REPUBLIC OF TÜRKİYE

ALTINBAŞ UNIVERSITY

Institute of Graduate Studies

Electrical and Computer Engineering

A NEW FRAME WORK BASED APPLICATION FOR MALWARE DETECTION IN IOT

Mahmood Alaa Sadeq ALHADEETHI

Master's Thesis

Supervisor

Prof. Dr. Hasan Hüseyin BALIK

Istanbul, 2023

A NEW FRAME WORK BASED APPLICATION FOR MALWARE DETECTION IN IOT

Mahmood Alaa Sadeq ALHADEETHI

Electrical and Computer Engineering

Master's Thesis

ALTINBAŞ UNIVERSITY

2023

The thesis titled A NEW FRAME WORK BASED APPLICATION FOR MALWARE DETECTION IN IOT prepared and presented by MAHMOOD ALAA SADEQ ALHADEETHI and submitted on 23/08/2023 as been **accepted unanimously** for the degree of Master of Science in Electrical and Computer Engineering.

____
Prof. Dr. Hasan Hüseyin BALIK
Supervisor

Thesis Defense Jury Members:

Prof. Dr. Hasan Hüseyin BALIK	Department of Computer Engineering, İstanbul Aydin University	________________
Asst. Prof. Dr. Oğuz ATA	Department of Software Engineering, Altınbaş University	________________
Asst. Prof. Dr. Aytuğ BOYACI	Department of Engineering and Architecture, Milli Savunma University	________________

I hereby declare that this thesis meets all format and submission requirements of a Master's thesis.

Submission date of the thesis to Institute of Graduate Studies: ___/___/___

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare all unoriginal meterials and conclusion have been cited in the text and all references mentioned in the Reference List have been cited in the text, and vice versa as required by the abovementioned ruled and conduct.

Mahmood Alaa Sadeq ALHADEETHI

Signature

DEDICATION

I would like to express my gratitude to my father, mother, friends, Dr. Aljarah, Mr. Omer Alshami, and all those who provided assistance and support throughout my academic journey. I am sincerely thankful for your contributions.

ABSTRACT

A NEW FRAME WORK BASED APPLICATION FOR MALWARE DETECTION IN IOT

ALHADEETHI, Mahmood Alaa Sadeq
M.Sc., Electrical and Computer Engineering, Altınbaş University,
Supervisor: Prof. Dr. Hasan Hüseyin BALIK
Date: 08/2023
Pages: 82

Recent studies indicate a significant rise in the prevalence of malicious software (malware), which has become a cause for concern. It has been observed that certain types of malware possess the ability to hide within computer systems by employing diverse obfuscation methods. It is crucial to recognize malware before a large number of systems are infected with it in order to protect computers and the Internet from harm. In recent years, there have been several investigations conducted on strategies for detecting malware. Despite this fact, the issue of identifying malware continues to be challenging. When it comes to finding previously identified malware, signature-based and heuristic-based detection approaches perform well. However, it should be noted that signature-based detection methods are not capable of detecting unknown malware. However, it should be noted that behavior-based, model checking-based, and cloud-based methods have shown effectiveness in dealing with unfamiliar and intricate malware. Additionally, there is a growing trend in utilising deep learning-based, mobile device-based, and IoT-based techniques to identify both known and unknown malware. Nonetheless, it is impossible for any technique to identify all forms of malicious software available.

This clearly illustrates the immense challenge of creating a reliable approach for identifying malware, highlighting the pressing demand for original research and methodologies.

This thesis offers an extensive examination of various techniques for detecting malware and the recent methodologies that utilise these techniques. In order to assist researchers in acquiring a broad comprehension of malware detection methods, their advantages and disadvantages, and the optimal means of achieving optimal outcomes, we have identified the most efficient protocol.

Keywords: Malware Detection Approaches, Malware Features, Internet of Things, Machine Learning, Deep Learning.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABBREVIATIONS

AI	:	Artificial Intelligent
AV	:	Antivirus
AMDA	:	Automatic Malware Detection Algorithm
IoT	:	Internet of Thing
PUP	:	Potentially Unwanted Programs
BN	:	Bayesian Network
NB	:	Naive Bayes
LMT	:	Logistic Model Trees
RF	:	Random Forest Tree
KNN	:	K-Nearest Neighbor
MLP	:	Multilayer Perceptron
SLR	:	Simple Logistic Regression
SVM	:	Support Vector Machine
SMO	:	Sequential Minimal Optimization
ACC	:	Accuracy

1. INTRODUCTION

1.1 INTRODUCTION

Using the Internet in our daily lives. The reason for this is that it is extremely difficult to complete any task without the use of the Internet. The Internet is essential for various activities such as social interactions, online finance, health-related transactions, and marketing, among others. As a result of the Internet's exponential growth, criminals have shifted their focus to committing online crimes instead of engaging in physical offences. Criminals often utilise malicious software to launch cyberattacks on targeted computers. Malware is a type of software that intentionally carries out harmful actions on targeted devices such as computers, smartphones, computer networks, and so on. A wide range of malicious software types can be found, encompassing viruses, worms, Trojan horses, rootkits, and ransomware. Every variety and category of malware is specifically crafted to impact the initial target device in a distinct manner, such as inducing harm to the system, enabling remote code execution, pilfering confidential information, and so on. Certain types of malware have the ability to exhibit traits found in various classifications. In an era where malware is becoming more complex, it is becoming progressively challenging to classify it.

In the past ten years, the number of cyberattacks has exploded. The significance can be confirmed by the frequency with which cyber attack incidents are reported in global news stories. The causes are twofold: increasing numbers of individuals and businesses hold social gatherings and conduct business transactions digitally, and the emergence of attack generation toolkits that make cyber attacks easier to execute. Malicious software, also known as malware, is one of the primary causes of daily cyber attacks.

Malware remains one of the most significant cybersecurity concerns today [1, 2]. According to the 2018 Cybersecurity Threat Report [3] from Symantec: In 2017, the number of malware variants detected reached a staggering 669 million, which represents a significant surge of 87.7 percent compared to the previous year. The number of new variants of malware targeting mobile devices, like cell phones, has experienced a surge of around 55%, rising from 17,000 to 27,000. As a result of the advancement and extensive acceptance of IoT technology, there has been a significant surge in malware infections impacting IoT devices in recent years.

According to the report, there has been a significant rise of around 600% in malware variants targeting IoT devices in recent years. As depicted in Figure 1, it poses a significant risk to the presently available IoT device ecosystem.

The public, enterprises, and national security are gravely jeopardized by the rising prevalence of malware.

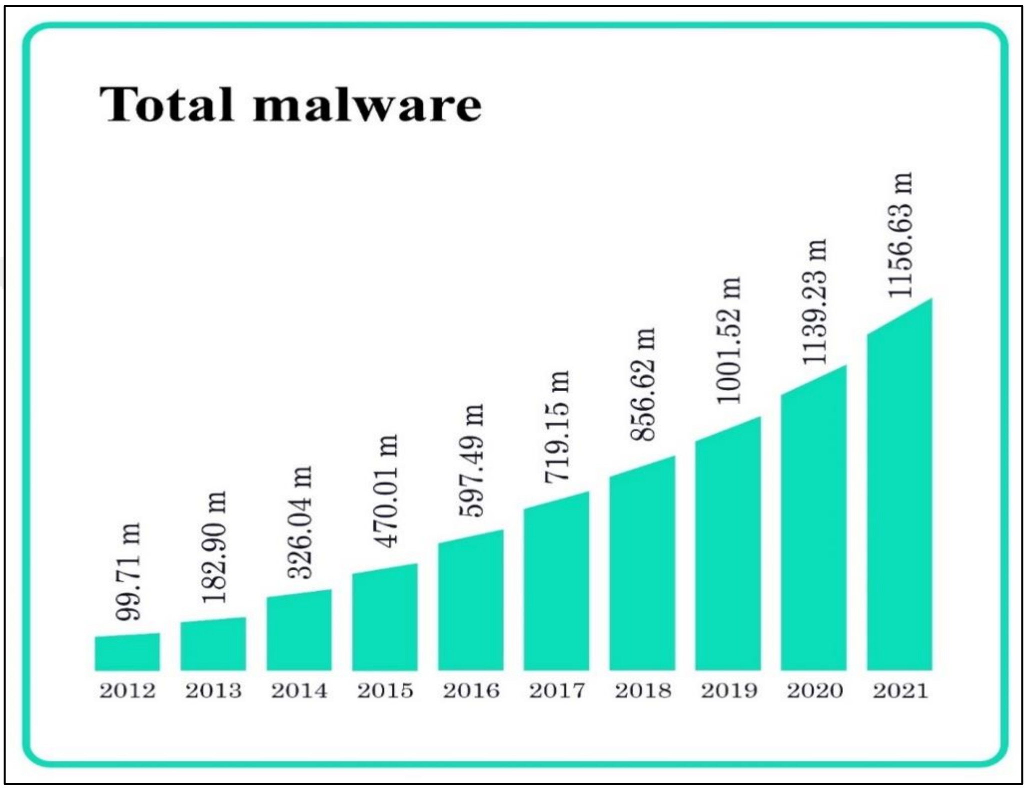


Figure 1.1: Total Malware Increasing [4].

It's a threat to people's privacy and the stability of computer networks. Malware intrusions may have serious repercussions, such as the theft of sensitive information or the shutting down of essential services. To counter malware-based assaults, defense mechanisms need to detect and eliminate infected host machines and network traffic.

1.2 PROBLEM STATEMENT

As computing technology continues to advance, the ability to hide malware attacks from detection becomes increasingly easier and more compact. The severity of unconventional malware in relation to the tools and techniques for analysis and detection is uncertain.

The evaluation of current techniques for detection and protection, along with the analysis and reverse engineering of malware methods, is necessary. In order to effectively and efficiently detect and mitigate emerging and unconventional malware threats, it is crucial to have a reliable detection mechanism in place.

1.3 MALWARE IN IOT DEVICES

The Internet of Things (IoT) refers to a network of interconnected devices and systems that have the ability to communicate independently [5]. The IoT environment is advantageous and convenient for consumers. Given that a significant number of IoT devices are linked to the internet, they become an appealing target for advanced malware. As per a report from HP, it has been found that approximately 70% of Internet of Things devices are susceptible to potential attacks due to weaknesses in password security and encryption [6]. Ransomware is one of the most prevalent types of malware severe hazards to IoT device security. the Ransomware can primarily encrypt and disable the fundamental functions of IoT devices. IoT Ransomware can primarily encrypt and disable the core functionality of IoT devices.

Zhang-Kennedy et al. The authors of the ransomware emphasized that it will predominantly encrypt and disable the fundamental functions of IoT devices. The primary objectives of the study were to determine the IoT attack vectors, self-spreading techniques of IoT ransomware, and the specific IoT application class that is most susceptible to ransomware attacks. The techniques for encrypting Internet of Things devices have been identified by the authors.

Malicious software, commonly known as malware, is any piece of software designed to do harm to its user. All of these and other malicious programs are together referred to as malware. The perpetrator's motivation is no longer to steal data or cause system outages. In the present day, the malware industry has become an incredibly lucrative field.

Various techniques, including as encryption, binary packers, and self-modifying code, are used by malware developers to avoid being identified by malware classifiers. Based on the approach used to analyze code, malware detection methods are often divided into two categories: static analysis and dynamic analysis. The static analysis aims to gather details regarding the organisation and substance of the document.

According to sources [8, 9], malicious software refers to any software that disrupts machine operations, gathers sensitive data, or breaches private computer systems. Cybercriminals develop malware with the intention of achieving particular objectives. The objectives listed in the source may involve acquiring confidential information, gathering login details and credentials, distributing unsolicited emails, initiating Denial of Service attacks, engaging in extortion, and committing identity theft [9]. According to sources, cybercriminals have been utilising Crypto Locker as a means to infect and encrypt files on computers. This malicious software is then used to demand a ransom in exchange for decrypting the files [10].

1.4 THE ADVANCEMENT OF MALWARE

On the basis of the time period during which it emerged, malware evolution is typically divided into five phases. The earliest aspect of malware is when it first appeared. The second stage of malware development saw the introduction of Windows and email attacks. The Internet entered its third stage after its development. Network viruses become the most widespread danger with the introduction of the Internet. The most dangerous forms of malware, such as rootkits and ransomware, emerged in the fourth stage of development. The final stage of malware development encompasses the production of espionage malware purposes by the secret services of certain nations [11].

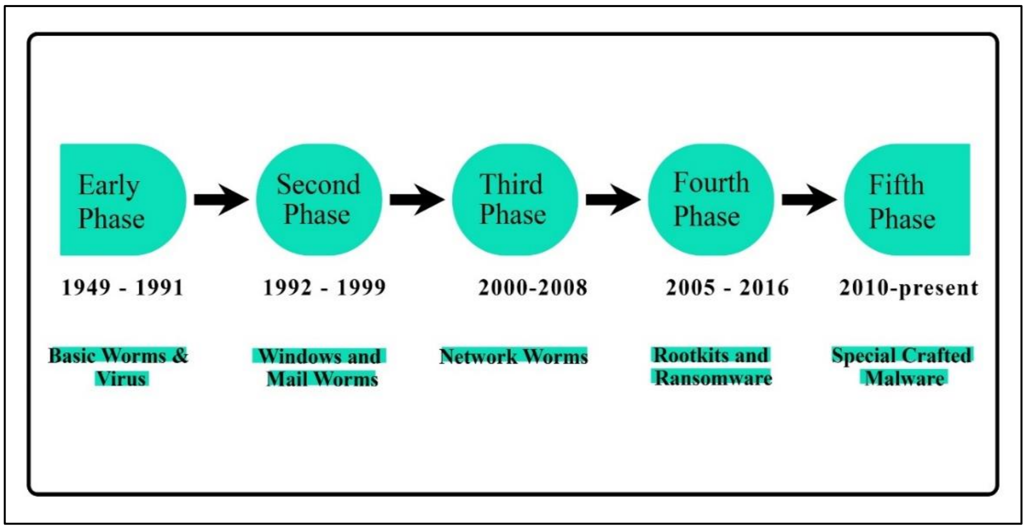


Figure 1.2: Timeline of The Five Phases of Malware Advancement [12].

1.4.1 Initial Development Stages of Malware (1949-1991)

Malware poses a substantial risk to the digital realm at present. The original intention of malware was not to cause harm, steal, or manipulate, but it has now become a threat to the security and safety of our society. In this section, we will examine some of the earliest forms of malware. In 1949, John Von Neumann coined the term "self-replicating string of code" to describe the first virus. He devised a "self-reproducing automata" that was capable of generating a new variant on its own [13].

It was utilized primarily to highlight vulnerabilities in MS-DOS systems. As a result of the utilization of system resources, the malware's payload in this phase was a transient system crash.

In this phase, viruses and worms have spread via infected floppy drives or the ARPANET. Fred Cohen formulated the generally idea of computer virus is defined as "a program that can infect other programs by modifying them to include a possibly evolved copy of itself" [14].

1.4.2 The Second Stage of Malware Creation (1992-1999)

As a result of Windows' user interface's simplicity and power, Hackers and perpetrators evolved into intrigued the operating system Windows. Windows PCs were the primary target of malware during this time period. The history of Windows malware, early email worms, and macro worms are all chronicled here. Anti-virus software was also created around this time. In 1992 [15], WinVir was released as the first Windows malware. It was malware, and it did very little damage to victim files as it copied itself [16]. Furthermore, in 1992, a virus known as V-sign attacked the boot sector, attempting to deactivate the machine by displaying a V-sign on the screen [17].

1.4.3 The Third Stage in The Development of Malware (2000-2008)

The broad use of the Internet marked the beginning of the third generation of malicious software. This is the early life cycle phase of several illnesses, including network worms. During this time period, malicious software was most often spread by phished email attachments, free malware downloads from malicious websites, and unsecured shared drives on a local area network. The Morris worm was able to spread widely in the early days of the

Internet because security was not a top focus. The goal of any Internet worm is to spread to every unprotected machine it can find on a network using a scanning algorithm. It starts by wreaking havoc on the system, and then it tries to expand outward from that point [18].

1.4.4 The Fourth Stage of Malware (2005-2016)

Rootkits and ransomware appear in the fourth stage. In the final stage, users are infected with malware using a variety of vectors include emails, remote desktop protocols, compromised website downloads, USB drives, and other external media. At this point in malware development, the major motivation was financial gain or unauthorized exploitation of infiltrated computers. A rootkit is a software that secretly takes over an operating system in order to steal information from the infected machine. Rootkits stand out from other malware because of their stealthy nature, whereas viruses and nematodes are classified according to their capacity to multiply. Rootkits seek to conceal the presence of an attacker on an infected system [19].

1.5 MALWARE SUBTYPES

Malware is an acronym for malicious software, which refers to any software that is intended to cause harm to the user. The issue with each definition of malware is that they are overly general and lack rigor. Malicious activities may include interfering with the operation of a computer, gathering sensitive data, or gaining illicit access to a computer system or network.

The classification of malware can consider the purpose or functionality of the malware. However, it may not be feasible to classify malware based on its functionality, as malware can possess a wide range of functions. Malware, for example, has the ability to mimic a worm by scanning the network and exploiting vulnerabilities. It can also obtain other forms of malware, like a backdoor [20]. The subsequent categories of malware are not incompatible, and their sole purpose is to acquaint the reader with the various malware categories.

a. Backdoor: It is categorized as a Trojan horse and grants the perpetrator system access and command execution.
b. Bot: Enables a hacker (known as the botmaster) to manage the compromised system remotely. A botnet is a linked network of bots that are remotely managed by a botmaster using control and command (C&C) software [21]. Common bad botnet behavior includes

Distributed Denial-of-Service (DDoS) assaults, spyware use, spam email distribution, and virus distribution.

c. Downloader The injector has been created to acquire and deploy more malware or malicious components.
d. Ransomware, also known as malware, is a type of malicious software that has the ability to freeze the screen of the targeted individual and encrypt various file formats commonly used, including but not limited to .xsl, .docx, .txt, .sql, .jpg, .cpp, and .mp3. Elliptic curve cryptography, RSA, and AES are the most used encryption transformations.The culprit requests payment (often in Bitcoin [22] in return for the decryption key.
e. Rootkit: It modifies the operating system in order for a rival to retain administrator access. Rootkits can be identified by their capacity to hide their existence or that of other malicious software [23].
f. Spyware: Retrieves and sends sensitive information via the victim's device to the offender. Examples include credit card numbers, passwords, a record of websites visited, emails, and various documents. such sensitive information , A key logger is an example of spyware because it records keystrokes and sends them to the perpetrator. A sniffer that monitors Internet traffic is a second example of spyware [24].
g. Trojan horse: It masquerades as benign software while performing malicious actions But it does the opposite [25].
h. Virus: is malicious software which is connected to a host. When a host that is infected its executes itself, the virus becomes active, performs malicious actions, and propagates to rest of computers. [26].
i. Worm: resembles a virus, However, its activation doesn't need an installation file or human intervention. A worm is capable of self-replication and spreading across other computers [27].

Rootkits and ransomware are favored by cybercriminals due to their efficacy for accomplishing their objectives (such as obtaining system access) and because, in the scenario of rootkits, they may conceal to prevent detection by employing hard-to-detect modification techniques. The majority of the time, cybercriminals install rootkits after gaining root or administrative privileges. Obtaining administrative or root access to a system requires a direct attack on the system.

Potentially Unwanted Programs (PUP) are a form It is a kind of software that exists among malicious and secure files. There may be uncertainty regarding the intent of PUP, or for certain users, the potential advantage might outweigh the potential harm.

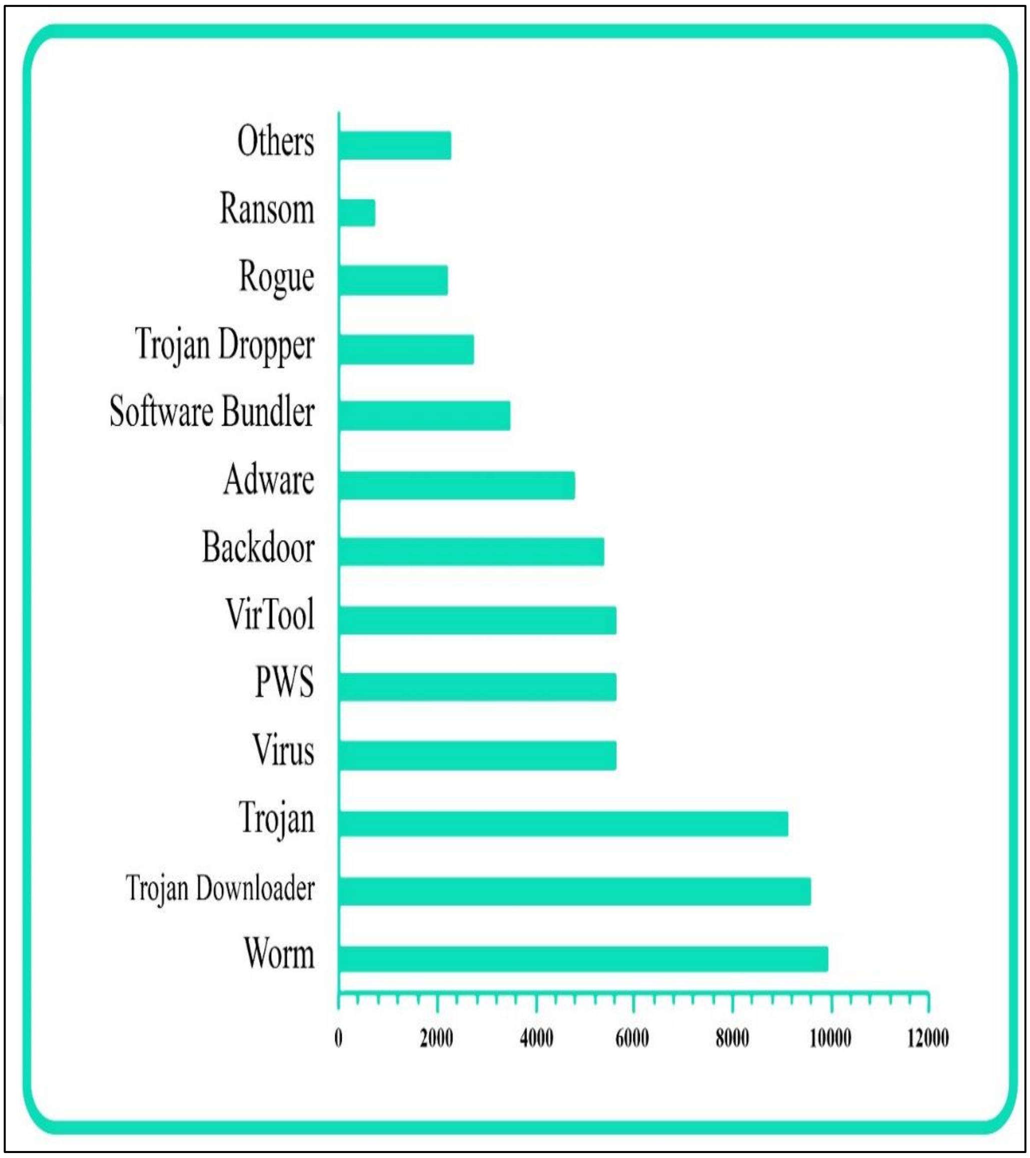


Figure 1.3: The Number of Malware Type Which it Detected by Microsoft AV [28].

1.6 CYBER ATTACKS CAUSED BY MALWARE

A cyber-attack is any act or attempt to obtain unauthorized access, whether successful or unsuccessful. Stuxnet, a worm with rootkit capabilities discovered for the first time in 2010 [29], is a well-known cyberattack induced by malware.

Similar to Stuxnet, numerous malware have and continue to target industrial control systems. Shamoon [30] and Dragonfly [31, 32] are examples. Alureon and Game Over Zeus are additional examples of such malware. Alureon, which is also referred to as TDL, is a trojan with rootkit capabilities that was initially detected in 2008. The purpose of this programme was to extract data by intercepting the network traffic of a system. Its main objective was to locate and retrieve sensitive user information, including banking login credentials, credit card details, and social security numbers.

1.7 MALWARE OBFUSCATION TECHNIQUES

Malware authors frequently modify malware in order To render detection harder. The methods mentioned are commonly known as encryption techniques, which are employed to hide malware from analysts specialising in malware and engineers specialising in reverse engineering. These methods effectively bypass detection systems that rely on signatures. Numerous layers, including code, an instruction sequence, and binary, are susceptible to obfuscation. In the following paragraphs, we will discuss some techniques which render malware detection more challenging. Most common methods include compression, encryption, polymorphism, and metamorphism.

The procedure of hiding a file that is executable using compression (one or more layers) is known as packing. This process generates a new executable file that may be used to store encoded data. To go back to the original file (or source code), a decompression process must be used.

Encryption: Is comparable to compression, but encryption is utilized in place of compression. Malware that is encrypted and compressed must include a Software for encoding that can be utilized for detection based on signatures.

Polymorphism: it uses encryption, and the decryptor mechanism is also mutated, which eliminates the need for a signature. However, the decryption of polymorphic malware is necessary, and the detection process can utilise the original (non-obfuscated) code through the implementation of signatures.

Metamorphism: Malware authors' most sophisticated and challenging obfuscation technique. While retaining its original functionality, malware with a metamorphic framework changes its internal framework.

Packaging as well as encryption are two widely used methods for avoiding detection based on signatures and static analysis. Malware that is polymorphic or metamorphic can alter its code after every iteration. When this type of malware runs, it may be re-obfuscated in order to evade detection based on signatures.

1.8 TRADITIONAL MALWARE AND NEW GENERATION

Throughout history, malware has been crafted with clear-cut intentions, which has facilitated its identification process. This particular form of malware is commonly referred to as traditional (fundamental) malware. Malware that operates in kernel mode and possesses greater destructive capabilities and detection evasion compared to conventional malware is The term "next-generation malware" is presently used to describe this type of malicious software. This malware has the capability to evade kernel-mode security tools like firewalls, antivirus programmes, and so on.Traditional malware generally consists of a solitary process and does not employ intricate methods to conceal its presence. The malware in its latest version, however, utilises various pre-existing or original methods. The malware operates simultaneously and employs methods of obfuscation to hide its presence and remain in the system. Never-before-seen destructive attacks, such as targeted and persistent, can be launched by malware of the new generation, and multiple types of malware are used in these attacks.
Table 1 displays a comparison between traditional and novel generations of malware.

