67]	A fast, powerful, flexible and easy to use open source data analysis and manipulation tool, built on top of the Python programming language.	1.4.2
Scikit-learn [68]	Is a free software machine learning library for the Python programming language. It features various classification, regression and clustering algorithms.	1.0.2
Matplotlib [69]	Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python.	3.5.2

Table 4.1: List of python libraries used.

4.3 Dataset

The dataset chosen for this study is the CIC-IDS2017 dataset, provided by the Canadian Institute for Cybersecurity (CIC) [70], it covers the eleven criteria necessary to build a reliable benchmark dataset that none of the previous datasets could cover [71]. Generating realistic background traffic was the main priority in building this dataset and was achieved by abstracting the behavior of human interactions via the B-Profile system [72]. The dataset is fully annotated and has more than 80 network traffic features extracted for each NetFlow observation and evaluated for all benign and invasive flows using CICFlowMeter software which is freely obtainable from the Canadian Cybersecurity Institute website [73].

4.3.1 Descriptions of CIC-IDS2017 dataset

According to the author [72] of CIC-IDS2017, the data capture period started at 9:00 am on Monday, July 3, 2017 and ended at 5:00 pm on Friday, July 7, 2017, a total of 5 days. The dataset spanned over eight different files, a brief description of all the files is presented in Table 4.2.

Name of Files	**Day Activity**	**Size (MB)**	**Attacks found**
Monday-WorkingHours .pcap_ISCX.csv	Monday	262.354 MB	Benign (Normal human activities)
Tuesday-WorkingHours .pcap_ISCX.csv	Tuesday	170.603 MB	Benign, FTP-Patator, SSH-Patator
Wednesday-WorkingHours .pcap_ISCX.csv	Wednesday	278.949 MB	Benign, DoS GoldenEye, DoS Hulk, DoS Slowhttptest, DoS Slowloris, Heartbleed
Thursday-WorkingHours-Morning -WebAttacks.pcap_ISCX.csv	Thursday	89.874 MB	Benign, Web Attack – Brute Force, Web Attack – Sql Injection, Web Attack – XSS
Thursday-WorkingHours-Afternoon -Infilteration.pcap_ISCX.csv	Thursday	106.175 MB	Benign, Infiltration
Friday-WorkingHours-Morning .pcap_ISCX.csv	Friday	73.620 MB	Benign, Bot
Friday-WorkingHours-Afternoon -PortScan.pcap_ISCX.csv	Friday	99.488 MB	Benign, PortScan
Friday-WorkingHours-Afternoon -DDos.pcap_ISCX.csv	Friday	95.019 MB	Benign, DDoS

Table 4.2: Description of files containing CIC-IDS2017 dataset.

The dataset contains 2830743 instances and 85 features containing 15 class labels (1 normal + 14 attack labels) [74]. The characteristics of combined dataset and the detailed class wise occurrence has been presented in Table 4.3 and Table 4.4.

Dataset Name	CIC-IDS2017
Dataset Type	Multi class
Year of release	2017
Total number of instances	2830743
Number of features	85
Number of distinct classes	15

Table 4.3: Overall characteristics of CIC-IDS2017 dataset.

Class Labels	**Number of instances**
BENIGN	2359289
DoS Hulk	231073
PortScan	158930
DDoS	41835
DoS GoldenEye	10293
FTP-Patator	7938
SSH-Patator	5897
DoS Slowloris	5796
DoS Slowhttptest	5499
Bot	1966
Web Attack - Brute Force	1507
Web Attack - XSS	652
Infiltration	36
Web Attack - Sql Injection	21
Heartbleed	11

Table 4.4: Class wise instance occurrence of CIC-IDS2017 dataset.

Features	**Type**	**Features**	**Type**
Flow ID	int64	Bwd Packets/s	float64
Source IP	int64	Min Packet Length	float64
Source Port	int64	Max Packet Length	float64
Destination IP	int64	Packet Length Mean	float64
Destination Port	int64	Packet Length Std	float64
Protocol	int64	Packet Length Variance	float64
Timestamp	int64	FIN Flag Count	int64
Flow Duration	int64	SYN Flag Count	int64
Total Fwd Packets	int64	RST Flag Count	int64
Total Backward Packets	int64	PSH Flag Count	int64
Total Length of Fwd Packets	float64	ACK Flag Count	int64
Total Length of Bwd Packets	float64	URG Flag Count	int64
Fwd Packet Length Max	float64	CWE Flag Count	int64
Fwd Packet Length Min	float64	ECE Flag Count	int64
Fwd Packet Length Mean	float64	Down/Up Ratio	float64
Fwd Packet Length Std	float64	Average Packet Size	float64
Bwd Packet Length Max	float64	Avg Fwd Segment Size	float64
Bwd Packet Length Min	float64	Avg Bwd Segment Size	float64
Bwd Packet Length Mean	float64	Fwd Avg Bytes/Bulk	int64
Bwd Packet Length Std	float64	Fwd Avg Packets/Bulk	int64
Flow Bytes/s	int64	Fwd Avg Bulk Rate	int64
Flow Packets/s t	int64	Bwd Avg Bytes/Bulk	int64
Flow IAT Meant	float64	Bwd Avg Packets/Bulk	int64
Flow IAT Std	float64	Bwd Avg Bulk Rate	int64
Flow IAT Max	float64	Subflow Fwd Packets	int64
Flow IAT Min	float64	Subflow Fwd Bytes	int64
Fwd IAT Total	float64	Subflow Bwd Packets	int64
Fwd IAT Mean	float64	Subflow Bwd Bytes	int64
Fwd IAT Std	float64	Init_Win_bytes_forward	int64
Fwd IAT Max	float64	Init_Win_bytes_backward	int64
Fwd IAT Min	float64	act_data_pkt_fwd	int64
Bwd IAT Total	float64	min_seg_size_forward	int64
Bwd IAT Mean	float64	Active Mean	float64
Bwd IAT Std	float64	Active Std	float64
Bwd IAT Max	float64	Active Max	float64
Bwd IAT Min	float64	Active Min	float64
Fwd PSH Flags	int64	Idle Mean	float64
Bwd PSH Flags	int64	Idle Std	float64
Fwd URG Flags	int64	Idle Max	float64
Bwd URG Flags	int64	Idle Min	float64
Fwd Header Length	int64	External IP	object
Bwd Header Length	int64	Label	object
Fwd Packets/s	float64		

Table 4.5: All 85 features present in the CIC-IDS2017 dataset.

4.4 Implementation

A summary of the work done in this chapter is represented in Figure 4.1.

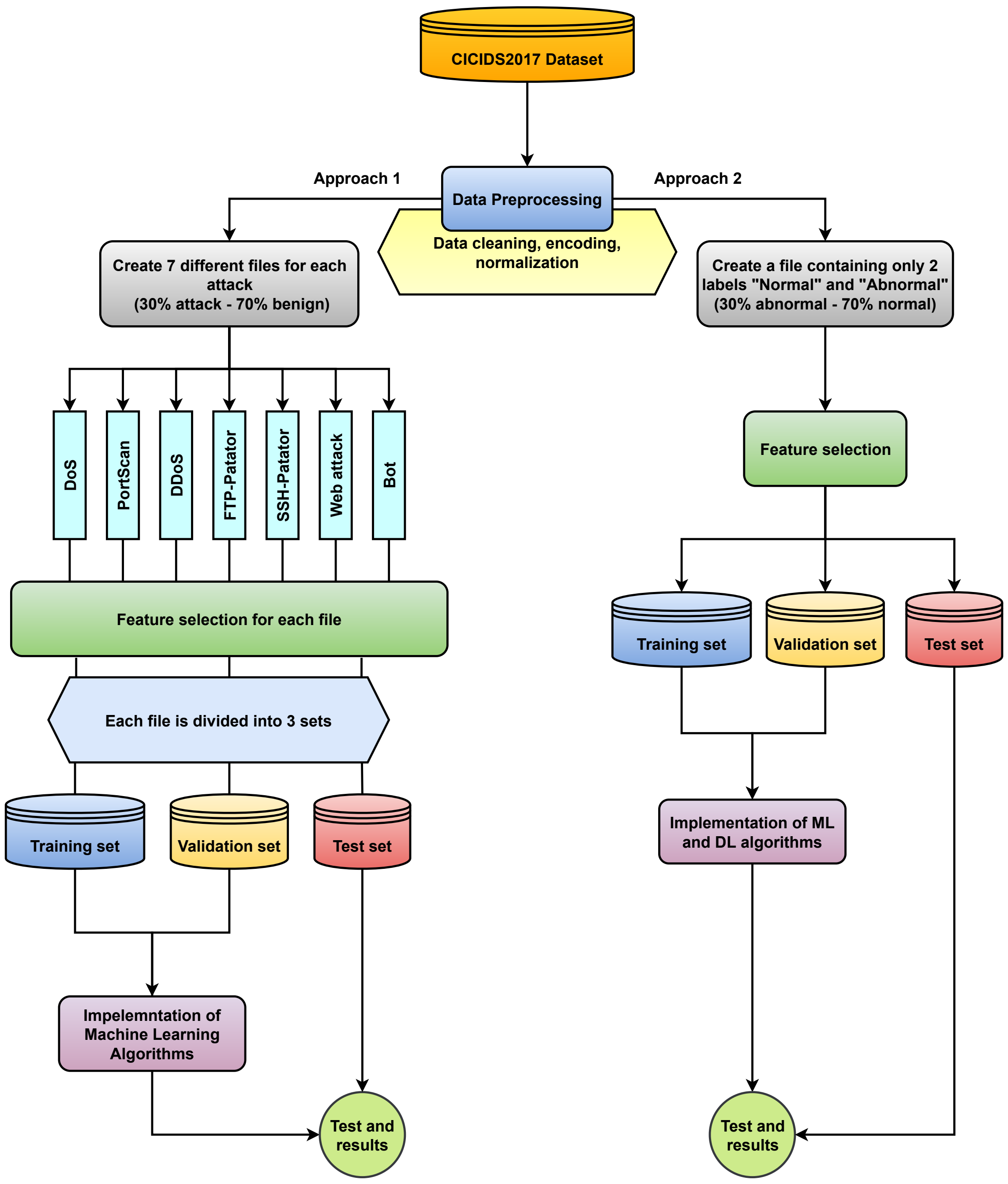

CICIDS2017 Dataset
Data Preprocessing
Data cleaning, encoding, normalization
Approach 1
Approach 2
Create 7 different files for each attack (30% attack - 70% benign)
Create a file containing only 2 labels "Normal" and "Abnormal" (30% abnormal - 70% normal)
DoS
PortScan
DDoS
FTP-Patator
SSH-Patator
Web attack
Bot
Feature selection for each file
Each file is divided into 3 sets
Training set
Validation set
Test set
Impelemntation of Machine Learning Algorithms
Test and results
Feature selection
Training set
Validation set
Test set
Implementation of ML and DL algorithms
Test and results

Figure 4.1: Summary of the work conducted.

4.5 Data pre-processing

Machine Learning algorithms learn from the data that is provided to them, therefore if this data is of poor quality, i.e. incorrect, corrupted, incorrectly formatted, duplicate, or incomplete, then the resulting algorithm will itself be quite bad since it is supposed to reflect what it sees in the data. For this reason, It is imperative to pre-process our data before feeding it to the learning model and this phase is known as data pre-processing.
We have seen in Table 4.2 that the data of CIC-IDS2017 dataset is divided into 8 different files and because processing each file individually is a tedious task, we decided to facilitate the processing of our data by merging all 8 files into a single CSV file called "all_data.csv" which contains a total of 2830743 instances.

4.5.1 Data cleaning

Following an examination of our dataset, we proceeded to remove all missing values, such as **NaN** (Not A Number) and **INF** (Infinity) values, as well as all redundant rows, as these can have a negative impact on the performance of our model.
Categorical and string values cannot be interpreted by our model as it can only learn from numerical values. Therefore, another modification was made to our dataset by transforming all strings and categorical values present in the features (*Flow ID, Source IP, Destination IP, Timestamp, External IP, Flow Bytes/s, Flow Packets/s*) into numeric values using the Label Encoder [75] module from Scikit-learn library.

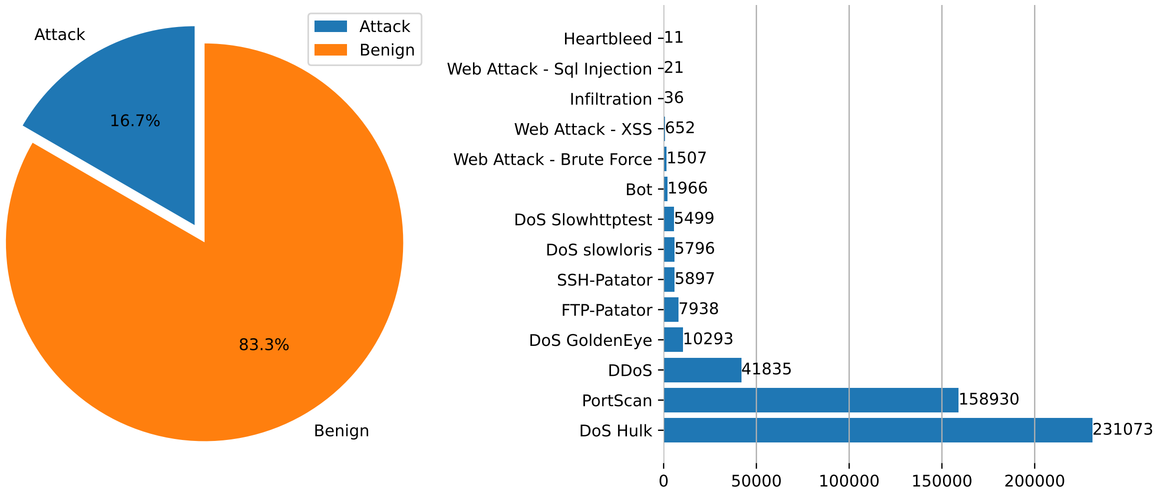


Figure 4.2: The distribution of data flow and attack types in the dataset.

After examining the number of attacks in the dataset shown in Figure 4.2, we can clearly see that there is a huge imbalance in the distribution that is due to the nature of these attacks. During the attack, DoS and PortScan attacks generate an excessive amount of data and packet flow. Therefore, the increase in traffic from normal use and other types of attacks is quite natural.

4.5.1.1 First approach

In this approach, we will mainly focus on 7 types of attacks (DoS, PortScan, DDoS, FTP-Patator, SSH-Patator, Bot and Web attack). Due to the low number of instances in the remaining attacks, it would be quite difficult to develop a model that can effectively detect and classify these attacks, as ML algorithms require a good amount of data to achieve the best results. Therefore, we decided to combine all the different types of DoS and Web attacks into two types of attacks labeled "DoS" and "Web Attack" respectively (see Table 4.6) and discard the rest of the attacks.
Next, we generated 7 new CSV files, each containing a single type of attack and benign stream. However, we did not use the entire benign traffic (2359289 instances) for each attack file in order to avoid having a huge imbalance between attack and normal traffic, which makes it harder for the model to converge in the learning phase. Instead, we used a ratio of (attack=30%, benign=70%) in each attack file. The results are shown in Table 4.7.

Class Labels	**New Labels**	**Number of instances**
DoS Hulk DoS GoldenEye DoS slowloris DoS Slowhttptest	DoS	252660
Web Attack - Brute Force Web Attack - XSS Web Attack - Sql Injection	Web	2180

Table 4.6: Merging all types of DoS and Web attacks into two class labels "DoS" and "Web".

CSV files	**Number of attack instances (30%)**	**Number of benign instances (70%)**	**Files Size (MB)**
"DoS.csv"	252660	589540	359.629 MB
"PortScan.csv"	158930	370836	198.964 MB
"DDoS.csv"	41835	97615	54.594 MB
"FTP-Patator.csv"	7938	18522	10.279 MB
"SSH-Patator.csv"	5897	13759	7.489 MB
"Web.csv"	2180	5086	2.810 MB
"Bot.csv"	1966	4587	2.453 MB

Table 4.7: The 7 different attack files that were used to train the models in the first approach.

4.5.1.2 Second approach

In this second approach, we created a new csv file with only two label classes: "Normal" and "Abnormal". The "Normal" label class contains only benign traffic, whereas the "Abnormal" label class contains all 14 attacks.

Class Labels	**New Labels**	**Number of instances**
BENIGN	Normal	2359289
DoS Hulk PortScan DDoS DoS GoldenEye FTP-Patator SSH-Patator DoS slowloris DoS Slowhttptest Bot Web Attack - Brute Force Web Attack - XSS Infiltration Web Attack - Sql Injection Heartbleed	Abnormal	471454

Table 4.8: Converting the entire dataset into two labels: "Normal" and "Abnormal".

CSV file	**Number of attack instances (30%)**	**Number of benign instances (70%)**	**File size (GB)**
"N_A.csv"	471454	1100059	1.1 GB

Table 4.9: The CSV file that was used to train the models in the second approach.

For each CSV file, the Label column, which represents the class of each instance, has been encoded using the Label Encoder module. The encoding converts each string value in the Label column to either 1 or 0. 1 means that the instance belongs to the class and 0 means otherwise.
Next, in order to enhance the performance and reliability of our ML model and help it converge quickly, we normalized our data by placing all quantitative variables on the same scale.

4.5.2 Creation of Training and Test Data

In Machine Learning, it is never appropriate to evaluate the performance of a model on the same data that was used for its training. The CIC-IDS2017 dataset contains a single unbundled dataset rather than dedicated training and test data.

As a result, the data must be divided into two parts: a Train set, whose data is used for training the model, and a Test set, reserved solely for evaluating the model's performance. The generally preferred partitioning is 20% test data, 80% training data. To accomplish this, we use the function train_test_split() [76] which comes from the module model_selection of Sklearn. This function randomly shuffles our dataset before splitting it into two parts (see Figure 4.3).

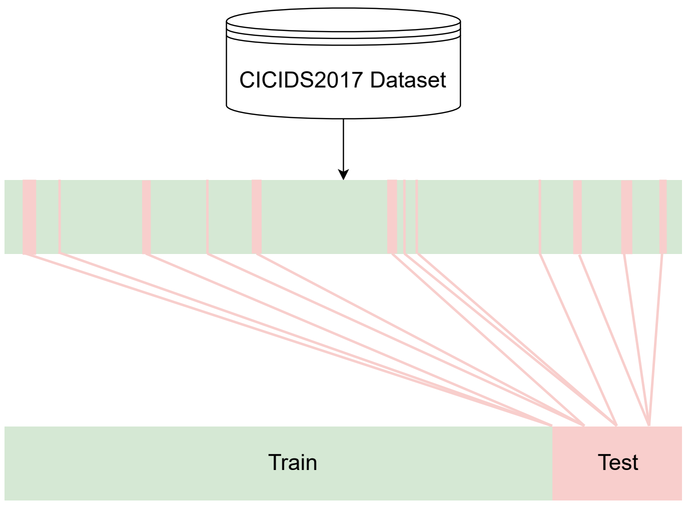


Figure 4.3: Splitting the CIC-IDS2017 dataset into two parts.

Some ML algorithms require an adjustment of their hyper-parameters in order to obtain the best results. The issue in this case is that by tweaking these parameters while looking for the best results, the model will eventually end up seeing all of the test data, making it unusable for the evaluation phase. For this reason, a third set must be cut from our dataset, called the validation set. This section allows us to search for the model parameters that give the best performance while keeping the test set data only for evaluation.
If we want to compare two ML models, for example a KNN with 3 neighbors and a KNN with 6 neighbors, we will first train these two models on the Train set, and then select the one with the best performance on the Validation set. We can then evaluate this model on the Test set to get an idea of its performance (as shown in Figure 4.4).

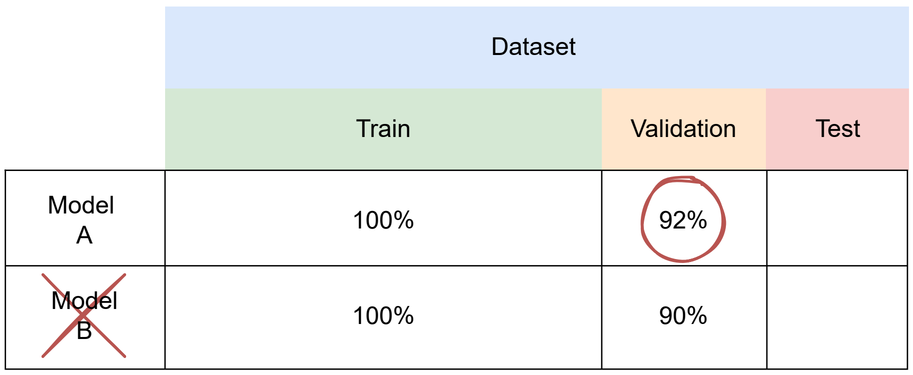


Figure 4.4: Performance comparison between two models using the validation set.

The problem with the validation set is that there is no guarantee that our data is split in the right way. If this is the case, training and validating our two models on another portion of the data may reveal that in fact model B is better. In this case, there is a solution known as Cross-Validation.

4.5.2.1 Cross-Validation

Cross-Validation consists in training and validating our model on various portions of the Train set. For example, by dividing the Train set into five parts, we can train our model on the first four and validate it on the fifth. The process will then be repeated for all possible configurations. At the end, we will calculate the average of the five scores obtained, and when comparing two models, we will be confident in selecting the one with the best overall performance (see Figure 4.5).