Table 1.1: Traditional Versus New Generation Malware [33].

Comparison Parameters	Traditional	New Generation
Implementation Level	Simple Coded	Hard Coded
State of Behavior	Static	Dynamic
Proliferation	Each Copy is Similar	Each Copy Is Different
Through Spreading	Use .exe Extension	Use Also Different Extension
Permanence on the System	Temporal	Persistent
Interaction with Processes	A Few Process	Multiple Process
Use Concealment Technique	None	Yes
Attack Type	General	Targeted
Defensive Challenge	Easy	Difficult
Targeted Devices	General Computer	Many Different Devises

1.9 MLWARE DETECTION TECHNIQUE AND ALGORITHM

Datamining and machine learning (ML) In recent years, malware detection has made extensive use of algorithms. Malware detection is a method of examining the data of a software program in order to determine if it contains malware, to know if it is malicious or benign. Malware analysis, feature extraction, and classification are the three steps of the malware detection procedure.

a. Malware Analysis

Malware analysis techniques are primarily static and dynamic [34]. Malware is examined via static analysis without executing the actual code [35]. In contrast, The dynamic analysis of malware investigates its actions while its software is running. The examination of malware begins with fundamental static analysis and concludes with advanced dynamic analysis. Malware is analyzed via reverse engineering [36] and a number of other malware analysis tools that represent the malware in various formats. format. Reverse engineering process can be seen in Figure 4.

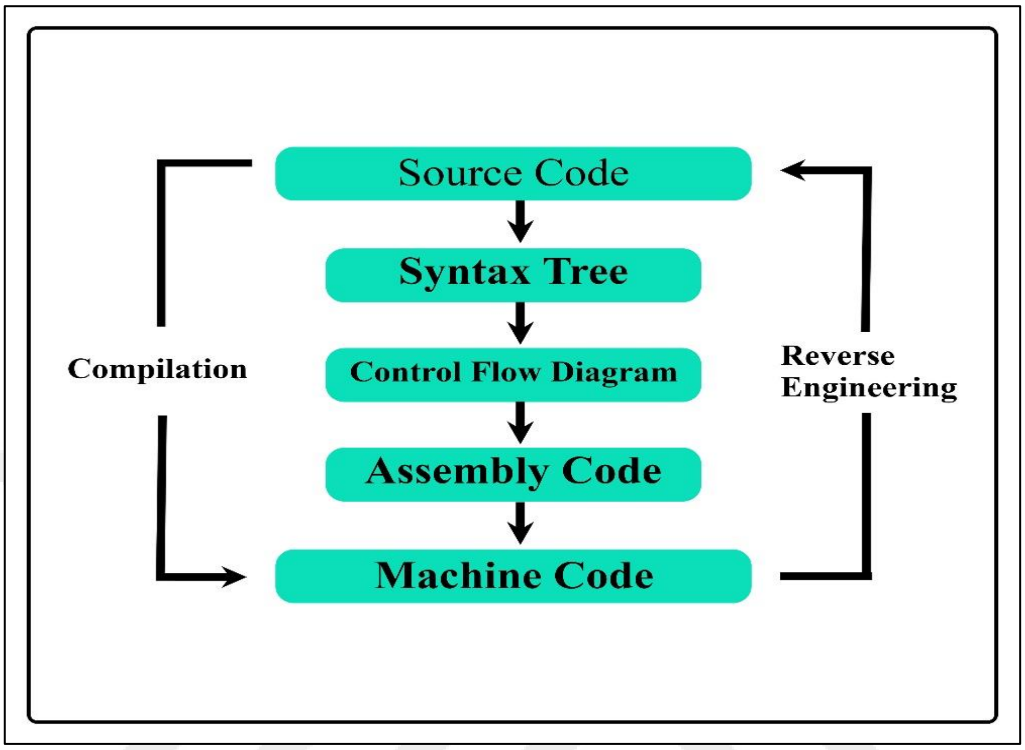


Figure 1.4: A Flow Chart of Reverse Engineering Process [37].

b. Malware Feature Extraction

The characteristics of malware are extracted using data mining techniques. Large datasets or databases are mined for previously obscure, novel, and significant information. Using datamining, novel models and datasets have been constructed in recent years,Various models, such as the n-gram and graph models, are utilized for the creation of malware datasets and features.

c. Malware Classification

Machine learning (ML) refers to a collection of algorithms that can reliably anticipate the results of applications without being specifically designed to do so. Using statistical analysis, the Machine Learning aims for transform the data supplied into valid value intervals. Machine learning (ML) may perform a number of tasks on the linked data, including classification, regression, and grouping. For a long time now, ML algorithms have been used effectively in the identification of malware [38]. The Bayesian network, naive Bayes, C4.5 variation decision tree, logistic model tree, random forest tree, k-nearest neighbor, multilayer perceptron, simple logistic regression, support vector machine, and sequential minimum optimisation methods are all examples of popular ML techniques. Particularly, behavior-based detection and other forms of detection make use of these techniques. However, each algorithm has its own strengths and weaknesses.

1.10 OUTLINE OF THESIS

The whole project is divided into the following five chapters:

Chapter One: provides A brief overview of malware before discussing the problem statement and objectives. The development of malware through the years, Types of Malware, malware-based cyberattacks This thesis employs Malware Obfuscation Techniques, traditional malware and new generation, Technique And Algorithm.

Chapter Two: Presented the theoretical portion and additional information regarding the thesis and related work.

Chapter Three: introduced the research methodology and procedures for obtaining the outcome results.

Chapter Four: This section contained the simulation configuration and a discussion of the project's results.

Chapter Five: Included completion and prospective work.

2. OVERVIEW

2.1 INTRODUCTION

This section provides a thorough overview of the thesis's context. Firstly, an overall description of malware is provided and malware detection, it will explain some comparisons between different kind of them. The final section will focus on the most possible future obstacles to malware detection. Finally, discuss significant work that is related.

2.2 DESCRIPTION OF MALWARE

Malware, short for "malicious software," is any program or file that is intentionally harmful to a computer, network, or server. Malware can be designed to cause damage to a stand-alone computer or a networked PC. Its goal is to compromise, destroy, or render useless a target system or device, including but not limited to personal computers, servers, networks, tablets, and mobile phones. Malicious software may be installed on a computer without the user's knowledge or permission and used to steal data, encrypt it, erase it, or change or hijack essential computer operations, or even spy on user behavior. Computer viruses, worms, Trojan horses, ransomware, spyware, adware, and phony security programs are only few examples of malware. When disseminating malware that infects devices and networks, cybercriminals use a wide range of techniques. Malware detection is crucial for maintaining computer security and protecting against potential threats [39].

2.3 TAXONOMY OF MALWARE

Malware can be classified based on various characteristics, but typically it is categorised according to its family type. The classification of malware files takes into account naming patterns [41] associated with them. Due to the rapid growth of malware, it made sense to revise the original terminology convention to reflect its rapid evolution.

The standard name an original bit of malware typically consists from four components, it shown in Figure (2.1).

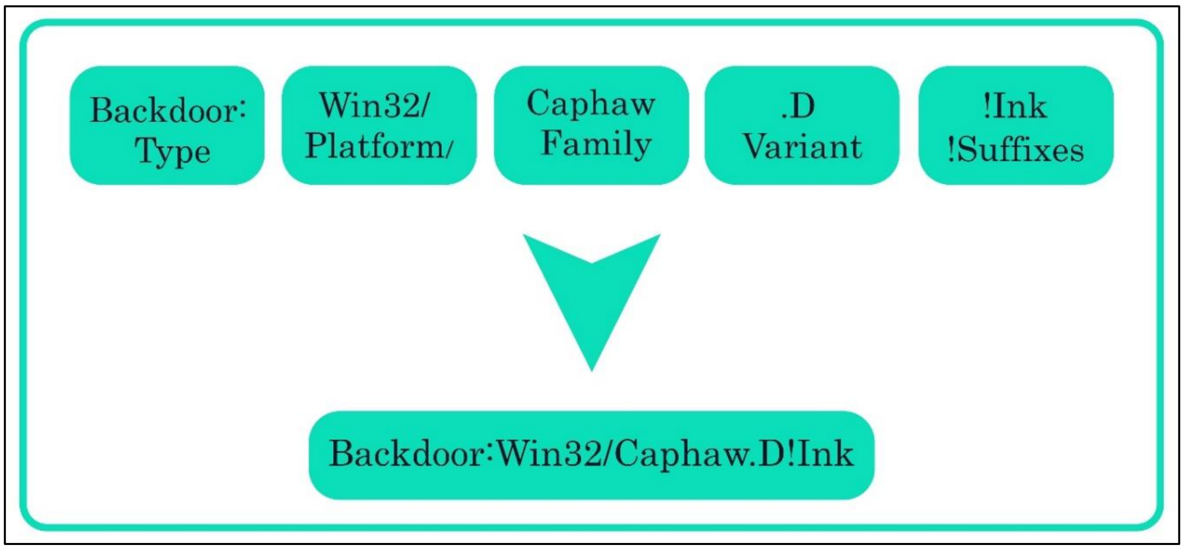


Figure 2.1: Naming Standard for Malware [40].

Type and Family are the relevant portions of the appellation, as they provide the broad distinctions between categories. Malware is classified by Type when the malware's behavior and effect on the victim system are taken into account.

Examples include:

a. Backdoor: installed on the victim's computer, granting remote administration to the perpetrator.
b. Botnet: similar to a (back door), however the victim device is part from the network of compromised computers connected to a (command-and-control server);
c. Downloader: precursor coding that downloads additional malicious code;
d. Information-stealing malware: coding that extracts information (such as financial credentials) from the computer of the victim and sends it to a designated source.
e. Launcher: utilized to launch other malicious programs via illegitimate means to preserve anonymity.
f. Rootkit: software that provides unauthorized use of a computer and attempts to conceal the presence of other malicious code.
g. Scareware: software that frequently mimics an anti-virus (AV) warning in order to frighten ignorant users into acquiring malicious software.
h. Spam-sending malware: leads the targeted person computer to unwittingly transmit spam.
i. Virus: self-replicating coding that may infect more computers but demands a host file.

j. Worm: Malware that spreads itself through protocols used by networks.
k. Trojan: Malware that is typically disguised as harmless software and cannot spread on itself, but grants entry to the victim's computer.
l. Ransomware: Malware that locks a victim's data and demands payment from the victim person to decrypt them.
m. Dropper: Engineered to evade anti-virus scanning and containing additional malware that is released from the initial code after installation. Malware frequently falls into categories or uses multiple mechanisms, making it hard to create a distinct taxonomy system [41] (McGraw and Morrison, 2000). Then, deferring to household the same as the singular nominal attribute may offer an adequate description for cited malware. For instance, well-known and highly developed cyberweapon (Stuxnet), that was utilized to harm and damage the Iranian nuclear program, encompasses various malware groups.

The initial attacking vector was a USB drive contaminated with the worm module as well as a.lnk file pointing towards the worm by itself. (Stuxnet) uses multiple weaknesses, One of the those enables USB devices to automatically execute , link files, The worm module includes the payload routines, and a rootkit module conceals The illegal actions as well as protocols away from the consumer [42]. this is a complex attack, it may be challenging to precisely define the malware employing conventional methods. However, the family of variable permits an informative refer to entire malware.

2.4 MALWARE RECOGNITION TECHNIQUES

The development of methods for recognizing malicious resembles an effort to get to the bottom against malware authors. When antivirus companies utilize new methods for detecting malware, malware authors develop new techniques to evade detection.

Figure 2.2 depicts a classification of prevalent malware methods for detection currently in utilize. The primary a wide range of investigation are described in the following section:

2.4.1 Signature-Based

By comparing the program's data to a library of known malicious code signatures, signature-based detection may identify malicious software [43].

It takes a lot of time and work to extract, store, and disseminate these sequences [45].

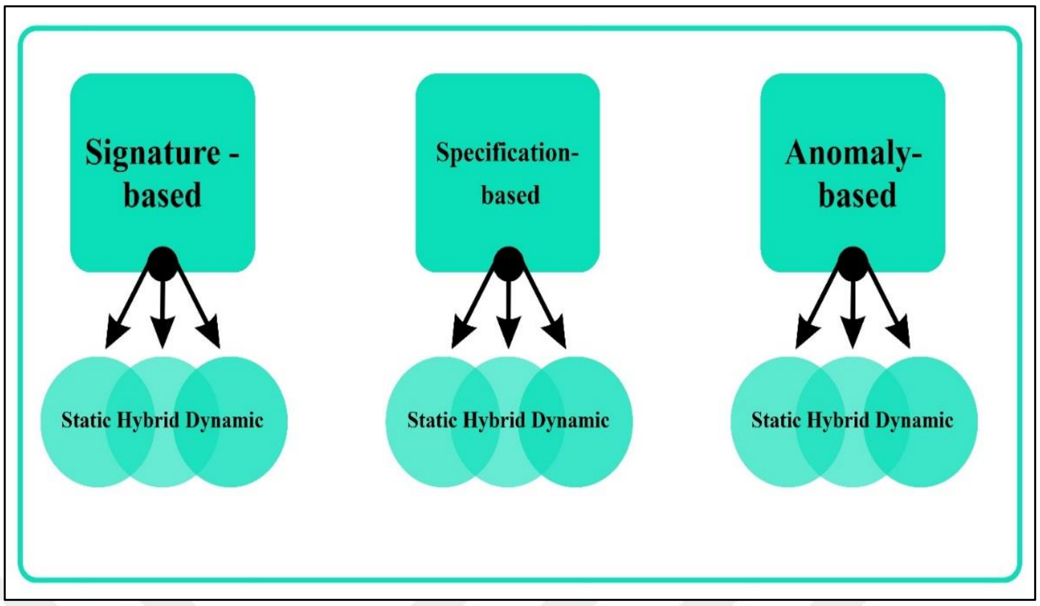


Figure 2.2: Malware Detection Methods [44].

The construction of signatures is "slow and prone to error," according to experts from Symantec, a leading anti-virus company [46].

In showing the results of their study on a computerized signature extract system, the researchers reported enhanced latencies of)1,278(min (>21 hours) for generating potential signatures using the data of 46,288 malevolent samples, and between 5 and 17 minutes for determining final signatures. The detection process is frequently carried out statically, which has the main drawbacks becoming ineffective toward unclear normal expressions as it is unable to find encrypted code to such sequences. it makes signature-based techniques less effective to the a wave of malware that is multiplying exponentially.

2.4.2 Anomaly-Based

Anomaly-based detection systems classify and form file or system behaviours, and deviations from these are labelled as anomalies [48].

The training phase generates a model that represents the typical behaviour or structure of the file or system. The monitoring stage then identifies deviations from the training starting points [49].

For instance, a tainted PDF's format may deviate significantly from that of a typical or anticipated PDF. In the same vein, a TV show file lasting one hour could be around 350 MB in size. If the file size is smaller than 120KB, it may be considered suspiciousAnomaly detection has the notable benefit of being able to detect zero-day threats. Due to its reliance on common characteristics, the system is capable of detecting unusual traits without the need for prior knowledge of said traits. However, anomaly-based systems have limitations because they often produce false-positive ratings and require a significant number of features for effective system modelling through inspection [49].

2.4.3 Specification-Based

Designed to address the issue of greater false-positive rates, the specification-based approach for detection is a close relative of anomaly detection. This detection technique focuses on the creation of the rule set that matches what the system needs instead of its execution [47].

The appropriate modeling of massive system is a challenging endeavor, and it is a result, specification-based detection may suffer from the same drawback as an anomaly-based system, namely that its model fails to accurately represent the behaviors of an intricate system.

2.4.4 Static-Analysis

Static analysis can be performed to the original code and the generated binary format [45].

The objective of the examination is to figure out the purpose of the program through examination of its structures data and code.

Digest approximations of the executable and like (MD5) and (SHA1), may be compared to set of data containing previous identified malware hashes. Call the graphs can illustrate the software's design and the potential transfers among functions. According to a recent study (Namanya et al., 2015), URLs, IP addresses, command line arguments, Windows PE files, and passwords are all fair game for string analysis [49].

The primary benefit when utilizing static analysis methods is the fact that there is no danger from the sample of malware because it is never executed; therefore, it can be securely analyzed in depth. In contrast to static analysis, which examines the whole source code, dynamic analysis just looks at the currently running code [50].

Testing all potential code paths may not always be advantageous since the disclosed details could consist of superfluous inactive code designed to conceal the actual purpose of the file. Against more sophisticated malware that may evade detection until the time it's activated, static analysis techniques are losing relevance. The true code is not always revealed [50].

by the hiding approach since the source code is never executed during static analysis.

2.4.5 Dynamic Analysis

In order to do a dynamic analysis, the file being looked at must first be implemented. To get over the limitations of static analysis introduced by different obfuscation schemes, approaches are used to extract information during memory access, runtime, and post-execution. Conditions that could arise include: a) instances of plagiarism The host's operating system is used for the analysis, therefore there is no need to switch between computers for the process. Malware analysis is complicated due of the malware's impact on the host computer. Deep Freeze [51] (Faronics Corporation, 2018) is a software that offers a snapshot of the original native environment, which can be recovered following any malware activation (Faronics Corporation, 2018). According to Light et al. (2010), the process of restoring everything in the system to its original state can be quite time-consuming [52].

Another method is emulation, where the host uses software to imitate hardware and manage the visitor environment. The software's dependence on the host architecture might cause simulating to run slowly or not at all. c) simulated by a computer program; a virtual machine provides a separate, host-controlled environment for its processes to run in. In a perfect world, the heart of the situation would be exposed for everyone to see. Malware authors have responded by developing a wide variety of approaches to avoid detection, such as those that are anti-debugging, anti-instrumentation, and anti-Virtual Machine [53] (Bulazel and Yener, 2017). One major drawback of dynamic techniques is the temporal complexity of execution, which arises from the need to execute the program for a certain amount of time [50].

2.4.6 Hybrid-Analysis

Hybrid methods' recognition algorithms combine static and dynamic analysis. Roundy and Miller's (2010) research used processing approaches to statically examine potentially harmful code before to execution [54].

They focused on analysing the code's framework and used dynamic processing for disguised code. The algorithm developed by the authors utilises a combination of dynamic and static techniques to analyse code, even when it is obfuscated.

2.5 RELATED RESEARCH

The present danger landscape caused by malware's rapid development. Although methods of detecting malware have been developed for many years, the increasing rates of infection suggest that the anti-malware environment is failing under the strain of swiftly evolving malware. As reported by Symantec (2018), infections with ransomware have steadily risen each year since 2015, reaching an all-time high of 1,271 per day in 2018. The following part examines the academic study conducted in the area so far, which has tried to offer solutions for the failings of present detection methods, and identifies the remaining obstacles.

2.5.1 Malware Detection by Using Machine Learning

Obfuscation techniques have posed significant challenges for identifying malware, and additionally from the point of view of consumer AV products. In the past, the evasion techniques employed by malware authors have hindered efforts to increase malware detection numbers. Throughout history, numerous scholars have explored the utilisation of data mining and machine learning techniques for the detection of malicious software in undisclosed datasets. [55]According to Schultz et al. (2001), the concept of utilising machine learning for analysing malware binary files was initially proposed by them. In this study, the findings of three machine learning classifiers (RIPPER, Nave Bayes, and Multi-Nave Bayes) were compared to a signature-based classification method known as AV scanner. The investigation focused on examining programme headers, string features, and byte sequence features. Typical AV scanner effectiveness was inferior to that of machine learning gets closer, with a pair of classifiers and characteristics tripling the device's accuracy. In their study, Kolter and Maloof (2004) [56] employed n-gram analysis on the hexadecimal representation of executable malware files. Instead of utilising count or frequency, they opted for the Boolean characteristic of present or not present. Enhanced decision trees outperform other classifiers with an Area Under the ROC Curve of 0.996%. In a study conducted by Moskovitch et al. (2008) [57], the researchers examined the detection of imperceptible worms through the analysis of the computer's operating system's arrangement, background activity, and user-facing characteristics. The information gathered resulted in observations for 323 features per second, compared 5 worm types and a typical behavioral structure. With 20 characteristics, the average detection accuracy was greater than 90%, with detection of specific nematodes reaching 99%.

2.5.2 Opcode-Analysis

The most recent study on novel methods to detecting malware has centered on the execution behaviors of malware, i.e. what the software performs, instead of how it performs it, and the way this varies from the behavior of benign code. Operating codes (opcodes) are machine-readable instructions used for database abuse, logical operations, and program flow control. O'Kane et al. (2014) [58] By analyzing the host environment's native opcodes at execution, it is possible to identify the existence of malware while avoiding encryption (O'Kane et al., 2013) [59]. Santos et al. (2011) used n-gram models of opcodes in statically created datasets. Even though their study only looked at pairs of twos (n = 2), the researchers were able to get accuracy and F-measure rates higher than 85% by using ROC-SVM. Anderson et al. (2011) [60] used chains of Markov models to create dynamically created run-time traces that were shown as weighted directed graphs. These graphs gave information about how likely it was that one opcode was being watched by another. The main purpose of the writers' study was to show that their method was better than n-gram models and signature-based recognition techniques. The data set had 1615 samples of malware and 615 examples of harmless programmes. However, the provided analyses lack information about the malware type, family, age, obfuscation, or size. This limitation greatly hinders the interpretability of the findings. The performance of nine AV scanners was compared to the n-gram feature selection method, which selects the topmost n-grams based on Information Gain ranking, as described by Kolter and Maloof (2004) [56].

With a best accuracy of 73.32 percent, the AV algorithm performed worst of the three models tested. A total of 595 false negatives were recorded, while there were no false positives. With

300 FPs and 98 FNs, the best n-gram approach achieved an accuracy of 82.15%. The results showed that the Markov model was the most effective, with a success rate of 96.41%. Additionally, it recorded 47 false positives (FPs) and 33 false negatives (FNs). The authors note that the FN ranking may potentially be a result of sampling bias, as the dataset is skewed towards certain preferences. In their study, Runwal et al. (2012) [61] utilized graph techniques to analyze PE file operation codes, drawing inspiration from the previous work of Anderson et al. (2011) in the field of metamorphic malware. By leveraging similarity scores, the model successfully differentiated metamorphic malware from benignware, along with other similar families within the same category. According to O'Kane et al. (2013, 2014) [58] [59] , they have devised a technique to examine the utilization of opcodes in malware detection using supervised machine learning approaches. By employing a hypervisor, specifically a virtual machine, the file under investigation was executed, allowing for the collection of run-time traces for both benign and malicious code files. By incorporating an operational layer onto the host machine's OS, this technique of virtualization facilitated the execution of unauthorized programs in a sanctioned environment, as observed by the researchers. The separation allowed for the successful implementation of the malware, ensuring efficient data acquisition while also isolating it from any potential impact the malware may have had on the hosting system. The execution trace of each investigated program was obtained using a debug program called Ollydbg and a masking utility known as StrongOD. The provided data includes a comprehensive record of each opcode and its corresponding operand, along with supplementary details such as the memory register's address. By employing a bespoke parser, the opcodes within this trace file were extracted, and the frequencies of their occurrences were calculated. To avoid introducing impact interactions caused by different run lengths, the density of each opcode was calculated based on its frequency within the sample. The features most likely to provide the maximum amount of information to the Support Vector Machine (SVM) were selected by applying a pre-filter. This pre-filter helps reduce the total size problem caused by using the technique of n-gram analysis to examine all possible opcode configurations. Every opcode's relevance to the categorization through SVM assignment was ranked using Principal Component Analysis (PCA). In order to create a selection of crucial components, principal component analysis (PCA) reduces the data size while keeping the variety in the data. According to the researchers, it was found that the top 8 opcodes accounted for 99.5% of the overall variation

in the data. This resulted in a reduction of information from the initial 150 opcodes. The effectiveness of this reduction was verified through the use of an SVM.

2.5.3 Existing Dataset in the Context of Other Datasets

O'Kane et al. (2013) [59] provided the present dataset, which includes both benign and malicious PE samples (300 and 350, respectively). Despite the higher size of the dataset compared to earlier studies like Bilar's (2007) [62], it falls short in terms of scale when compared to similar studies. In a groundbreaking study conducted by Schultz et al. (2001), a total of 3265 malicious files and 1001 benign files were carefully chosen for analysis. In their study, Santos et al. (2011) [63] employed a dataset comprising 2,000 files, which were evenly distributed among the groups. Moskovitch et al. (2008b) and Shabtai et al. (2012) [64], who claim to have analysed the biggest dataset available at the time, report comparing 7688 malicious files with 22735 benign files. All sorts of comparisons with a more comprehensive collection of variables are made possible by the aforementioned databases. However, it is important to note that there are limitations associated with the process of collecting the data. The sample data (Shabtai et al., 2012) [64] was disassembled using IDAPro, a renowned disassembler in the industry. According to the authors, the software was able to actively disassemble only 74% of the original files. According to the data provided, the attrition rate for malicious software in 2011 was approximately 26% based on a sample of files. Similarly, the attrition rate for benign software was approximately 10% based on a larger sample of 2319 files. The remaining 26% of files have not been examined, except for the assertion that the excluded documents were either compressed or bundled. However, it is important to note that the authors do acknowledge the potential application of extraction software in malware analysis. However, they fail to provide any justification for why they did not utilize the readily available program mentioned in their study on the complete dataset.

The dataset used by Santos et al. (2011) [63] comprised of 2000 files due to technical limitations that were not clearly specified. The harmful files were chosen arbitrarily from a collection of 170,000 sample files kept on the VXHeaven website.

Significantly, the use of compressed files was not employed during the static analysis. The study's important drawback is that it failed to thoroughly investigate disguised malware, resulting in an unsuccessful process. More than 50% of the sample set included three types of malware: backdoors, hacking tools, and email worms. These findings suggest the possibility of within-class inequalities, as elaborated further in the subsequent explanation. In addition, the malware sampled does not adequately reflect the whole scope of malware since it does not include all known variants. Table 2.1 displays the comparison of file sizes across different categories. The average file size for all categories was similar, with 299KB compared to 222KB. However, there were variations in the distribution of files within each size category. The research has a notable drawback, as it compared files using PE file size instead of considering the number of opcodes in a specific application segment or a fixed run-time. The quantity of information, specifically the number of opcodes, does not have any impact. Consequently, the data may vary depending on the number of opcodes present in each class.

Table 2.1: Distribution of File Types in Santos et al (2011) [63].

	Malware	**Benignware**
<100KB	43.8%	69.6%
100-1000KB	49.6%	25.4%
>1000KB	6.6%	5.0%

The input is being passed to the classifier. In their study, Kang et al. (2014) classified a total of 6721 malware samples [65] and obtained from VxHeaven into three distinct categories: backdoor, trojan, and worm. The researchers identified a comprehensive range of 26 malware types, each with over 100 variations within their respective families. There were no measures implemented to guarantee that sampling across categories was aligned, leading to the discovery of 497 worm variants, 3048 backdoor variants, and 3176 trojan variants. The analysis was conducted dynamically with the utilisation of VMWare and the Pin debugger, leading to the generation of execution traces which were later examined. The overall implementation trace for the entire data set was 201GB, with the worm category accounting for only 14GB, or 7%, of the trace, which is indicative of the impact of unequal sampling. The consistency of the number of opcodes transmitted to the machine learning models may be affected, which can impact the results of models that rely on traces and are dependent on different types of malware. According to the authors, there was a set maximum of one million orders, with no specified minimum requirement.

The dataset created by O'Kane et al. (2013) [59] has the benefit of being dynamically generated through the use of the original PE source, leading to a statically analysed raw file. The running durations have been adjusted, thereby controlling the quantity of opcodes allocated to the machine learning component.

In their study, Nappa et al. (2013) [66] investigated the identification of attack servers managed by the identical malicious individuals.Over a period of eleven months, 500 attack servers in the open were tracked by the researchers. A total of 11,688 malicious binaries were compiled from the collated artifacts. The authors describe a technique called "milking" where unpatched VMs, known as honeyclients, were instructed to visit a list of malware-serving URLs that had been identified beforehand. The purpose was to gather data on the resulting malware infections. The dataset was an illustration of a practical attack vector and was collected in the field. The specimens consist primarily of members of a small number of Trojan family groups.

3. METHODOLGY

3.1 OVERVIEW

This chapter will present the methodology used for detecting malware through static analysis of private data obtained from smart home IoT devices. The data will be classified into two categories: benign and malware. Additionally, the chapter will discuss the process of data collection and aggregation, the dataset utilised, and the experimental setup employed. Please elucidate the various stages of the framework. Subsequently, we showcased the performance metrics that assess favourable outcomes such as throughput, prison, f1-score, and recall. In conclusion, we now offer the synopsis of this chapter.

3.2 INTRODUCTION

We have assembled a diverse assortment of three distinct smart home IoT devices, encompassing both unadulterated and compromised units. The dataset utilised is conn.log. labelled, which corresponds to the Zeek conn.log file derived from the Zeek network analyzer utilising the original pcap file. After that, the system continuously monitors and records their activities, sending the data to a database (Dataset). This dataset is then utilised as the input for machine learning and deep learning classification procedures. Once the dataset has been gathered, the classifier is trained using it. Subsequently, a testing sample is created and outcomes are obtained. AI is employed in this domain to determine whether the samples are normal (benign) or infected (malware).

3.3 PROPOSED MODEL

The primary goal of this approach is to identify the malware employed in this research project. The objective of this thesis was to assess the effectiveness of machine learning and deep learning in relation to three distinct smart home IoT devices. It employed a range of algorithms in various scenarios. The proposed model is explained in Figure 3.1.

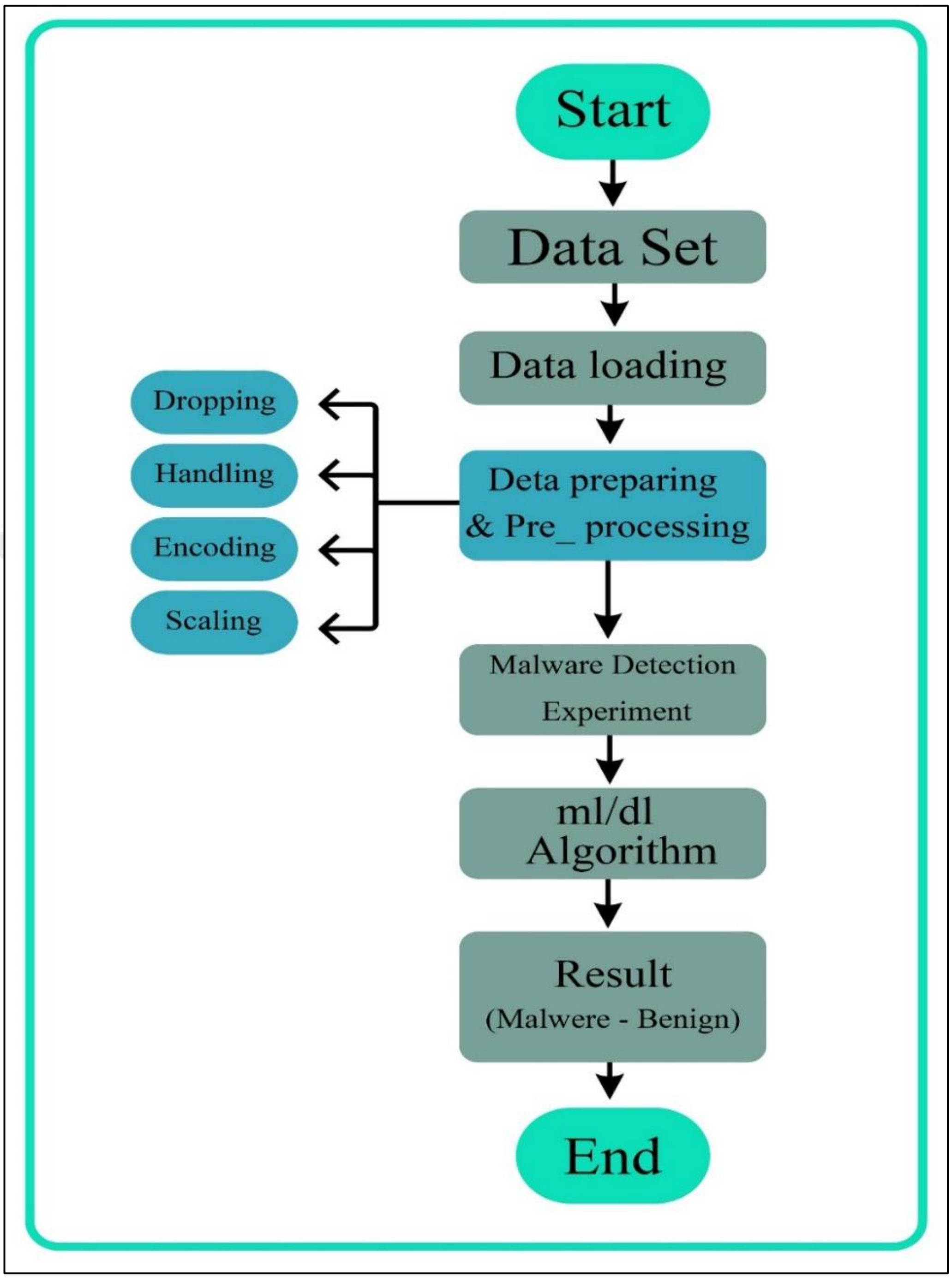

Start
Data Set
Data loading
Dropping
Handling
Encoding
Scaling
Deta preparing & Pre_ processing
Malware Detection Experiment
ml/dl Algorithm
Result (Malwere - Benign)
End

Figure 3.1: Methodology of The Proposed Model [67].

3.4 DATABASE (DATASET)

In January 2020, a dataset consist from 23 captures was released. This dataset contains network traffic data from three distinct smart home IoT devices. The dataset contains a total of 23 captures, with 20 of them being identified as malicious and 3 as benign. The benign captures account for approximately 13.05% of the dataset, while the remaining 86.95% are classified as malware.The dataset contains malware labels such as Attack like (C&C, C&C-FileDownload, C&C-HeartBeat, C&C-HeartBeatAttack, C&C-HeartBeat-FileDownload, C&C-Mirai, C&CTorii, DDoS, FileDownload, Okiru, Okiru-Attack, and Part Of A Horizontal Port Scan.The dataset utilised is conn.log.labelled, which corresponds to the Zeek conn.log file derived from the Zeek network analyzer utilising the original pcap file.

3.5 DATA LOADING

Firstly, loading all 23 datasets from captures separately, where each capture in the format of conn.log.labeled represents a one dataset , Each capture (dataset) has been loaded into separated data frame, where each one contains 20 features in addition to the label (ts, uid, id.orig_h, id.orig_p, id.resp_h, id.resp_p, proto, service, duration, orig_bytes, resp_bytes, conn_state, local_orig, local_resp, missed_bytes, history,orig_pkts,orig_ip_bytes, resp_pkts, resp_ip_bytes, label).

Then we combined all 23 dataset into a one dataset (data frame), where the final dataset consists of 1048575 samples with 20 features in addition to the label, These samples are combination of malicious and benign samples, as shown in the following Figure (3.2):

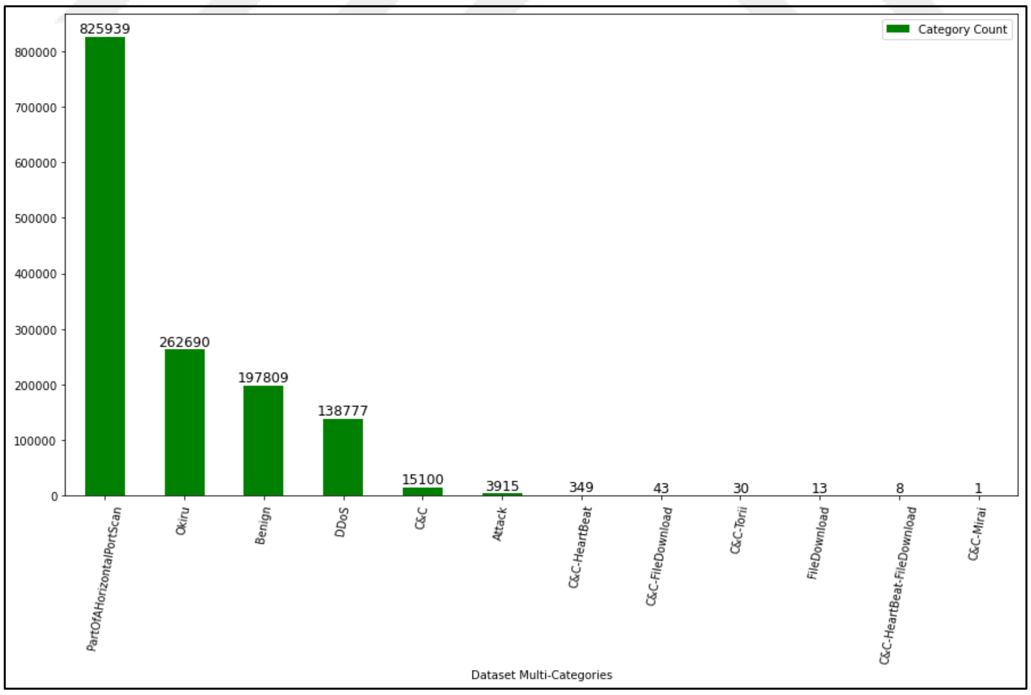


Figure 3.2: Malicious and Benign Samples [68].

3.6 DATA PREPARING AND PREPROCESSING

We combined all malicious categories into one category (Malware), while the other category is the benign samples, as table (3.1) below show the data type and number of it:

Table 3.1: The Data Type and Number [69].

Data Type	Number of Samples
Malware	849498
Benign	199077

The distribution of samples after combination illustrated, To make Preparing and Preprocessing: in the following Figure (3.3):

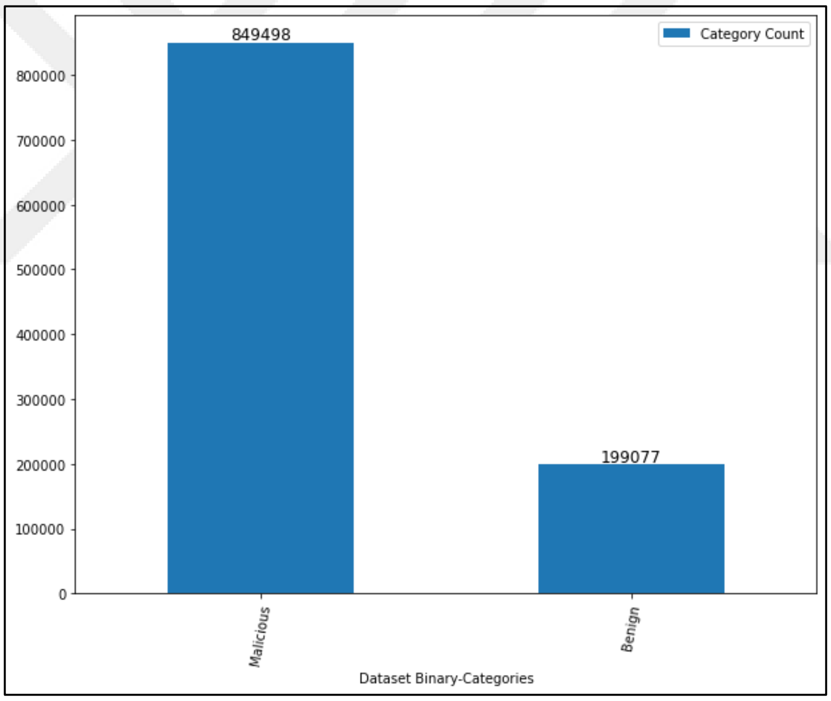


Figure 3.3: Data Binary-Categories [70].

a. Dropping Unnecessary features: it's essential to understand how to deal with highly correlated features. Highly correlated features refer to variables that have a strong linear relationship with each other. When two or more variables are highly correlated, they carry almost the same information like :
('ts','uid','id.orig_p','id.resp_h','id.resp_p','service','local_orig','local_resp','history')

b. Handling corrupted features: there are some features contains many (-) values that must be handled, we replace all these values by 0.

c. Features Encoding: we have applied the Encoding technique to encode some categorical feature, which means convert categorical features into neumeric. These features that have been encoded are: (proto, conn_state, id.orig_h).

d. Features Scaling: We have performed the Standard Scaler to perform features scaling, which means standardize features by removing the mean and scaling to unit variance. The final dataset after preprocessing consists of 1048575 samples with 12 features in addition to the binary class label in two categories (Malware and Benign).

3.7 EXPERIMENTS SETUP

install the Windows 10 operating system on a dedicated laptop with CPU Intel Core i7 6820HQ CPU 64-bit operating system, 2.70GHz , RAM 32.0 GB, system make changes and modification , finally building our model testing it and get result on jupyter notebook 2022 python 3.10.6. program.

3.8 PERFORMANCE METRICS

Classifiers have numerous metrics for performance. There is an overview in Refs. [67] as well as [68], and additional metrics are proposed in Refs. [69] and [70]. This part presents several performance measures commonly employed in the field of Internet of Things malware detection. As an ordinary binary classification problem, the outcomes of predicting whether an application includes malware may be categorized into four categories, as a confusion matrix demonstrates. [71][72] in Table (3.2):

Table 3.2: Confusion Matrix Demonstrates [72].

	Actually Positive	**Actually Negative**
Predicted positive	TP	FP
Predicted negative	FN	TN

Following are definitions of the terms FP, FN, TP, and TN:

a. True positive (TP): The program is malicious and its malicious nature was rightly foretold.
b. False positive (FP): The program isn't malicious, despite erroneous predictions to the contrary.
c. True negative (TN): The app isn't malicious and its non-malicious nature was rightly predicted.
d. False negative (FN): The app was incorrectly forecasted as non-malicious despite being malicious.

The sum of the above four incompatible results equals the total amount of samples tested. On the basis of each of these fundamental concepts, a number performance metrics have been developed.

The following metrics are widely employed:

a. Accuracy (Acc): indicates the proportion of successful estimates relative to the overall amount of test samples.

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FM} \tag{3.1}$$

Equation (3.1) known as accuracy equation.

b. Precision (P): indicates the proportion of samples that are positive reliably anticipated among every positive sample expected.

$$Precision = \frac{TP}{TP+FP} \tag{3.2}$$

Equation (3.2) known as precision equation.

c. Recall (R): indicates the ratio of accurately anticipated samples that are positive to the total samples that are positive.

$$Recall = \frac{TP}{TP+FN} \tag{3.3}$$

Equation (3.3) known as recall equation.

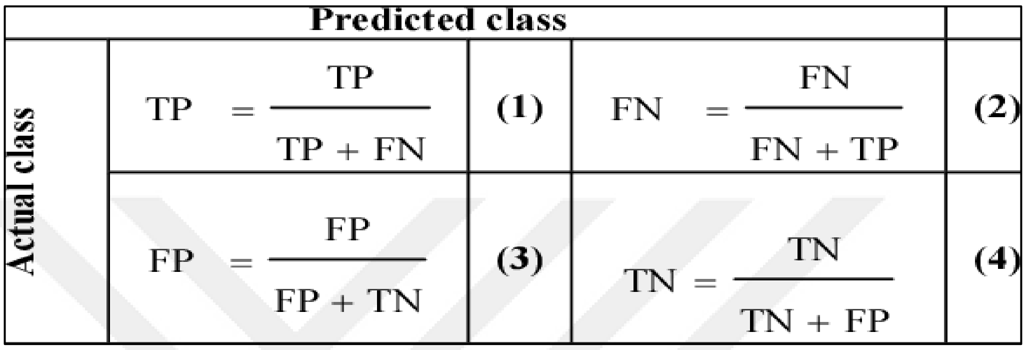
Precision and recall are crucial indicators of performance, however they offer a partial assessment.

Precision Equation

For an additional assessment of the classifier's performance, the harmonic average of precision and recall, also known as the F1 score, can be utilized for combining both of these numbers.

$$F1\ Score = \frac{2}{\frac{1}{Precision}+\frac{1}{Recall}} \tag{3.4}$$

Equation (3.4) known as F1-score.



	Predicted class			
Actual class	$TP = \frac{TP}{TP + FN}$	(1)	$FN = \frac{FN}{FN + TP}$	(2)
	$FP = \frac{FP}{FP + TN}$	(3)	$TN = \frac{TN}{TN + FP}$	(4)

Figure 3.4: Confusion Matrix Measurement [72].

3.9 MACHINE AND DEEP LEARNING ALGORITHMS FOR MALWARE DETECTION

3.9.1 Machine Learning Algorithm

a. Random Forest (RF) : Random Forest is an algorithm for supervised Machine Learning developed by Leo Breiman in 1997. The above method makes use of a collection of classification trees [72].

The group learning method creates multiple learners and combines their outcomes into a single set. Random Forest employs a modification of the Bagging [73] technique. In Bagging, each classifier is constructed independently using a bootstrap collection of the input data. At a node splitting in a conventional decision tree classification algorithm, a determination is made depending on all feature attributes. In contrast, Random Forest determines the optimal parameter at every node of a decision tree by picking a number of features. This random choice of features enables Random Forest methods to not only scale well when there are numerous features per characteristic vector, but also reduces the degree of interdependence (correlation) among the feature attributes. Thus, this technique is less susceptible to innate data disturbance.

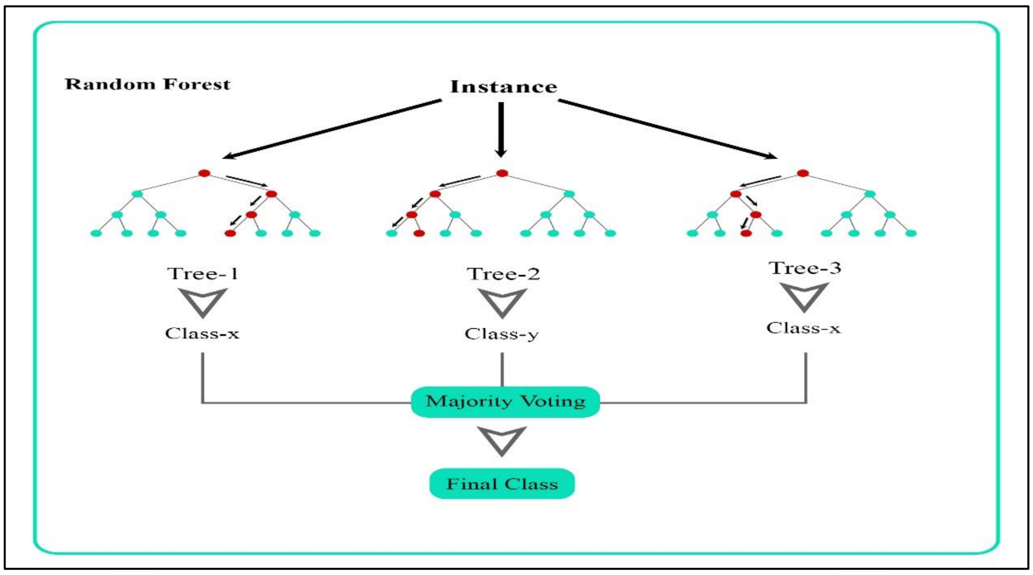


Figure 3.5: Random Forest Classification Process Based [73].

b. Decision Trees (DT): Using data items stored in a structure resembling a tree, a decision tree is used to make choices. As in a mode of cognition and processing analogous to that utilized by humans when confronted with decisions that are difficult, its fundamental process adheres to the straightforward and obvious "divide and conquer" approach [74].

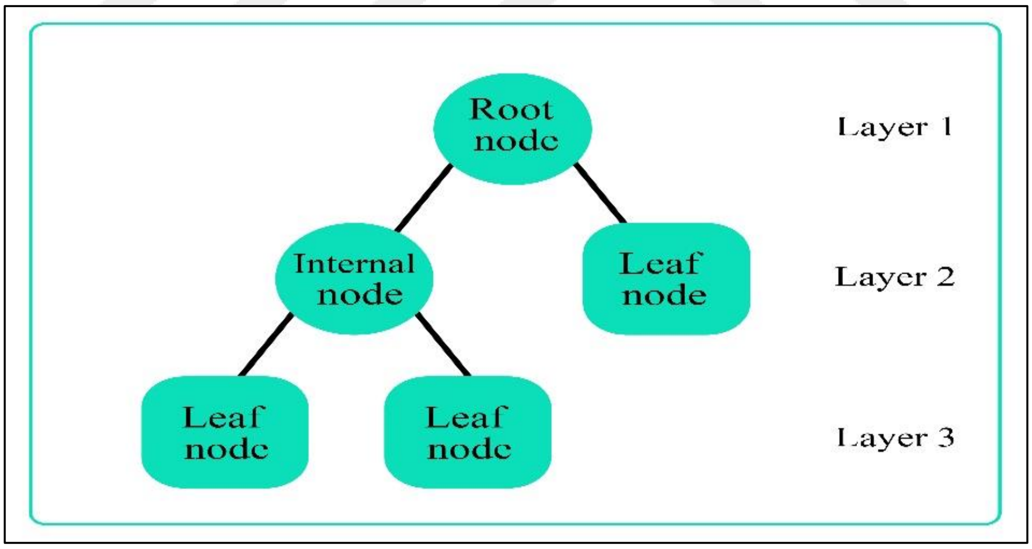


Figure 3.6: Decision Trees Classification Process Based [74].

c. Adaboost: is a 1995 machine learning meta-algorithm developed by Yoav Freund and Robert Schapire, it is a method of ensemble learning which combines numerous weak classifiers to produce a strong classifier. The algorithm operates by training weak classifiers repeatedly on various subsets of the training data and giving weights to every classifier according to its performance. The final classifier is the weighted average of each of the classifiers [75].

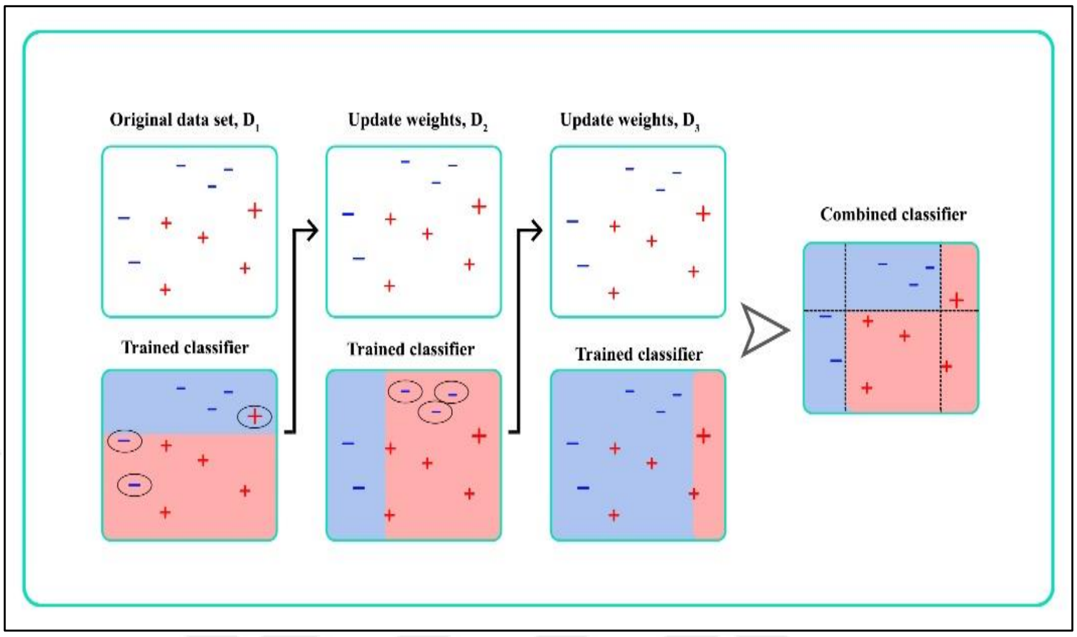


Figure 3.7: Adaboost Classification Process Based [75].

d. XG Boost: (Extreme Gradient Boosting) is a machine learning algorithm developed to enhance the efficacy and efficiency of gradient boosting decision trees. It is a kind of software library created to enhance the effectiveness and efficacy of machine learning designs, it is a form of ensemble learning that utilizes the predictions from various weak models to produce a stronger prediction, and it is a method of learning that combined the forecasts of numerous weak models to produce a stronger forecast. The algorithm is extremely flexible and permits the optimization of numerous model parameters [76].