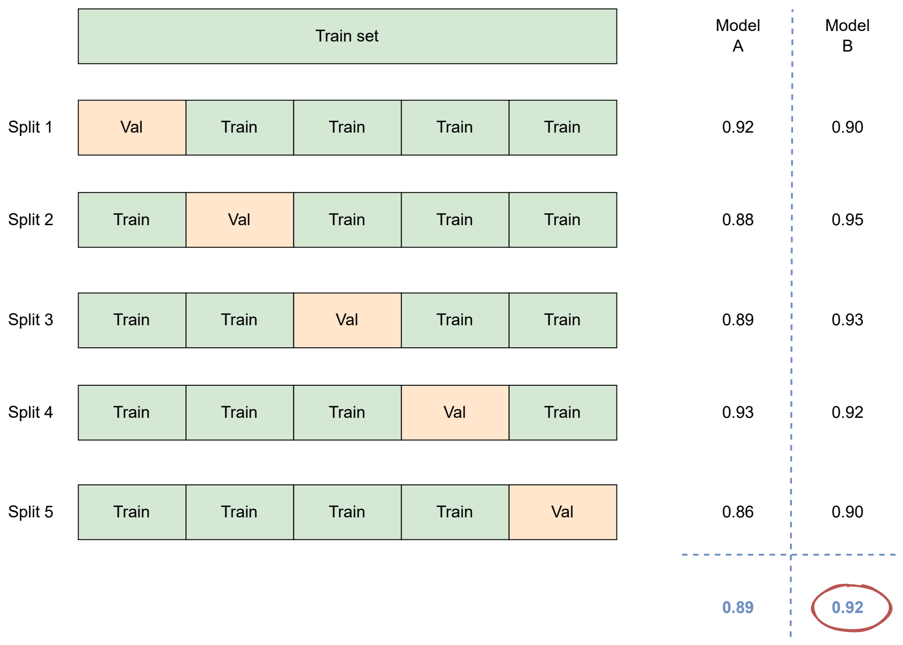


Figure 4.5: Performance comparison between two models using the Cross-Validation method.

A ML algorithm will only be able to learn if the training data contains enough relevant features and not too many irrelevant features. A critical part of a successful ML project is finding a suitable set of features to train on [42]. This process is called **Feature selection**.

4.5.3 Feature selection

Feature selection is one of the most crucial phases during model development. It is the process of selecting the most important features to input into ML algorithms. Feature selection techniques are used to reduce the number of input variables by removing redundant and irrelevant features, which can negatively impact the performance, accuracy and computational cost of the learning algorithm.

4.5.3.1 Feature Selection According to Attack Types

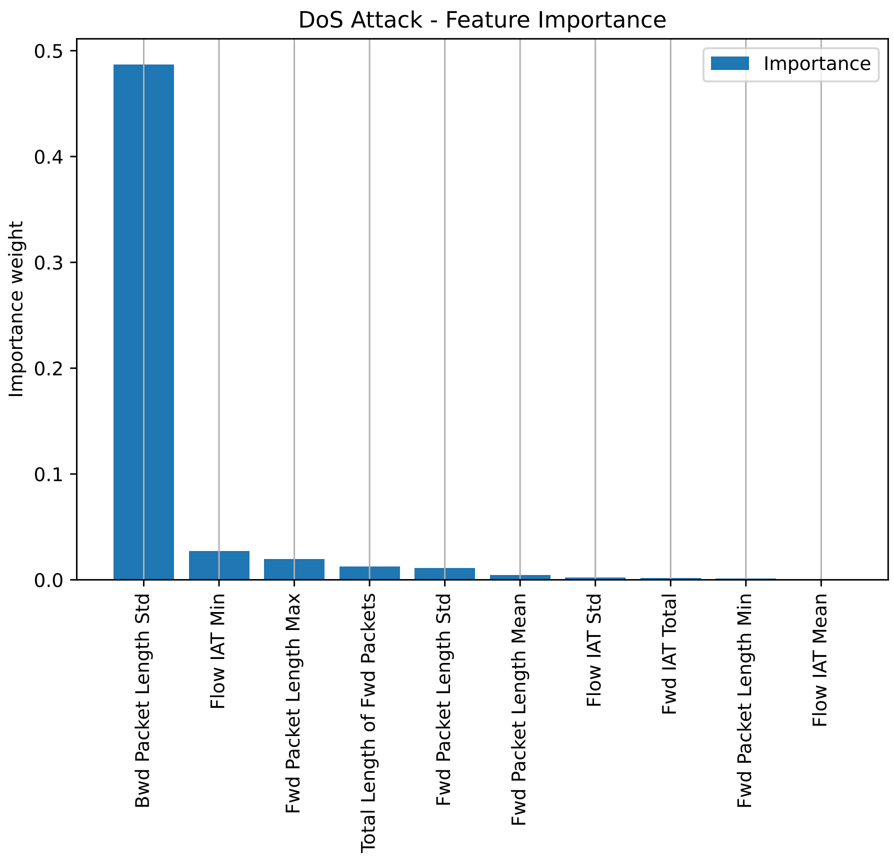
In order to determine which features are important in defining an attack, we used the Random Forest Regressor [77] module of Sklearn. This is an algorithm that creates a Decision Forest where each feature is given an importance weight based on its utility in building the Decision Tree. When the process is complete, these feature importance weights are compared and ranked. The sum of the importance weights of all the features gives the total importance weight of the Decision Tree. Comparing the score of a feature with the score of the entire tree gives information about the importance of that feature in the Decision Tree [8].

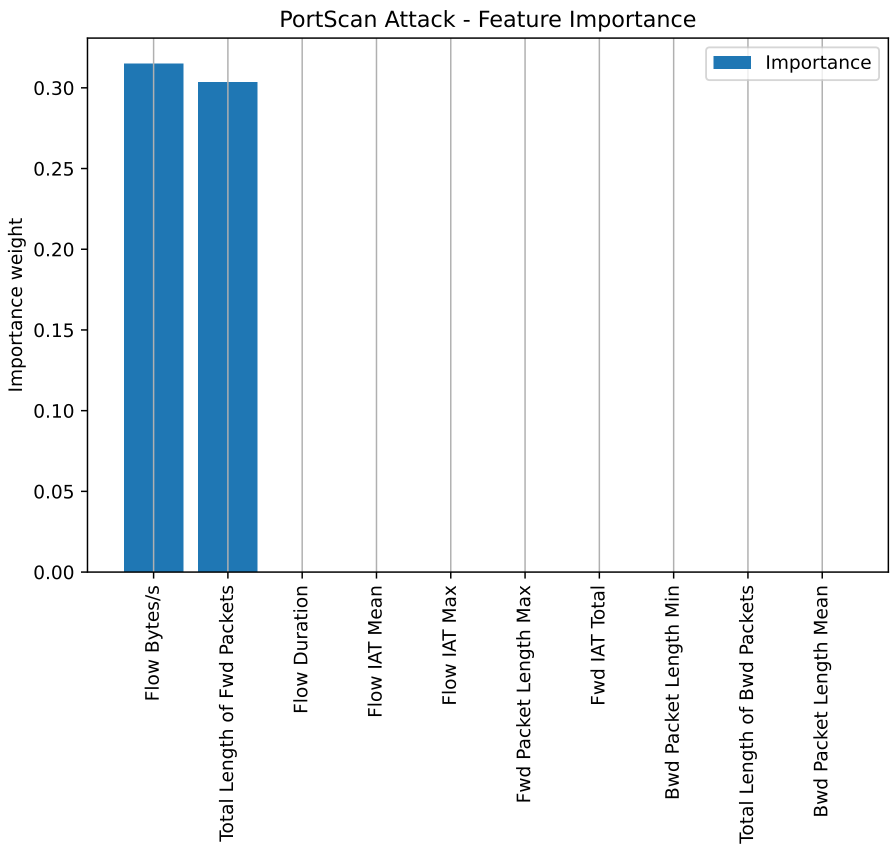
Features such as (*Flow ID, Source IP, Source Port, Destination IP, Destination Port, Protocol, Timestamp, External IP*) were excluded from the calculation because they are related to connection information and do not represent properties of any specific attack [78]. The 5 features with the highest importance weight for each attack are listed in Table 4.10.

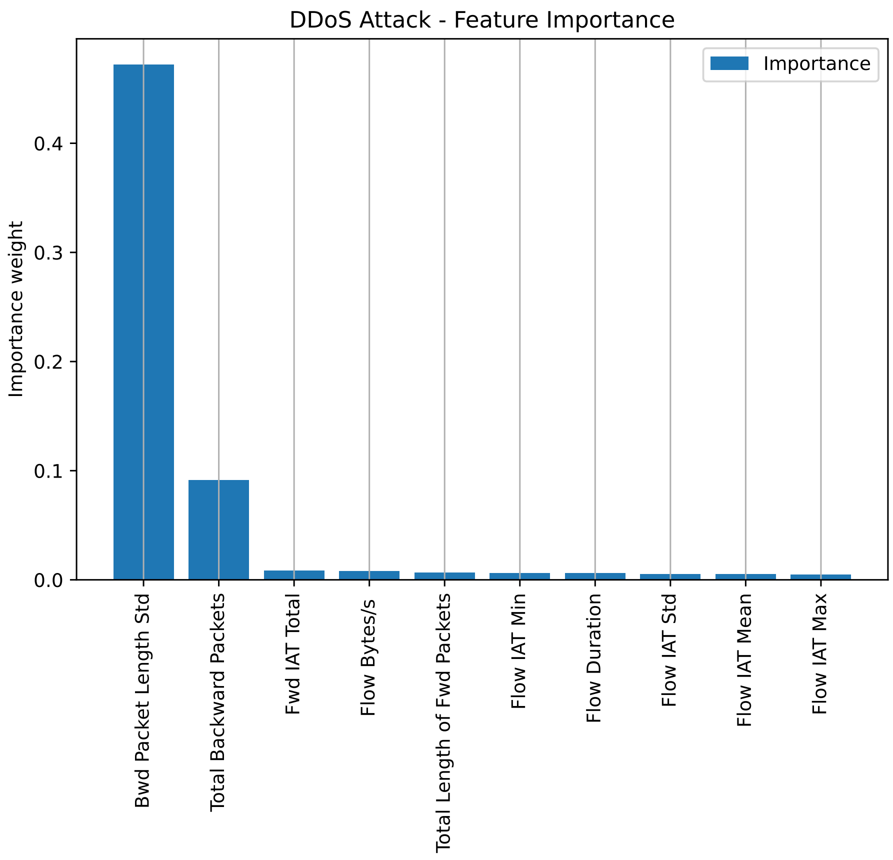
Attack / Feature name	**Importance Weight**	**Attack / Feature name**	**Importance Weight**
DoS attack		**DDoS attack**	
Bwd Packet Length Std	0.486887	Bwd Packet Length Std	0.472082
Flow IAT Min	0.026953	Total Backward Packets	0.091310
Fwd Packet Length Max	0.019744	Fwd IAT Total	0.008562
Total Length of Fwd Packets	0.012743	Flow Bytes/s	0.008193
Fwd Packet Length Std	0.010881	Total Length of Fwd Packets	0.006510
PortScan attack		**FTP-Patator attack**	
Flow Bytes/s	0.315061	Fwd Packet Length Max	0.388980
Total Length of Fwd Packets	0.303497	Fwd Packet Length Std	0.027879
Flow Duration	0.000368	Fwd Packet Length Mean	0.014869
Flow IAT Mean	0.000234	Bwd Packet Length Std	0.000242
Flow IAT Max	0.000205	Bwd Packet Length Mean	0.000213
SSH-Patator		**Web attack**	
Fwd Packet Length Max	0.000887	Bwd Packet Length Std	0.004635
Total Length of Fwd Packets	0.000565	Total Length of Fwd Packets	0.003736
Flow IAT Mean	0.000556	Total Fwd Packets	0.002832
Flow Duration	0.000516	Flow IAT Min	0.002257
Flow IAT Max	0.000509	Bwd Packet Length Max	0.001523

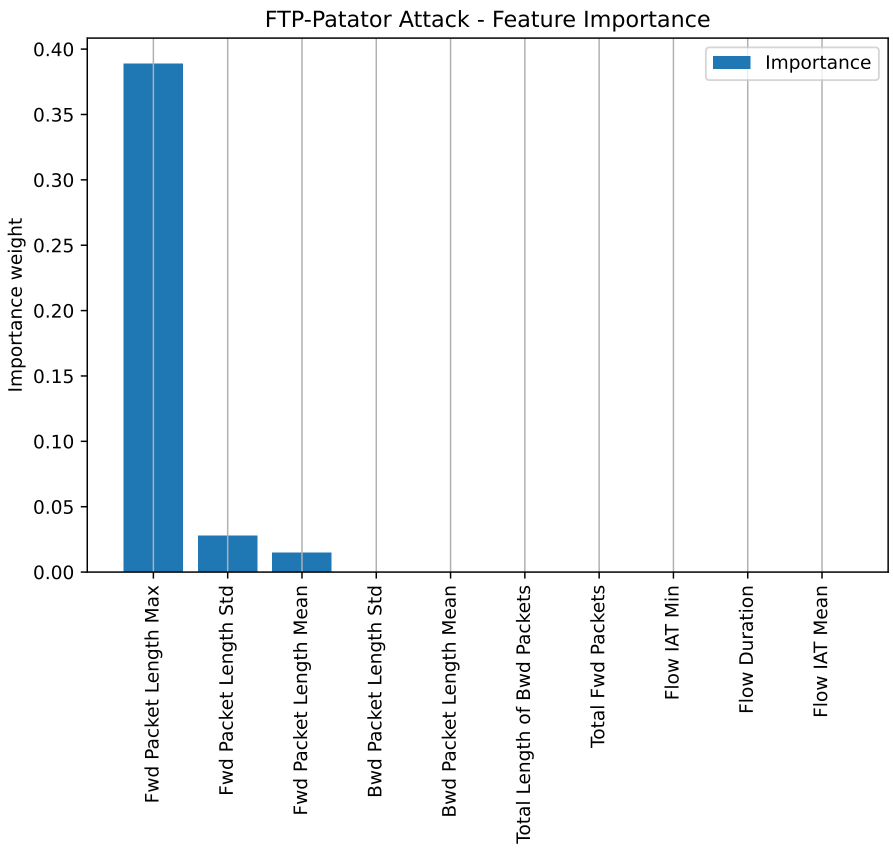
Bot attack	
Feature name	**Importance weight**
Bwd Packet Length Mean	0.358487
Flow IAT Min	0.027334
Flow IAT Max	0.010331
Flow IAT Std	0.005678
Flow IAT Mean	0.004784

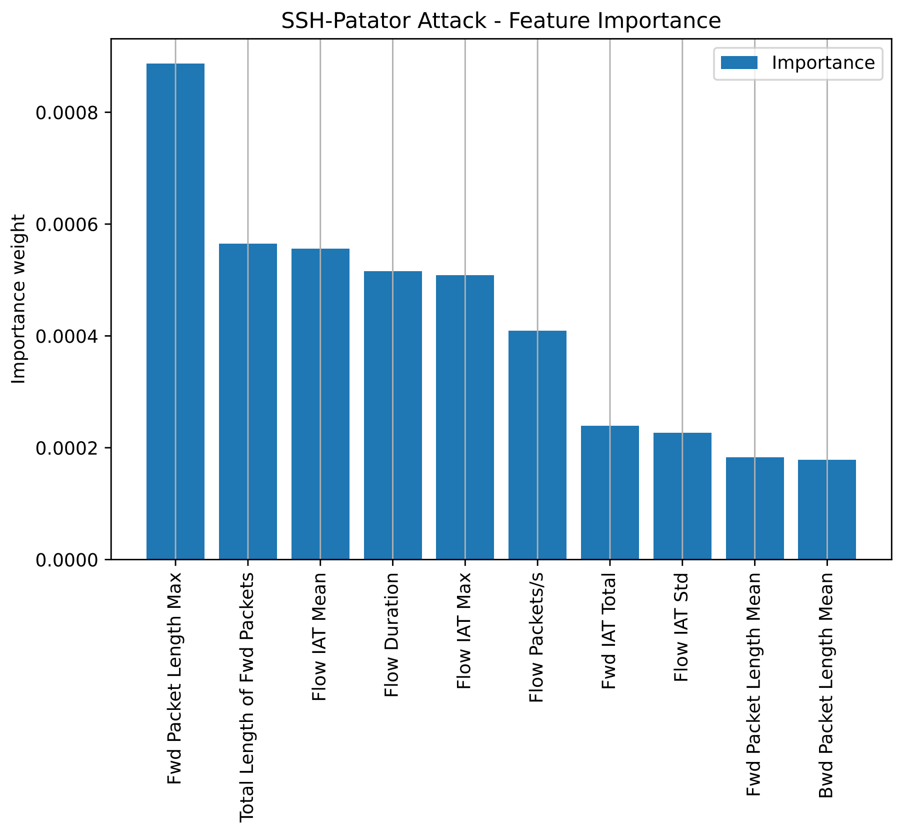
Table 4.10: List of the 5 most relevant features for each attack.

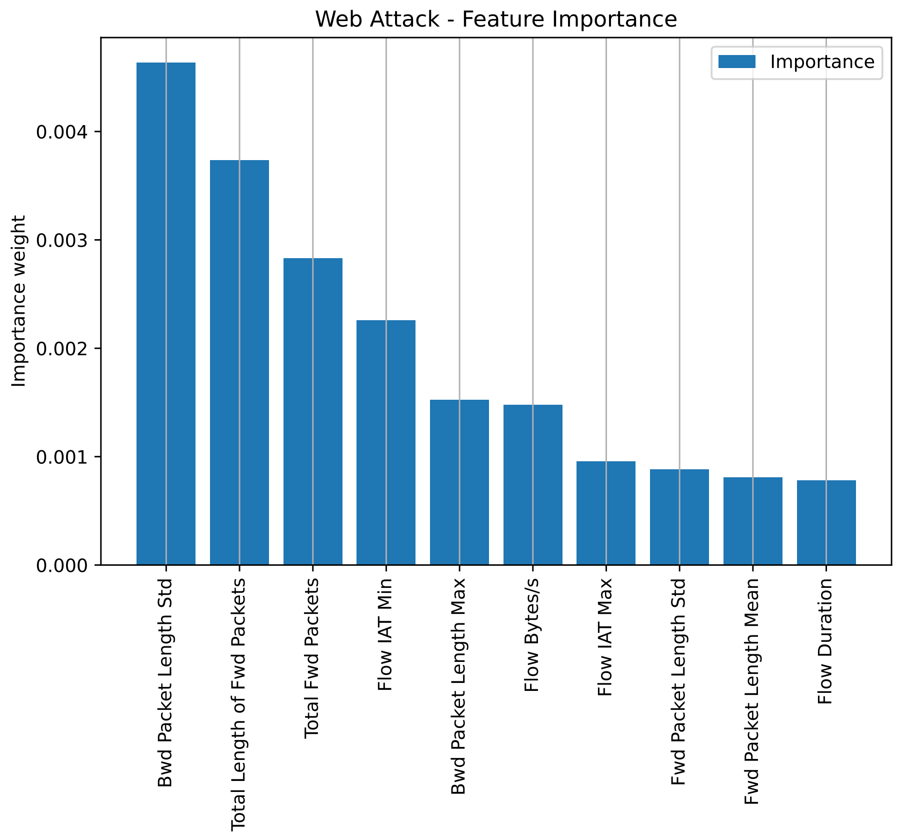

DoS Attack - Feature Importance
Importance
Importance weight
0.0
0.1
0.2
0.3
0.4
0.5
Bwd Packet Length Std
Flow IAT Min
Fwd Packet Length Max
Total Length of Fwd Packets
Fwd Packet Length Std
Fwd Packet Length Mean
Flow IAT Std
Fwd IAT Total
Fwd Packet Length Min
Flow IAT Mean


PortScan Attack - Feature Importance
Importance
Importance weight
0.00
0.05
0.10
0.15
0.20
0.25
0.30
Flow Bytes/s
Total Length of Fwd Packets
Flow Duration
Flow IAT Mean
Flow IAT Max
Fwd Packet Length Max
Fwd IAT Total
Bwd Packet Length Min
Total Length of Bwd Packets
Bwd Packet Length Mean


DDoS Attack - Feature Importance
Importance
Importance weight
0.0
0.1
0.2
0.3
0.4
Bwd Packet Length Std
Total Backward Packets
Fwd IAT Total
Flow Bytes/s
Total Length of Fwd Packets
Flow IAT Min
Flow Duration
Flow IAT Std
Flow IAT Mean
Flow IAT Max


FTP-Patator Attack - Feature Importance
Importance
Importance weight
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.35
0.40
Fwd Packet Length Max
Fwd Packet Length Std
Fwd Packet Length Mean
Bwd Packet Length Std
Bwd Packet Length Mean
Total Length of Bwd Packets
Total Fwd Packets
Flow IAT Min
Flow Duration
Flow IAT Mean


SSH-Patator Attack - Feature Importance
Importance
Importance weight
0.0000
0.0002
0.0004
0.0006
0.0008
Fwd Packet Length Max
Total Length of Fwd Packets
Flow IAT Mean
Flow Duration
Flow IAT Max
Flow Packets/s
Fwd IAT Total
Flow IAT Std
Fwd Packet Length Mean
Bwd Packet Length Mean


Web Attack - Feature Importance
Importance
Importance weight
0.000
0.001
0.002
0.003
0.004
Bwd Packet Length Std
Total Length of Fwd Packets
Total Fwd Packets
Flow IAT Min
Bwd Packet Length Max
Flow Bytes/s
Flow IAT Max
Fwd Packet Length Std
Fwd Packet Length Mean
Flow Duration

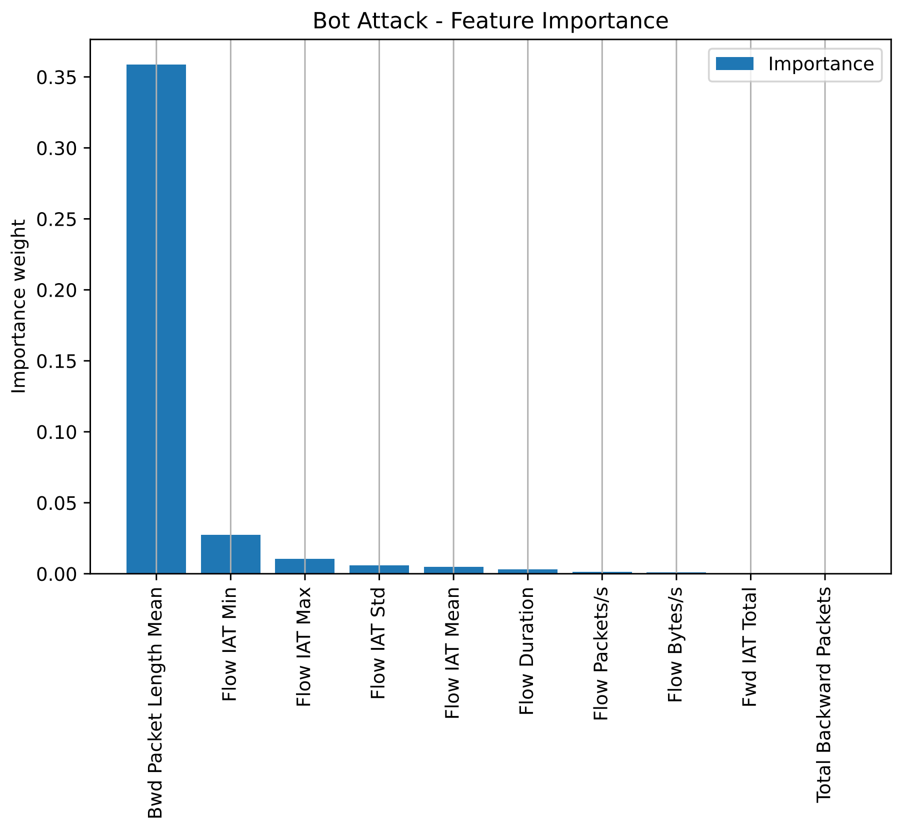


Figure 4.6: Graphs of features ranked by their importance weight for each attack.