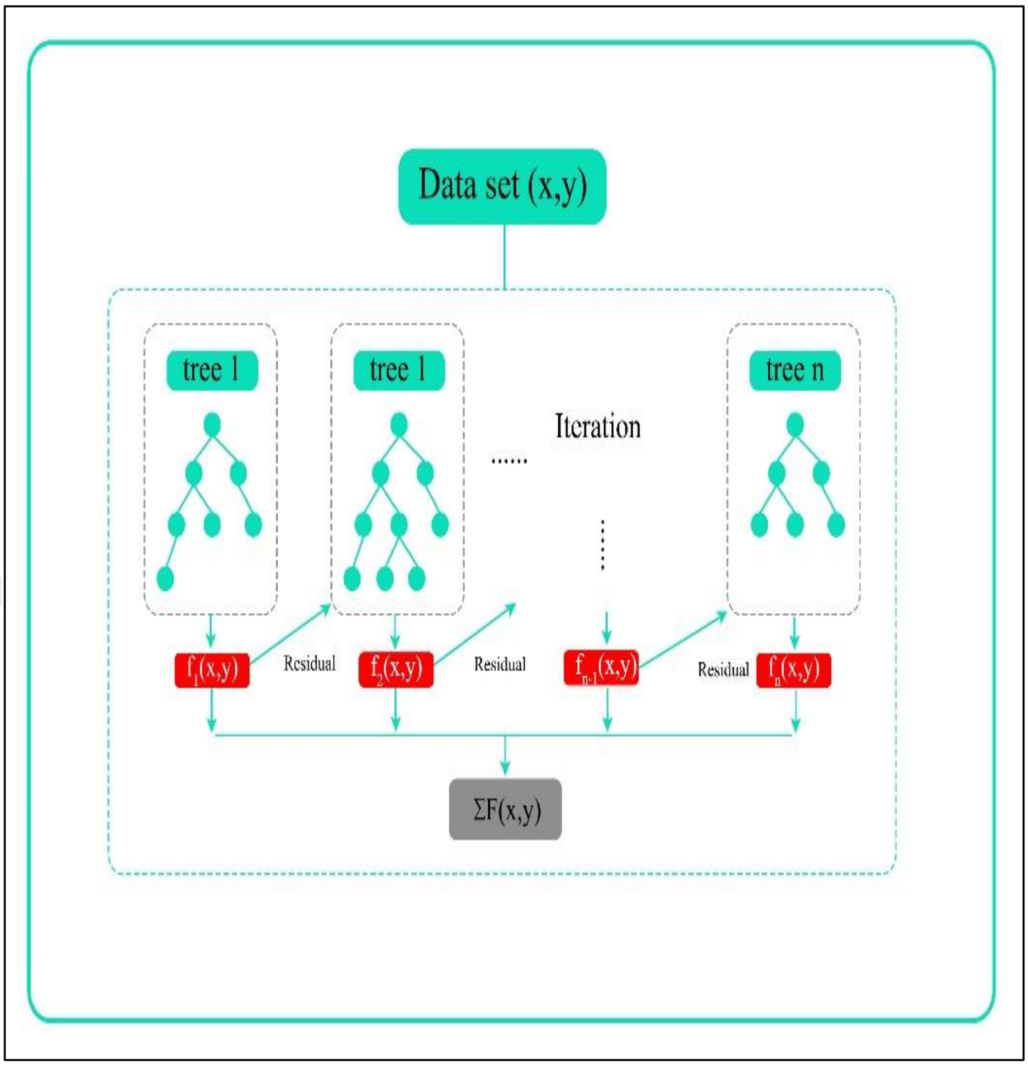


Figure 3.8: XG Boost Classification Process Based [76].

e. Gradient Boosting: Gradient Boosting is an algorithm for machine learning applied to construct models that are predictive. It is a method that integrates multiple poor learners into a single strong learner. The algorithm iteratively trains insufficient learners on gradient-based functions and integrates them into the model as "boosted" participants. The learners who are weak are typically decision trees. Gradient boosting is a potent method for developing predictive models that can be applied to tasks involving regression as well as classification. Gradient tree boosting is additionally known as gradient descent boosting and gradient boosting machines. (GBM) [77].

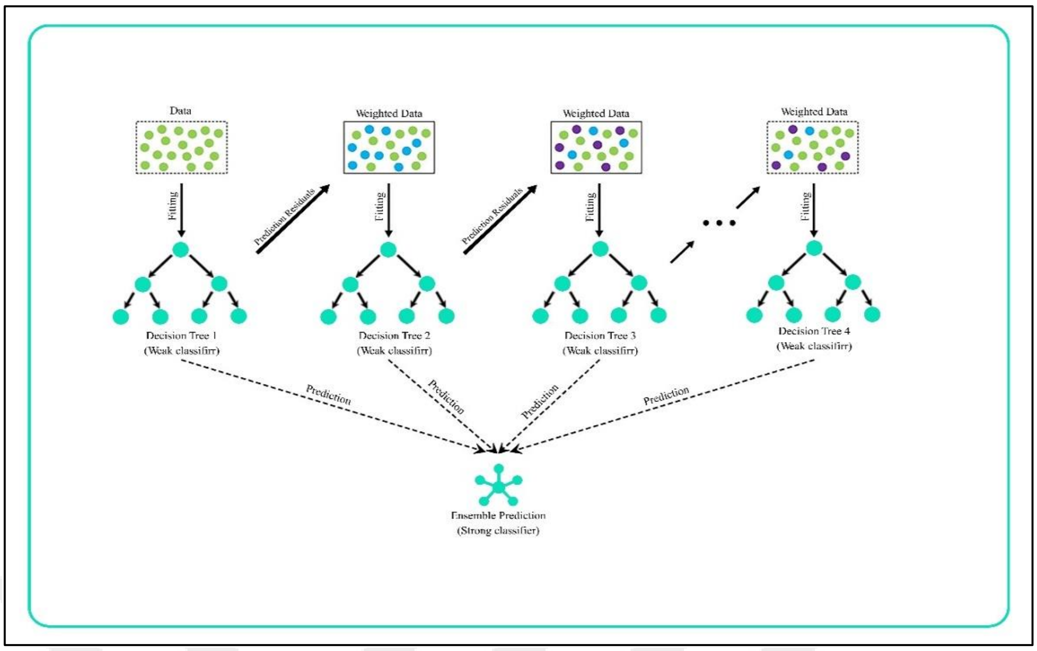


Figure 3.9: The architecture of Gradient Boosting [78].

f. Hard Voting Classifier: The combination standards used to classify labels include the simple majority vote, where the prediction that appears most frequently in the base classifiers is selected. This method is commonly used in bagging. In the context of binary scenarios, the need for a majority vote system aligns with the classical Condorcet criterion. This criterion states that for a class to be declared the winner, it must consistently outperform each of the other classes in individual evaluations or one-on-one matches. In other words, it must be preferred over the other class when assessed individually [79].

g. Soft Voting Classifier: The combination standards were derived by polling the constant outputs of each base classifier using a formula (average, maximum, minimum, product [79].

This formula selects the class label that maximises the numerical value of the utilised function for forecast probabilities. From a predictive power standpoint, the average is considered the most effective. The common combiner is a direct competitor to the prevailing preference for packaging.

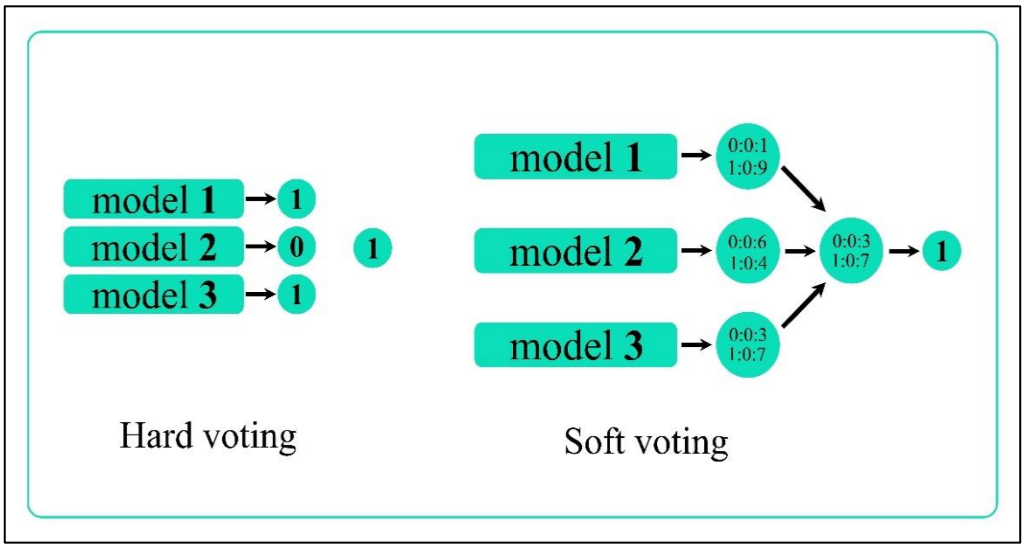


Figure 3.10: How the Hard and Soft Voting Work [79].

3.9.2 Deep Learning Algorithm

a. Fully Connected Neural Network (NN):

Its known as a dense neural network, is a type of artificial neural network where all the nodes or neurons in one layer are connected to the neurons in the next layer.

It is a feedforward neural network that is used for supervised learning. The fully connected neural network consists of a series of fully connected layers, where each output dimension depends on each input dimension. The mathematical form of a fully connected network is represented by the input to a fully connected layer, and the output from the fully connected layer is computed using the back-propagation algorithm for computing the error. The Hard Voting Classifier algorithm is an ensemble machine learning algorithm that combines the predictions of multiple classifiers, including fully connected neural networks, to make a final prediction. The Hard Voting Classifier works by taking the majority vote of the predictions made by each of the individual models [80].

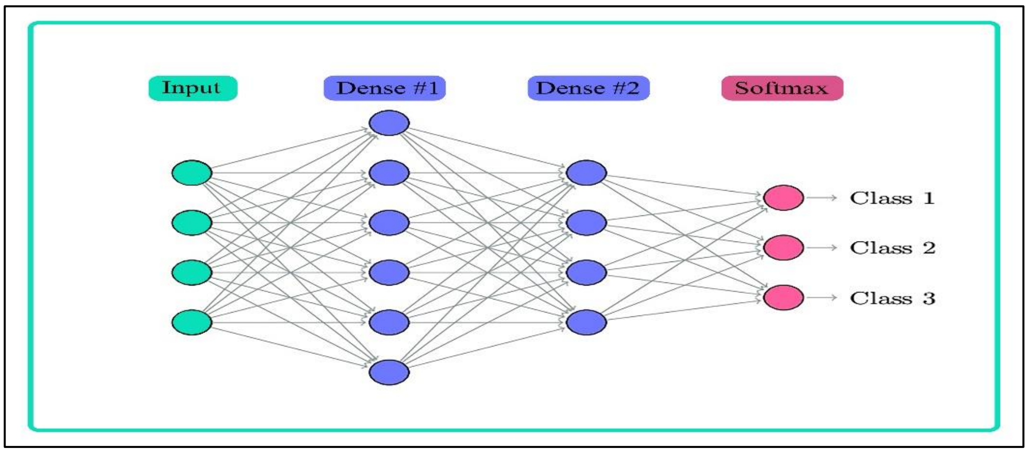


Figure 3.11: Example of Fully-Connected Neural Network [81].

b. Long Short-Term Memory (LSTM):

Multiple LSTM architectures are pertinent to the malware analysis problem, and malware datasets are appropriate for sequential analysis. It is a storage cell, that is a linear unit with a fixed amount of weight self-connection. Multiplicative input and output gateway units regulate the steady error flow so that unrelated inputs and memory contents have no effect on it. The input and output gates discover which defects must be scaled or trapped [82].

c. Bi-Long Short-Term Memory (Bi-LSTM):

The Bidirectional LSTM (BiLSTM) is a type of recurrent neural network that is commonly used in the field of natural language processing (NLP). In contrast to traditional LSTM, the input moves bidirectionally, allowing for the utilisation of data from both directions. It is also a powerful tool for modelling the logical relationships between words and phrases in both directions of the sequence. To summarise, BiLSTM incorporates an extra LSTM layer that reverses the information flow. To put it concisely, it suggests that the given sequence passes through an extra LSTM layer in reverse [83].

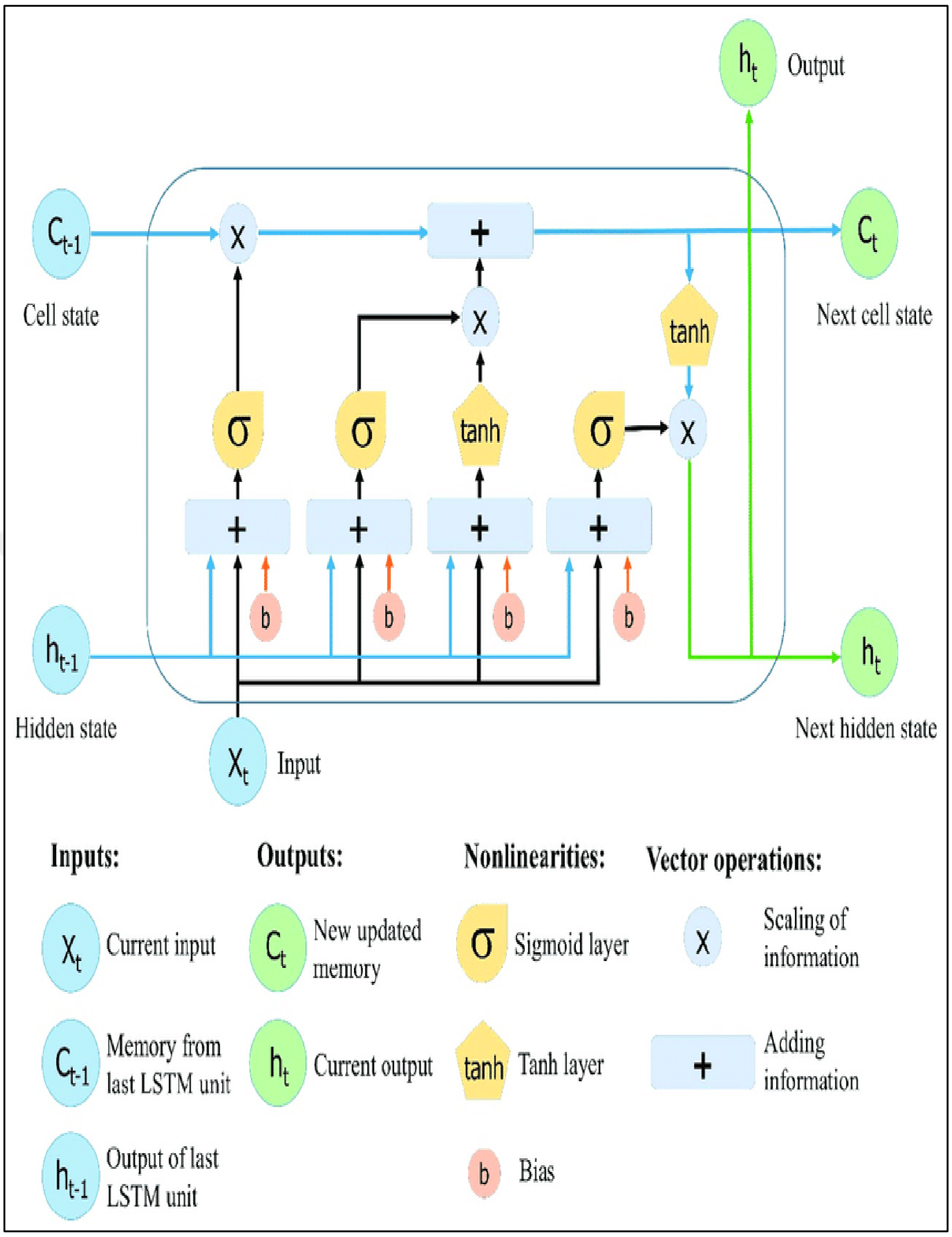


Figure 3.12: The Structure of The Long Short-Term Memory (LSTM) Neural Network [84].

d. Convolutional Neural Network (CNN):

CNN is an efficient recognitionalgorithm which is utilized extensively in recognition of patterns and processing of images. It has numerous characteristics, including a simple framework, fewer training variables, and flexibility. Invoice analysis and recognition of images have become a trending subject. Its weight-shared network design resembles biological neural networks more closely. It decreases the network model's complexity and the total amount of weights. includes two layers one is feature extraction layer, the input of each neuron is connected to the local receptive fields of the previous layer, and extracts the local feature. Once the local features is extracted, the positional relationship between it and other features also will be determined [85].

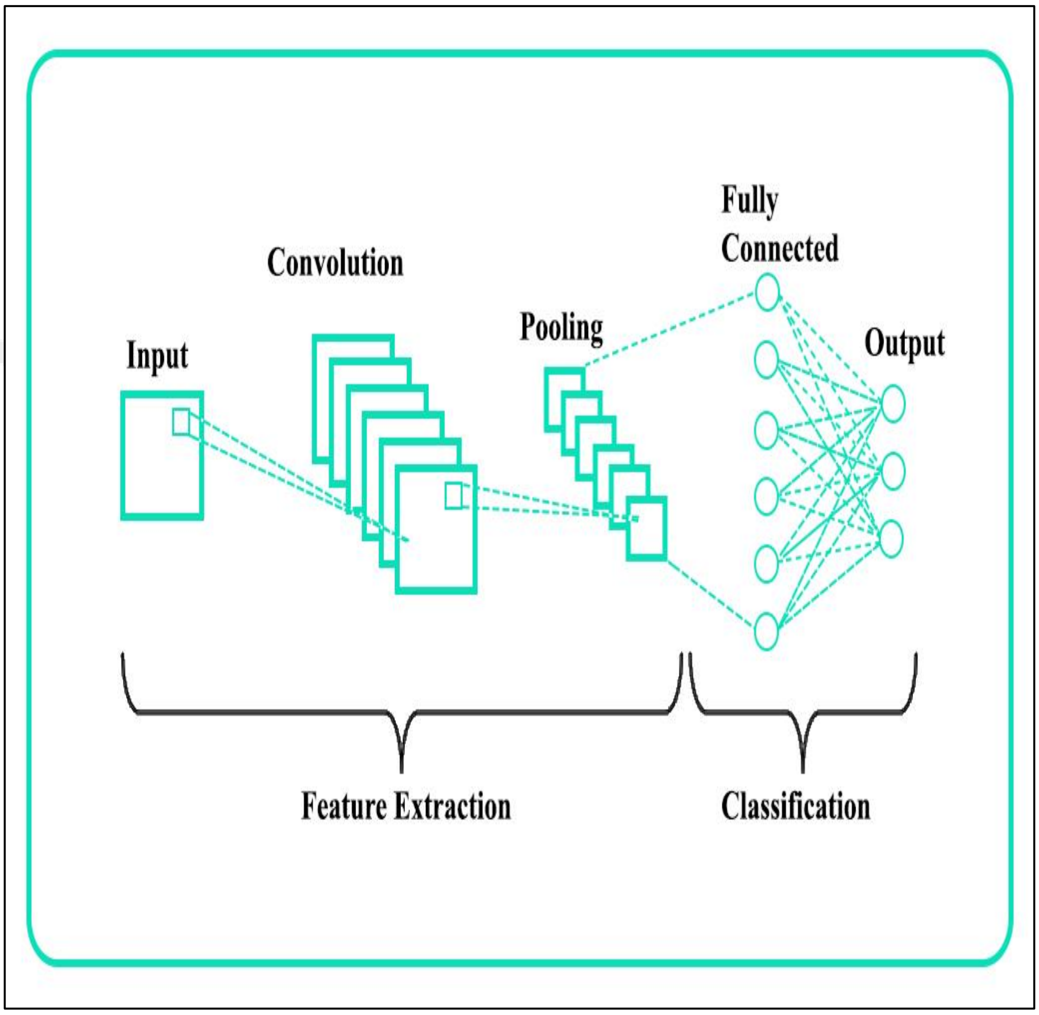


Figure 3.13: Basic Convolutional Neural Network (CNN) Architecture [86].

4. RESULTS AND DISCUSSIONS

4.1 INTRODUCTION

In this chapter, the results achieved through the suggested method, which was extensively explained in the previous chapter, are presented and analysed. In our particular scenario, the evaluation and comparison of machine learning and deep learning algorithms were conducted, with a focus on four performance metrics: accuracy, precision, recall, and F1-score that found by coding using python on jupyter notebook.

4.2 RESULT FOR MACHINE AND DEEP LEARNING

4.2.1 Machine Learning Algorithmes Results

a. Random Forest:

Results: Confusion Matrix

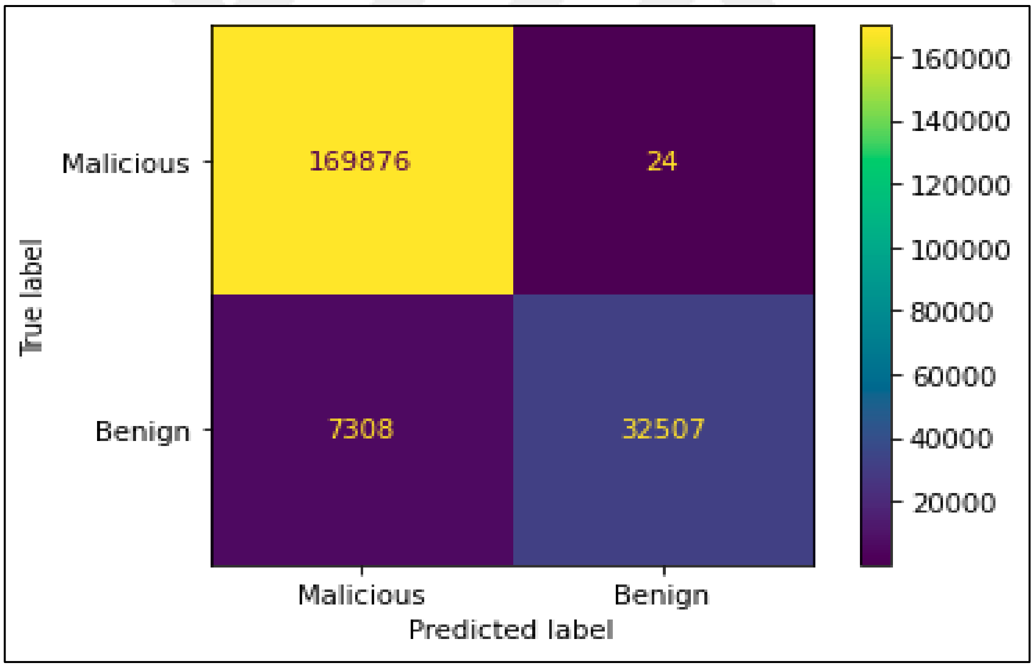


Figure 4.1: Confusion Matrix of Random Forest.

Here The randome forest algorithm recognized 169876 of malicious data and it Failed 24 time to recognized it.

Its also succeed on recognized 32507 of benign data and it failed 7308 time to recognized it.

It is clearly obvious that malicious detecting are better more than benign detecting because of number of data for training the model.

Classification Evaluation Metrics:

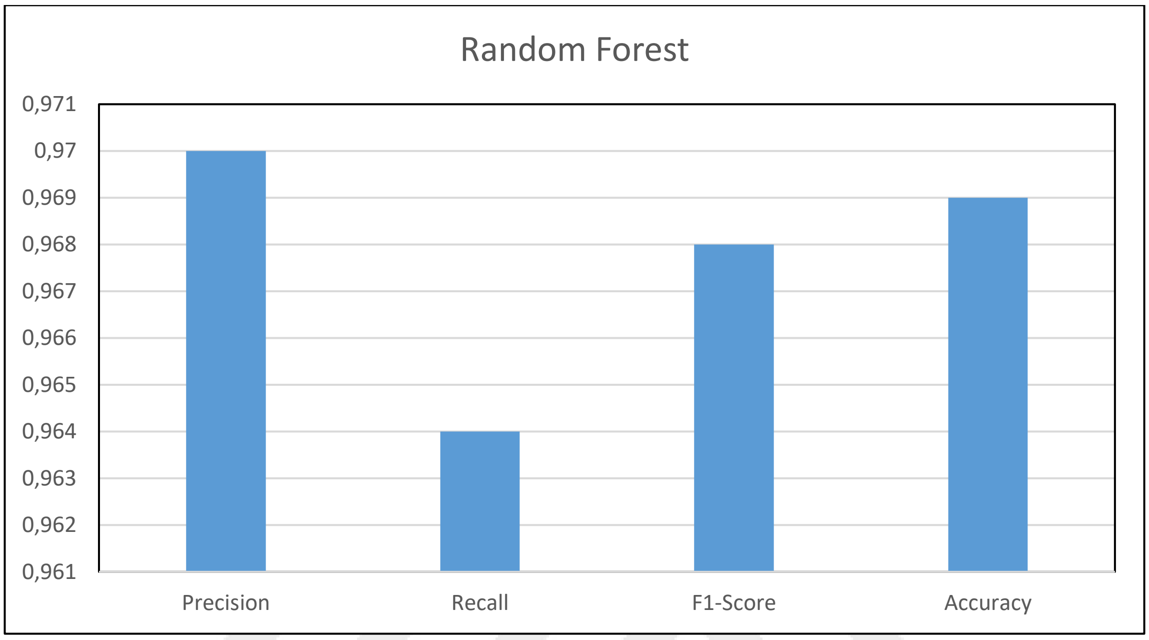


Figure 4.2: Classification Evaluation Metrics of Random Forest.

The metrics of this algorithm is (precision = 97, recall = 96.4, F1-score = 96.8 , accuracy =96.9).

b. Decision Trees:

Results:

Confusion Matrix:

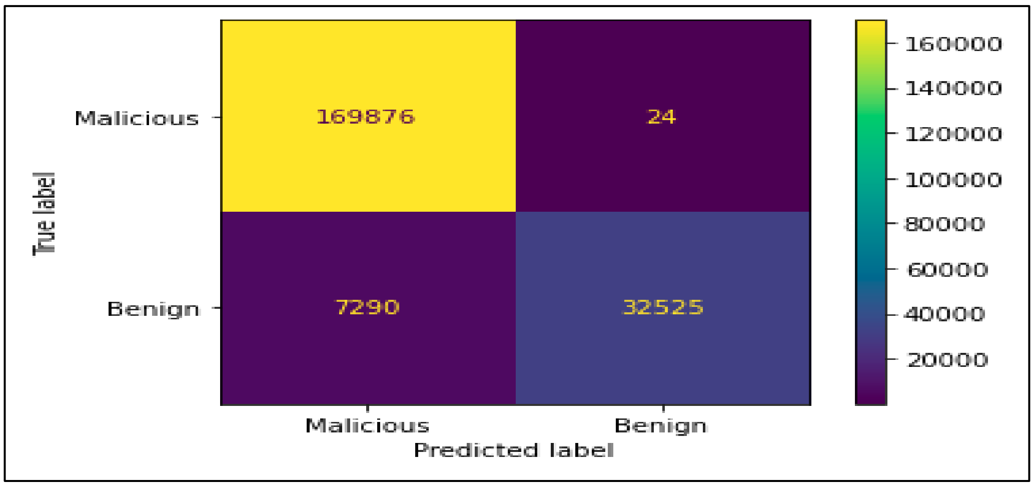


Figure 4.3: Confusion Matrix of Decision Trees.

Here The Decision Trees algorithm recognized 169876 of malicious data and it Failed 24 time to recognized it.

Its also succeed on recognized 32525 of benign data and it failed 7290 time to recognized it.

Classification Evaluation Metrics:

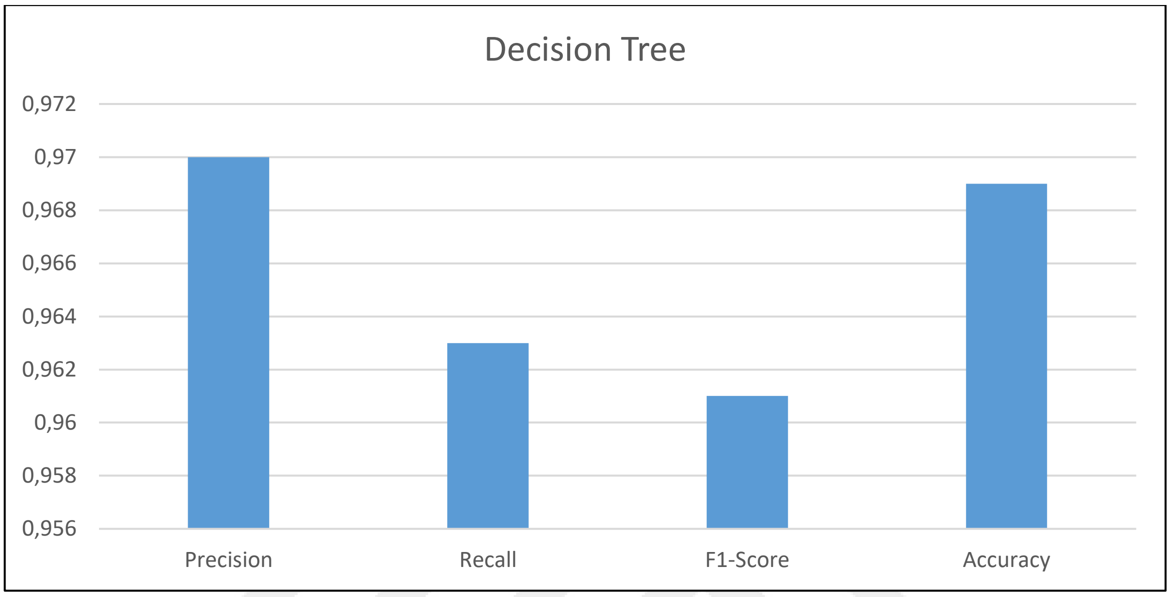


Figure 4.4: Classification Evaluation Metrics of Decision Trees.

The metrics of this algorithm is (precision = 97, recall = 96.3, F1-score = 96.1 , accuracy =96.9) .

c. AdaBoost :

Results:

Confusion Matrix:

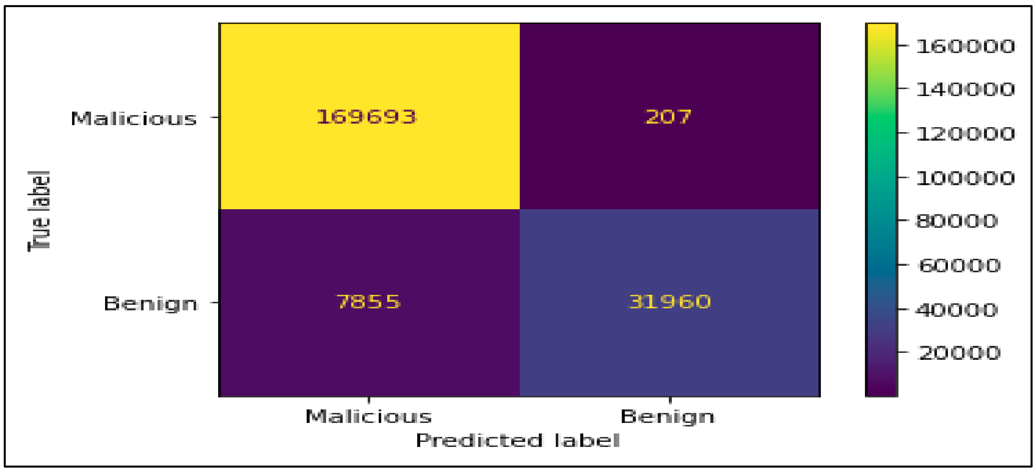


Figure 4.5: Confusion Matrix of AdaBoost.

Here The AdaBoost algorithm recognized 169693 of malicious data and it Failed 207 time to recognized it .

Its also succeed on recognized 31960 of benign data and it failed 7855 time to recognized it .

Classification Evaluation Metrics:

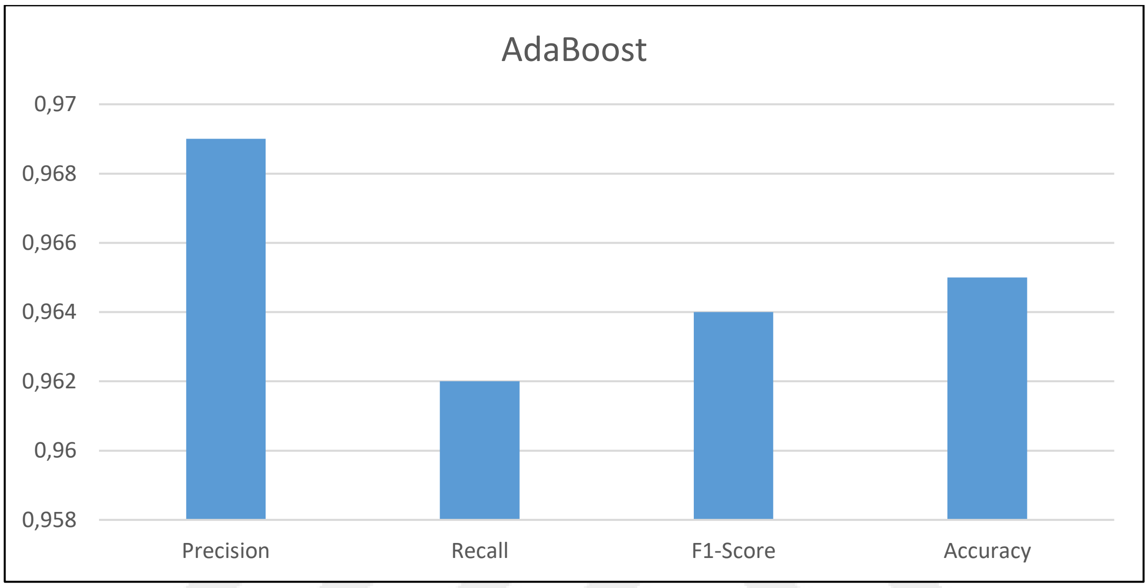


Figure 4.6: Classification Evaluation Metrics of AdaBoost.

The metrics of this algorithm is (precision = 96.9 ,recall = 96.2 , F1-score = 96.4, accuracy =96.5).

d. XGBoost :

Results : Confusion Matrix

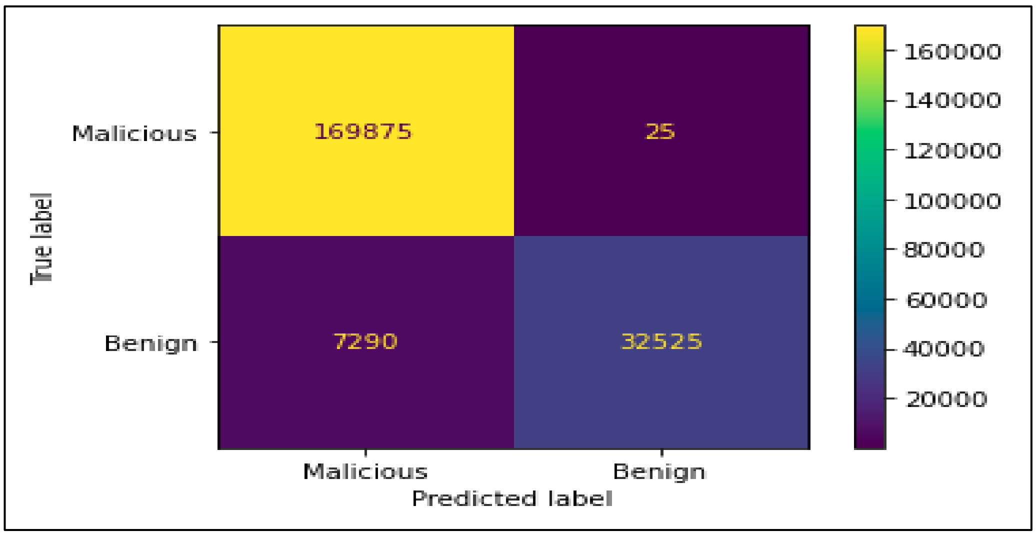


Figure 4.7: Confusion Matrix of XGBoost.

Here The XGBoost algorithm recognized 169875 of malicious data and it Failed 25 time to recognized it .

Its also succeed on recognized 32525 of benign data and it failed 7290 time to recognized it.

Classification Evaluation Metrics:

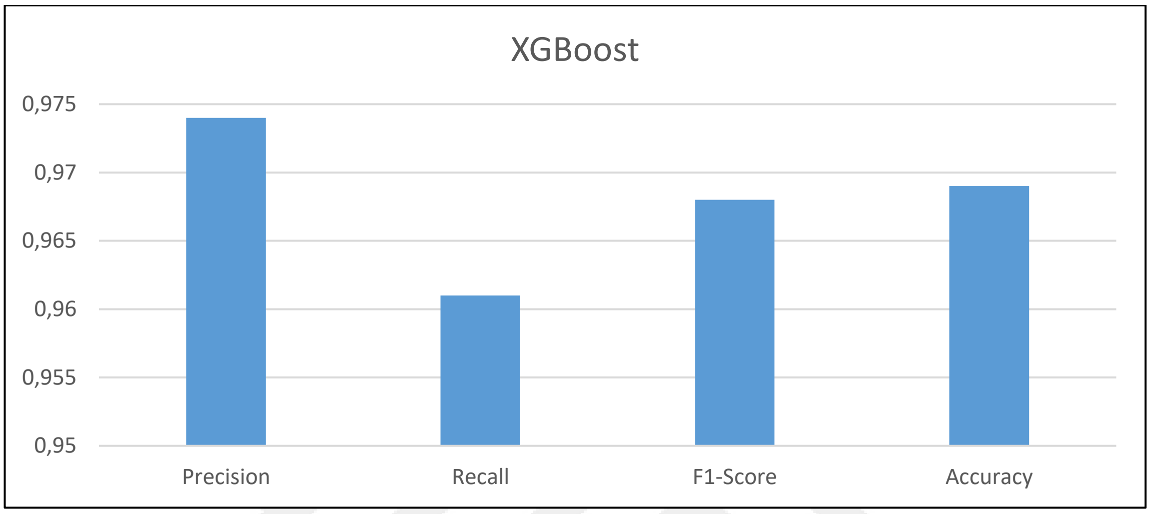


Figure 4.8: Classification Evaluation Metrics of XGBoost.

The metrics of this algorithm is (precision = 97.4 , recall = 96.1 , F1-score = 96.8, accuracy =96.9).

5. Gradient Boosting

Results: Confusion Matrix

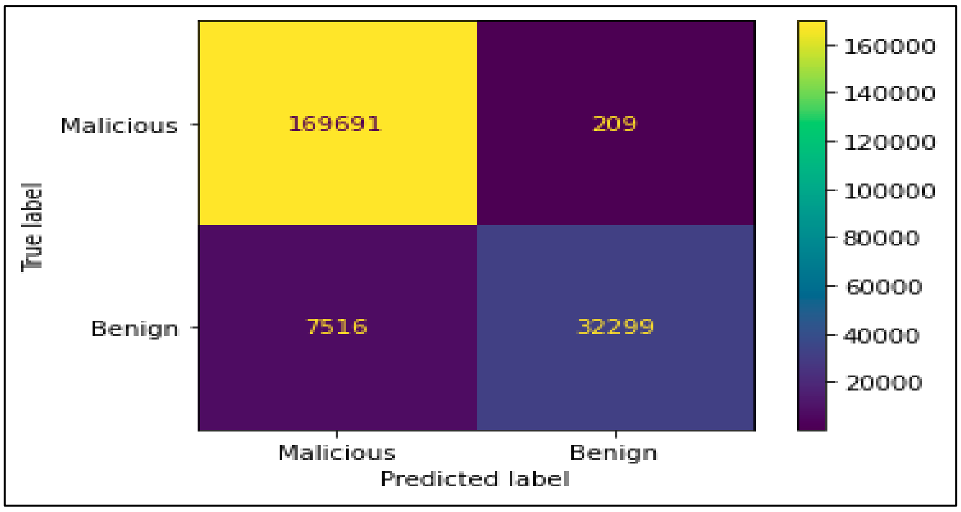


Figure 4.9: Confusion Matrix of Gradient Boosting.

Here The Gradient Boosting algorithm recognized 169691 of malicious data and it Failed 209 time to recognized it.

Its also succeed on recognized 32299 of benign data and it failed 7516 time to recognized it Classification Evaluation Metrics:

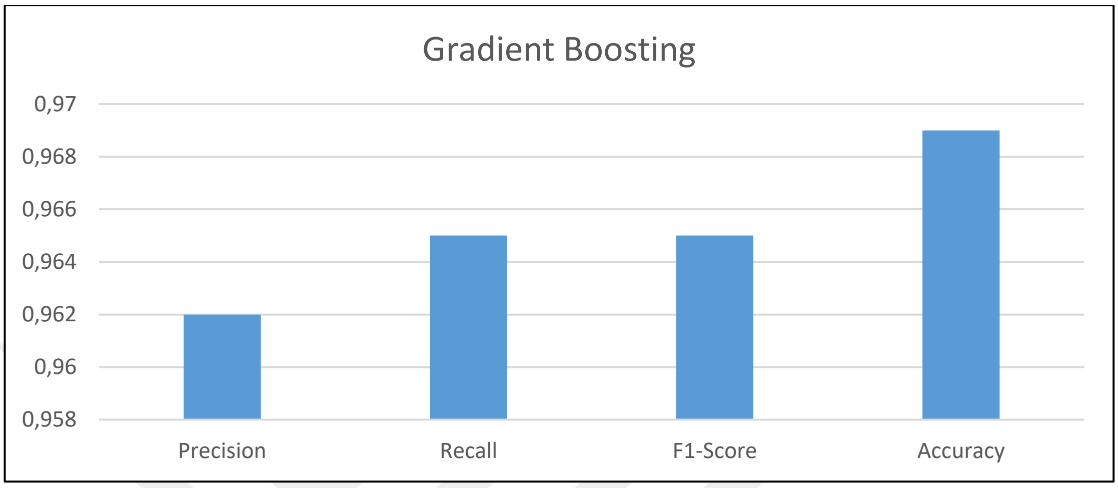


Figure 4.10: Classification Evaluation Metrics of Gradient Boosting.

The metrics of this algorithm is (precision = 96.2 , recall = 96.5 , F1-score = 96.5 , accuracy =96.9).

6. Hard Voting Classifier of (Randomforest , Decisiontree, Adaboost, Xgboost, Gradient boosting).

Results:

Confusion Matrix:

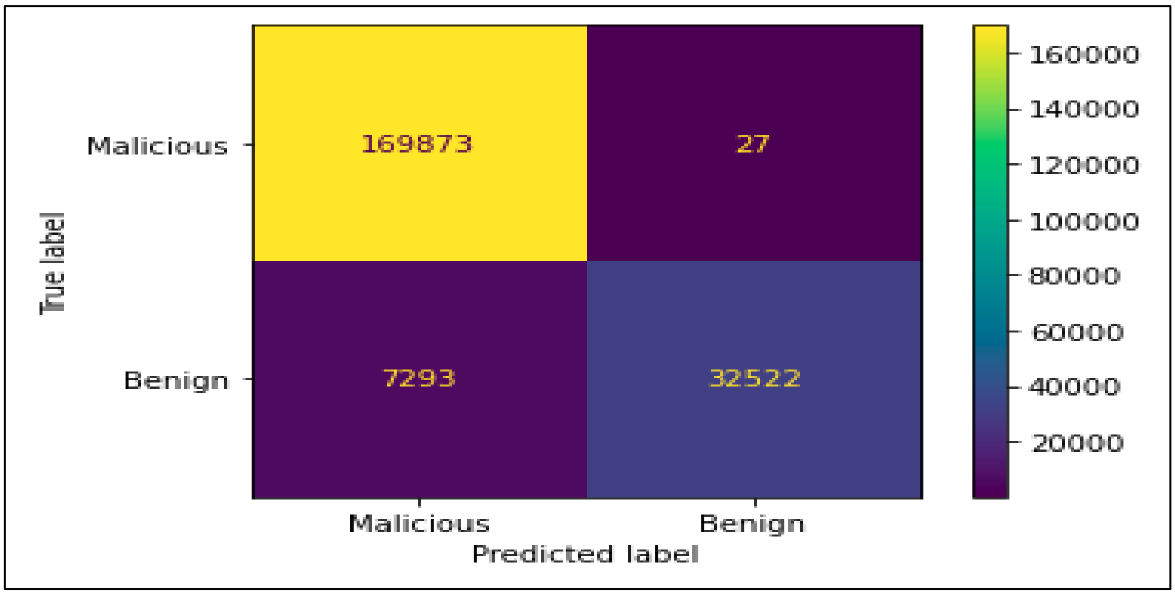


Figure 4.11: Confusion Matrix of Hard Voting Classifier of Previous Algorithmes.

Here Hard Voting Classifier of (Randomforest , Decisiontree, Adaboost, Xgboost, Gradient boosting) algorithm recognized 169873 of malicious data and it Failed 27 time to recognized it .

Its also succeed on recognized 32522 of benign data and it failed 7293 time to recognized it Classification Evaluation Metrics:

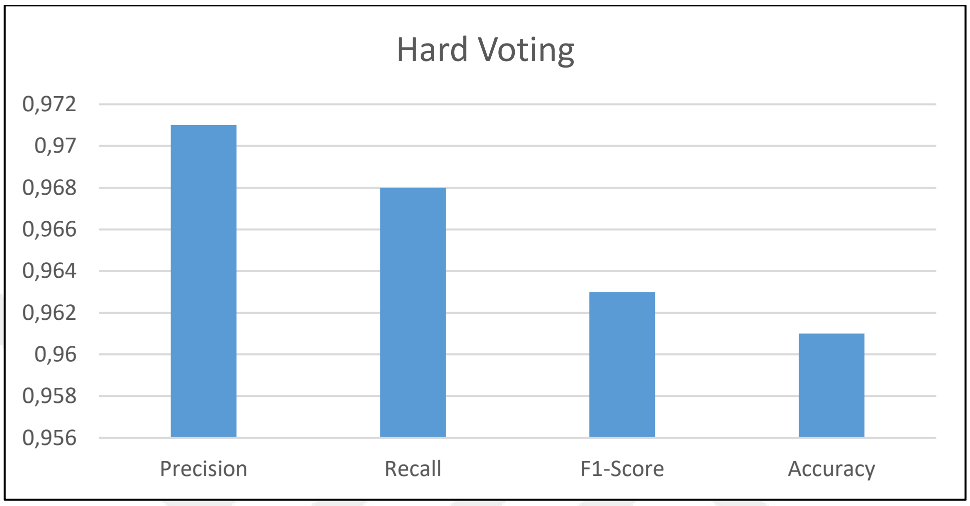


Figure 4.12: Classification Evaluation Metrics of Hard Voting Classifier of Previous Algorithms

The metrics of this algorithm is (precision = 97.1 , recall = 96.8 , F1-score = 96.3 , accuracy = 96.1) .

7. Soft Voting Classifier of (RandomForest , DecisionTree, AdaBoost, XGBoost, Gradient Boosting).

Results: Confusion Matrix:

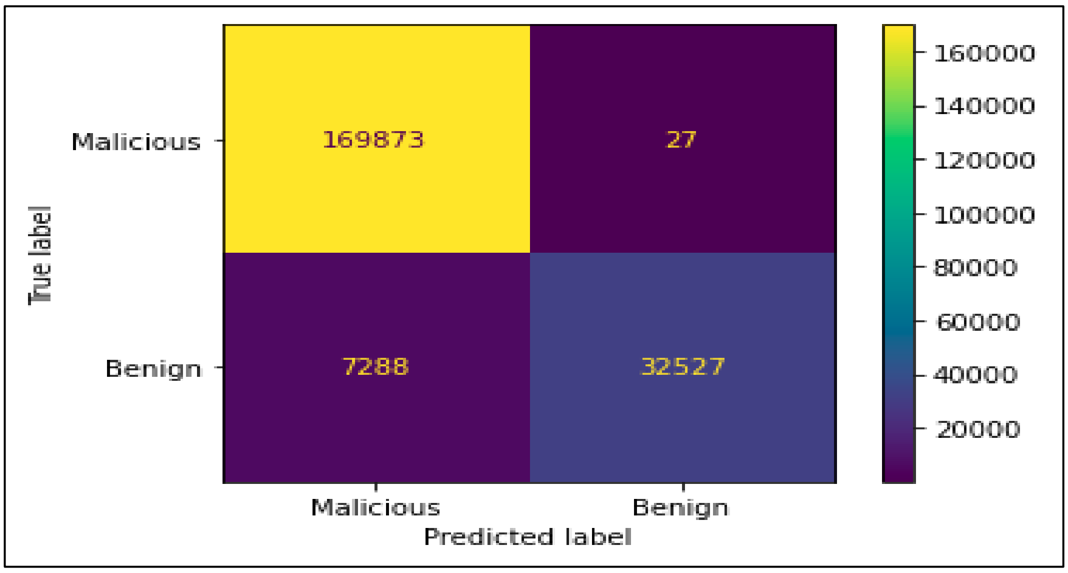


Figure 4.13: Confusion Matrix of Soft Voting Classifier of Previous Algorithms.

Here soft Voting Classifier of (Randomforest , Decisiontree, Adaboost, Xgboost, Gradient boosting) algorithm recognized 169873 of malicious data and it Failed 27 time to recognized it.

Its also succeed on recognized 32527 of benign data and it failed 7288 time to recognized it.

Classification Evaluation Metrics:

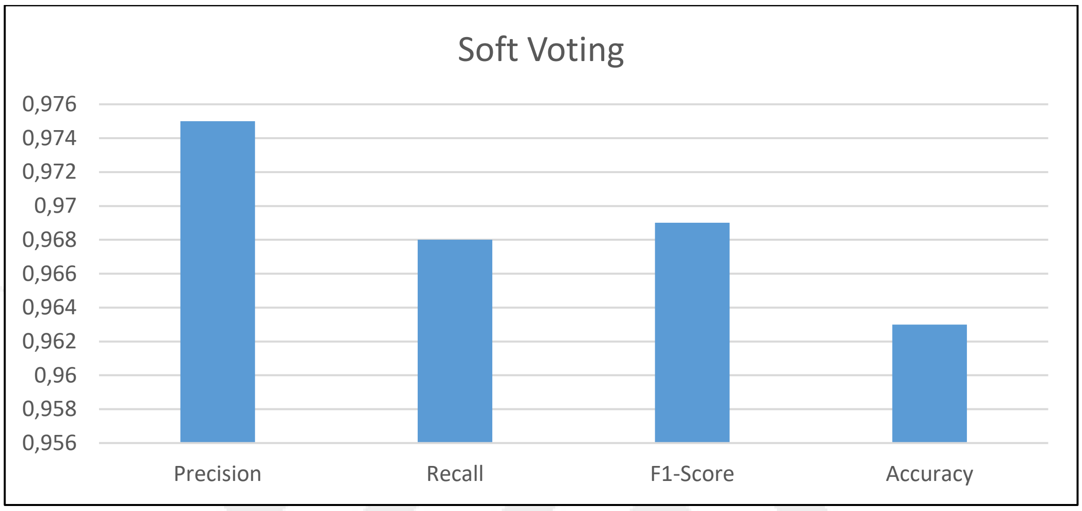


Figure 4.14: Classification Evaluation Metrics of soft Voting Classifier of Previous Algorithms.

The metrics of this algorithm is (precision = 97.5 ,recall = 96.8 , F1-score = 96.9, accuracy =96.3) .

The Results Summary of all Machine Learning algorithms as we can see below in the table (4.1):

Table 4.1: Result Summury of Machine Learning Algorithms.