4.5.3.2 Feature Selection According to Normal or Abnormal

The second approach to feature selection is to use the data from the file shown in Table 4.9 which contains only Normal and Abnormal labels. Similar to the first approach, we used the Random Forest Regressor to extract the most relevant features for this dataset, the list of features obtained is presented in Table 4.11 and the feature graphs in Figure 4.7.

Feature name	Importance weight
Bwd Packet Length Std	0.252245
Total Length of Fwd Packets	0.145528
Fwd Packet Length Std	0.032334
Flow Bytes/s	0.015275
Flow IAT Std	0.007259
Fwd IAT Total	0.006500
Flow IAT Min	0.005038
Flow Duration	0.004401
Total Length of Bwd Packets	0.003380
Flow IAT Max	0.003134

Table 4.11: List of the 10 most relevant features for the dataset containing only Normal and Abnormal labels.

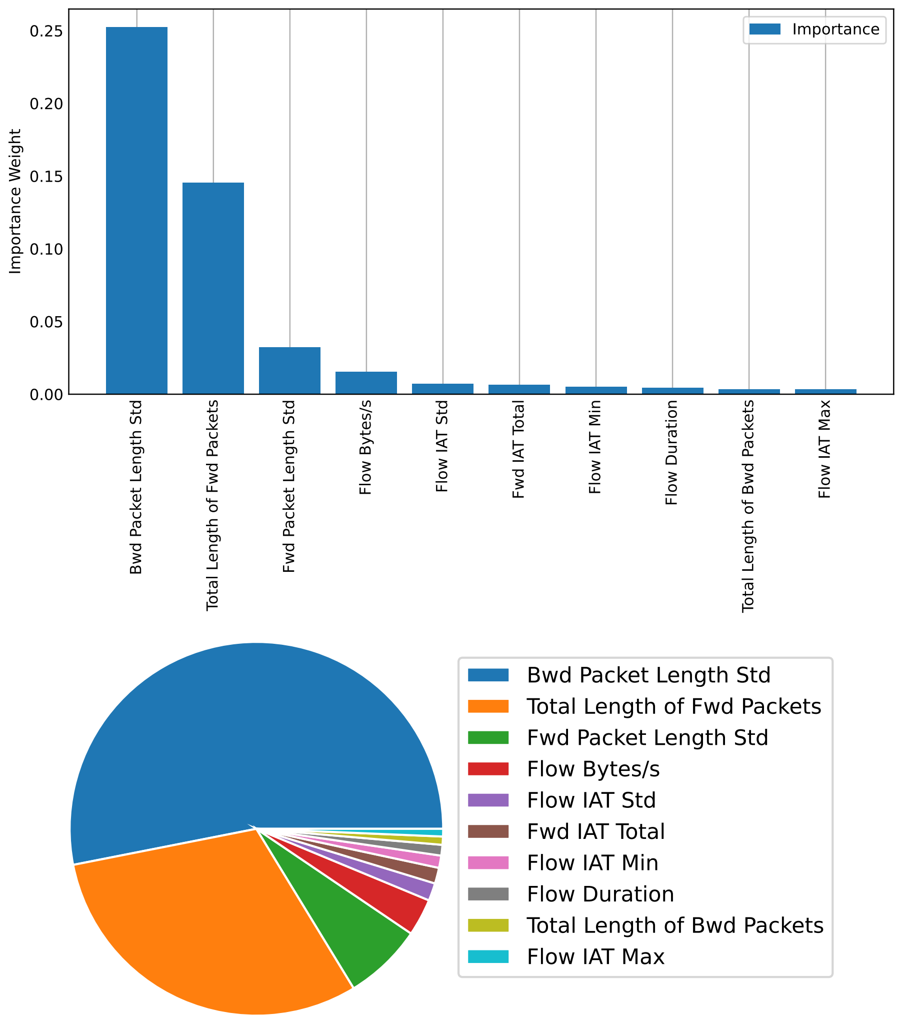


Figure 4.7: Graphs of feature importance weights according to Normal and Abnormal Labels.

4.6 Implementation of Machine Learning Algorithms

Two different approaches have been used to apply ML algorithms to the dataset. In the first approach, we used ML classification techniques to implement five types of ML algorithms: Random Forest, Decision Tree, K-Nearest Neighbors, Support Vector Machine and Multi-Layer Perceptron. These models were built and tested on seven different attack files, which are listed in Table 4.7.
In the second approach, we applied the same five ML algorithms to the dataset containing only two labels "Normal" and "Abnormal" which we discussed in Table 4.9.

Deep Neural Networks and other complex models can detect subtle patterns in the data, but if the training set is noisy or contains too many features, the model is likely to detect patterns in the noise itself, leading to overfitting of the model as well as the inability to generalize to new instances [42]. To avoid this problem while reducing computational costs, we included only the 10 most relevant features for each attack in the model learning phase for both approaches.

4.6.1 Decision Trees

Decision Trees are versatile ML algorithms that can perform classification and regression tasks. They are built by analyzing a set of training examples for which the class labels are known. They are then applied to classify previously unseen examples.
Each Decision Tree consists of nodes, branches, and leaves. Each node in the tree acts as a test case for an attribute, and each child node descending from the node corresponds to possible answers to the test case. An element is sorted into a class by following the path from the root node down to the leaf node based on the answers that apply to the element. An element is assigned to the class that has been associated with the leaf it reaches [79].

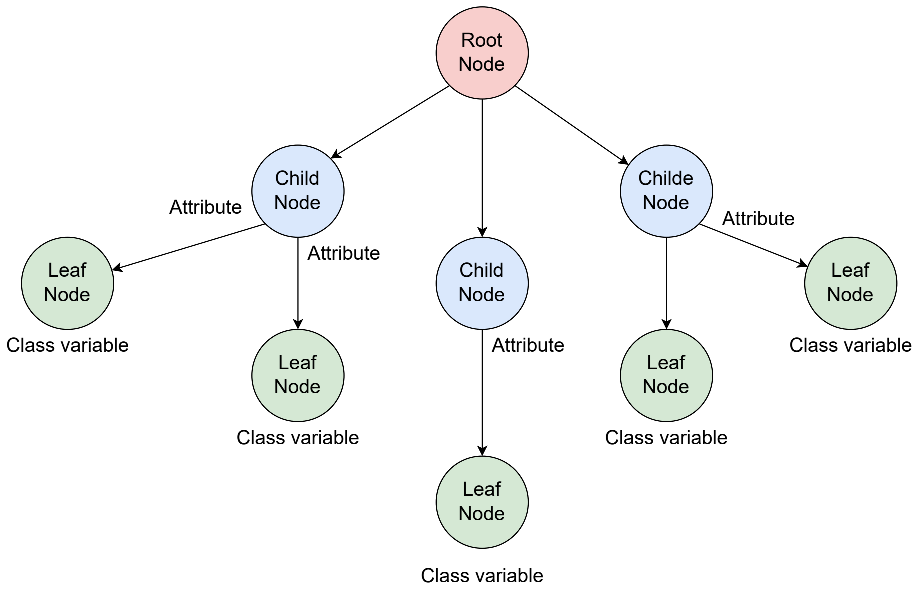


Figure 4.8: Decision Tree Structure.

4.6.2 Random Forest

Random Forest is one of the most popular ML techniques used to solve both classification and regression problems. It runs a number of Decision Trees on different subsets of a given dataset and averages the results to improve the predictive accuracy of that dataset. Instead of relying on a single Decision Tree, the Random Forest takes the predictions from each tree and predicts the final output based on the majority votes of the predictions [80].

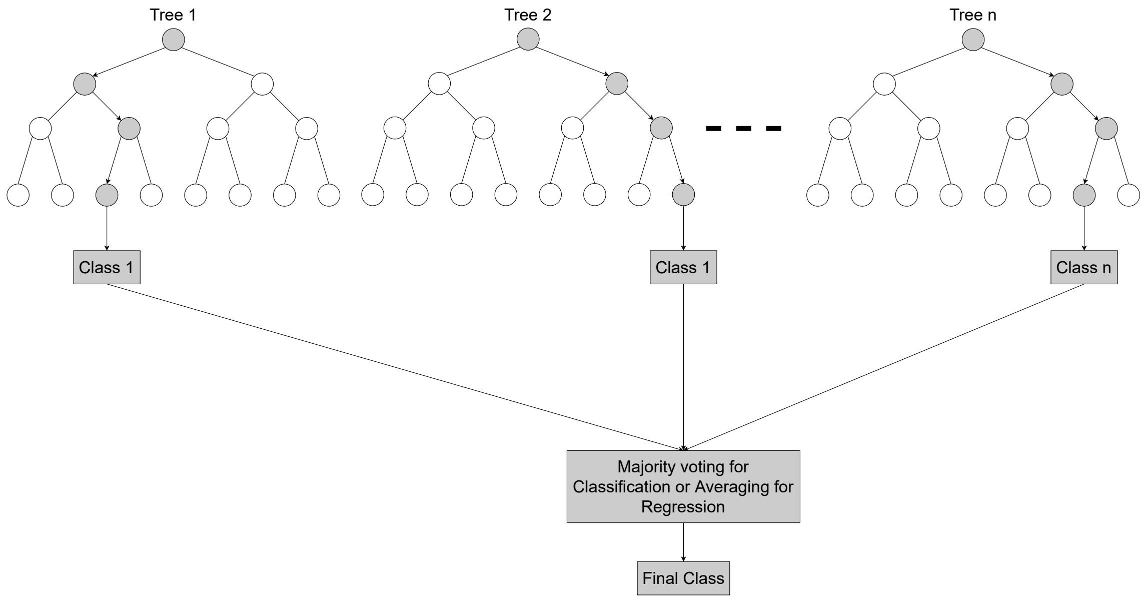

Tree 1
Tree 2
Tree n
Class 1
Class 1
Class n
Majority voting for Classification or Averaging for Regression
Final Class

Figure 4.9: Random Forest structure.

4.6.3 K-Nearest Neighbors

K-Nearest Neighbor (KNN) is one of the most basic yet essential classification algorithms in ML, which uses proximity to make classifications or predictions about the grouping of an individual data point. To determine if a data point belongs to a specific class, it first selects the number of neighbors (k) and calculates the distance to find the unclassified data point's closest k neighbors. From these k neighbors, the number of data points is calculated in each class, which then classifies the unknown sample into the class with the highest number of neighbors [81].

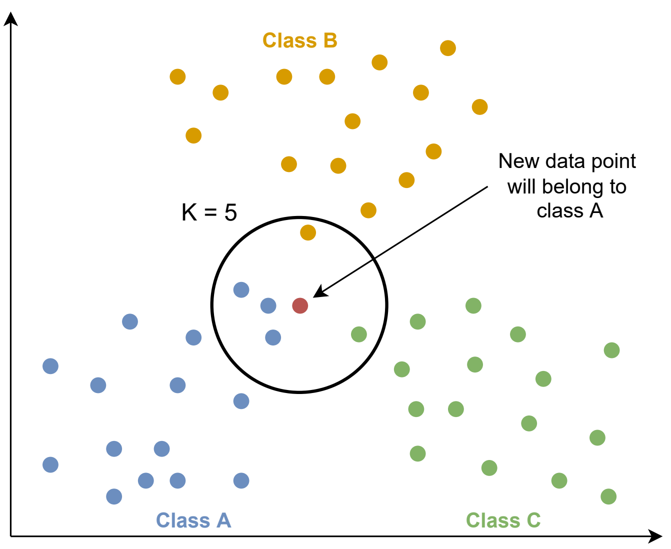


Figure 4.10: Operation of KNN algorithm for K = 5 value.

4.6.4 Support Vector Machine

Support Vector Machine (SVM) is a supervised ML algorithm primarily used for classification problems. The SVM algorithm's goal is to find a separating hyperplane in an N-dimensional space that separates data points, each in its own class. To successfully classify these data points, the algorithm finds the optimal hyperplane, which is the one with the highest distance between the data points of the two classes [82].

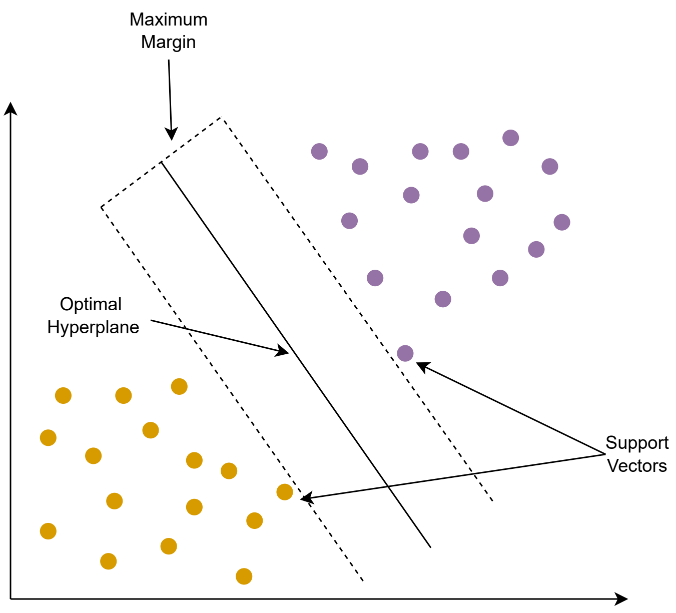


Figure 4.11: Classification of data points using SVM algorithm.

4.6.5 Multi Layer Perceptron

This topic has been elaborated in Chapter 3 (see The Perceptron and The Multi-Layer Perceptron).

4.7 Results and Discussion

In this section, the results of the studies conducted in the implementation section are presented.
Several tests have been performed to obtain the right hyper-parameters for each model. These parameters include for example (the number of neighbors k for KNN or the number of layers and iterations for MLP etc...). To do so, we begin by randomizing the hyper-parameters, then train our model on the training set while validating it on the validation set, then repeat this process with different parameter values until we achieve the best results, and finally confirm these results by testing the final model on the test set. The best model is the one with the highest accuracy score and the lowest error rate when compared to all other models.

4.7.1 First approach - Using 7 attack types

Five different ML methods are applied to 7 different attack types and the obtained results are presented in Tables (4.12 - 4.18). For each attack, there is a best and worst model, each is represented by a Confusion matrix as well as the ROC curve.

DoS attack				
Machine Learning Algorithms	**Accuracy**	**Precision**	**Recall**	**F1-score**
RF	98.48%	97.71%	98.60%	98.14%
DT	98.42%	97.64%	98.55%	98.08%
KNN	97.96%	97.07%	97.98%	97.51%
MLP	90.63%	90.04%	88.59%	88.65%
SVM	86.10%	89.11%	76.08%	79.62%

Table 4.12: Performance comparison of the 5 ML algorithms on the DoS attack.

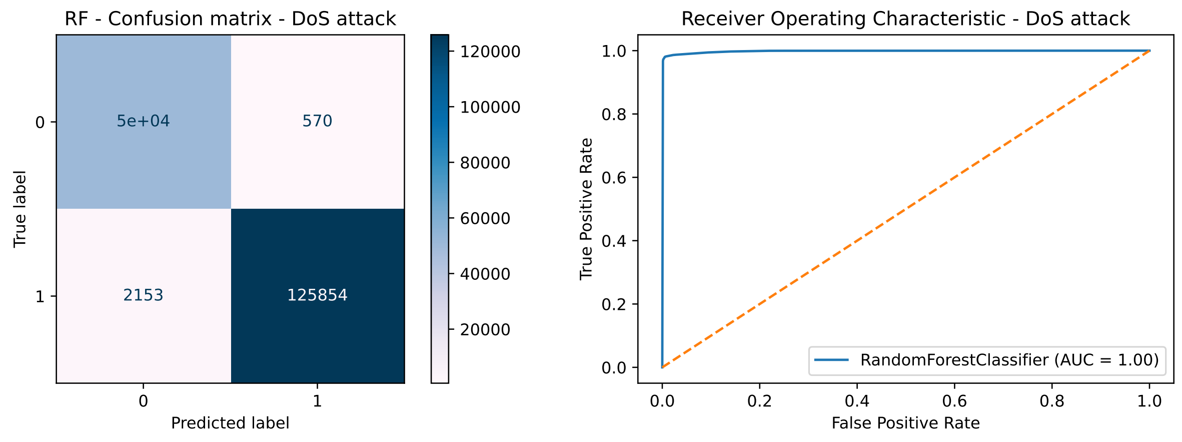


Figure 4.12: Performance of the RF algorithm on the DoS attack.

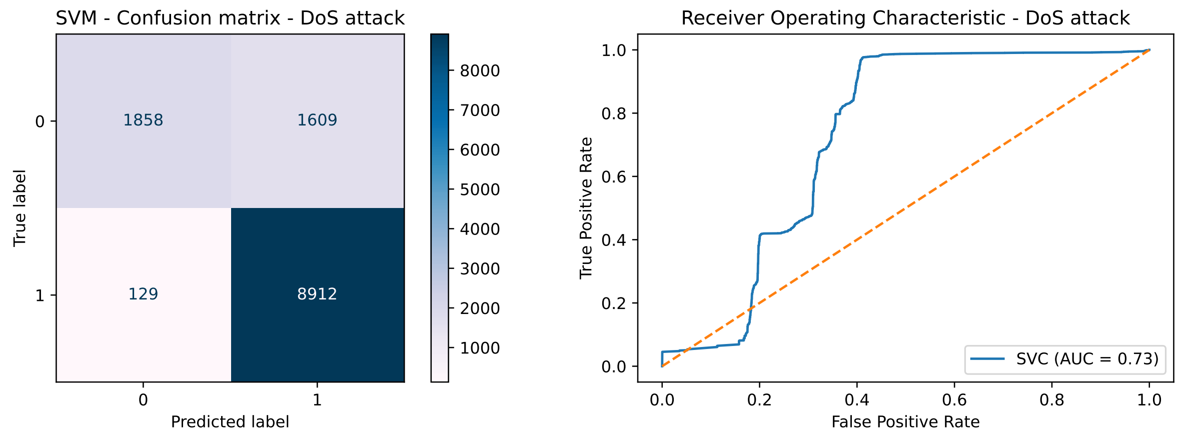


Figure 4.13: Performance of the SVM algorithm on the DoS attack.

PortScan attack				
Machine Learning Algorithms	**Accuracy**	**Precision**	**Recall**	**F1-score**
RF	99.94%	99.92%	99.94%	99.93%
DT	99.94%	99.91%	99.93%	99.92%
KNN	99.83%	99.75%	99.84%	99.79%
SVM	81.75%	79.32%	78.06%	77.82%
MLP	75.21%	71.09%	70.54%	62.76%

Table 4.13: Performance comparison of the 5 ML algorithms on the PortScan attack.

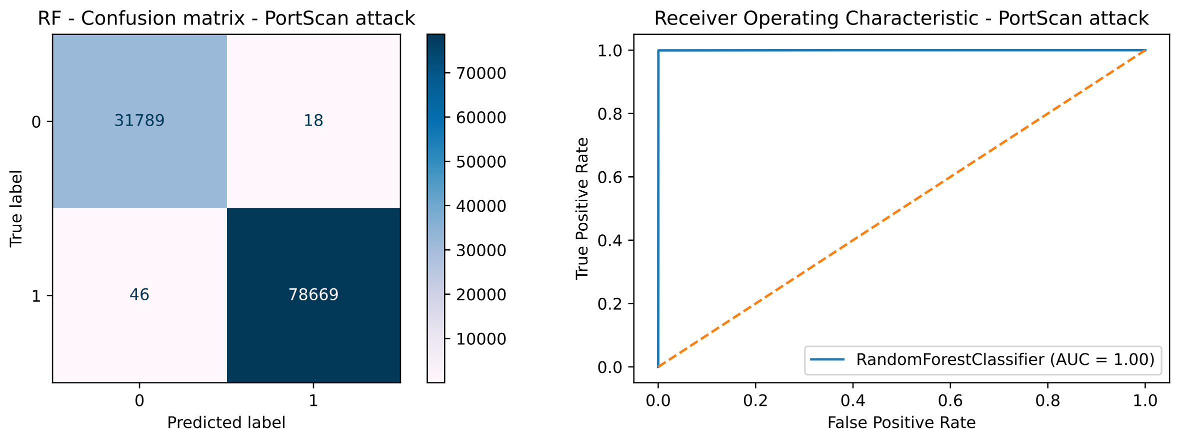


Figure 4.14: Performance of the RF algorithm on the PortScan attack.