Model	**Precision**	**Recall**	**F1-Score**	**Accuracy**
Random Forest	0.97	0.964	0.968	0.969
Decision Tree	0.97	0.963	0.961	0.969
AdaBoost	0.969	0.962	0.964	0.965
XGBoost	0.974	0.961	0.968	0.969
Gradient Boosting	0.962	0.965	0.965	0.969
Hard Voting	0.971	0.968	0.963	0.961
Soft Voting	0.975	0.968	0.969	0.963

Classification Evaluation Metrics of all Machine Learning algorithms was as the Figure (4.15) below:

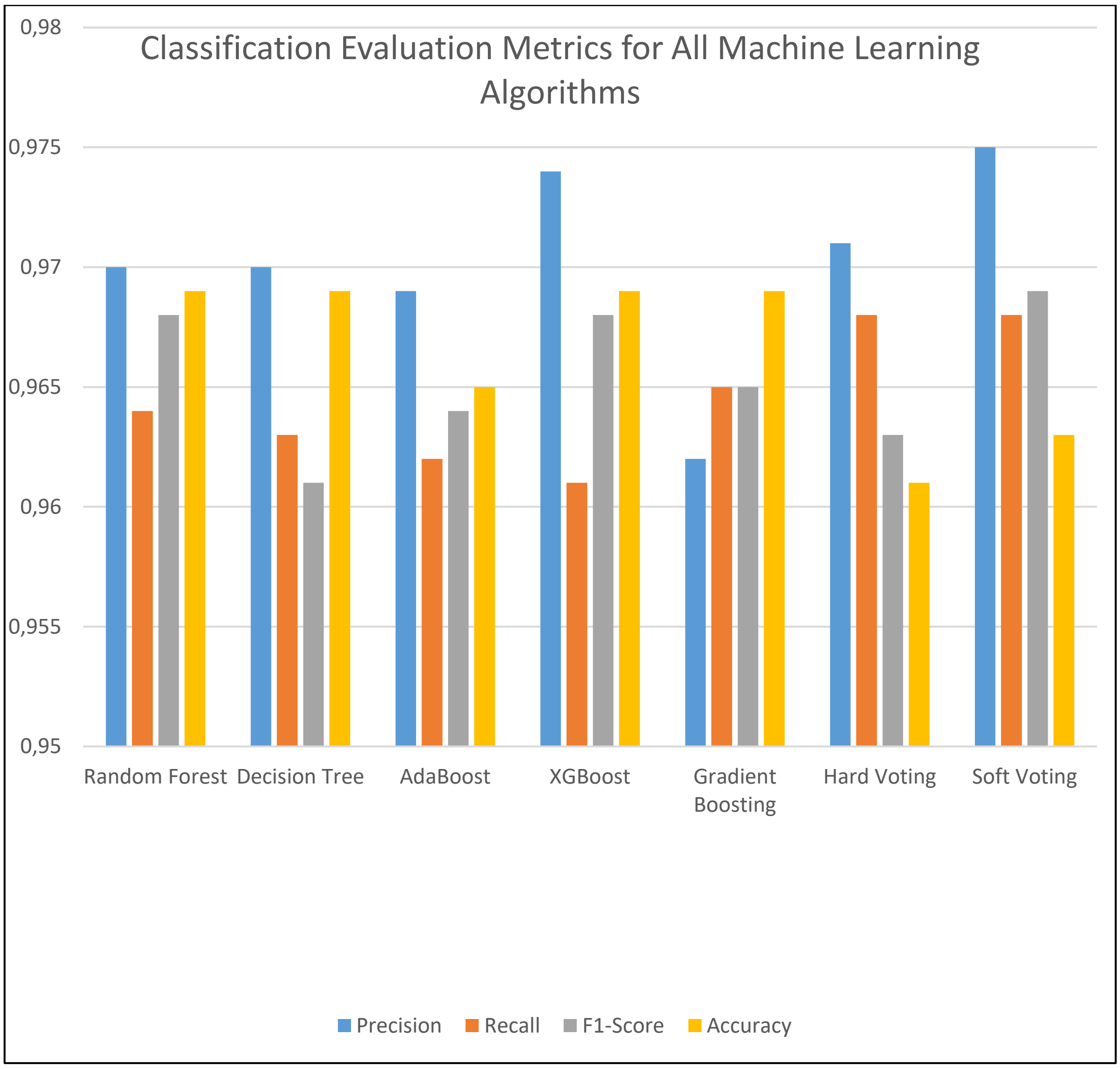
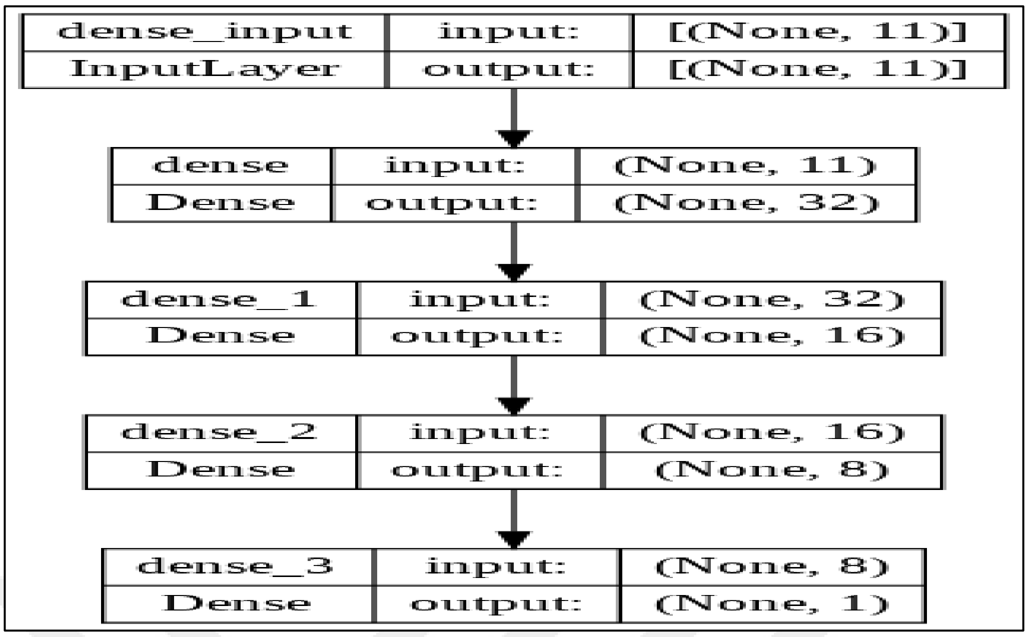


Figure 4.15: Classification Evaluation Metrics for All Machine Learning Algorithms.

4.2.2 Deep Learning Algorithmes Results

a. Fully Connected Neural Network (NN):

Model Architecture:



dense_input | input: | [(None, 11)]
InputLayer | output: | [(None, 11)]
dense | input: | (None, 11)
Dense | output: | (None, 32)
dense_1 | input: | (None, 32)
Dense | output: | (None, 16)
dense_2 | input: | (None, 16)
Dense | output: | (None, 8)
dense_3 | input: | (None, 8)
Dense | output: | (None, 1)

Figure 4.16: Model Architecture Fully Connected Neural Network (NN).

Results:

Confusion Matrix:

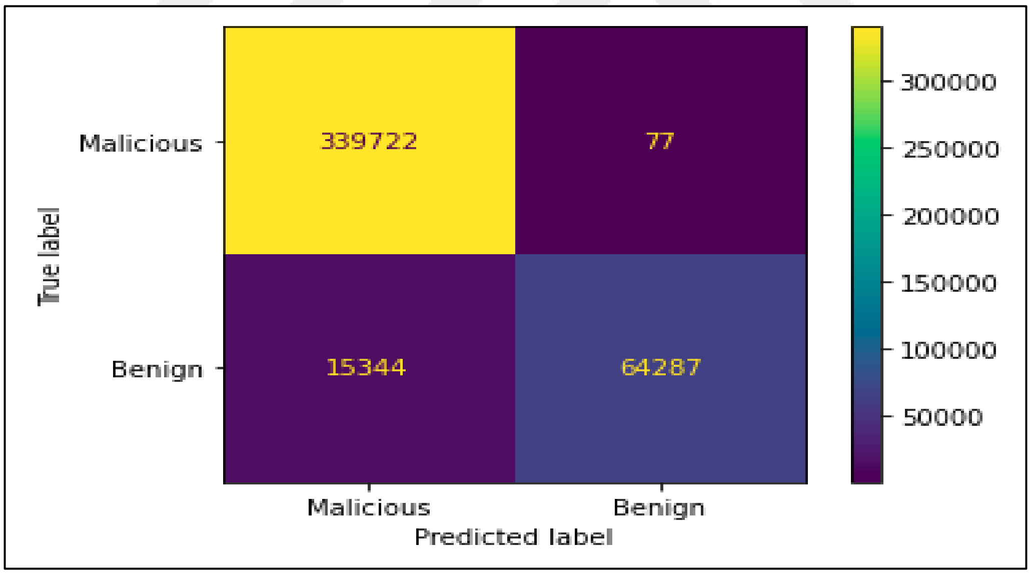


Figure 4.17: Confusion Matrix of Fully Connected Neural Network (NN).

Here the Fully Connected Neural Network (NN) algorithm recognized 339722 of malicious data and it Failed 77 time to recognize it.

Its also succeed on recognized 64287 of benign data and it failed 15344 time to recognized it Classification Evaluation Metrics:

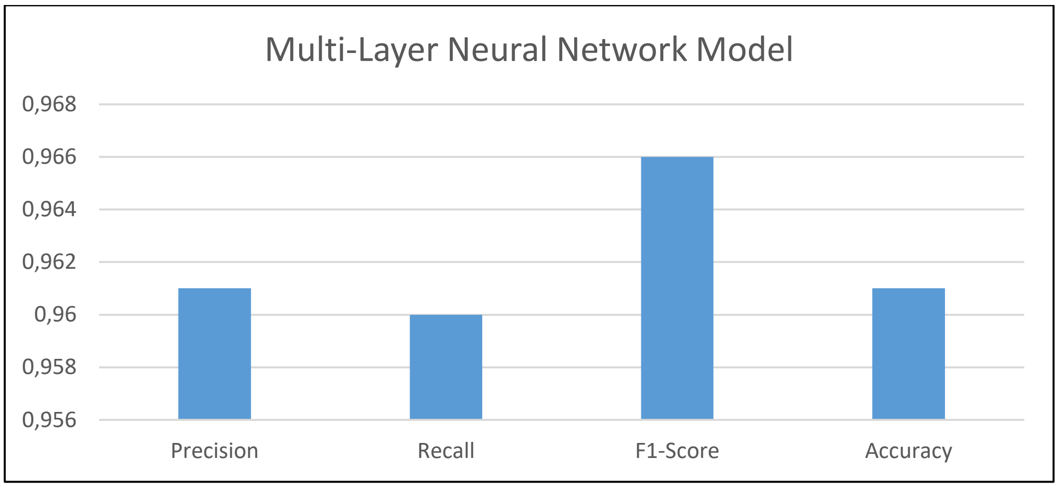


Figure 4.18: Classification Evaluation Metrics of Fully Connected Neural Network (NN).

The metrics of this algorithm is (precision = 96.1,recall = 0.96, F1-score = 96.6, accuracy =96.1).

Model Performance (Accuracy):

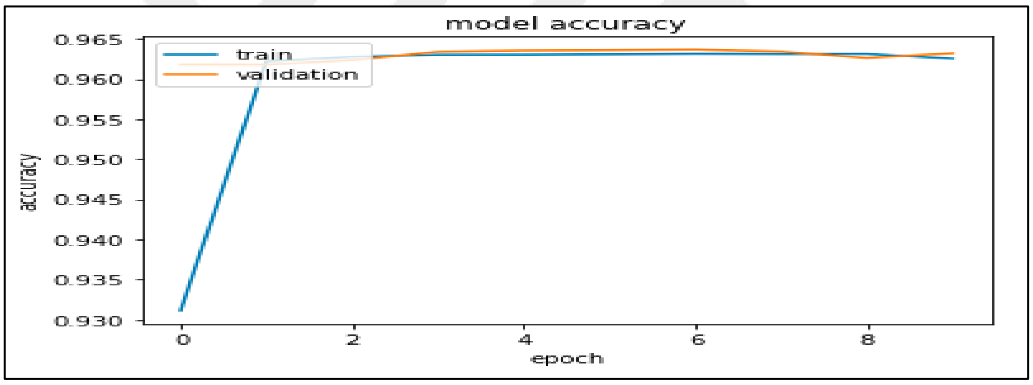


Figure 4.19: Model Performance (Accuracy) of Fully Connected Neural Network (NN).

Model Performance (Loss):

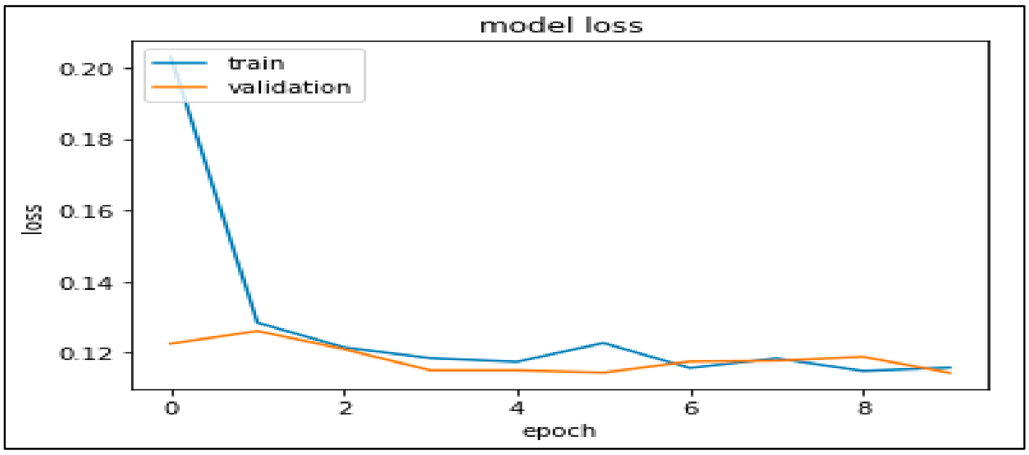
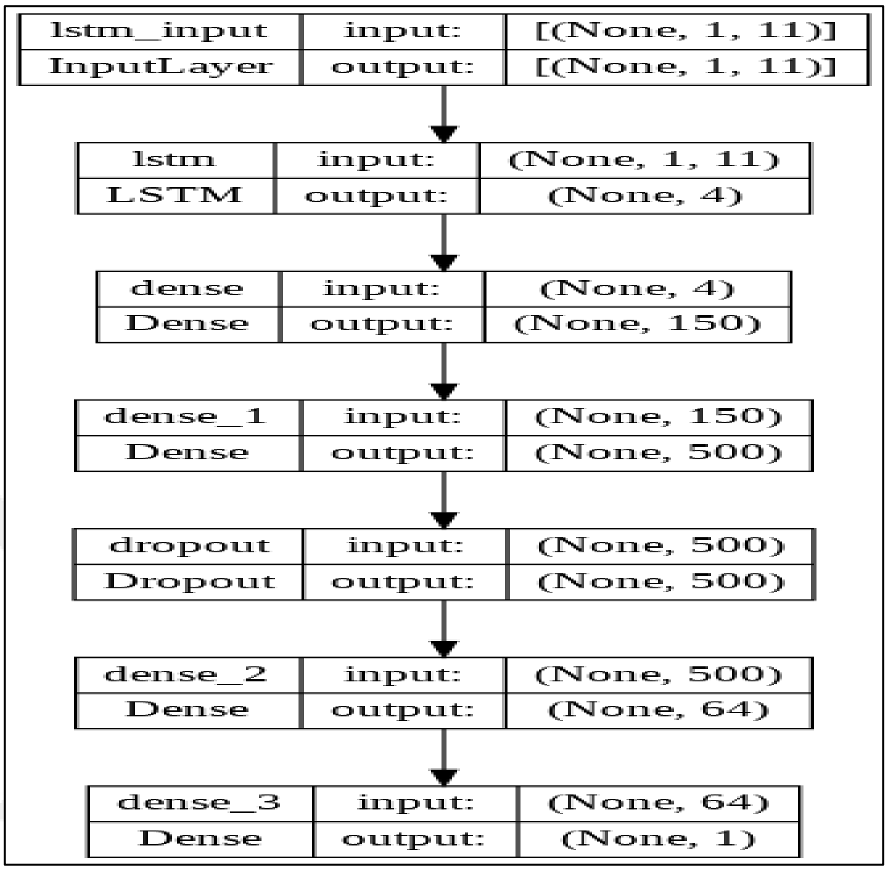


Figure 4.20: Model Performance (Loss) of Fully Connected Neural Network (NN).

b. Long Short -Term Memory (LSTM)

Model Architecture :



lstm_input | input: | [(None, 1, 11)]
InputLayer | output: | [(None, 1, 11)]
lstm | input: | (None, 1, 11)
LSTM | output: | (None, 4)
dense | input: | (None, 4)
Dense | output: | (None, 150)
dense_1 | input: | (None, 150)
Dense | output: | (None, 500)
dropout | input: | (None, 500)
Dropout | output: | (None, 500)
dense_2 | input: | (None, 500)
Dense | output: | (None, 64)
dense_3 | input: | (None, 64)
Dense | output: | (None, 1)

Figure 4.21: Model Architecture of Long Short-Term Memory (LSTM).

Results:

Confusion Matrix

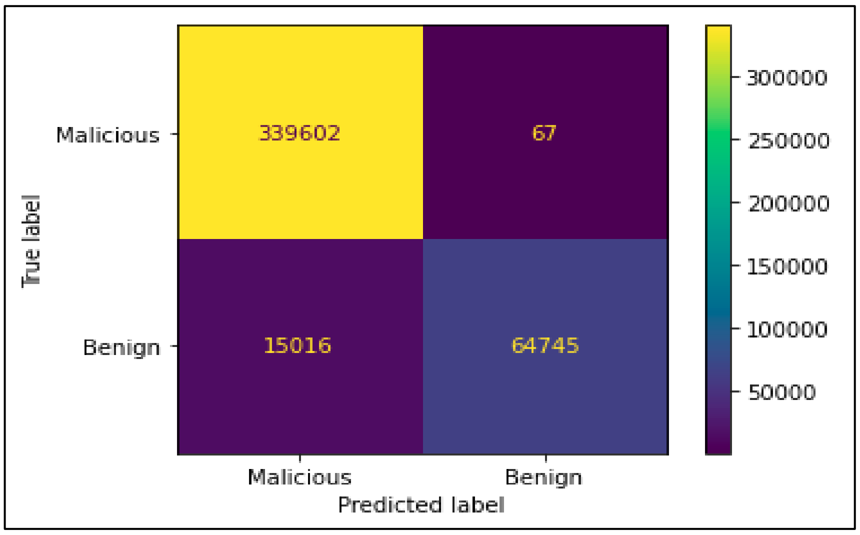


Figure 4.22: Confusion Matrix of Long Short -Term Memory (LSTM).

Here the Fully Connected Neural Network (NN) algorithm recognized 339602 of malicious data and it Failed 67 time to recognize it.

Its also succeed on recognized 64745 of benign data and it failed 15016 time to recognized it.

Classification Evaluation Metrics:

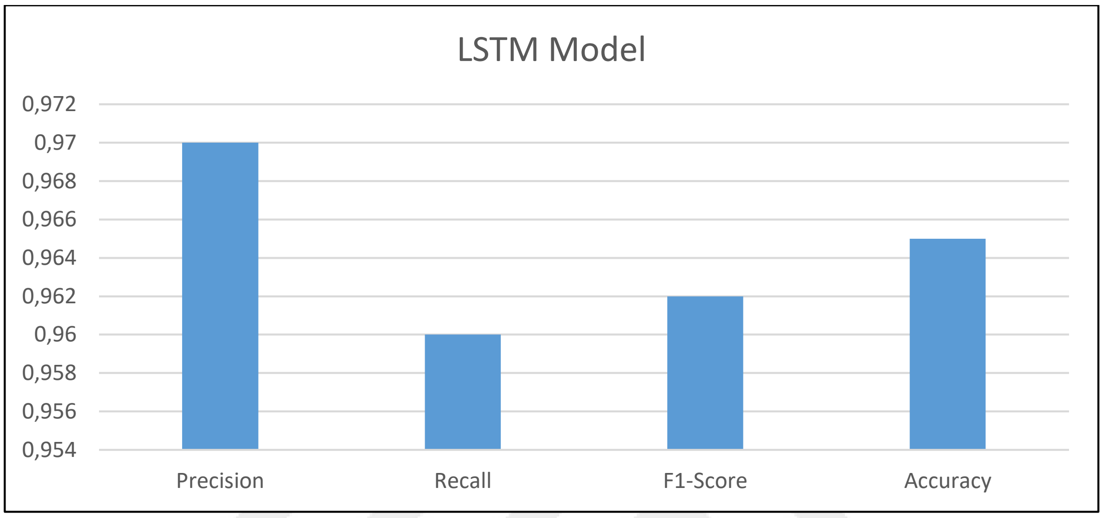


Figure 4.23: Classification Evaluation Metrics of Long Short -Term Memory (LSTM).

The metrics of this algorithm is (precision = 97, recall = 96, F1-score = 96.2 , accuracy =96.5).

Model Performance (Accuracy):

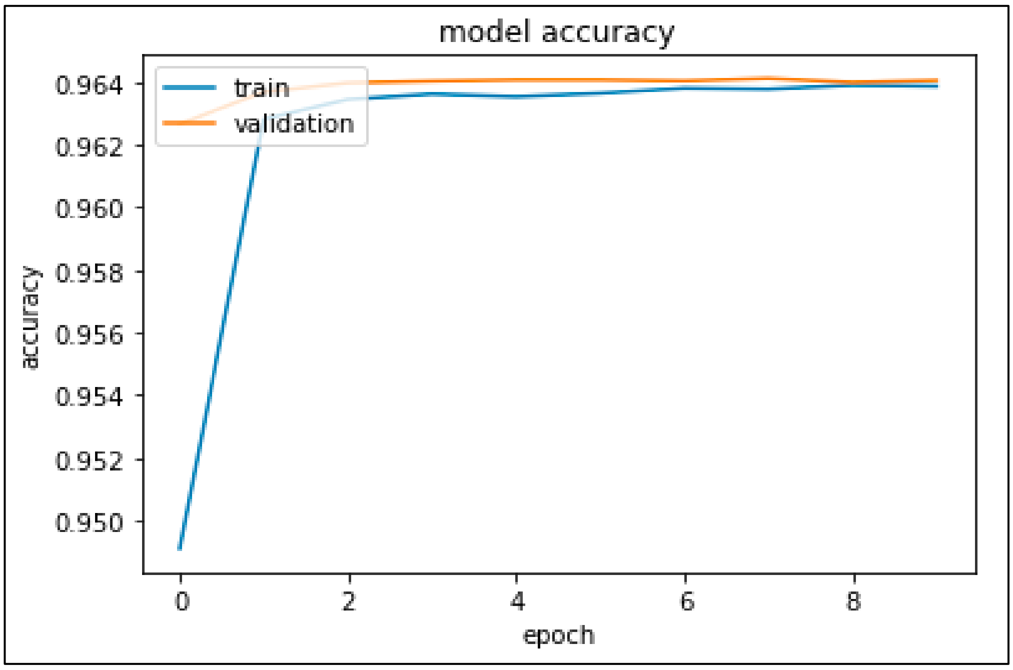


Figure 4.24: Model Performance (Accuracy) of Long Short -Term Memory (LSTM).

Model Performance (Loss):

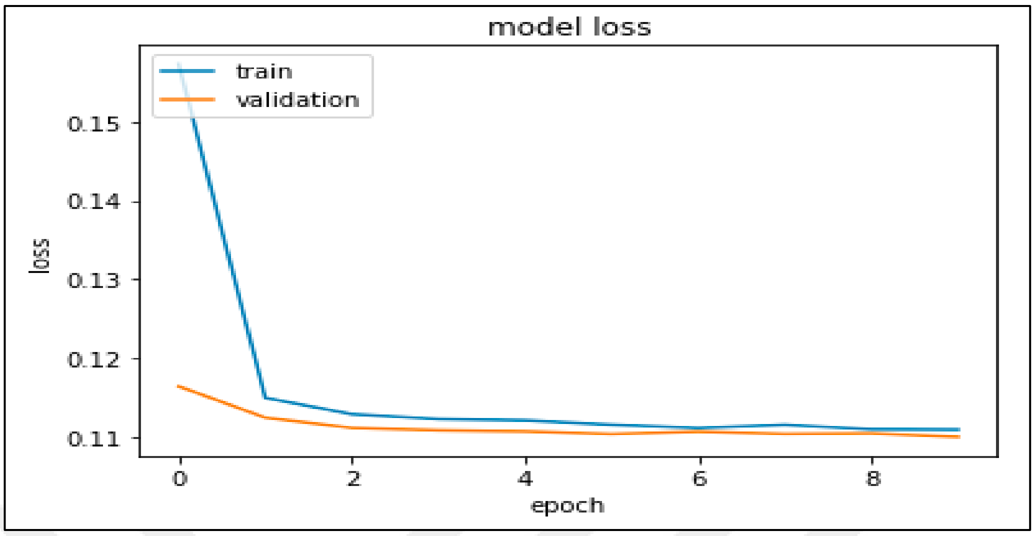
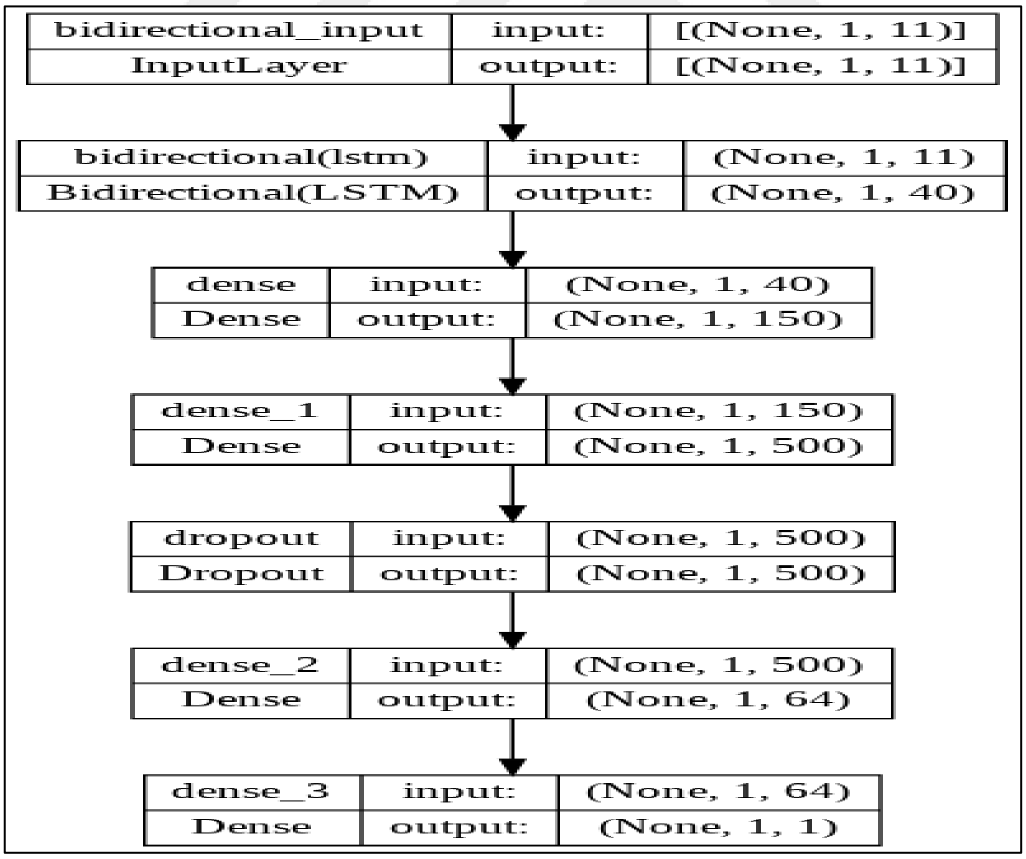


Figure 4.25: Model Performance (Loss) of Long Short -Term Memory (LSTM).

c. Bi-Long Short -Term Memory (Bi-LSTM)

Model Architecture:



bidirectional_input | input: | [(None, 1, 11)]
InputLayer | output: | [(None, 1, 11)]
bidirectional(lstm) | input: | (None, 1, 11)
Bidirectional(LSTM) | output: | (None, 1, 40)
dense | input: | (None, 1, 40)
Dense | output: | (None, 1, 150)
dense_1 | input: | (None, 1, 150)
Dense | output: | (None, 1, 500)
dropout | input: | (None, 1, 500)
Dropout | output: | (None, 1, 500)
dense_2 | input: | (None, 1, 500)
Dense | output: | (None, 1, 64)
dense_3 | input: | (None, 1, 64)
Dense | output: | (None, 1, 1)

Figure 4.26: Model Architecture of Bi-Long Short-Term Memory (Bi-LSTM).

Results :

Confusion Matrix :

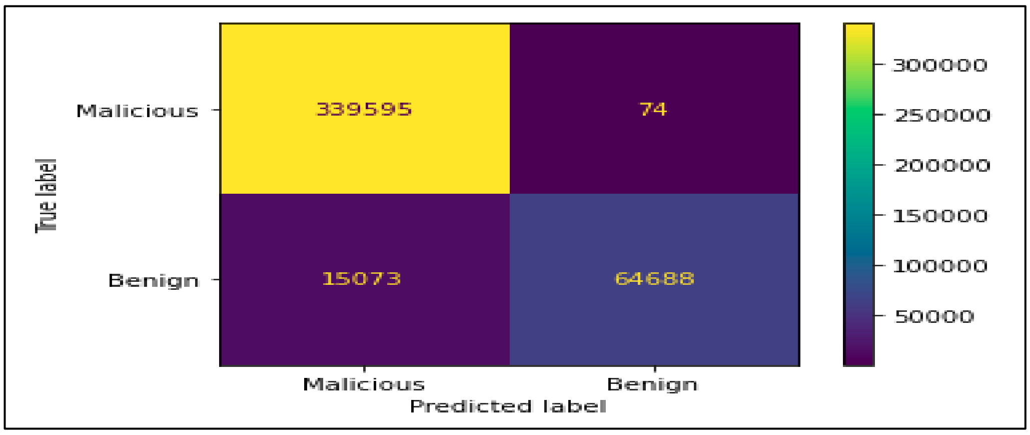


Figure 4.27: Confusion Matrix of Bi-Long Short-Term Memory (Bi-LSTM).

Here the Bi-Long Short-Term Memory (Bi-LSTM) algorithm recognized 339595 of malicious data and it Failed 74 time to recognize it.

Its also succeed on recognized 64688 of benign data and it failed 15073 time to recognized it.

Classification Evaluation Metrics:

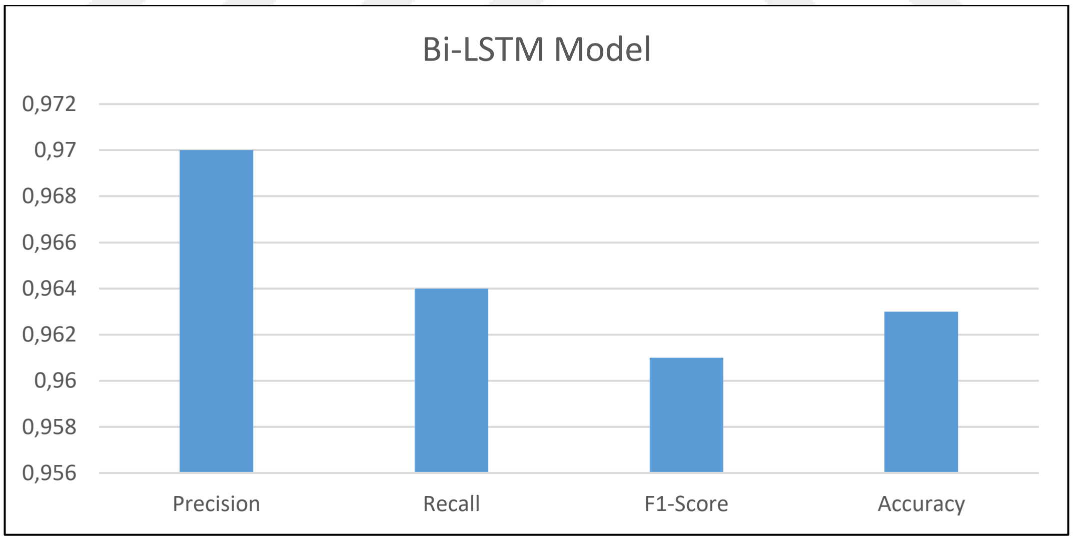


Figure 4.28: Classification Evaluation Metrics of Bi-Long Short-Term Memory (Bi-LSTM).

The metrics of this algorithm is (precision = 97, recall = 96.4, F1-score = 96.1 , accuracy = 96.3).

Model Performance (Accuracy):

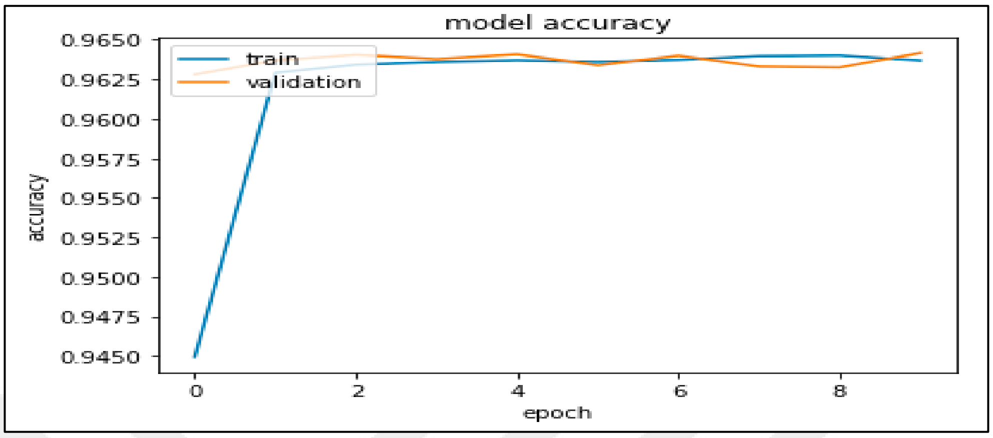


Figure 4.29: Model Performance (Accuracy) of Bi-Long Short-Term Memory (Bi-LSTM).

Model Performance (Loss):

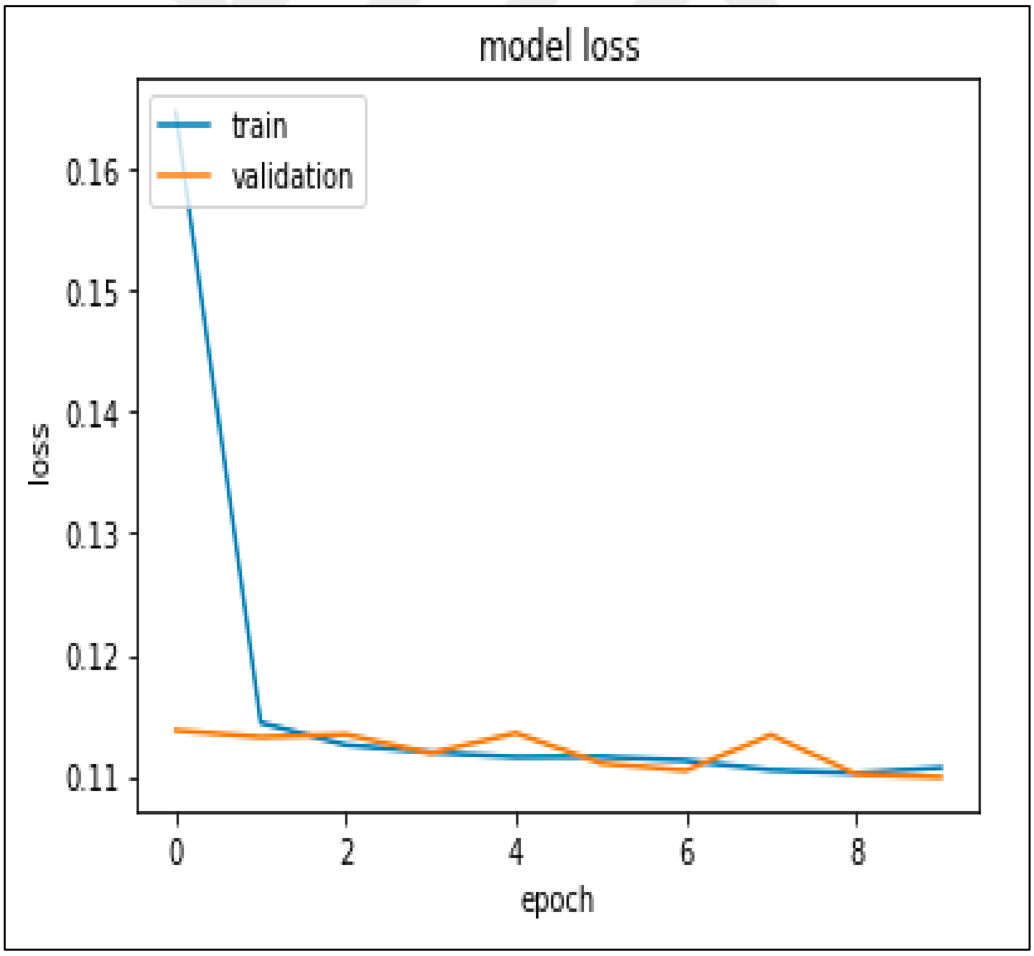


Figure 4.30: Model Performance (Accuracy)of Bi-Long Short-Term Memory (Bi-LSTM).

d. Convolutional Neural Network (CNN):

Model Architecture:

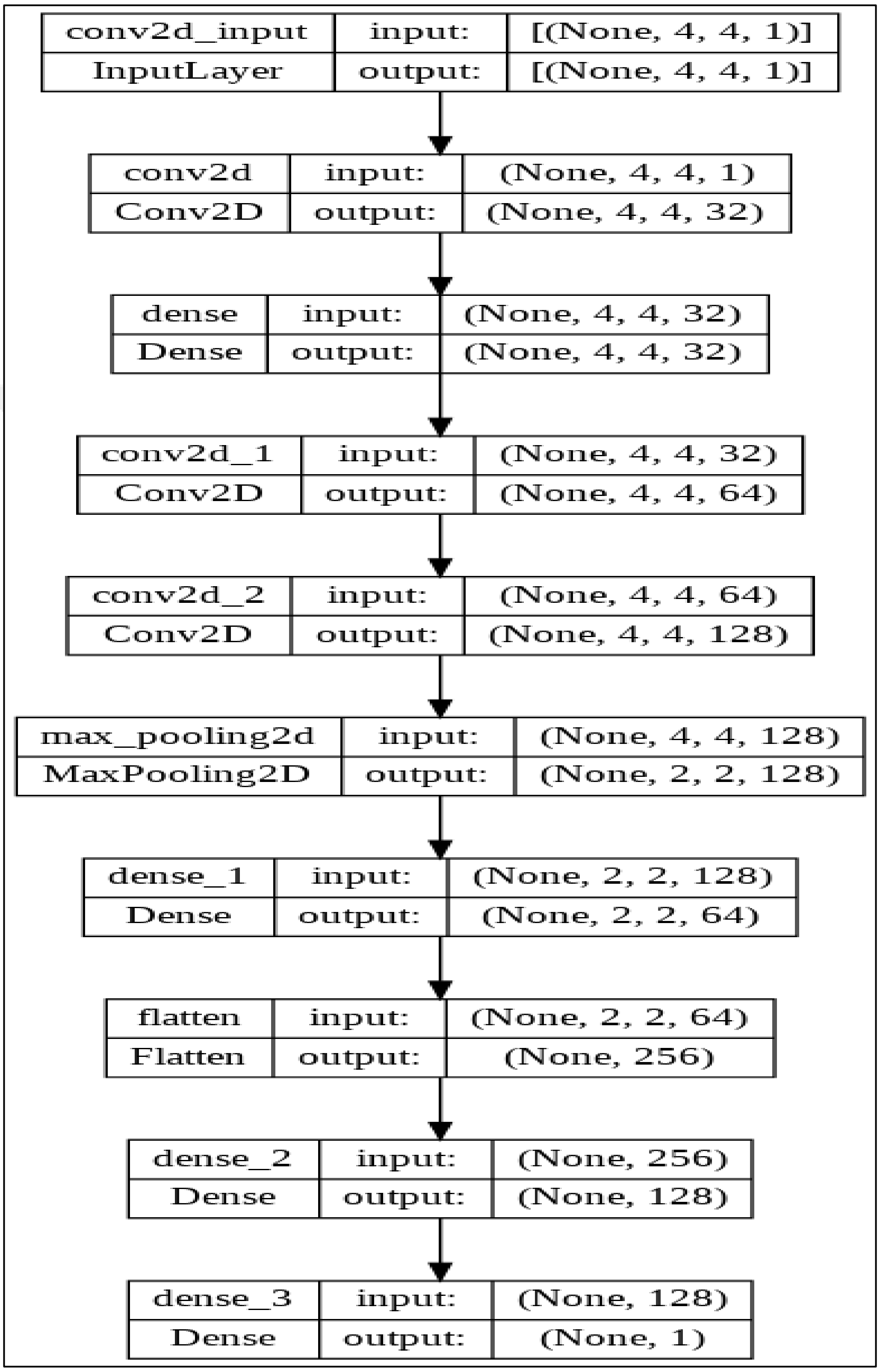

conv2d_input | InputLayer | input: [(None, 4, 4, 1)] | output: [(None, 4, 4, 1)]
conv2d | Conv2D | input: (None, 4, 4, 1) | output: (None, 4, 4, 32)
dense | Dense | input: (None, 4, 4, 32) | output: (None, 4, 4, 32)
conv2d_1 | Conv2D | input: (None, 4, 4, 32) | output: (None, 4, 4, 64)
conv2d_2 | Conv2D | input: (None, 4, 4, 64) | output: (None, 4, 4, 128)
max_pooling2d | MaxPooling2D | input: (None, 4, 4, 128) | output: (None, 2, 2, 128)
dense_1 | Dense | input: (None, 2, 2, 128) | output: (None, 2, 2, 64)
flatten | Flatten | input: (None, 2, 2, 64) | output: (None, 256)
dense_2 | Dense | input: (None, 256) | output: (None, 128)
dense_3 | Dense | input: (None, 128) | output: (None, 1)

Figure 4.31: Model Architecture of Convolutional Neural Network (CNN).

Results:

Confusion Matrix:

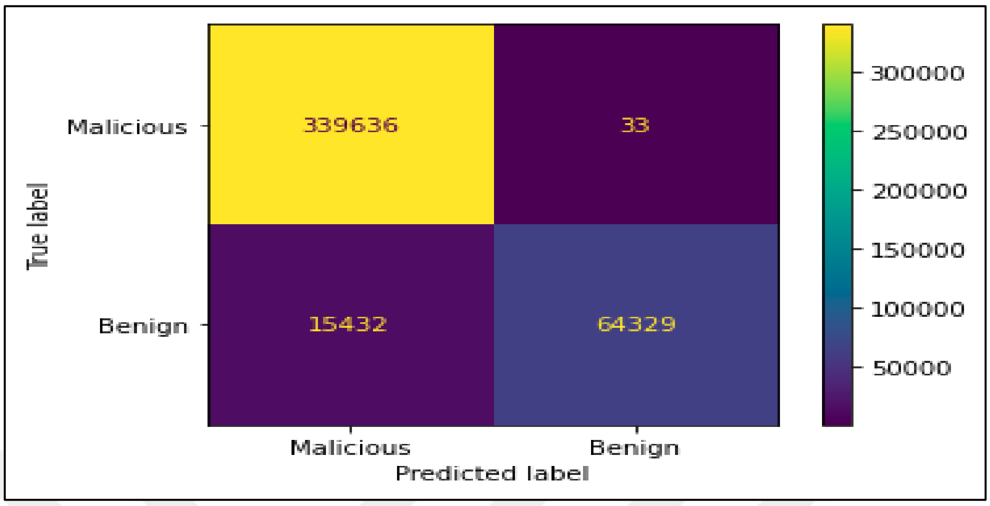


Figure 4.32: Confusion Matrix of Convolutional Neural Network (CNN).

Here the Convolutional Neural Network (CNN) algorithm recognized 339636 of malicious data and it Failed 33 time to recognize it.

Its also succeed on recognized 64329 of benign data and it failed 15432 time to recognized it.

Classification Evaluation Metrics:

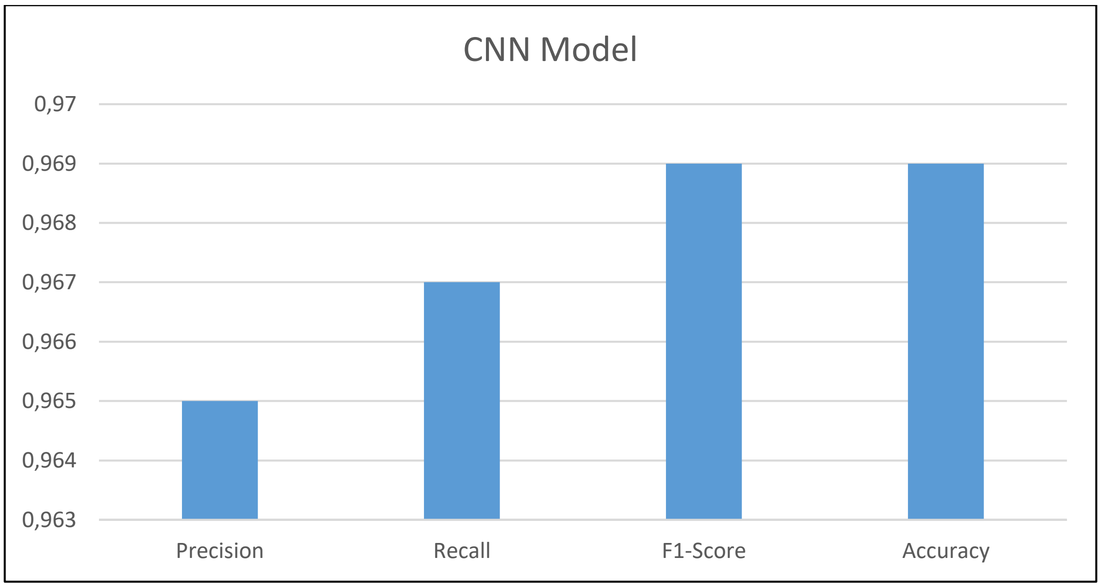


Figure 4.33: Classification Evaluation Metrics of Convolutional Neural Network (CNN).

The metrics of this algorithm is (precision = 96.5 , recall = 96.7 , F1-score = 96.9, accuracy = 96.9).

Model Performance (Accuracy):

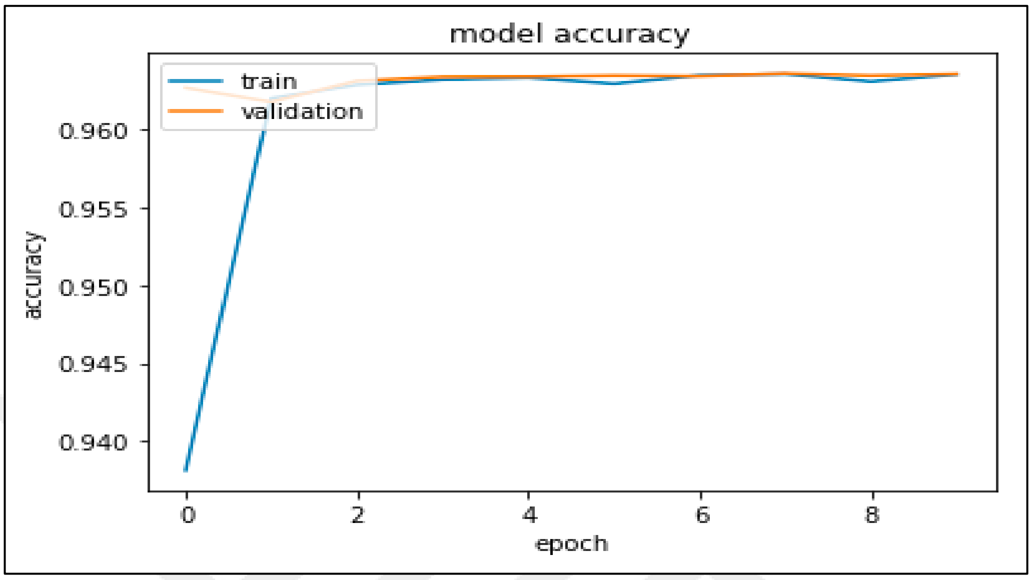


Figure 4.34: Model Performance (Accuracy) of Convolutional Neural Network (CNN).

Model Performance (Loss):

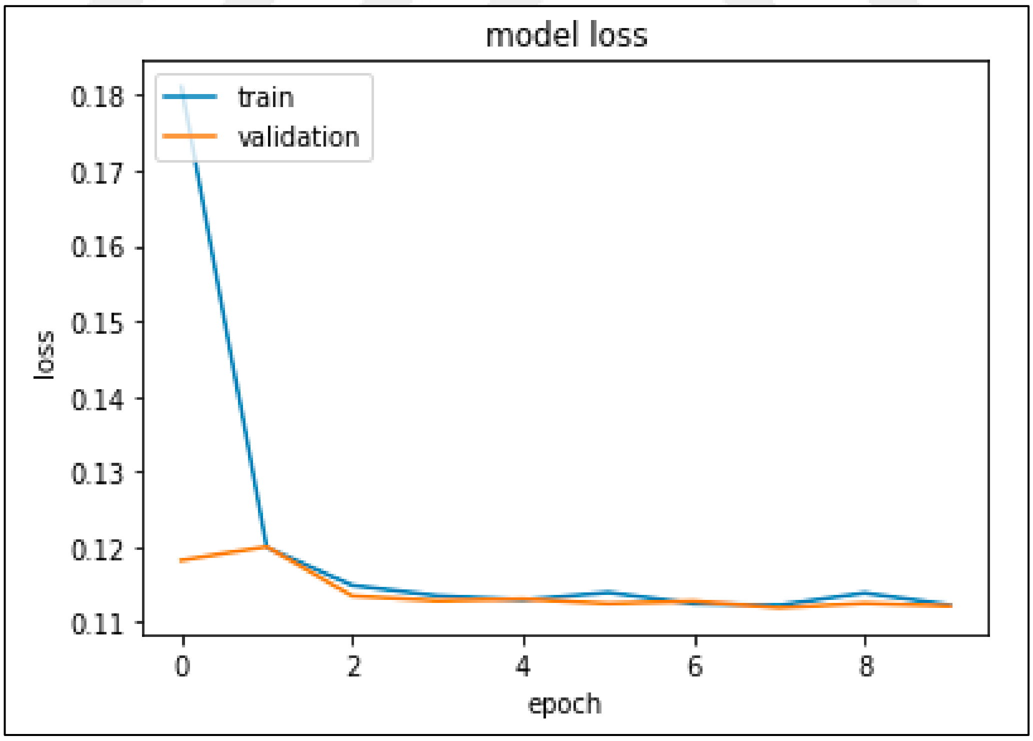


Figure 4.35: Model Performance (Loss) of Convolutional Neural Network (CNN).

Results Summary for all Deep Learning Models:

Table 4.2: Result Summary of Deep Learning Algorithms.