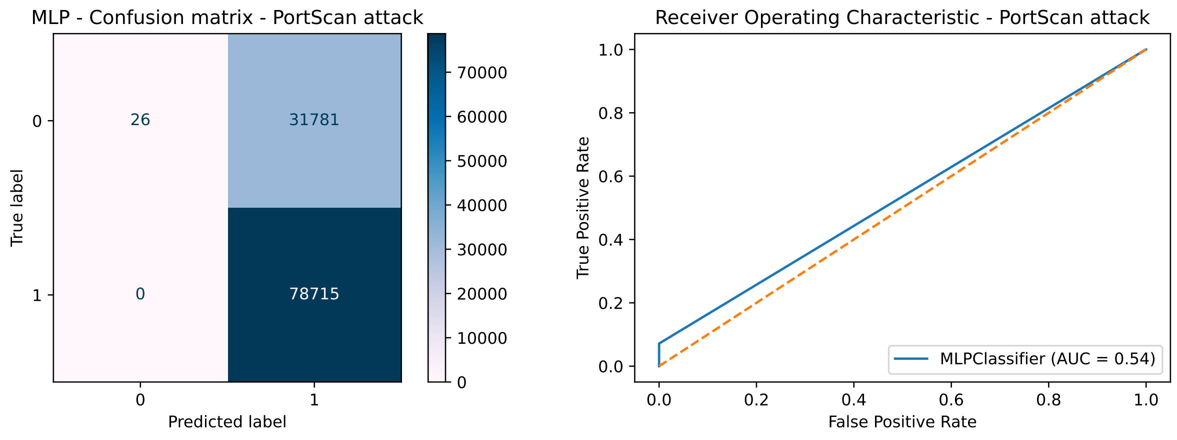


Figure 4.15: Performance of the MLP algorithm on the PortScan attack.

DDoS attack				
Machine Learning Algorithms	**Accuracy**	**Precision**	**Recall**	**F1-score**
RF	97.43%	96.28%	97.84%	97.01%
KNN	96.51%	95.07%	97.02%	95.96%
DT	96.14%	95.38%	95.48%	95.43%
SVM	75.41%	71.42%	74.28%	70.52%
MLP	75.59%	77.25%	68.24%	67.24%

Table 4.14: Performance comparison of the 5 ML algorithms on the DDoS attack.

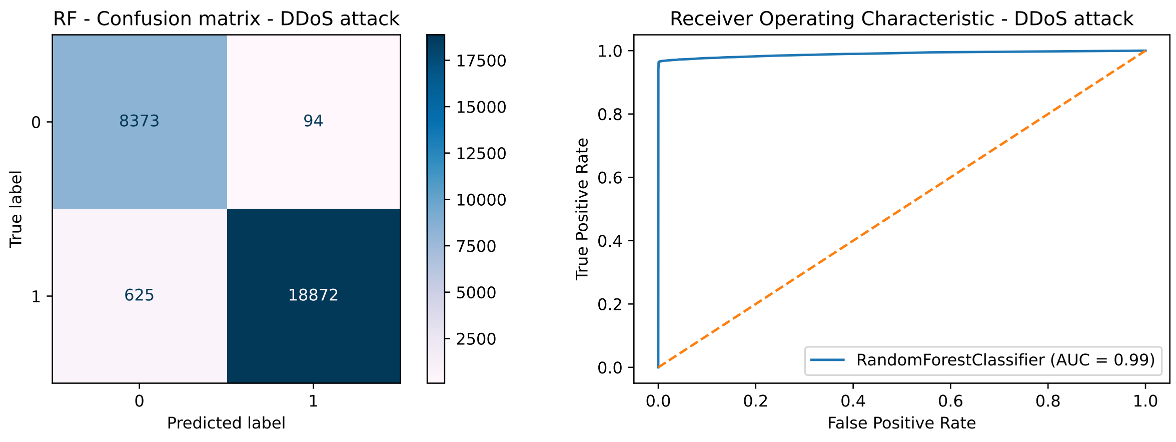


Figure 4.16: Performance of the RF algorithm on the DDoS attack.

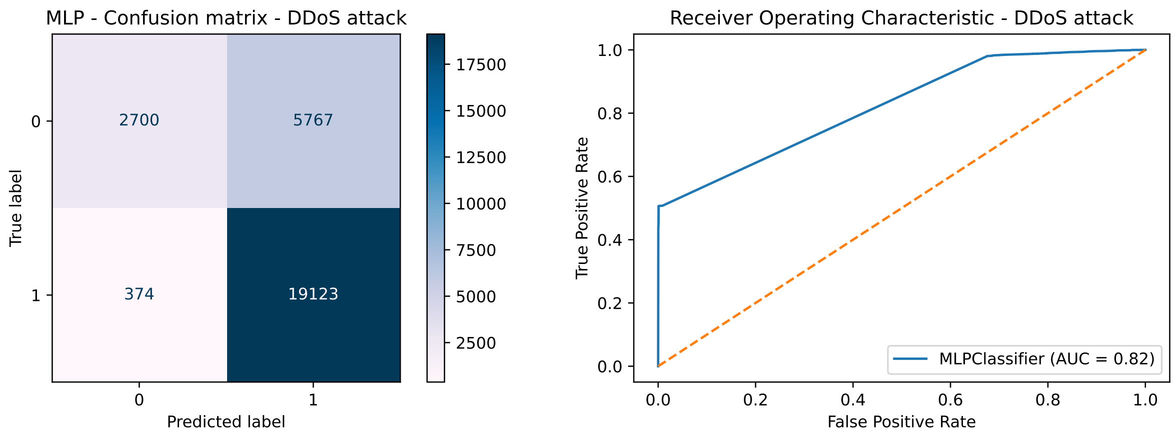


Figure 4.17: Performance of the MLP algorithm on the DDoS attack.

FTP-Patator attack				
Machine Learning Algorithms	**Accuracy**	**Precision**	**Recall**	**F1-score**
RF	99.91%	99.92%	99.86%	99.89%
DT	99.89%	99.88%	99.85%	99.87%
KNN	99.46%	99.23%	99.47%	99.35%
MLP	91.72%	92.16%	87.93%	88.83%
SVM	82.93%	84.17%	73.48%	76.21%

Table 4.15: Performance comparison of the 5 ML algorithms on the FTP-Patator attack.

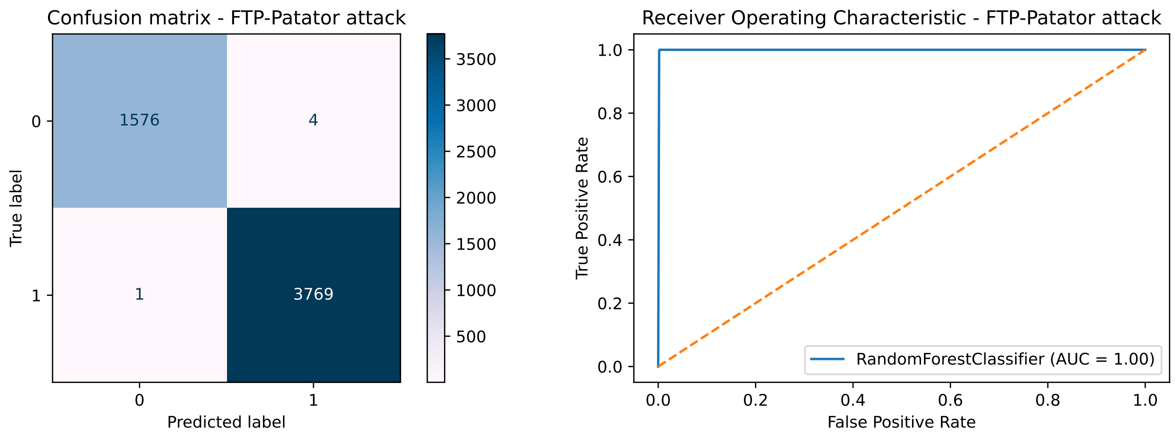


Figure 4.18: Performance of the RF algorithm on the FTP-Patator attack.

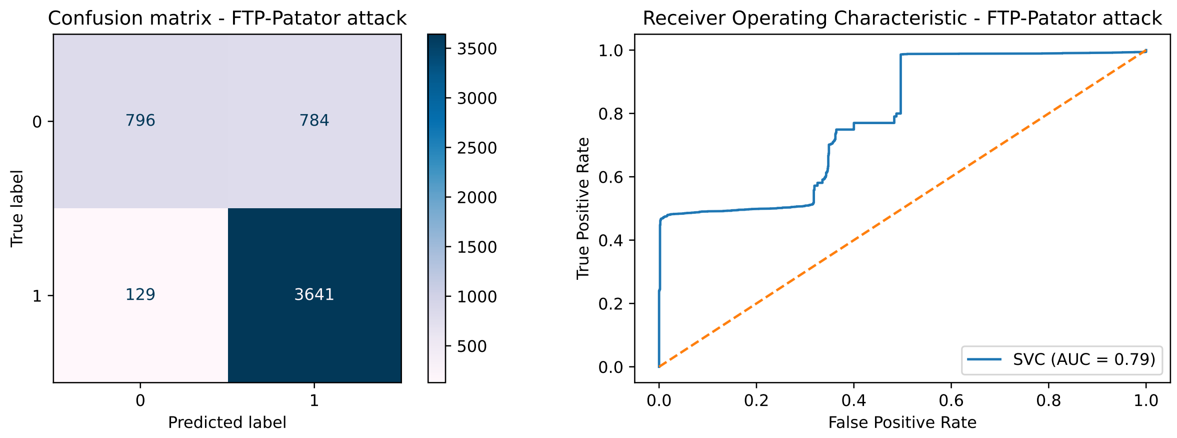


Figure 4.19: Performance of the SVM algorithm on the FTP-Patator attack.

SSH-Patator attack				
Machine Learning Algorithms	**Accuracy**	**Precision**	**Recall**	**F1-score**
RF	97.23%	95.96%	97.66%	96.75%
DT	97.13%	95.80%	97.61%	96.64%
KNN	95.71%	94.16%	95.91%	94.97%
MLP	84.86%	88.47%	76.45%	78.49%
SVM	84.56%	90.83%	74.11%	77.57%

Table 4.16: Performance comparison of the 5 ML algorithms on the SSH-Patator attack.

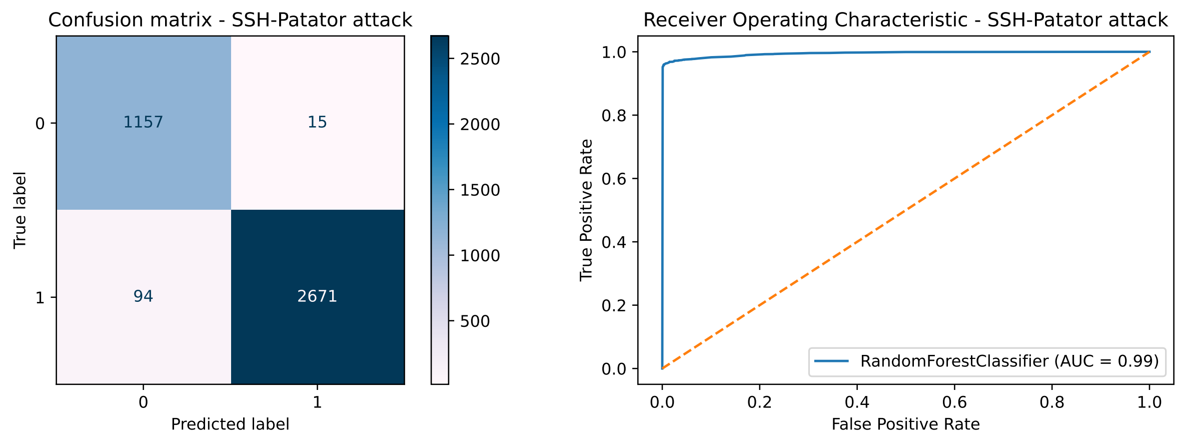


Figure 4.20: Performance of the RF algorithm on the SSH-Patator attack.

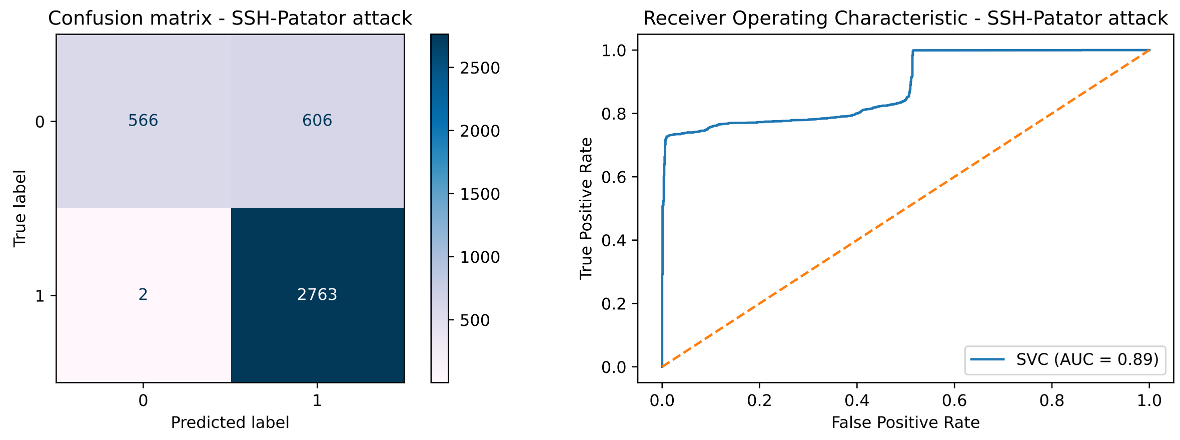


Figure 4.21: Performance of the SVM algorithm on the SSH-Patator attack.

Web attack				
Machine Learning Algorithms	**Accuracy**	**Precision**	**Recall**	**F1-score**
DT	97.10%	96.58%	96.69%	96.63%
RF	97.10%	96.68%	96.57%	96.62%
KNN	95.44%	94.36%	95.18%	94.75%
SVM	89.56%	87.23%	90.48%	88.43%
MLP	74.88%	63.85%	62.12%	56.58%

Table 4.17: Performance comparison of the 5 ML algorithms on the Web attack.

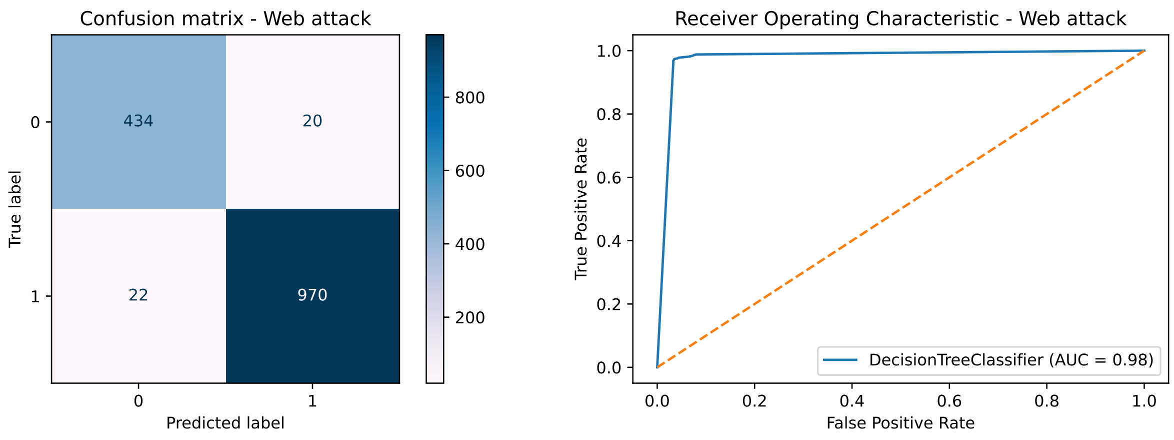


Figure 4.22: Performance of the DT algorithm on the Web attack.

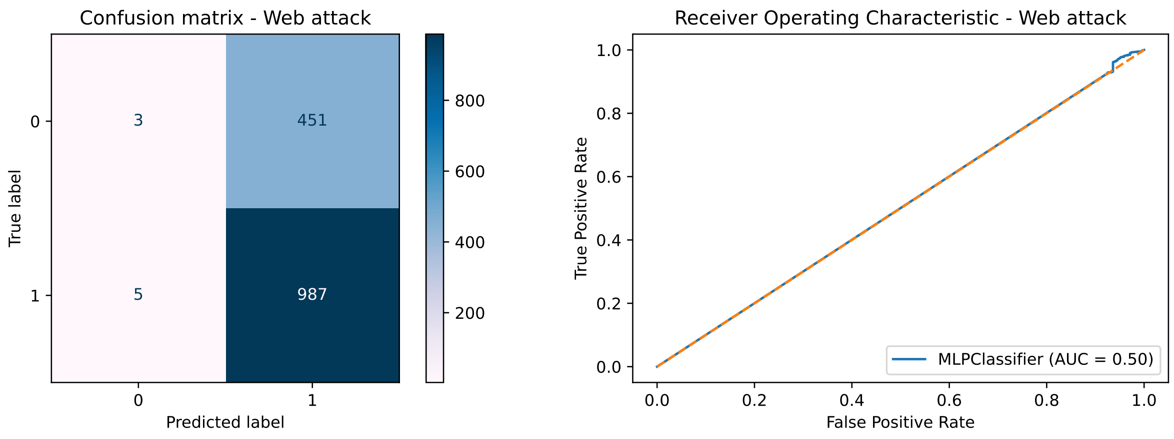


Figure 4.23: Performance of the MLP algorithm on the Web attack.

Bot attack				
Machine Learning Algorithms	**Accuracy**	**Precision**	**Recall**	**F1-score**
RF	97.24%	96.07%	97.43%	96.71%
DT	97.16%	95.96%	97.38%	96.63%
KNN	95.08%	93.12%	95.68%	94.25%
MLP	80.71%	78.58%	73.77%	73.81%
SVM	70.66%	35.33%	50%	41.4%

Table 4.18: Performance comparison of the 5 ML algorithms on the Bot attack.

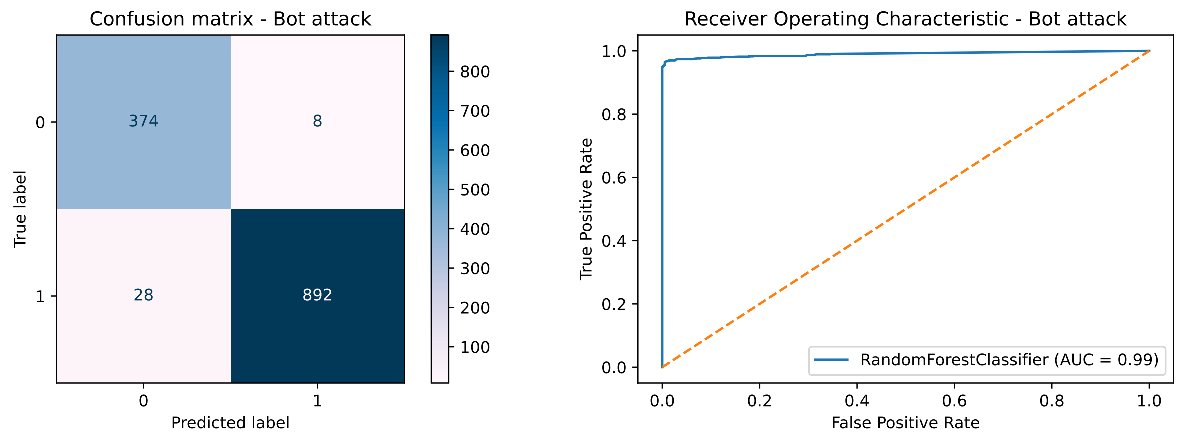


Figure 4.24: Performance of the RF algorithm on the Bot attack.

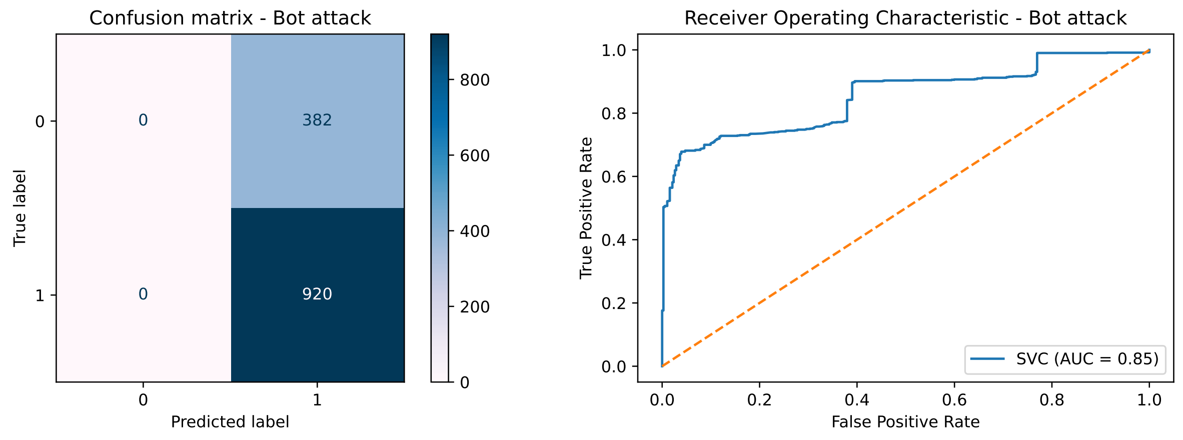


Figure 4.25: Performance of the SVM algorithm on the Bot attack.