Model	Precision	Recall	F1-Score	Accuracy
Multi-Layer Neural Network Model	0.961	0.96	0.966	0.961
LSTM Model	0.97	0.96	0.962	0.965
Bi-LSTM Model	0.97	0.964	0.961	0.963
CNN Model	0.965	0.967	0.969	0.969

Classification Evaluation Metrics:

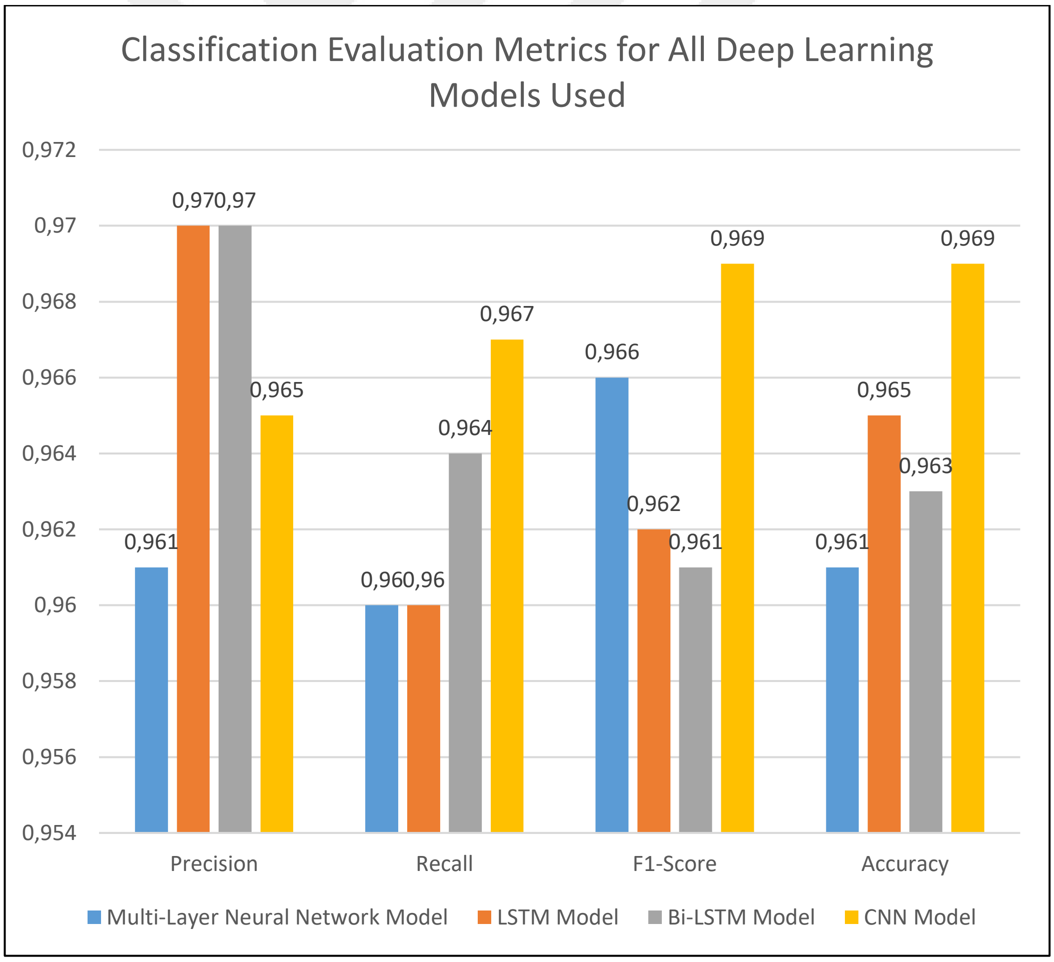


Figure 4.36: Classification Evaluation Metrics for All Deep Learning Models Used.

4.3 COMPARISON FOR THE RESULT OF CLASSIFIER

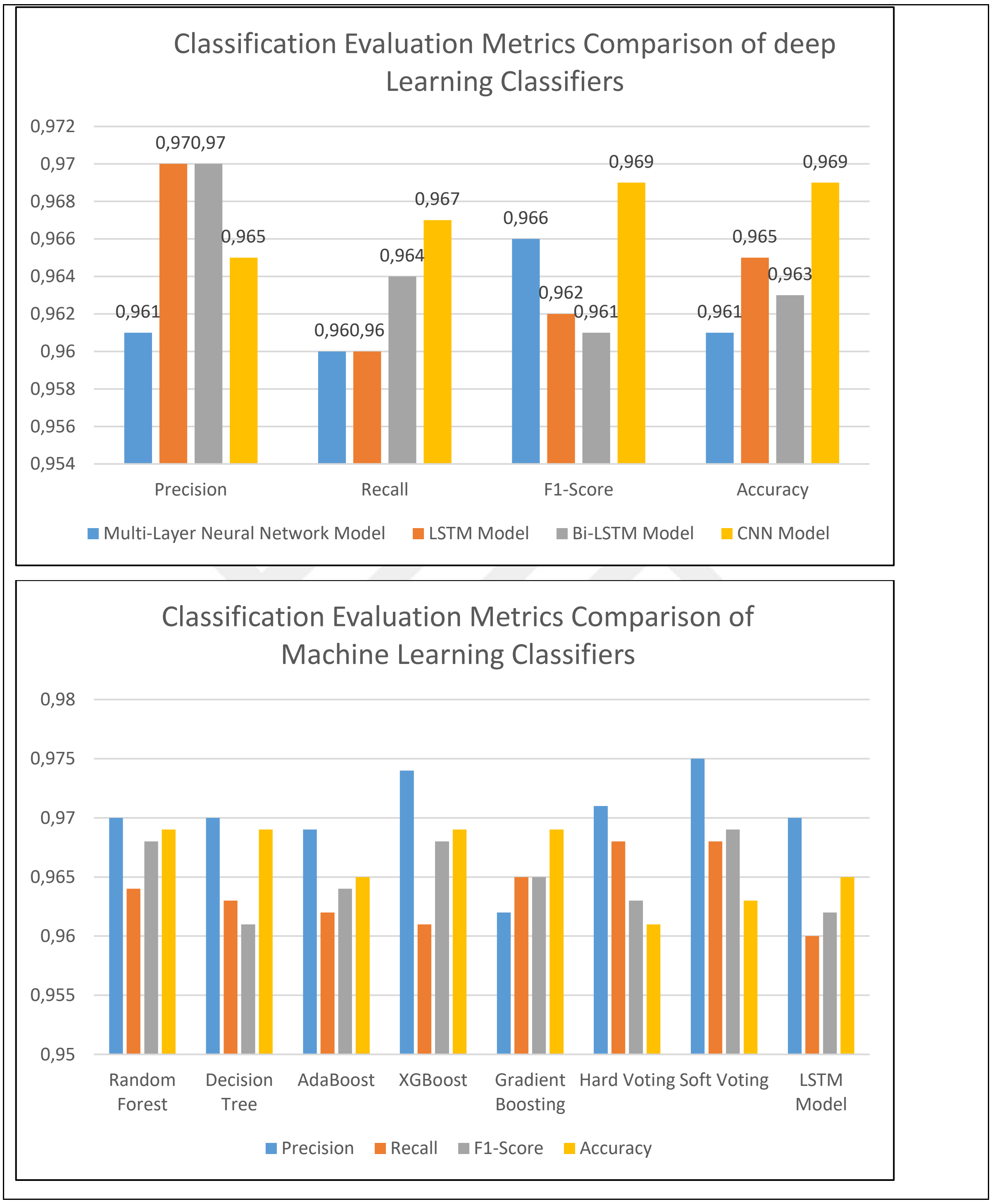


Figure 4.37: Classification Evaluation Metrics Comparison of Machine Learning and Deep Learning Classifiers.

4.4 SUMMARY

In this chapter, we examined and talked about a machine learning like (Random Forest, Decision Tree, Ada Boost, XG Boost, Gradient Boosting , Hard and soft voiting) and deep learning classifier like (NN, LSTM, BI-LSTM and CNN), for The data of three different smart home IOT devise In addition ,We used four performance metric like this (accuracy, Recall, f1-score, precision) and because that the number data (malicious and benign) are imbalance we take f1-score as the major metric for our algorithms and found that all of our algorithms achieve the best result which is (96.9%) for accuracy and f1-score , This section allowed us to compute the efficacy criteria evaluations for each classifier that we used.

So its good if we compare it with other result achieved by other researcher as below :

a. Malware detection by using MADAM detector which is based on multilevel anomaly detector achieved 96 % for accuracy.
b. Using android analysis for detection this inspection using two unrelated intent objects and the accuracy was about 91 %.
c. There is study by Bhatia and kaushal on 2017 its based on dynamic analysis for android malware detection its achieved more than 80 % for accuracy.
d. Another study by liang et al. , on 2017 based on end to end detection using call structure it achieved 93 % for accuracy.
e. Liu et al , on 2016 did detection based on hybrid method for malware detection and it achieved 93.33 % for accuracy.

5. CONCLUSION AND FUTURE WORK

5.1 CONCLUSION

This work has supplied an assessment of the effectiveness of machine learning and deep learning algorithms by get result and comparing two different major ways like (RandomForest , DecisionTree, AdaBoost, XGBoost, GradientBoosting , NN, CNN, LSTM,BI-LSTM). These algorithms tested by using four performance metrics: (accuracy, Re call, F1-scor and precision).

The results of our simulation are fairly convincing, so the conclusions that follow can be:
Because the number data (malicious and benign) which is imbalance we take f1-score as the major metrics and found that all of our algorithms achieve good and excellent result.
We take the raw data and also used the reverse engineering and choosing the right algorithm and make the suitable tuning.

5.2 FUTURE WORK

The next step is to collect additional samples for every type of it to develop malware detection by machine and deep learning algorithms , it should focus on to separate the low-level system details that are used for various analyses from the high-level information that is communicated to the user.

The capability to recognize sophisticated malware attacks, such as Zero-day attacks, requires additional development. Development of permission-based Behavior-based analysis is also possible.

We should demonstrate that the use of deep neural networks for static malware detection is viable and has potential for further improvement Malware is increasingly posing a serious security threat to computer systems. It is essential to analyze the behavior of malware and categorize samples so that robust programs to prevent malware attacks can be developed.

Towards this endeavor, we have proposed a deep convolutional neural network (CNN) architecture for malware classification.

We train a CNN for classification. Experimental results on two benchmark malware classification datasets shows the effectiveness of our proposed method.

REFERENCES

[1] A. A. Al-khatib, W. A. Hammood, "Mobile malware and defending systems: Comparison study," International Journal of Electronics and Information Engineering, vol. 6, no. 2, pp. 116–123, 2017.

[2] S. Islam, H. Ali, A. Habib, N. Nobi, M. Alam, and D. Hossain, "Threat minimization by design and deployment of secured networking model," International Journal of Electronics and Information Engineering, vol. 8, no. 2, pp. 135–144, 2018.

[3] Symantec, Internet Security Threat Report, vol.23, 2018. (https://www.symantec.com/content/dam/ symantec/docs/reports/istr-23-2018-en.pdf).

[4] Zhang, Xiaoliang & Wu, Kehe & Chen, Zuge & Zhang, Chenyi. (2021). MalCaps: A Capsule Network Based Model for the Malware Classification. Processes. 9. 929. 10.3390/pr9060929.

[5] Park Mookyu, Oh Haengrok, L. K. (2020). Security risk measurement for information leakage in iot-based smart homes from a situational awareness perspective. In Sensors (Basel, Switzerland), volume 19, page 2148.

[6] Hopping, C. (2014). Hp: 70% of internet of things devices vulnerable to at- tack. https://www.itpro.co.uk/security/22804/hp-70-of-internet-of-things-devices-vulnerable-to-attack.

[7] Assal, H., Chiasson, S., Zhang-Kennedy, L., Rocheleau, J., Mohamed, R., and Baig, K. (2018). The aftermath of a crypto-ransomware attack at a large academic institution.

[8] (2018) Malware. [Online]. Available: https://en.wikipedia.org/wiki/Malware

[9] SANS. Ouch! What is malware. [Online]. Available: https://bit.ly/ 2N1EAUG

[10] (2017) Cryptolocker. [Online]. Available: https://en.wikipedia.org/wiki/ CryptoLocker

[11] N. Milosevi ˇ c, "History of malware," 2014. [Online].

[12] Alenezi, Mohammed & Alabdulrazzaq, Haneen & Alshaher, Abdullah & Alkharang, Mubarak. (2020). Evolution of Malware Threats and Techniques: A Review. International Journal of Communication Networks and Information Security. 12. 326. 10.17762/ijcnis.v12i3.4723.

[13] M. Gaudesi, A. Marcelli, E. Sanchez, G. Squillero, and A. Tonda, “Challenging anti-virus through evolutionary malware obfuscation,” in Lecture Notes in Computer Science. Springer, vol. 9598, pp. 149–162, 2016.

[14] F. Cohen, “Computer viruses: theory and experiments,” Computers & security, vol. 6, no. 1, pp. 22–35, 1987.

[15] Weisburd, D., Jonathan, T., & Perry, S. (2009). The Israeli model for policing terrorism: Goals, strategies, and open questions. Criminal Justice and behavior, 36(12), 1259-1278.

[16] F-Secure, “Winvir,” Link, [Online; accessed 01-May-2020]. [Online]

[17] F. Touchette, “The evolution of malware,” Network Security, vol. 2016, no. 1, pp. 11 – 14, 2016.

[18] S. Embleton, S. Sparks, and C. C. Zou, “Smm rootkit: a new breed of os independent malware,” Security and Communication Networks, vol. 6, no. 12, pp. 1590–1605, 2013.

[19] Monnappa, K. Learning Malware Analysis: Explore the concepts, tools, and techniques to analyze and investigate Windows malware. Packt Publishing Ltd, 2018.

[20] Nakamoto, S., & Bitcoin, A. (2008). A peer-to-peer electronic cash system. Bitcoin.– URL: https://bitcoin. org/bitcoin. pdf, 4(2).

[21] Rhee, J., Riley, R., Xu, D., & Jiang, X. (2009, March). Defeating dynamic data kernel rootkit attacks via vmm-based guest-transparent monitoring. In 2009 international conference on availability, reliability and security (pp. 74-81). IEEE.

[22] Stafford, T. F., & Urbaczewski, A. (2004). Spyware: The ghost in the machine. The Communications of the Association for Information Systems, 14(1), 49.

[23] Kiltz, S., Lang, A., & Dittmann, J. (2007). Malware: specialized trojan horse. In Cyber Warfare and Cyber Terrorism (pp. 154-160). IGI Global.

[24] Aycock, J. (2006). Computer viruses and malware (Vol. 22). Springer Science & Business Media.

[25] Christodorescu, M., Jha, S., Maughan, D., Song, D., & Wang, C. (Eds.). (2007). Malware detection (Vol. 27). Springer Science & Business Media.

[26] Bachwani, Rekha. (2015). Better Malware Ground Truth: Techniques for Weighting Anti-Virus Vendor Labels.

[27] A. Thabet , Stuxnet malware analysis paper," Freelancer Malware Re- searcher Report, 2011.

[28] Z. Dehlawi and N. Abokhodair, \Saudi Arabia's response to cyber conict : A case study of the Shamoon malware incident," in IEEE International Conference on Intelligence and Security Informatics (ISI), 2013, pp. 73-75.

[29] Nelson, N. (2016). The impact of dragonfly malware on industrial control systems. SANS Institute.

[30] Langill, J. T. (2014). Defending against the dragonfly cyber security attacks. Retrieved, 11, 2015.

[31] Aslan, Ö. A., & Samet, R. (2020). A comprehensive review on malware detection approaches. IEEE Access, 8, 6249-6271.

[32] Y. Alosefer, ``Analysing Web-based malware behavior through client honeypots," Ph.D. dissertation, School Compute. Sci. Inform., Cardiff Univ., Cardiff, U.K., 2012.

[33] N. Idika and P. Mathur, ``A survey of malware detection techniques," Purdue Univ., West Lafayette, IN, USA, Tech. Rep., vol. 48, 2007.

[34] E. Eilam, Reversing: Secrets of Reverse Engineering. Hoboken, NJ, USA: Wiley, 2011.

[35] Aslan, Ömer & Samet, Refik. (2020). A Comprehensive Review on Malware Detection Approaches. IEEE Access. 8. 1-1. 10.1109/ACCESS.2019.2963724.

[36] A. Souri and R. Hosseini, ``A state-of-the-art survey of malware detection approaches using data mining techniques," Hum.-Centric Comput. Inf. Sci., vol. 8, no. 1, p. 3, 2018.

[37] E. Gandotra, D. Bansal, and S. Sofat, ``Malware analysis and classi_cation:A survey," J. Inf. Secur., vol. 5, no. 2, pp. 56_64, 2014.

[38] Namanya, A. P., Cullen, A., Awan, I. U., & Disso, J. P. (2018, August). The world of malware: An overview. In 2018 IEEE 6th International Conference on Future Internet of Things and Cloud (FiCloud) (pp. 420-427). IEEE.

[39] Maggi, F., Bellini, A., Salvaneschi, G., & Zanero, S. (2011). Finding non-trivial malware naming inconsistencies. In Information Systems Security: 7th International Conference, ICISS 2011, Kolkata, India, December 15-19, 2011, Procedings 7 (pp. 144-159). Springer Berlin Heidelberg.

[40] Belaoued, Mohamed & Derhab, Abdelouahid & Mazouzi, Smaine & Khan, Farrukh. (2020). MACoMal : A Multi-Agent based Collaborative Mechanism for Anti-Malware Assistance. IEEE Access. PP. 1-1. 10.1109/ACCESS.2020.2966321.

[41] McGraw, G., & Morrisett, G. (2000). Attacking malicious code: A report to the infosec research council. IEEE software, 17(5), 33-41.

[42] VELUZ, D. (2010). Malware Gets Smart with Vodafone Smartphone. http://blog. trendmicro. com/malware-gets-smart-with-vodafone-smartphone.

[43] Ellis, D. R., Aiken, J. G., Attwood, K. S., & Tenaglia, S. D. (2004, October). A behavioral approach to worm detection. In Proceedings of the 2004 ACM workshop on Rapid malcode (pp. 43-53).

[44] Dynamic analyses of malware Carlin, D. (Author). Dec 2018.

[45] Szydlowski, M., Egele, M., Kruegel, C., & Vigna, G. (2012). Challenges for dynamic analysis of ios applications. In Open Problems in Network Security: IFIP WG 11.4 International Workshop, iNetSec 2011, Lucerne, Switzerland, June 9, 2011, Revised Selected Papers (pp. 65-77). Springer Berlin Heidelberg.

[46] Griffin, K., Schneider, S., Hu, X., & Chiueh, T. C. (2009, September). Automatic Generation of String Signatures for Malware Detection. In RAID (Vol. 5758, pp. 101-120).

[47] Idika, N., & Mathur, A. P. (2007). A survey of malware detection techniques. Purdue University, 48(2), 32-46.

[48] Sikorski, M., & Honig, A. (2012). Practical malware analysis: the hands-on guide to dissecting malicious software. no starch press.

[49] Namanya, A. P., Pagna-Disso, J., & Awan, I. (2015, September). Evaluation of automated static analysis tools for malware detection in Portable Executable files. In 31st UK Performance Engineering Workshop (Vol. 17, p. 81).

[50] Willems, C., Holz, T., & Freiling, F. (2007). Toward automated dynamic malware analysis using cwsandbox. IEEE Security & Privacy, 5(2), 32-39.

[51] Ireland, N. (2017). Dynamic Analyses of Malware.

[52] Ligh, M., Adair, S., Hartstein, B., & Richard, M. (2010). Malware analyst's cookbook and DVD: tools and techniques for fighting malicious code. Wiley Publishing.

[53] Bulazel, A., & Yener, B. (2017, November). A survey on automated dynamic malware analysis evasion and counter-evasion: Pc, mobile, and web. In Proceedings of the 1st Reversing and Offensive-oriented Trends Symposium (pp. 1-21).

[54] Roundy, K. A., & Miller, B. P. (2010). Hybrid analysis and control of malware. In Recent Advances in Intrusion Detection: 13th International Symposium, RAID 2010, Ottawa, Ontario, Canada, September 15-17, 2010. Proceedings 13 (pp. 317-338). Springer Berlin Heidelberg.

[55] Schultz, M. G., Eskin, E., Zadok, F., & Stolfo, S. J. (2000, May). Data mining methods for detection of new malicious executables. In Proceedings 2001 IEEE Symposium on Security and Privacy. S&P 2001 (pp. 38-49). IEEE.

[56] Kolter, J. Z., & Maloof, M. A. (2004, August). Learning to detect malicious executables in the wild. In Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining (pp. 470-478).

[57] Moskovitch, R., Elovici, Y., & Rokach, L. (2008). Detection of unknown computer worms based on behavioral classification of the host. Computational Statistics & Data Analysis, 52(9), 4544-4566.

[58] O'Kane, P., Sezer, S., & McLaughlin, K. (2014, January). N-gram density based malware detection. In 2014 World Symposium on Computer Applications & Research (WSCAR) (pp. 1-6). IEEE.

[59] O'Kane, P., Sezer, S., McLaughlin, K., & Im, E. G. (2013). SVM training phase reduction using dataset feature filtering for malware detection. IEEE transactions on information forensics and security, 8(3), 500-509.

[60] Islam, M. S., & Ahmed, R. (2011). Land use change prediction in Dhaka city using GIS aided Markov chain modeling. Journal of Life and Earth Science, 6, 81-89.

[61] Runwal, N., Low, R. M., & Stamp, M. (2012). Opcode graph similarity and metamorphic detection. Journal in computer virology, 8, 37-52.

[62] Bilar, D. (2007). Opcodes as predictor for malware. International journal of electronic security and digital forensics, 1(2), 156-168.

[63] Santos, I., Brezo, F., Sanz, B., Laorden, C., & Bringas, P. G. (2011). Using opcode sequences in single-class learning to detect unknown malware. IET information security, 5(4), 220-227.

[64] Shabtai, A., Moskovitch, R., Feher, C., Dolev, S., & Elovici, Y. (2012). Detecting unknown malicious code by applying classification techniques on opcode patterns. Security Informatics, 1(1), 1-22.

[65] Han, K., Kang, B., & Im, E. G. (2014). Malware analysis using visualized image matrices. The Scientific World Journal, 2014.

[66] Nappa, A., Rafique, M. Z., & Caballero, J. (2013). Driving in the cloud: An analysis of drive-by download operations and abuse reporting. In Detection of Intrusions and Malware, and Vulnerability Assessment: 10th International Conference, DIMVA 2013, Berlin, Germany, July 18-19, 2013. Proceedings 10 (pp. 1-20). Springer Berlin Heidelberg.

[67] Japkowicz, N., & Stephen, S. (2002). The class imbalance problem: A systematic study. Intelligent data analysis, 6(5), 429-449.

[68] Park, M. J., Choi, M. G., Shinagawa, Y., & Shin, S. Y. (2006). Video-guided motion synthesis using example motions. ACM Transactions on Graphics (TOG), 25(4), 1327-1359.

[69] Raskutti, B., & Kowalczyk, A. (2004). Extreme re-balancing for SVMs: a case study. ACM Sigkdd Explorations Newsletter, 6(1), 60-69.

[70] Akbani, R., Kwek, S., & Japkowicz, N. (2004). Applying support vector machines to imbalanced datasets. In Machine Learning: ECML 2004: 15th European Conference on Machine Learning, Pisa, Italy, September 20-24, 2004. Proceedings 15 (pp. 39-50). Springer Berlin Heidelberg.

[71] Santos, I., Brezo, F., Sanz, B., Laorden, C., & Bringas, P. G. (2011). Using opcode sequences in single-class learning to detect unknown malware. IET information security, 5(4), 220-227.

[72] Menahem, E., Shabtai, A., Rokach, L., & Elovici, Y. (2009). Improving malware detection by applying multi-inducer ensemble. Computational Statistics & Data Analysis, 53(4), 1483-1494.

[73] Japkowicz, N., & Stephen, S. (2002). The class imbalance problem: A systematic study. Intelligent data analysis, 6(5), 429-449.

[74] Kruegel, C., & Toth, T. (2003, September). Using decision trees to improve signature-based intrusion detection. In International workshop on recent advances in intrusion detection (pp. 173-191). Berlin, Heidelberg: Springer Berlin Heidelberg.

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

[76] Kumar, R., & Geetha, S. (2020). Malware classification using XGboost-Gradient boosted decision tree. Adv. Sci. Technol. Eng. Syst, 5, 536-549.

[77] Pham, H. D., Le, T. D., & Vu, T. N. (2018). Static PE malware detection using gradient boosting decision trees algorithm. In Future Data and Security Engineering: 5th International Conference, FDSE 2018, Ho Chi Minh City, Vietnam, November 28–30, 2018, Proceedings 5 (pp. 228-236). Springer International Publishing.

[78] Deng, Haowen & Zhou, Youyou & Wang, Lin & Zhang, Cheng. (2021). Ensemble learning for the early prediction of neonatal jaundice with genetic features. BMC Medical Informatics and Decision Making. 21. 10.1186/s12911-021-01701-9.

[79] de Oliveira, G. P., Fonseca, A., & Rodrigues, P. C. (2022). Diabetes diagnosis based on hard and soft voting classifiers combining statistical learning models. Brazilian Journal of Biometrics, 40(4), 415-427.

[80] Hussain, S., Mokhtar, M., & Howe, J. M. (2014). Sensor failure detection, identification, and accommodation using fully connected cascade neural network. IEEE Transactions on Industrial Electronics, 62(3), 1683-1692.

[81] Pelletier, Charlotte & Webb, Geoffrey & Petitjean, François. (2019). Temporal Convolutional Neural Network for the Classification of Satellite Image Time Series. Remote Sensing. 11. 523. 10.3390/rs11050523.

[82] Sai Charan, P. V., Gireesh Kumar, T., & Mohan Anand, P. (2019). Advance persistent threat detection using long short term memory (LSTM) neural networks. In Emerging Technologies in Computer Engineering: Microservices in Big Data Analytics: Second International Conference, ICETCE 2019, Jaipur, India, February 1–2, 2019, Revised Selected Papers 2 (pp. 45-54). Springer Singapore.

[83] Tran, T. K., Sato, H., & Kubo, M. (2019, November). Image-based unknown malware classification with few-shot learning models. In 2019 Seventh International Symposium on Computing and Networking Workshops (CANDARW) (pp. 401-407). IEEE.

[84] Le, Xuan Hien & Ho, Hung & Lee, Giha & Jung, Sungho. (2019). Application of Long Short-Term Memory (LSTM) Neural Network for Flood Forecasting. Water. 11. 1387. 10.3390/w11071387.

[85] Asam, M., Khan, S. H., Akbar, A., Bibi, S., Jamal, T., Khan, A., ... & Bhutta, M. R. (2022). IoT malware detection architecture using a novel channel boosted and squeezed CNN. Scientific Reports, 12(1), 15498.

[86] Phung, & Rhee,. (2019). A High-Accuracy Model Average Ensemble of Convolutional Neural Networks for Classification of Cloud Image Patches on Small Datasets. Applied Sciences. 9. 4500. 10.3390/app9214500.