Attack names	**F-measures**				
	RF	**DT**	**KNN**	**MLP**	**SVM**
DoS	**98.14%**	98.08%	97.51%	88.65%	<u>79.62%</u>
PortScan	**99.93%**	99.92%	99.79%	<u>62.76%</u>	77.82%
DDoS	**97.01%**	95.43%	95.96%	<u>67.24%</u>	70.52%
FTP-Patator	**99.89%**	99.87%	99.35%	88.83%	<u>76.21%</u>
SSH-Patator	**96.75%**	96.64%	94.97%	78.49%	<u>77.57%</u>
Web	96.62%	**96.63%**	94.75%	<u>56.58%</u>	88.43%
Bot	**96.71%**	96.63%	94.25%	73.81%	<u>41.4%</u>

Table 4.19: Overall performance of each algorithm for all attacks.

In Table 4.19, we used the F-measure (F1 score), which is calculated as the harmonic mean of precision and recall, giving each the same weight. It allows us to evaluate a model by considering both Precision and Recall using a single score, making it useful for describing model performance and comparing models.

Looking at the results, we notice that the Random Forest, Decision Tree, and K-Nearest Neighbors algorithms were successful in detecting all types of attacks by more than 94%. Among all algorithms, Random Forest is the most successful, having completed 6 of the 7 tasks with the highest score.

MLP has the second worst performance among the algorithms with the lowest score in 3 out of 7 tasks. After falling into overfitting for so many tests, we decided to reduce the number of included features to only 4 and the number of neurons in each hidden layer to reduce the computational cost and improve the performance of the model.

SVM had the worst performance with the lowest score in 4 of the 7 tasks, especially in the Bot attack and this is probably due to the limited amount of training data.

4.7.2 Approach 2 - Using Two Groups: Normal and Abnormal

Table 4.20 shows the results achieved by using 10 features that were obtained in the section Feature Selection According to Normal or Abnormal.

ML Algorithms	**Evaluation Metrics**				
	Accuracy	**Precision**	**Recall**	**F1-score**	**time (s)**
RF	**97.02%**	**96.08%**	**96.64%**	**96.35%**	3821.53
DT	96.74%	95.70%	96.37%	96.03%	**146.8**
KNN	96.24%	95.04%	95.82%	95.42%	700.23
MLP	87.57%	84.95%	87.32%	85.29%	2987.85
SVM	81.12%	83.68%	68.94%	71.57%	6157.98

Table 4.20: Overall performance of each algorithm according to Normal/Abnormal.

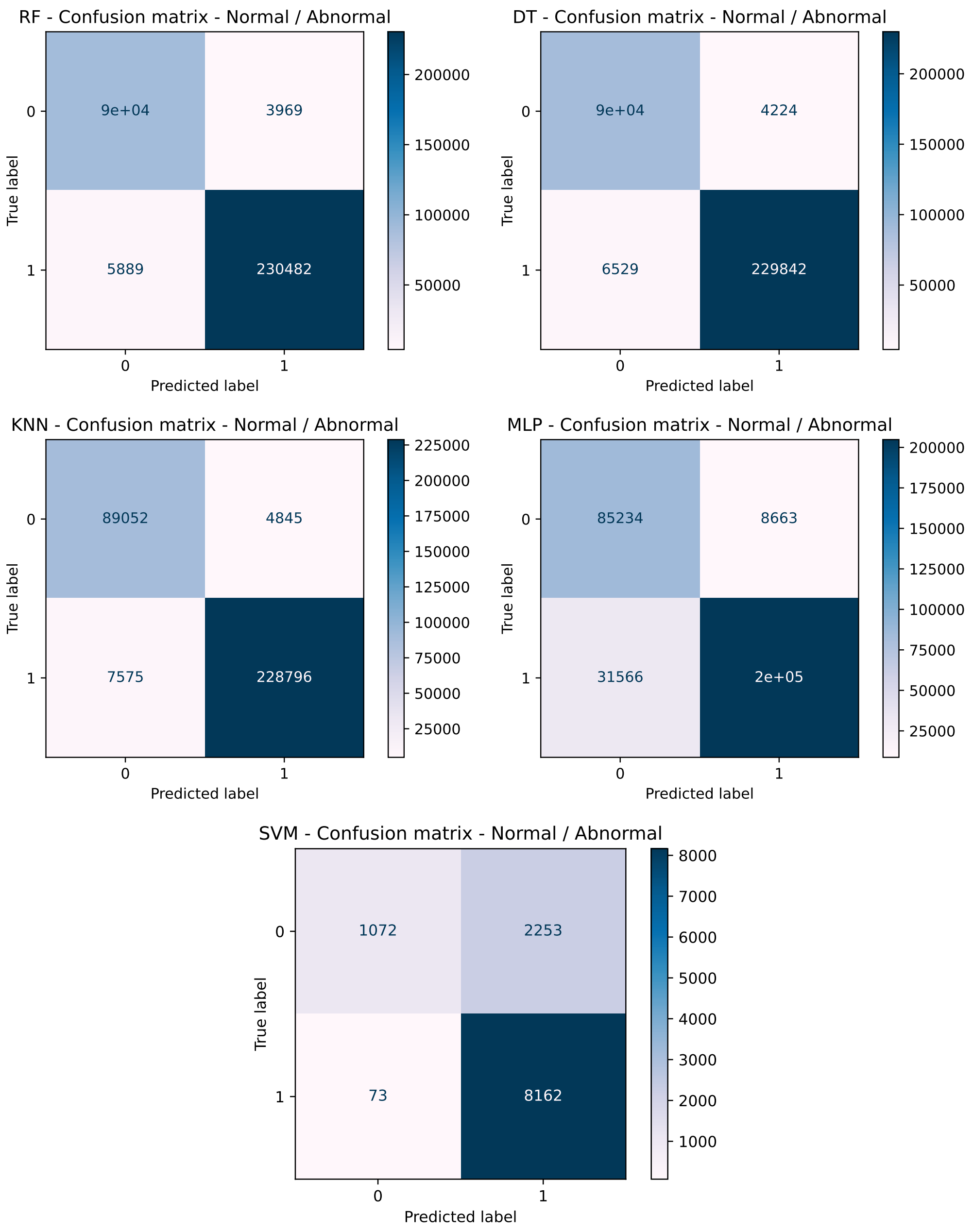


Figure 4.26: Confusion matrix of each algorithm according to Normal/Abnormal.

The table 4.20 shows that Random Forest achieved the best results, but has a drawback in terms of computation time. Decision Tree, on the other hand, had the second best result with the fastest computation time, making it overall a better choice.

MLP achieved decent results, which is due to the large amount of training data present in this second approach (over 1.5 million instances, see Table 4.9).

Given the limitations of the available hardware tools (GPU, CPU, memory), we could not apply the SVM on the whole dataset, as it takes an absurd amount of time to train. As a result, we opted to train it on only 10% of the total dataset.

The hyper-parameters of all algorithms employed during the training phase are listed in Table 4.21.

Classifiers	**Hyper-Parameters**
RF	RandomForestClassifier(random_state=0) [83]
DT	DecisionTreeClassifier(random_state=0) [84]
KNN	KNeighborsClassifier(n_neighbors=5) [85]
MLP	MLPClassifier(hidden_layer_sizes=(10,10,10),max_iter=500) [86]
SVM	SVC(kernel='rbf') [87]

Table 4.21: The hyper-parameters used for each classifier.

4.8 Comparative Study

In a 2017 study [72], seven commonly used ML methods (Naive-Bayes (NB), Random Forest (RF), K- Nearest Neighbours (KNN), Multi-layer perceptron (MLP), Adaboost, ID3, and Quadratic Discriminant Analysis (QDA)) were used to detect 14 different attack types. During this process, CIC-IDS2017 was used as the dataset. The performance ratios obtained in this study are as follows: Naive-Bayes 84%, KNN 96%, RF 97%, MLP 76%, Adaboost 77%, ID3 98%, QDA 92%.

In the study [88] conducted in 2019, a binary classification was used in the CIC-IDS2017 dataset using 3 different algorithms (Deep Neural Network (DNN), Gradient Boosted Tree (GBT) and Random Forest (RF)). The accuracy obtained by each model was as follows: DNN 97.73%, Random Forest 92.72%, Gradient Boosted Tree 99.97%.

4.9 Evaluation

Studies	ML Algorithms	Accuracy	F1-score
Faker, et al [88].	DNN	97.73%	/
	RF	92.72%	/
	GBT	99.97%	/
Sharafaldin, et al [72].	RF	/	97%
	DT	/	98%
	AdaBoost	/	77%
	MLP	/	76%
	KNN	/	96%
	Naive-Bayes	/	4%
	QDA	/	92%
Our study	RF	97.02%	96.35%
	DT	96.74%	96.03%
	KNN	96.24%	95.42%
	MLP	87.57%	85.29%
	SVM	81.12%	71.57%

Table 4.22: Comparison of classification prediction accuracy with previous studies used on CIC-IDS2017.

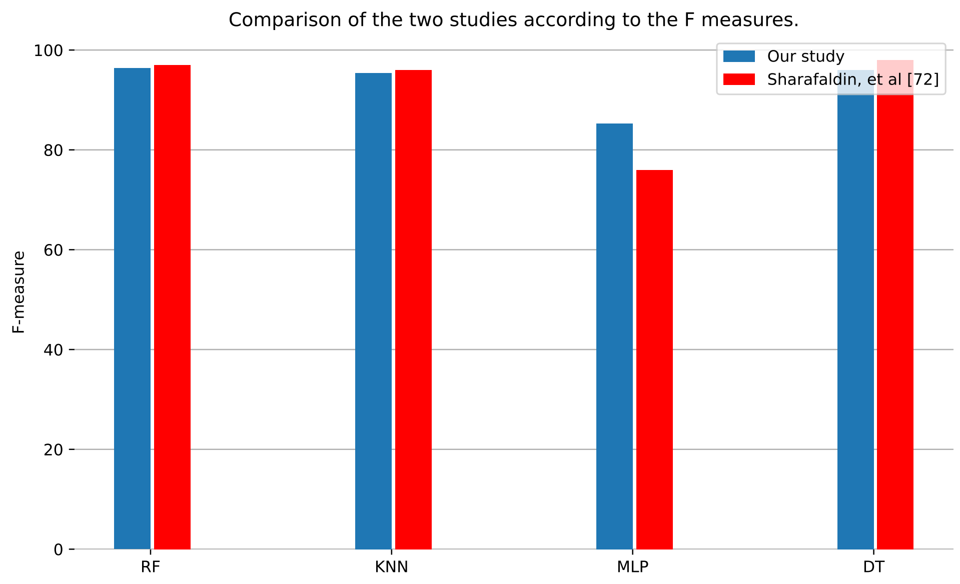


Figure 4.27: Comparison of the performance of the two studies with respect to F-measures.

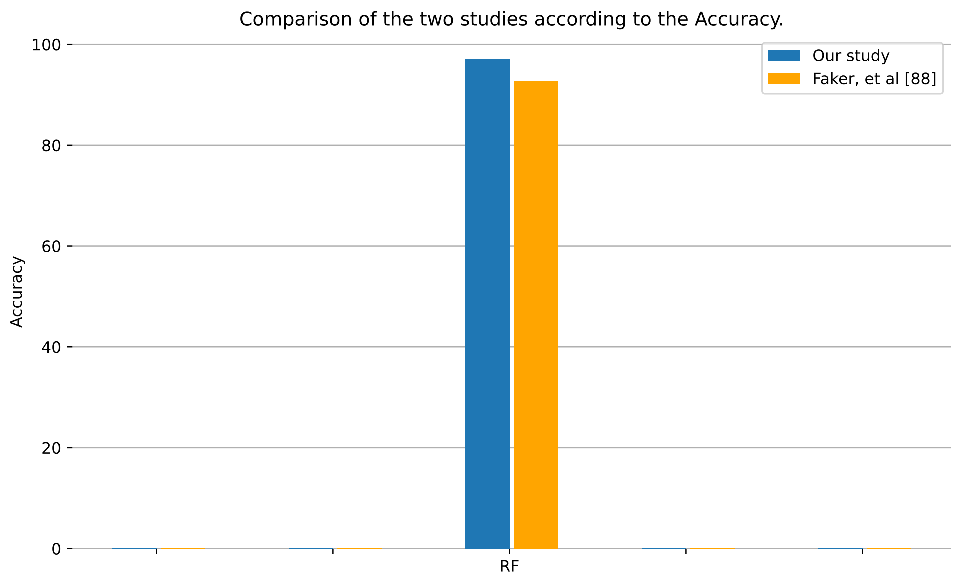


Figure 4.28: Comparison of the performance of the two studies with respect to Accuracy.

4.10 Conclusion

The objective of this last chapter was to propose a solution that best meets the different constraints of intrusion detection systems. The proposed solution improves the performance of IDS in terms of precision in detecting and classifying a wide range of attacks with high accuracy and low false alarm rate. Experiments show that the proposed approach yields very satisfactory results.

General conclusion

Driven by a growing interest in improving network security and preventing any security breaches that could have disastrous consequences, network security administrators are always looking for a solution to ensure a highly secure network environment.

The work conducted in this master thesis falls within the field of network anomaly detection, aiming to design and develop a model for the detection and classification of a wide range of attacks using ML and DL techniques. In the first chapter of this thesis, the generalities of network security as well as the most widespread types of attacks have been presented in a general way in order to get familiar with the basic concept of these notions. In the second chapter, we introduced intrusion detection systems given the fact that they constitute a good solution for detecting abnormal behavior within a network with the ability to alert system administrators to any potential security violations in their organization in order to prevent them before causing substantial damages.

The problem of detecting and classifying attacks in a real-world scenario cannot be solved properly without resorting to ML techniques that enable classification of attacks based on supervised learning.

We chose the CIC-IDS2017 dataset as training data for our algorithms (Random Forest (RF), Decision Tree (DT), K-Nearest Neighbors (KNN), Support Vector Machine (SVM) and Multi-layer Perceptron (MLP)). Knowing that the performance of our models depends entirely on the quality of this data we applied several data pre-processing techniques such as reducing the imbalance between classes and eliminating all irrelevant features and missing values that could negatively affect the performance of our models. After the learning phase, these algorithms were validated by different experiments on the validation set before being tested on the test set. The final results being very promising, we can consider the reliability and efficiency of our approach as satisfactory.

One of the limitations discovered during the experiments is the low precision for classification of attacks using the SVM algorithm, which is due to the fact that we only used 10% of the entire dataset to train the model, as SVM is a very powerful algorithm that requires a large amount of data and a lot of time to train in order to achieve the desired results. With more powerful hardware tools, we can drastically reduce the training phase's computation time, allowing us to improve the model's performance by devoting the entire data set to training the latter and eventually achieve much better results.

Finally, it is worth noting that three of the four ML algorithms outperformed the Multi-Layer perceptron. Although DL is unquestionably a more powerful technique than ML, it does have some drawbacks, such as the massive amount of data required to achieve good results as well as the need for powerful hardware tools to reduce computation time. DL is best suited for complex problems such as image recognition, speech recognition or natural language processing, provided we have enough data, computational power and patience.

Bibliography

[1] "2020 state of fraud and account security report." (), [Online]. Available: https://www.arkoselabs.com/wp-content/uploads/Fraud-Report-Q2-2020.pdf. (accessed: 15.06.2022).

[2] G. Gangemi and R. Lehtinen, Computer Security Basics. O'Reilly Media, Incorporated, 2006.

[3] M. Ciampa, Security+ guide to network security fundamentals. Cengage Learning, 2012.

[4] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," ACM computing surveys (CSUR), vol. 41, no. 3, pp. 1–58, 2009.

[5] M. Ahmed, A. N. Mahmood, and J. Hu, "A survey of network anomaly detection techniques," Journal of Network and Computer Applications, vol. 60, pp. 19–31, 2016.

[6] K. Kim, M. E. Aminanto, and H. C. Tanuwidjaja, Network intrusion detection using deep learning: a feature learning approach. Springer, 2018.

[7] A. Lamba, S. Singh, S. Bhardwaj, N. Dutta, and S. Rela, "Uses of artificial intelligent techniques to build accurate models for intrusion detection system," International Journal For Technological Research In Engineering, vol. 2, no. 12, 2015.

[8] K. Kostas, "Anomaly detection in networks using machine learning," Research Proposal, vol. 23, p. 343, 2018.

[9] S. Behal and K. Kumar, "Characterization and comparison of ddos attack tools and traffic generators: A review.," Int. J. Netw. Secur., vol. 19, no. 3, pp. 383–393, 2017.

[10] S. Haris, R. Ahmad, and M. Ghani, "Detecting tcp syn flood attack based on anomaly detection," in 2010 Second International Conference on Network Applications, Protocols and Services, IEEE, 2010, pp. 240–244.

[11] J. Choi, C. Choi, B. Ko, D. Choi, and P. Kim, "Detecting web based ddos attack using mapreduce operations in cloud computing environment.," J. Internet Serv. Inf. Secur., vol. 3, no. 3/4, pp. 28–37, 2013.

[12] "What is a slowloris ddos attack?" (), [Online]. Available: https://www.netscout.com/what-is-ddos/slowloris-attacks. (accessed: 07.06.2022).

[13] A. Sperotto, G. Schaffrath, R. Sadre, C. Morariu, A. Pras, and B. Stiller, "An overview of ip flow-based intrusion detection," IEEE communications surveys & tutorials, vol. 12, no. 3, pp. 343–356, 2010.

[14] M. Ciampa, CompTIA security+ guide to network security fundamentals. Cengage Learning, 2021.

[15] C. Badrick. "Defending against port scan attacks." (), [Online]. Available: https://www.turn-keytechnologies.com/blog/article/defending-against-port-scan-attacks/. (accessed: 09.04.2022).

[16] J. Gadge and A. A. Patil, "Port scan detection," in 2008 16th IEEE International Conference on Networks, IEEE, 2008, pp. 1–6.

[17] K. T. Hanna. "Syn scanning." (), [Online]. Available: https://www.techtarget.com/searchnetworking/definition/SYN-scanning#:~:text=SYN%5C%20scanning%5C%20is%5C%20a%5C%20tactic,known%5C%20as%5C%20half-open%5C%20scanning.. (accessed: 04.06.2022).

[18] M. Dorigny. "Les scans de port via tcp : Xmas, null et ack." (), [Online]. Available: https://www.it-connect.fr/les-scans-de-port-via-tcp-xmas-null-et-ack/. (accessed: 05.06.2022).

[19] ——, "Technique de scan de port udp." (), [Online]. Available: https://www.it-connect.fr/technique-de-scan-de-port-udp/. (accessed: 06.06.2022).

[20] P. Ilevičius. "What is an ip fragmentation attack?" (), [Online]. Available: https://nordvpn.com/blog/ip-fragmentation-attack/. (accessed: 06.06.2022).

[21] S. M. Kerner. "Ftp (file transfer protocol)." (), [Online]. Available: https://www.techtarget.com/searchnetworking/definition/File-Transfer-Protocol-FTP. (accessed: 07.06.2022).

[22] K. T. Hanna. "Brute-force attack." (), [Online]. Available: https://www.techtarget.com/searchsecurity/definition/brute-force-cracking. (accessed: 07.06.2022).

[23] P. Loshin. "Secure shell (ssh)." (), [Online]. Available: https://www.techtarget.com/searchsecurity/definition/Secure-Shell. (accessed: 07.06.2022).

[24] R. Ashraf. "How to brute-force ssh in linux." (), [Online]. Available: https://www.rootinstall.com/tutorial/bruteforce-ssh-in-linux/. (accessed: 07.06.2022).

[25] P. Likarish, E. Jung, and I. Jo, "Obfuscated malicious javascript detection using classification techniques," in 2009 4th International Conference on Malicious and Unwanted Software (MALWARE), IEEE, 2009, pp. 47–54.

[26] P. Loshin. "Structured query language (sql)." (), [Online]. Available: `https://www.techtarget.com/searchdatamanagement/definition/SQL#:~:text=Structured%5C%20Query%5C%20Language%5C%20(SQL)%5C%20is,on%5C%20the%5C%20data%5C%20in%5C%20them.`. (accessed: 09.06.2022).

[27] "What is sql injection (sqli) and how to prevent it." (), [Online]. Available: `https://www.acunetix.com/websitesecurity/sql-injection/`. (accessed: 09.06.2022).

[28] E. Cole, Network security bible. John Wiley & Sons, 2011.

[29] G. Karatas, O. Demir, and O. K. Sahingoz, "Deep learning in intrusion detection systems," in 2018 International Congress on Big Data, Deep Learning and Fighting Cyber Terrorism (IBIGDELFT), IEEE, 2018, pp. 113–116.

[30] U. Fiore, F. Palmieri, A. Castiglione, and A. De Santis, "Network anomaly detection with the restricted boltzmann machine," Neurocomputing, vol. 122, pp. 13–23, 2013.

[31] I. Ahmad, A. B. Abdullah, and A. S. Alghamdi, "Application of artificial neural network in detection of dos attacks," in Proceedings of the 2nd international conference on Security of information and networks, 2009, pp. 229–234.

[32] E. Hodo, X. Bellekens, A. Hamilton, C. Tachtatzis, and R. Atkinson, "Shallow and deep networks intrusion detection system: A taxonomy and survey," arXiv preprint arXiv:1701.02145, 2017.

[33] R. Di Pietro and L. V. Mancini, Intrusion detection systems. Springer Science & Business Media, 2008, vol. 38.

[34] H. Benmoussa, A. Abou El Kalam, and A. A. Ouahman, "Distributed intrusion detection system based on anticipation and prediction approach," in 2015 12th International Joint Conference on e-Business and Telecommunications (ICETE), IEEE, vol. 4, 2015, pp. 343–348.

[35] E. Vasilomanolakis, S. Karuppayah, M. Mühlhäuser, and M. Fischer, "Taxonomy and survey of collaborative intrusion detection," ACM Computing Surveys (CSUR), vol. 47, no. 4, pp. 1–33, 2015.

[36] S. M. Othman, N. T. Alsohybe, F. M. Ba-Alwi, and A. T. Zahary, "Survey on intrusion detection system types," International Journal of Cyber-Security and Digital Forensics, vol. 7, no. 4, pp. 444–463, 2018.

[37] N. Stakhanova, S. Basu, and J. Wong, "A taxonomy of intrusion response systems," International journal of information and computer security, vol. 1, no. 1-2, pp. 169–184, 2007.

[38] I. Goodfellow, Y. Bengio, and A. Courville, Deep learning. MIT press, 2016.

[39] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," nature, vol. 521, no. 7553, pp. 436–444, 2015.

[40] A. L. Samuel, "Some studies in machine learning using the game of checkers. ii—recent progress," IBM Journal of research and development, vol. 11, no. 6, pp. 601–617, 1967.

[41] J. G. Carbonell, R. S. Michalski, and T. M. Mitchell, "An overview of machine learning," Machine learning, pp. 3–23, 1983.

[42] A. Géron, Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems. " O'Reilly Media, Inc.", 2019.

[43] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," The bulletin of mathematical biophysics, vol. 5, no. 4, pp. 115–133, 1943.

[44] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain.," Psychological review, vol. 65, no. 6, p. 386, 1958.

[45] D. O. Hebb, The organization of behavior: A neuropsychological theory. Psychology Press, 2005.

[46] A. L. Maas, A. Y. Hannun, A. Y. Ng, et al., "Rectifier nonlinearities improve neural network acoustic models," in Proc. icml, Citeseer, vol. 30, 2013, p. 3.

[47] Y. LeCun, B. Boser, J. S. Denker, et al., "Backpropagation applied to handwritten zip code recognition," Neural computation, vol. 1, no. 4, pp. 541–551, 1989.

[48] D. H. Hubel and T. N. Wiesel, "Receptive fields and functional architecture of monkey striate cortex," The Journal of physiology, vol. 195, no. 1, pp. 215–243, 1968.

[49] K. Fukushima, "A neural network model for selective attention in visual pattern recognition," Biological Cybernetics, vol. 55, no. 1, pp. 5–15, 1986.

[50] S. Balaji. "Binary image classifier cnn using tensorflow." (), [Online]. Available: `https://medium.com/techiepedia/binary-image-classifier-cnn-using-tensorflow-a3f5d6746697`. (accessed: 09.05.2022).

[51] R. Yamashita, M. Nishio, R. K. G. Do, and K. Togashi, "Convolutional neural networks: An overview and application in radiology," Insights into imaging, vol. 9, no. 4, pp. 611–629, 2018.

[52] "What is a convolutional neural network?" (), [Online]. Available: `https://www.mathworks.com/discovery/convolutional-neural-network-matlab.html`. (accessed: 09.05.2022).

[53] I. C. Education. "Convolutional neural networks." (), [Online]. Available: `https://www.ibm.com/cloud/learn/convolutional-neural-networks#toc-convolutio-JgBTyG9C`. (accessed: 09.05.2022).

[54] M. Mandal. "Introduction to convolutional neural networks (cnn)." (), [Online]. Available: `https://www.analyticsvidhya.com/blog/2021/05/convolutional-neural-networks-cnn/`. (accessed: 09.05.2022).

[55] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," nature, vol. 323, no. 6088, pp. 533–536, 1986.

[56] S. Hochreiter and J. Schmidhuber, "Long short-term memory," Neural computation, vol. 9, no. 8, pp. 1735–1780, 1997.

[57] A. Graves, A.-r. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," in 2013 IEEE international conference on acoustics, speech and signal processing, Ieee, 2013, pp. 6645–6649.

[58] aishwarya.27. "Introduction to recurrent neural network." (), [Online]. Available: `https://www.geeksforgeeks.org/introduction-to-recurrent-neural-network/`. (accessed: 11.05.2022).

[59] G. Singhal. "Introduction to lstm units in rnn." (), [Online]. Available: `https://www.pluralsight.com/guides/introduction-to-lstm-units-in-rnn`. (accessed: 11.05.2022).

[60] Y. Le Cun and F. Fogelman-Soulié, "Modèles connexionnistes de l'apprentissage," Intellectica, vol. 2, no. 1, pp. 114–143, 1987.

[61] H. Bourlard and Y. Kamp, "Auto-association by multilayer perceptrons and singular value decomposition," Biological cybernetics, vol. 59, no. 4, pp. 291–294, 1988.

[62] G. E. Hinton and R. Zemel, "Autoencoders, minimum description length and helmholtz free energy," Advances in neural information processing systems, vol. 6, 1993.

[63] Anaconda software distribution, version Vers. 2-2.1.4, 2020. [Online]. Available: `https://docs.anaconda.com/`.

[64] G. Van Rossum and F. L. Drake Jr, Python reference manual. Centrum voor Wiskunde en Informatica Amsterdam, 1995.

[65] T. Kluyver, B. Ragan-Kelley, F. Pérez, et al., "Jupyter notebooks – a publishing format for reproducible computational workflows," in Positioning and Power in Academic Publishing: Players, Agents and Agendas, F. Loizides and B. Schmidt, Eds., IOS Press, 2016, pp. 87–90.

[66] C. R. Harris, K. J. Millman, S. J. van der Walt, et al., "Array programming with NumPy," Nature, vol. 585, no. 7825, pp. 357–362, Sep. 2020. DOI: `10.1038/s41586-020-2649-2`. [Online]. Available: `https://doi.org/10.1038/s41586-020-2649-2`.

[67] T. pandas development team, Pandas-dev/pandas: Pandas, version 1.4.2, Apr. 2022. DOI: `10.5281/zenodo.6408044`. [Online]. Available: `https://doi.org/10.5281/zenodo.6408044`.

[68] F. Pedregosa, G. Varoquaux, A. Gramfort, et al., "Scikit-learn: Machine learning in Python," Journal of Machine Learning Research, vol. 12, pp. 2825–2830, 2011.

[69] J. D. Hunter, "Matplotlib: A 2d graphics environment," Computing in Science & Engineering, vol. 9, no. 3, pp. 90–95, 2007. DOI: `10.5281/zenodo.6513224`.

[70] "Intrusion detection evaluation dataset (cic-ids2017)." (), [Online]. Available: `https://www.unb.ca/cic/datasets/ids-2017.html`.

[71] A. Gharib, I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, "An evaluation framework for intrusion detection dataset," in 2016 International Conference on Information Science and Security (ICISS), IEEE, 2016, pp. 1–6.

[72] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, "Toward generating a new intrusion detection dataset and intrusion traffic characterization.," ICISSp, vol. 1, pp. 108–116, 2018.

[73] A. H. Lashkari, G. Draper-Gil, M. S. I. Mamun, and A. A. Ghorbani, "Characterization of tor traffic using time based features.," in ICISSp, 2017, pp. 253–262.

[74] D. Stiawan, M. Y. B. Idris, A. M. Bamhdi, R. Budiarto, et al., "Cicids-2017 dataset feature analysis with information gain for anomaly detection," IEEE Access, vol. 8, pp. 132 911–132 921, 2020.

[75] "Sklearn.preprocessing.labelencoder." (), [Online]. Available: `https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelEncoder.html`. (accessed: 10.06.2022).

[76] "Sklearn.model$_selection.train_test_split$." (), [Online]. Available: `https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html`. (accessed: 11.06.2022).

[77] "Sklearn.ensemble.randomforestregressor." (), [Online]. Available: `https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html`. (accessed: 12.06.2022).

[78] R. Alshammari and A. N. Zincir-Heywood, "A flow based approach for ssh traffic detection," in 2007 IEEE international conference on systems, man and cybernetics, IEEE, 2007, pp. 296–301.

[79] C. Kingsford and S. L. Salzberg, "What are decision trees?" Nature biotechnology, vol. 26, no. 9, pp. 1011–1013, 2008.

[80] "Random forest algorithm." (), [Online]. Available: `https://www.javatpoint.com/machine-learning-random-forest-algorithm`. (accessed: 14.06.2022).

[81] T. Cover and P. Hart, "Nearest neighbor pattern classification," IEEE transactions on information theory, vol. 13, no. 1, pp. 21–27, 1967.

[82] W. S. Noble, "What is a support vector machine?" Nature biotechnology, vol. 24, no. 12, pp. 1565–1567, 2006.

[83] "Sklearn.ensemble.randomforestclassifier." (), [Online]. Available: `https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html`. (accessed: 15.06.2022).

[84] "Sklearn.tree.decisiontreeclassifier." (), [Online]. Available: `https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html`. (accessed: 15.06.2022).

[85] "Sklearn.neighbors.kneighborsclassifier." (), [Online]. Available: `https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html`. (accessed: 15.06.2022).

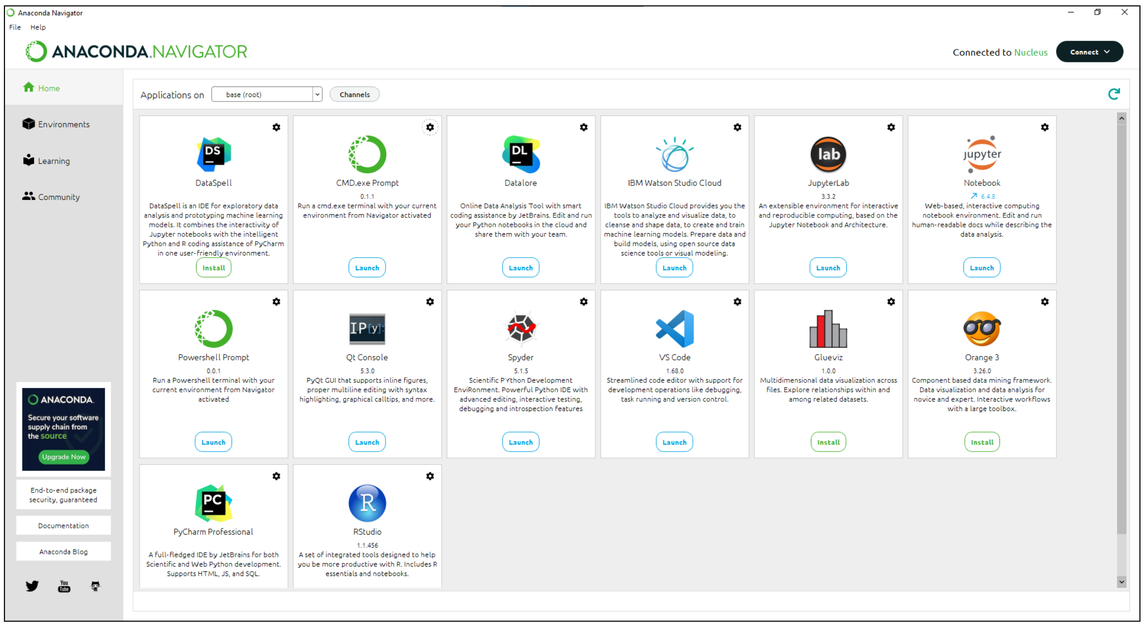
[86] "Sklearn.neural$_network.mlpclassifier$." (), [Online]. Available: `https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html`. (accessed: 15.06.2022).

[87] "Sklearn.svm.svc." (), [Online]. Available: `https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html`. (accessed: 15.06.2022).

[88] O. Faker and E. Dogdu, "Intrusion detection using big data and deep learning techniques," in Proceedings of the 2019 ACM Southeast Conference, 2019, pp. 86–93.

Appendix

A. Work Environment



DataSpell
DataSpell is an IDE for exploratory data analysis and prototyping machine learning models. It combines the interactivity of Jupyter notebooks with the intelligent Python and R coding assistance of PyCharm in one user-friendly environment.
Install

CMD.exe Prompt
0.1.1
Run a cmd.exe terminal with your current environment from Navigator activated
Launch

Datalore
Online Data Analysis Tool with smart coding assistance by JetBrains. Edit and run your Python notebooks in the cloud and share them with your team.
Launch

IBM Watson Studio Cloud
IBM Watson Studio Cloud provides you the tools to analyze and visualize data, to cleanse and shape data, to create and train machine learning models. Prepare data and build models, using open source data science tools or visual modeling.
Launch

JupyterLab
3.3.2
An extensible environment for interactive and reproducible computing, based on the Jupyter Notebook and Architecture.
Launch

Notebook
6.4.8
Web-based, interactive computing notebook environment. Edit and run human-readable docs while describing the data analysis.
Launch

Powershell Prompt
0.0.1
Run a Powershell terminal with your current environment from Navigator activated
Launch

Qt Console
5.3.0
PyQt GUI that supports inline figures, proper multiline editing with syntax highlighting, graphical calltips, and more.
Launch

Spyder
5.1.5
Scientific PYthon Development EnviRonment. Powerful Python IDE with advanced editing, interactive testing, debugging and introspection features
Launch

VS Code
1.68.0
Streamlined code editor with support for development operations like debugging, task running and version control.
Launch

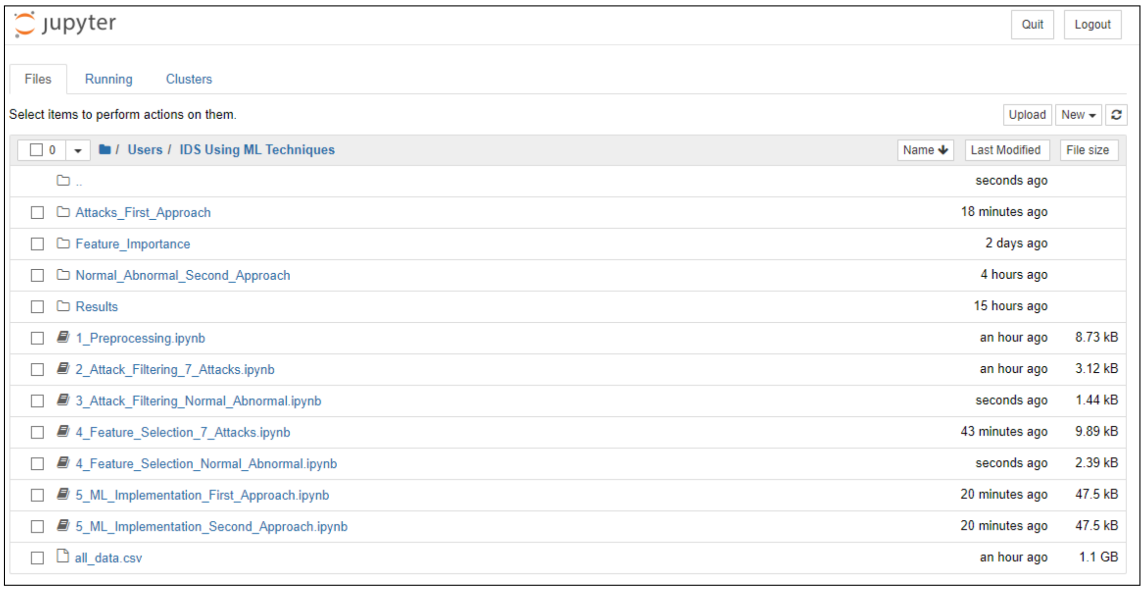
Glueviz
1.0.0
Multidimensional data visualization across files. Explore relationships within and among related datasets.
Install

Orange 3
3.26.0
Component based data mining framework. Data visualization and data analysis for novice and expert. Interactive workflows with a large toolbox.
Install

PyCharm Professional
A full-fledged IDE by JetBrains for both Scientific and Web Python development. Supports HTML, JS, and SQL.

RStudio
1.1.456
A set of integrated tools designed to help you be more productive with R. Includes R essentials and notebooks.

Figure A.1: Anaconda Navigator.



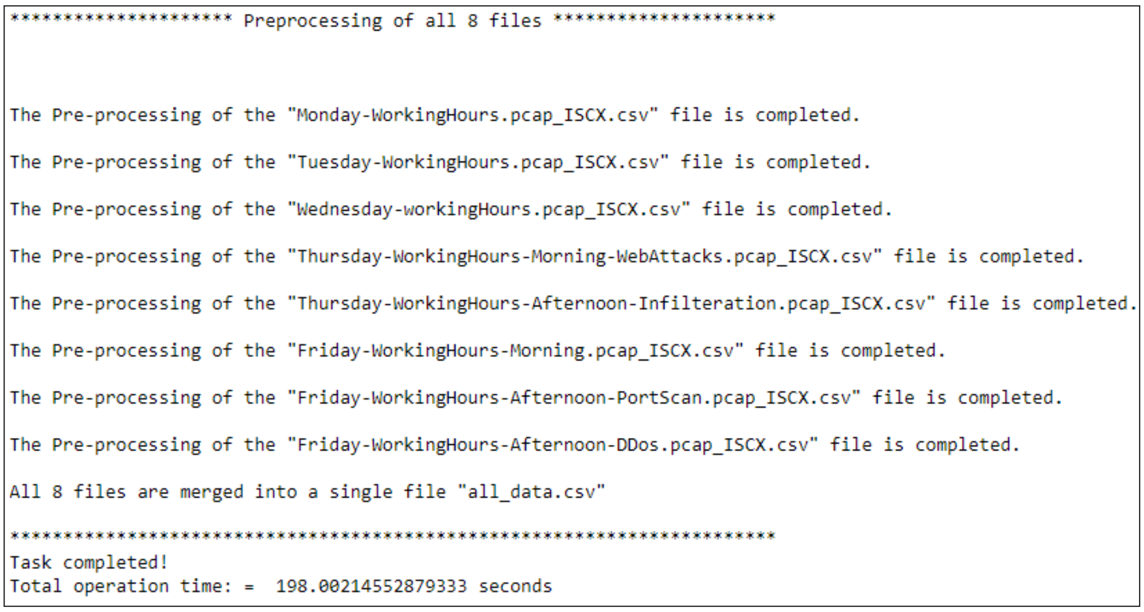
Select items to perform actions on them.

0 / Users / IDS Using ML Techniques

Name	Last Modified	File size
..	seconds ago	
Attacks_First_Approach	18 minutes ago	
Feature_Importance	2 days ago	
Normal_Abnormal_Second_Approach	4 hours ago	
Results	15 hours ago	
1_Preprocessing.ipynb	an hour ago	8.73 kB
2_Attack_Filtering_7_Attacks.ipynb	an hour ago	3.12 kB
3_Attack_Filtering_Normal_Abnormal.ipynb	seconds ago	1.44 kB
4_Feature_Selection_7_Attacks.ipynb	43 minutes ago	9.89 kB
4_Feature_Selection_Normal_Abnormal.ipynb	seconds ago	2.39 kB
5_ML_Implementation_First_Approach.ipynb	20 minutes ago	47.5 kB
5_ML_Implementation_Second_Approach.ipynb	20 minutes ago	47.5 kB
all_data.csv	an hour ago	1.1 GB

Figure A.2: Jupyter Notebook.

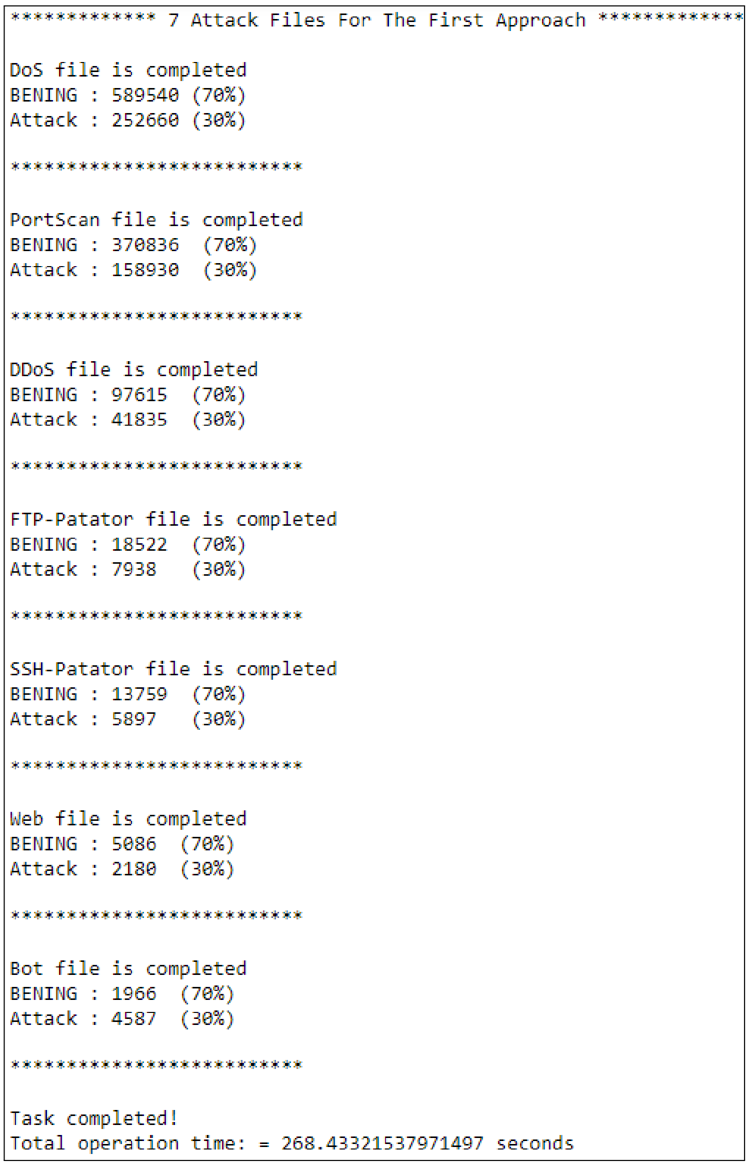
B. Data Preprocessing



```
******************** Preprocessing of all 8 files ********************


The Pre-processing of the "Monday-WorkingHours.pcap_ISCX.csv" file is completed.

The Pre-processing of the "Tuesday-WorkingHours.pcap_ISCX.csv" file is completed.

The Pre-processing of the "Wednesday-workingHours.pcap_ISCX.csv" file is completed.

The Pre-processing of the "Thursday-WorkingHours-Morning-WebAttacks.pcap_ISCX.csv" file is completed.

The Pre-processing of the "Thursday-WorkingHours-Afternoon-Infilteration.pcap_ISCX.csv" file is completed.

The Pre-processing of the "Friday-WorkingHours-Morning.pcap_ISCX.csv" file is completed.

The Pre-processing of the "Friday-WorkingHours-Afternoon-PortScan.pcap_ISCX.csv" file is completed.

The Pre-processing of the "Friday-WorkingHours-Afternoon-DDos.pcap_ISCX.csv" file is completed.

All 8 files are merged into a single file "all_data.csv"

**********************************************************************
Task completed!
Total operation time: =  198.00214552879333 seconds
```

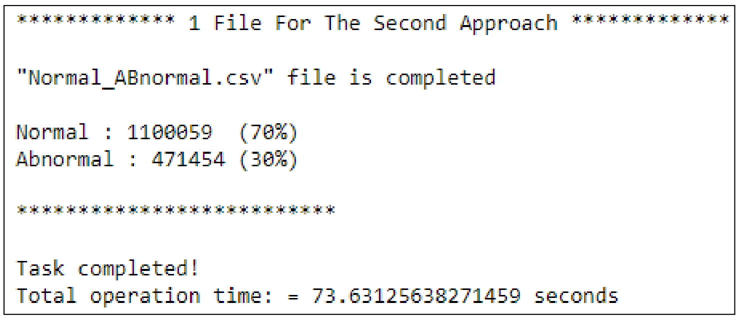
Figure B.1: The Preprocessing of all 8 files.

C. Attack Filtering



```
************* 7 Attack Files For The First Approach *************

DoS file is completed
BENING : 589540 (70%)
Attack : 252660 (30%)

**************************

PortScan file is completed
BENING : 370836  (70%)
Attack : 158930  (30%)

**************************

DDoS file is completed
BENING : 97615  (70%)
Attack : 41835  (30%)

**************************

FTP-Patator file is completed
BENING : 18522  (70%)
Attack : 7938   (30%)

**************************

SSH-Patator file is completed
BENING : 13759  (70%)
Attack : 5897   (30%)

**************************

Web file is completed
BENING : 5086  (70%)
Attack : 2180  (30%)

**************************

Bot file is completed
BENING : 1966  (70%)
Attack : 4587  (30%)

**************************

Task completed!
Total operation time: = 268.43321537971497 seconds
```

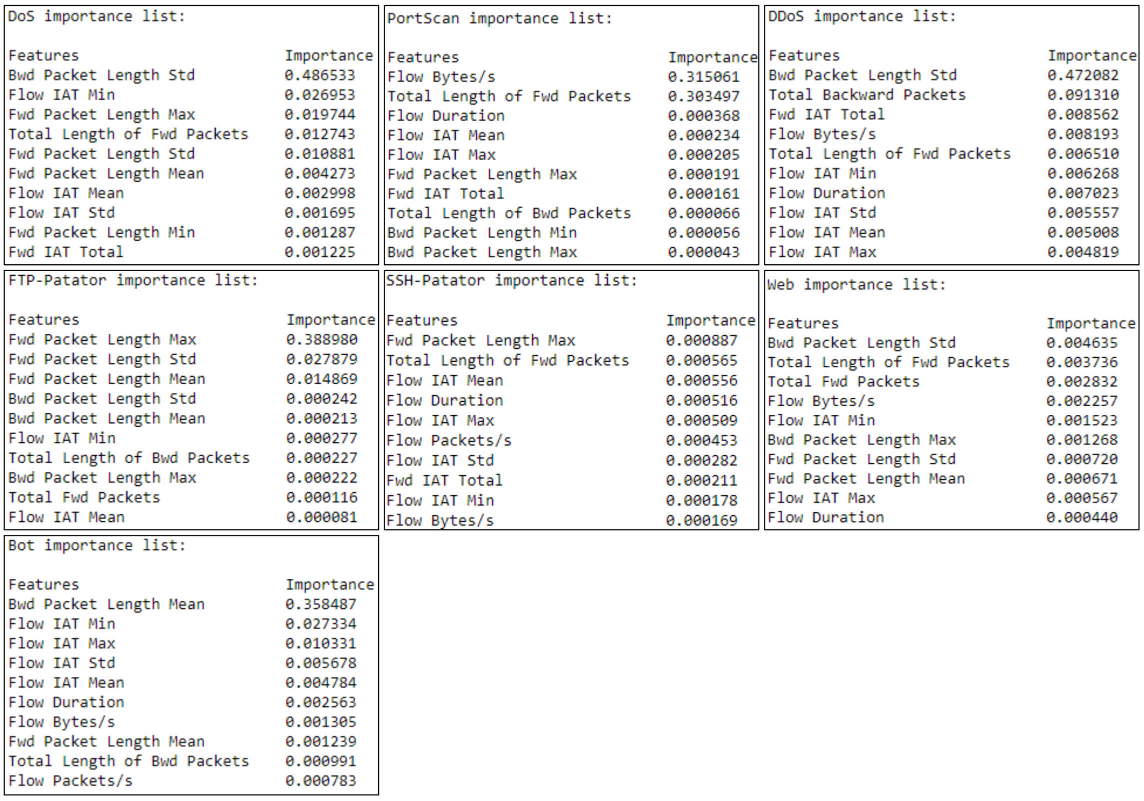
Figure C.1: Attack filtering for All 7 attacks files.



```
************* 1 File For The Second Approach *************

"Normal_ABnormal.csv" file is completed

Normal : 1100059  (70%)
Abnormal : 471454 (30%)

**************************

Task completed!
Total operation time: = 73.63125638271459 seconds
```

Figure C.2: Attack filtering for Normal/Abnormal file.

D. Feature Selection



```
DoS importance list:

Features                        Importance
Bwd Packet Length Std           0.486533
Flow IAT Min                    0.026953
Fwd Packet Length Max           0.019744
Total Length of Fwd Packets     0.012743
Fwd Packet Length Std           0.010881
Fwd Packet Length Mean          0.004273
Flow IAT Mean                   0.002998
Flow IAT Std                    0.001695
Fwd Packet Length Min           0.001287
Fwd IAT Total                   0.001225
```

```
PortScan importance list:

Features                        Importance
Flow Bytes/s                    0.315061
Total Length of Fwd Packets     0.303497
Flow Duration                   0.000368
Flow IAT Mean                   0.000234
Flow IAT Max                    0.000205
Fwd Packet Length Max           0.000191
Fwd IAT Total                   0.000161
Total Length of Bwd Packets     0.000066
Bwd Packet Length Min           0.000056
Bwd Packet Length Max           0.000043
```

```
DDoS importance list:

Features                        Importance
Bwd Packet Length Std           0.472082
Total Backward Packets          0.091310
Fwd IAT Total                   0.008562
Flow Bytes/s                    0.008193
Total Length of Fwd Packets     0.006510
Flow IAT Min                    0.006268
Flow Duration                   0.007023
Flow IAT Std                    0.005557
Flow IAT Mean                   0.005008
Flow IAT Max                    0.004819
```

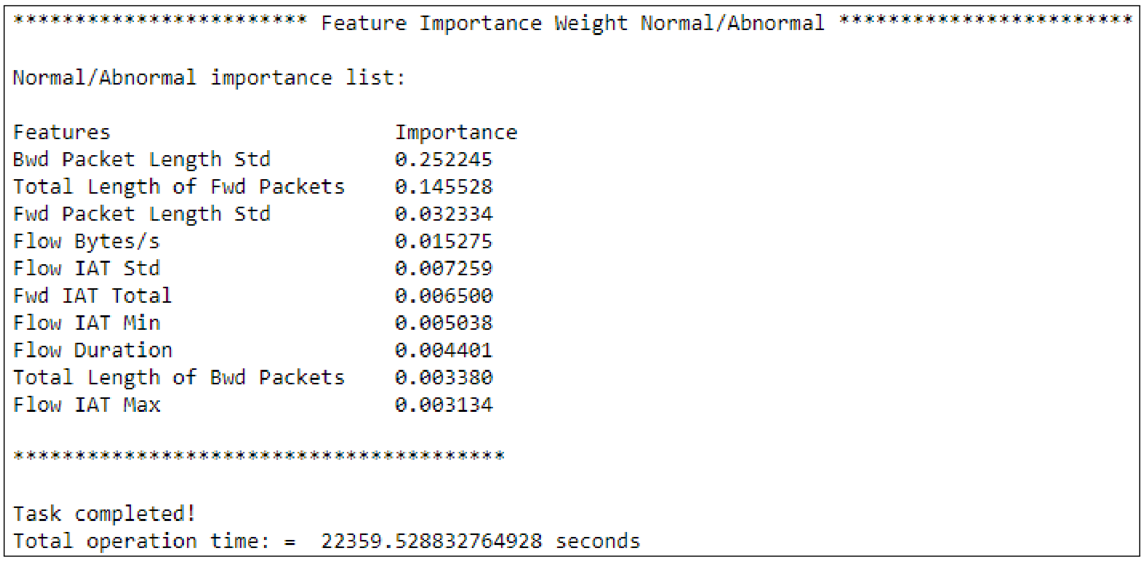
```
FTP-Patator importance list:

Features                        Importance
Fwd Packet Length Max           0.388980
Fwd Packet Length Std           0.027879
Fwd Packet Length Mean          0.014869
Bwd Packet Length Std           0.000242
Bwd Packet Length Mean          0.000213
Flow IAT Min                    0.000277
Total Length of Bwd Packets     0.000227
Bwd Packet Length Max           0.000222
Total Fwd Packets               0.000116
Flow IAT Mean                   0.000081
```

```
SSH-Patator importance list:

Features                        Importance
Fwd Packet Length Max           0.000887
Total Length of Fwd Packets     0.000565
Flow IAT Mean                   0.000556
Flow Duration                   0.000516
Flow IAT Max                    0.000509
Flow Packets/s                  0.000453
Flow IAT Std                    0.000282
Fwd IAT Total                   0.000211
Flow IAT Min                    0.000178
Flow Bytes/s                    0.000169
```

```
Web importance list:

Features                        Importance
Bwd Packet Length Std           0.004635
Total Length of Fwd Packets     0.003736
Total Fwd Packets               0.002832
Flow Bytes/s                    0.002257
Flow IAT Min                    0.001523
Bwd Packet Length Max           0.001268
Fwd Packet Length Std           0.000720
Fwd Packet Length Mean          0.000671
Flow IAT Max                    0.000567
Flow Duration                   0.000440
```

```
Bot importance list:

Features                        Importance
Bwd Packet Length Mean          0.358487
Flow IAT Min                    0.027334
Flow IAT Max                    0.010331
Flow IAT Std                    0.005678
Flow IAT Mean                   0.004784
Flow Duration                   0.002563
Flow Bytes/s                    0.001305
Fwd Packet Length Mean          0.001239
Total Length of Bwd Packets     0.000991
Flow Packets/s                  0.000783
```

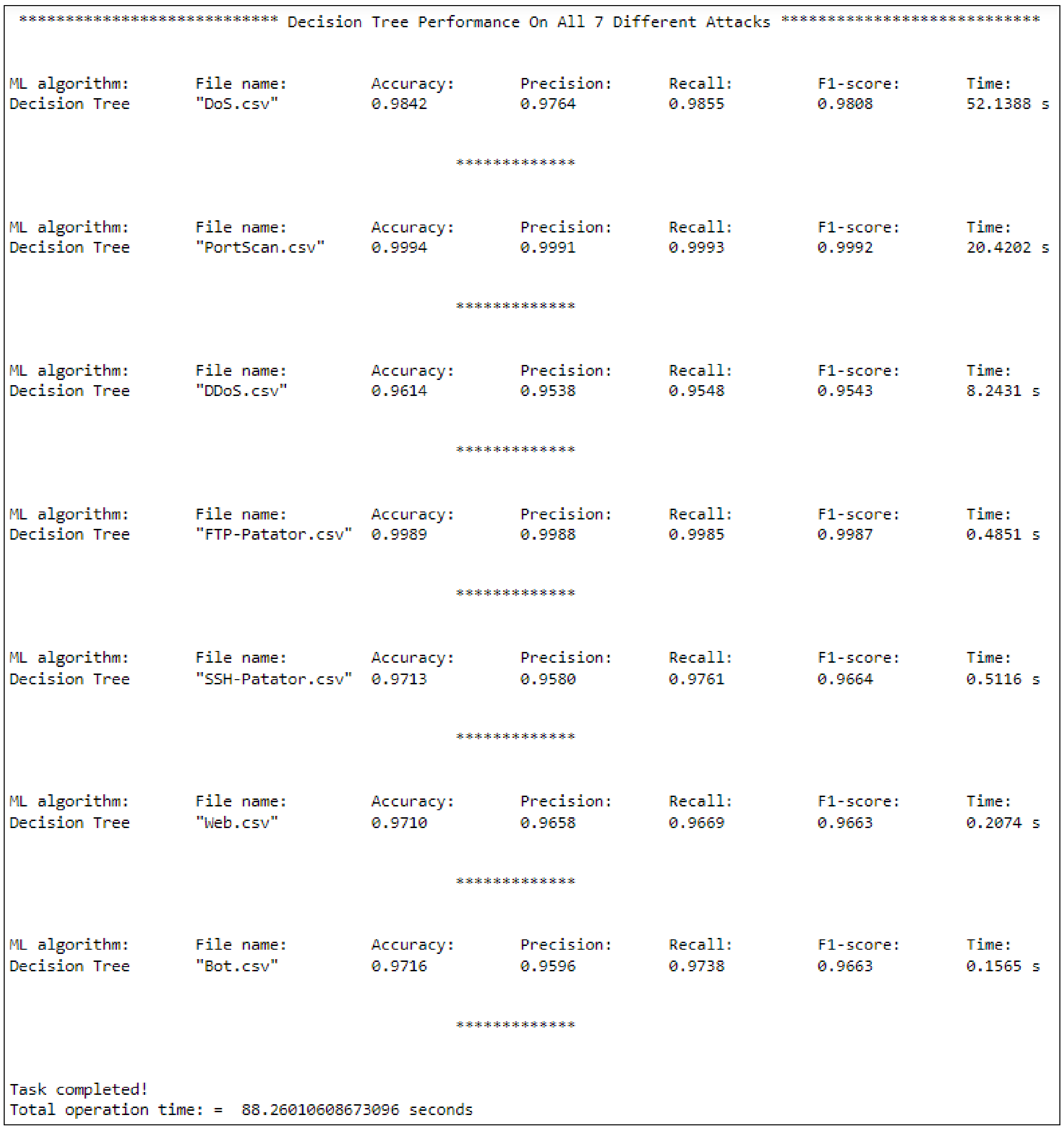
Figure D.1: The importance weights values of features according to 7 attacks.



```
*********************** Feature Importance Weight Normal/Abnormal ***********************

Normal/Abnormal importance list:

Features                        Importance
Bwd Packet Length Std           0.252245
Total Length of Fwd Packets     0.145528
Fwd Packet Length Std           0.032334
Flow Bytes/s                    0.015275
Flow IAT Std                    0.007259
Fwd IAT Total                   0.006500
Flow IAT Min                    0.005038
Flow Duration                   0.004401
Total Length of Bwd Packets     0.003380
Flow IAT Max                    0.003134

****************************************

Task completed!
Total operation time: =  22359.528832764928 seconds
```

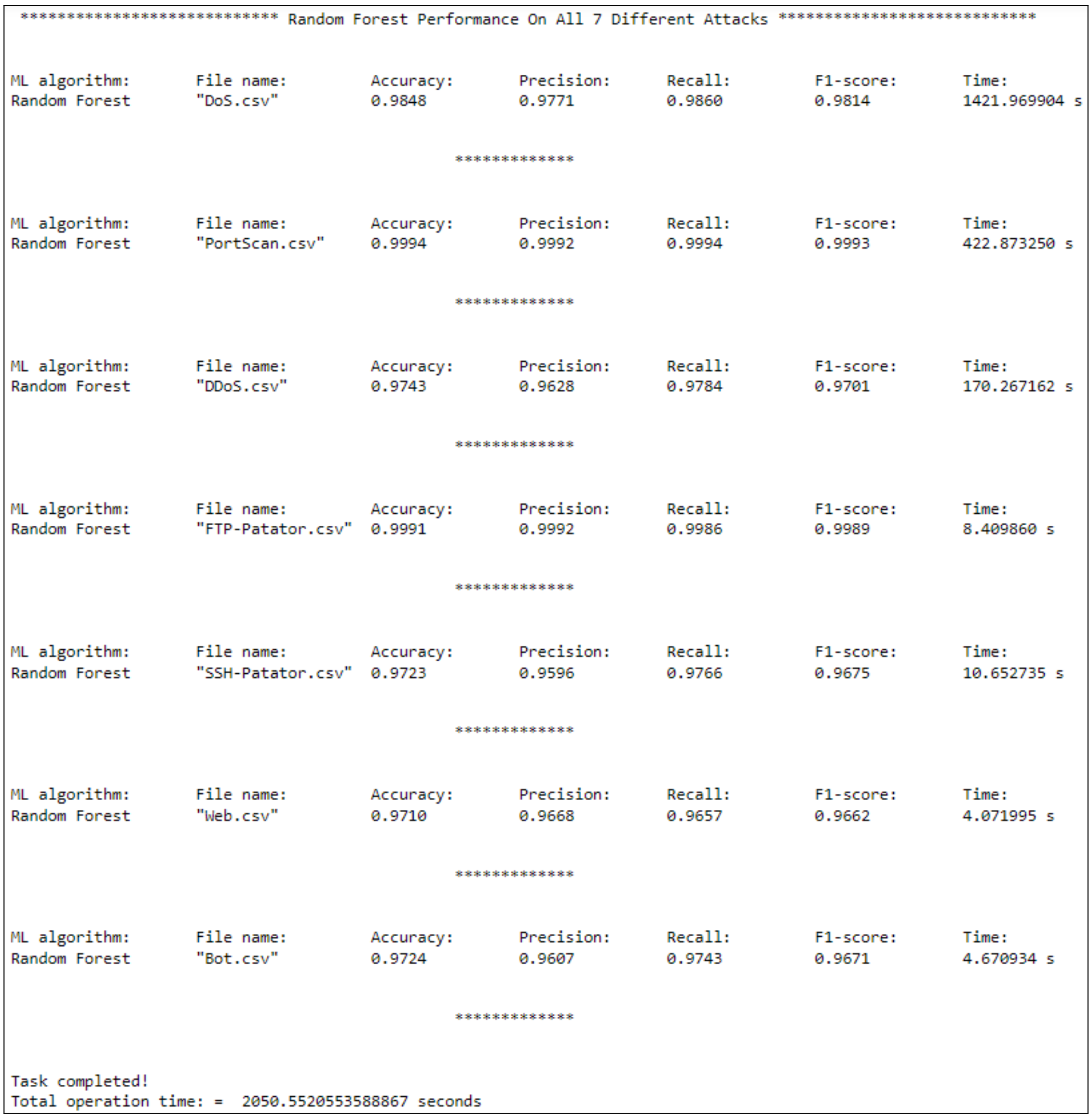
Figure D.2: The importance weights values of features according to Normal/Abnormal.

E. Results



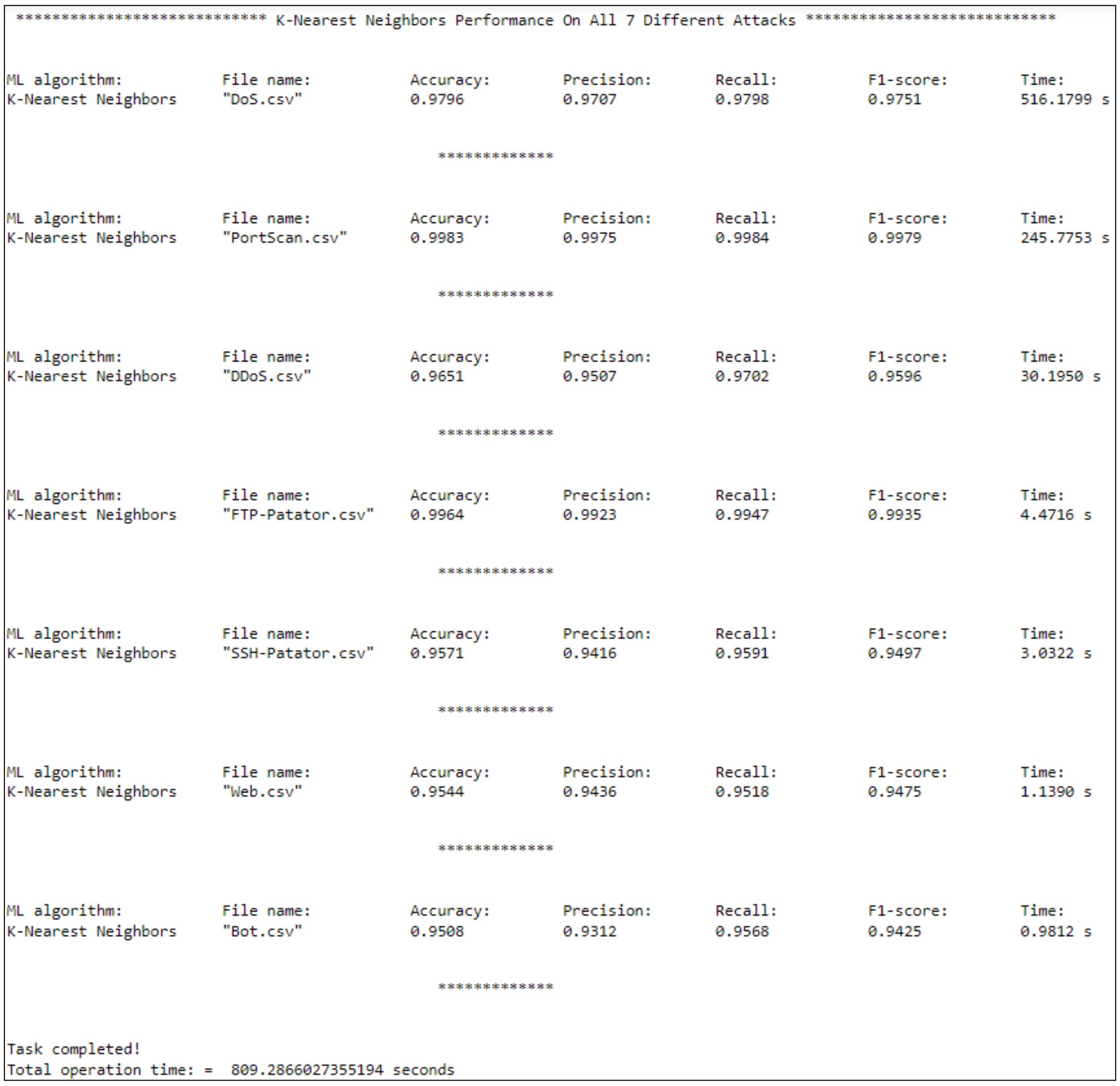
```
**************************** Decision Tree Performance On All 7 Different Attacks ****************************


ML algorithm:       File name:          Accuracy:       Precision:      Recall:         F1-score:       Time:
Decision Tree       "DoS.csv"           0.9842          0.9764          0.9855          0.9808          52.1388 s


                                                    *************


ML algorithm:       File name:          Accuracy:       Precision:      Recall:         F1-score:       Time:
Decision Tree       "PortScan.csv"      0.9994          0.9991          0.9993          0.9992          20.4202 s


                                                    *************


ML algorithm:       File name:          Accuracy:       Precision:      Recall:         F1-score:       Time:
Decision Tree       "DDoS.csv"          0.9614          0.9538          0.9548          0.9543          8.2431 s


                                                    *************


ML algorithm:       File name:          Accuracy:       Precision:      Recall:         F1-score:       Time:
Decision Tree       "FTP-Patator.csv"   0.9989          0.9988          0.9985          0.9987          0.4851 s


                                                    *************


ML algorithm:       File name:          Accuracy:       Precision:      Recall:         F1-score:       Time:
Decision Tree       "SSH-Patator.csv"   0.9713          0.9580          0.9761          0.9664          0.5116 s


                                                    *************


ML algorithm:       File name:          Accuracy:       Precision:      Recall:         F1-score:       Time:
Decision Tree       "Web.csv"           0.9710          0.9658          0.9669          0.9663          0.2074 s


                                                    *************


ML algorithm:       File name:          Accuracy:       Precision:      Recall:         F1-score:       Time:
Decision Tree       "Bot.csv"           0.9716          0.9596          0.9738          0.9663          0.1565 s


                                                    *************


Task completed!
Total operation time: =  88.26010608673096 seconds
```

Figure E.1: Decision Tree performance in the first approach.



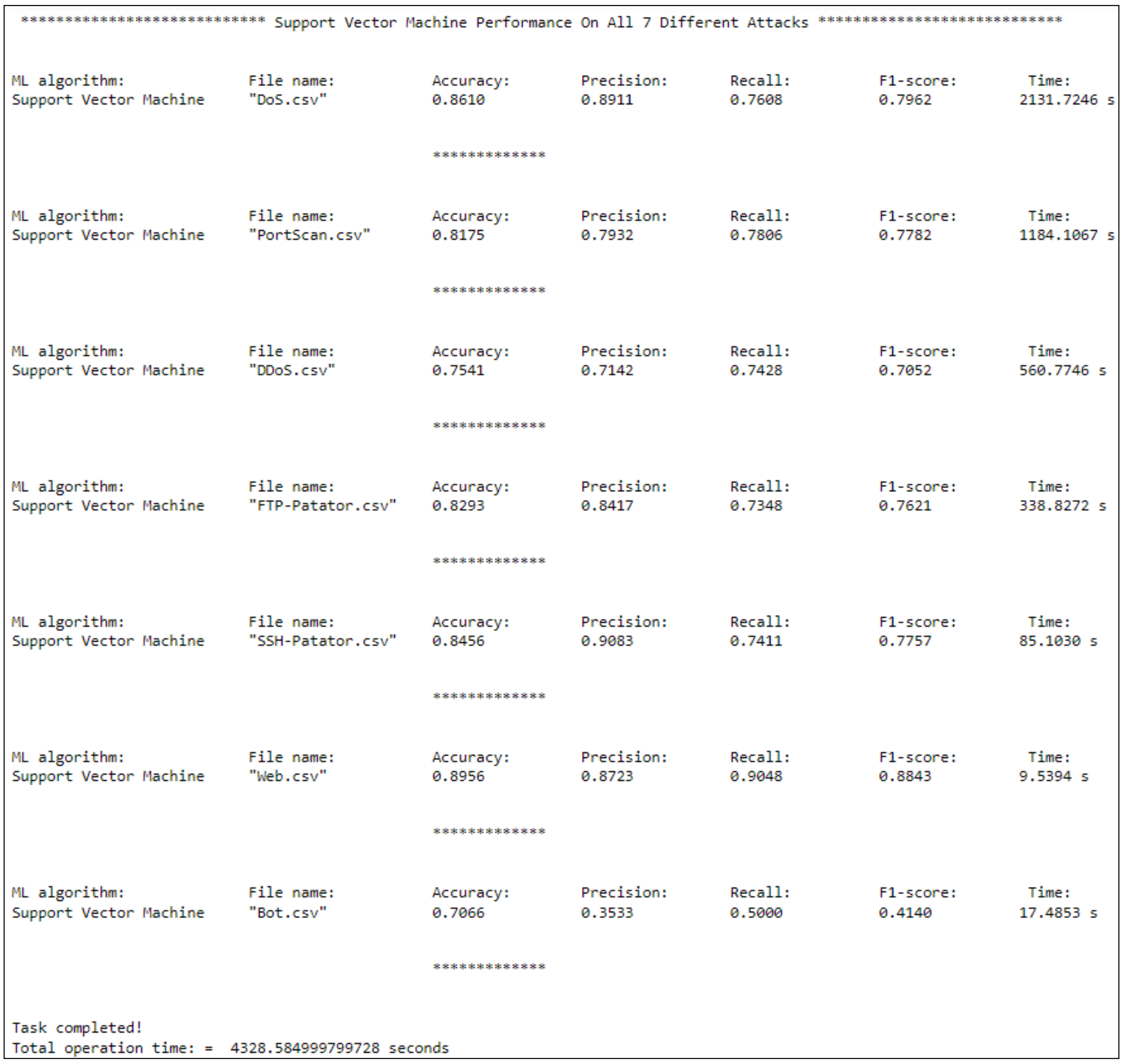
```
 *************************** Random Forest Performance On All 7 Different Attacks ***************************


ML algorithm:       File name:         Accuracy:       Precision:      Recall:         F1-score:       Time:
Random Forest       "DoS.csv"          0.9848          0.9771          0.9860          0.9814          1421.969904 s


                                                    *************


ML algorithm:       File name:         Accuracy:       Precision:      Recall:         F1-score:       Time:
Random Forest       "PortScan.csv"     0.9994          0.9992          0.9994          0.9993          422.873250 s


                                                    *************


ML algorithm:       File name:         Accuracy:       Precision:      Recall:         F1-score:       Time:
Random Forest       "DDoS.csv"         0.9743          0.9628          0.9784          0.9701          170.267162 s


                                                    *************


ML algorithm:       File name:         Accuracy:       Precision:      Recall:         F1-score:       Time:
Random Forest       "FTP-Patator.csv"  0.9991          0.9992          0.9986          0.9989          8.409860 s


                                                    *************


ML algorithm:       File name:         Accuracy:       Precision:      Recall:         F1-score:       Time:
Random Forest       "SSH-Patator.csv"  0.9723          0.9596          0.9766          0.9675          10.652735 s


                                                    *************


ML algorithm:       File name:         Accuracy:       Precision:      Recall:         F1-score:       Time:
Random Forest       "Web.csv"          0.9710          0.9668          0.9657          0.9662          4.071995 s


                                                    *************


ML algorithm:       File name:         Accuracy:       Precision:      Recall:         F1-score:       Time:
Random Forest       "Bot.csv"          0.9724          0.9607          0.9743          0.9671          4.670934 s


                                                    *************


Task completed!
Total operation time: =  2050.5520553588867 seconds
```

Figure E.2: Random Forest performance in the first approach.



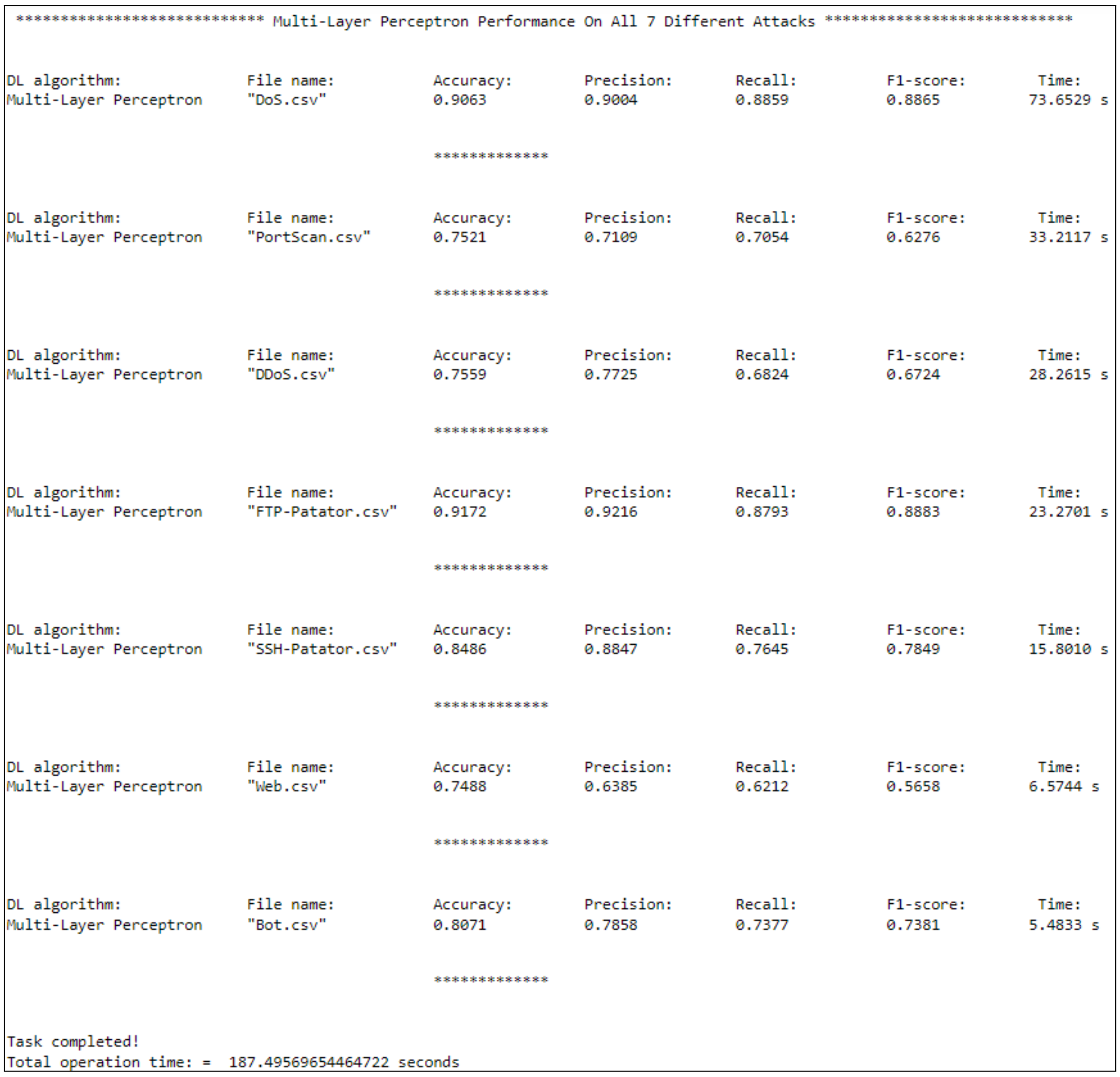
```
**************************** K-Nearest Neighbors Performance On All 7 Different Attacks ****************************


ML algorithm:           File name:          Accuracy:       Precision:      Recall:         F1-score:       Time:
K-Nearest Neighbors     "DoS.csv"           0.9796          0.9707          0.9798          0.9751          516.1799 s

                                                  *************


ML algorithm:           File name:          Accuracy:       Precision:      Recall:         F1-score:       Time:
K-Nearest Neighbors     "PortScan.csv"      0.9983          0.9975          0.9984          0.9979          245.7753 s

                                                  *************


ML algorithm:           File name:          Accuracy:       Precision:      Recall:         F1-score:       Time:
K-Nearest Neighbors     "DDoS.csv"          0.9651          0.9507          0.9702          0.9596          30.1950 s

                                                  *************


ML algorithm:           File name:          Accuracy:       Precision:      Recall:         F1-score:       Time:
K-Nearest Neighbors     "FTP-Patator.csv"   0.9964          0.9923          0.9947          0.9935          4.4716 s

                                                  *************


ML algorithm:           File name:          Accuracy:       Precision:      Recall:         F1-score:       Time:
K-Nearest Neighbors     "SSH-Patator.csv"   0.9571          0.9416          0.9591          0.9497          3.0322 s

                                                  *************


ML algorithm:           File name:          Accuracy:       Precision:      Recall:         F1-score:       Time:
K-Nearest Neighbors     "Web.csv"           0.9544          0.9436          0.9518          0.9475          1.1390 s

                                                  *************


ML algorithm:           File name:          Accuracy:       Precision:      Recall:         F1-score:       Time:
K-Nearest Neighbors     "Bot.csv"           0.9508          0.9312          0.9568          0.9425          0.9812 s

                                                  *************


Task completed!
Total operation time: =   809.2866027355194 seconds
```

Figure E.3: K-Nearest Neighbors performance in the first approach.



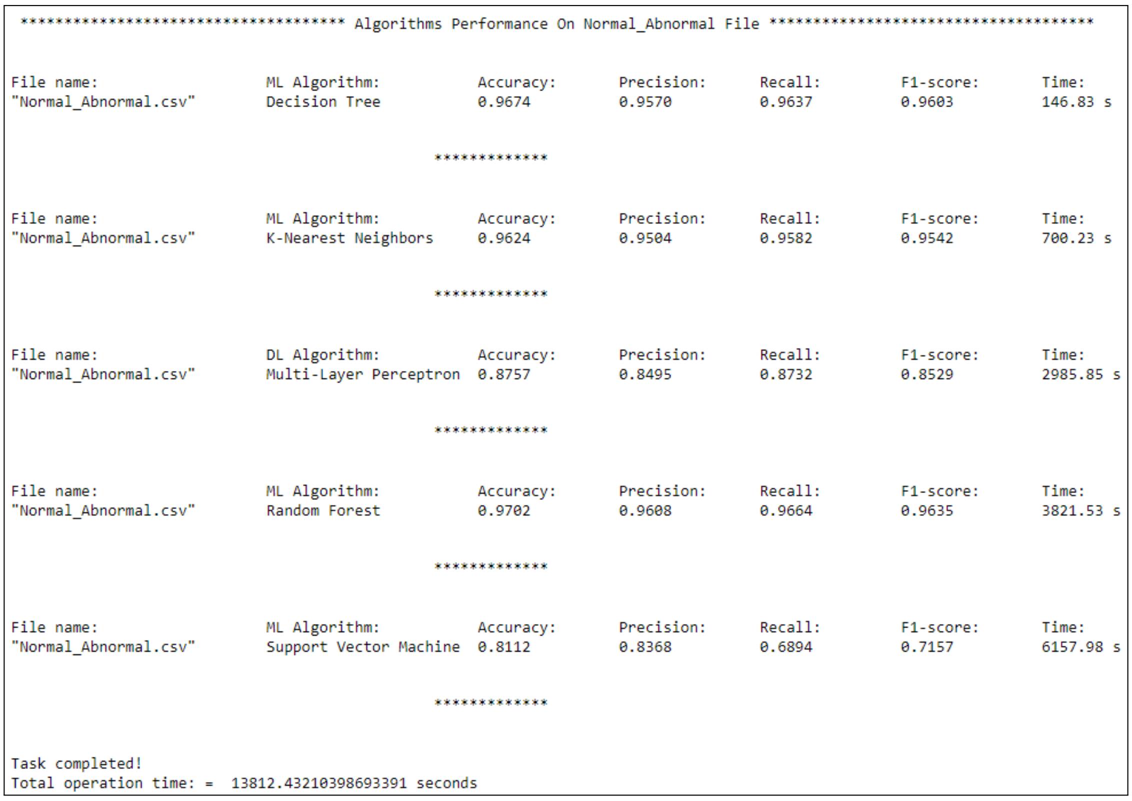
```
*************************** Support Vector Machine Performance On All 7 Different Attacks ***************************


ML algorithm:             File name:          Accuracy:        Precision:       Recall:          F1-score:        Time:
Support Vector Machine    "DoS.csv"           0.8610           0.8911           0.7608           0.7962           2131.7246 s


                                              *************


ML algorithm:             File name:          Accuracy:        Precision:       Recall:          F1-score:        Time:
Support Vector Machine    "PortScan.csv"      0.8175           0.7932           0.7806           0.7782           1184.1067 s


                                              *************


ML algorithm:             File name:          Accuracy:        Precision:       Recall:          F1-score:        Time:
Support Vector Machine    "DDoS.csv"          0.7541           0.7142           0.7428           0.7052           560.7746 s


                                              *************


ML algorithm:             File name:          Accuracy:        Precision:       Recall:          F1-score:        Time:
Support Vector Machine    "FTP-Patator.csv"   0.8293           0.8417           0.7348           0.7621           338.8272 s


                                              *************


ML algorithm:             File name:          Accuracy:        Precision:       Recall:          F1-score:        Time:
Support Vector Machine    "SSH-Patator.csv"   0.8456           0.9083           0.7411           0.7757           85.1030 s


                                              *************


ML algorithm:             File name:          Accuracy:        Precision:       Recall:          F1-score:        Time:
Support Vector Machine    "Web.csv"           0.8956           0.8723           0.9048           0.8843           9.5394 s


                                              *************


ML algorithm:             File name:          Accuracy:        Precision:       Recall:          F1-score:        Time:
Support Vector Machine    "Bot.csv"           0.7066           0.3533           0.5000           0.4140           17.4853 s


                                              *************


Task completed!
Total operation time: =   4328.584999799728 seconds
```

Figure E.4: Support Vector Machine performance in the first approach.



```
*************************** Multi-Layer Perceptron Performance On All 7 Different Attacks ***************************


DL algorithm:            File name:          Accuracy:       Precision:      Recall:         F1-score:        Time:
Multi-Layer Perceptron   "DoS.csv"           0.9063          0.9004          0.8859          0.8865           73.6529 s

                                             *************


DL algorithm:            File name:          Accuracy:       Precision:      Recall:         F1-score:        Time:
Multi-Layer Perceptron   "PortScan.csv"      0.7521          0.7109          0.7054          0.6276           33.2117 s

                                             *************


DL algorithm:            File name:          Accuracy:       Precision:      Recall:         F1-score:        Time:
Multi-Layer Perceptron   "DDoS.csv"          0.7559          0.7725          0.6824          0.6724           28.2615 s

                                             *************


DL algorithm:            File name:          Accuracy:       Precision:      Recall:         F1-score:        Time:
Multi-Layer Perceptron   "FTP-Patator.csv"   0.9172          0.9216          0.8793          0.8883           23.2701 s

                                             *************


DL algorithm:            File name:          Accuracy:       Precision:      Recall:         F1-score:        Time:
Multi-Layer Perceptron   "SSH-Patator.csv"   0.8486          0.8847          0.7645          0.7849           15.8010 s

                                             *************


DL algorithm:            File name:          Accuracy:       Precision:      Recall:         F1-score:        Time:
Multi-Layer Perceptron   "Web.csv"           0.7488          0.6385          0.6212          0.5658           6.5744 s

                                             *************


DL algorithm:            File name:          Accuracy:       Precision:      Recall:         F1-score:        Time:
Multi-Layer Perceptron   "Bot.csv"           0.8071          0.7858          0.7377          0.7381           5.4833 s

                                             *************


Task completed!
Total operation time: =  187.49569654464722 seconds
```

Figure E.5: Multi-Layer Perceptron performance in the first approach.



```
************************************* Algorithms Performance On Normal_Abnormal File *************************************


File name:                    ML Algorithm:          Accuracy:        Precision:       Recall:          F1-score:        Time:
"Normal_Abnormal.csv"         Decision Tree          0.9674           0.9570           0.9637           0.9603           146.83 s

                                                *************


File name:                    ML Algorithm:          Accuracy:        Precision:       Recall:          F1-score:        Time:
"Normal_Abnormal.csv"         K-Nearest Neighbors    0.9624           0.9504           0.9582           0.9542           700.23 s

                                                *************


File name:                    DL Algorithm:          Accuracy:        Precision:       Recall:          F1-score:        Time:
"Normal_Abnormal.csv"         Multi-Layer Perceptron 0.8757           0.8495           0.8732           0.8529           2985.85 s

                                                *************


File name:                    ML Algorithm:          Accuracy:        Precision:       Recall:          F1-score:        Time:
"Normal_Abnormal.csv"         Random Forest          0.9702           0.9608           0.9664           0.9635           3821.53 s

                                                *************


File name:                    ML Algorithm:          Accuracy:        Precision:       Recall:          F1-score:        Time:
"Normal_Abnormal.csv"         Support Vector Machine 0.8112           0.8368           0.6894           0.7157           6157.98 s

                                                *************


Task completed!
Total operation time: =  13812.43210398693391 seconds
```

Figure E.6: Performance of all algorithms in the second approach.