POLITECNICO DI TORINO

DEPARTMENT OF CONTROL AND COMPUTER ENGINEERING (DAUIN)

Master Degree in Computer Engineering

Master Degree Thesis

Enhancing Malware Classification Through LSTM Algorithm Integration in Binary Classification Models

Author: Andrea Toscano

Advisor: Alessandro Savino
Co-Advisor(s): Nicoló Maunero

February, 2024

Abstract

The latest trends in cybercrime show increasing damage to society and companies' economic assets by cyber attacks, most often conducted with the help of extremely versatile and effective attack tools called malware. Countermeasures taken to limit the impact of this threat often prove to be invalid, or at least can be circumvented through phishing techniques. This is why tireless research is needed to limit the advancement of such significant damage and such malicious techniques.

The thesis explores the most widely used techniques for recognizing malicious programs through classification: the ability, from an unknown file, to recognize its characteristics and assign it an identifying label, in order to distinguish malware from safe programs. In addition, the purpose of this work, aims at the application of a type of Machine Learning to the field of malware classification, evaluating benefits and performance obtained from the help of Recurrent Neural Networks.

Given the huge amount of malware found on a daily basis, another aspect that was given importance in the work done was the need to automate as much as possible the chain of extraction, behavioral analysis, and analysis by machine learning. The first proposed step sees an in-depth study regarding the most widely used techniques in the scientific literature for malware analysis, diversifying them according to the context in which the sample is studied: in fact, the main study methods used in static and dynamic analysis will be presented.

Since the need to automate the process is a crucial part of the work, useful tools for automating the discovery of malicious samples and the extraction of key features from them will also be presented.

The core of this research is based on the applicability of a Machine Learning algorithm, called Long Short-Term Memory, in analyzing certain features extracted from malware, which are essential based on their sequentiality. In fact, the algorithm will learn to recognize a malicious sample based on the sequential usage of system APIs that the Operating System provide to the running program.

Starting with the composition of a data collection containing, one by one, the first 100 APIs called by each malware extracted from an automated type of analysis, the main settings necessary for the above algorithm to achieve the best possible performance will be identified. Therefore, the choices provided in the work performed, will be based on the objectivity of the tests performed with different parameters to evaluate its efficiency. Finally, a proof-of-concept and useful hints will be provided in order to facilitate the future work of those who wish to continue the type of research proposed.

Contents

List of Tables

List of Figures

Chapter 1

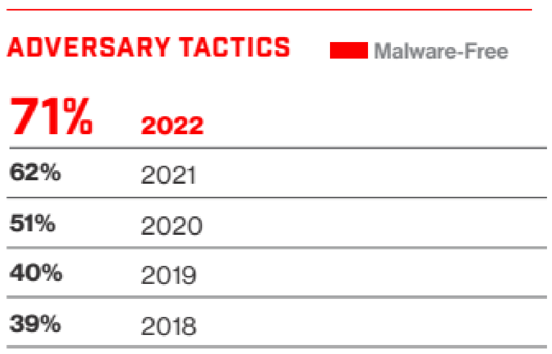
Introduction

In a world racing toward digital innovation, more and more pieces are being devised every day to make up the new model society of the future. Rich in benefits and accessibility, but equally rich in risks and threats. In an ecosystem where information becomes the new currency of exchange, the possibilities of being able to break through and obtain that information become more and more appealing. The growing trend of the phenomenon is also represented by the enormous exchange of this resource.

From 2010 to nowadays, the amount of data produced each year is increased by 9000% [65, Volume of Data Created]. While data is growing with such speed, the supply of IT-related jobs is not growing at the same rate, it then becomes much easier to be able to exploit distractions or flaws in a less controlled system in order to gain illicit access.

The work proposed in this Thesis aims to apply one of the greatest technological researches of our decade to simplify the role of defense in one of the most consistent and frequent threats of the cyber era: the malware.

In the fog of all the possible threats that can materialize in a system, one of the most feared is the category of malware, a malicious software that may be used by an attacker for multiple purpose: Spying, Blackmailing, Stopping a Service, Political or Financial Purposes, and so on.



ADVERSARY TACTICS 🟥 Malware-Free

71%	2022
62%	2021
51%	2020
40%	2019
39%	2018

Figure 1.1. Percentage of Malware-Free Attacks for each year [37]

Nowadays cyberspace represents a new battleground, rising geopolitical tensions and humanitarian disasters often result in an increase in cyber attacks and malware incidents. In 2020, malware attacks increased by 358% compared to 2019, and in 2021 raised by another 125% [37, Latest Cyber Crime Statistics]. Despite the incredible increase in malware threats, The CrowdStrike 2023 Global Threat Report shows that more and more attackers are deciding to engage in malware-free attacks (Figure 1.1) [19].

That is not why the cyberdefense side of conducting autopsies on new forms of malware can take a back seat. An increase in malware-free attack rates has a basis in a significant increase in attacks of another nature, so it cannot be proof that attacks conducted by malware have decreased. As an illustration of the phenomenon of cyberattacks as a new battleground of modern wars, one can look at the last two events of international significance: the beginning of the Ukraine-Russia war and the deflagration of the Israel-Palestine conflict.

> *"In the third quarter of 2022, Europe was dragged into a high-intensity hybrid cyber-war at a turning point in the conflict, with a massive wave of DDoS attacks, particularly in the Nordic and Baltic countries and Eastern Europe. Cyber is now a* ***crucial weapon in the arsenal of new instruments of war****, alongside disinformation, manipulation of public opinion, economic warfare, sabotage and guerrilla tactics [27, Pierre-Yves Jolivet, VP Cyber Solutions, Thales.]."*

Indeed, in the wake of the tragic events, Europe's network, cloud, and information systems infrastructure has reported continuous attacks, which in some cases do not seem to be remedied. This is where the branch of algorithms that has reported the fastest growth in performance over the past twenty years may come into play: Machine Learning.

As reported in [32], this technology grew by doubling its capacity each year, following Moore's law, until 2012. From 2012 onward, the rate of improvement has increased dramatically. The average growth was by a factor of 3.4 times its previous capacity, and not annually, but quarterly (see Figure 1.2).

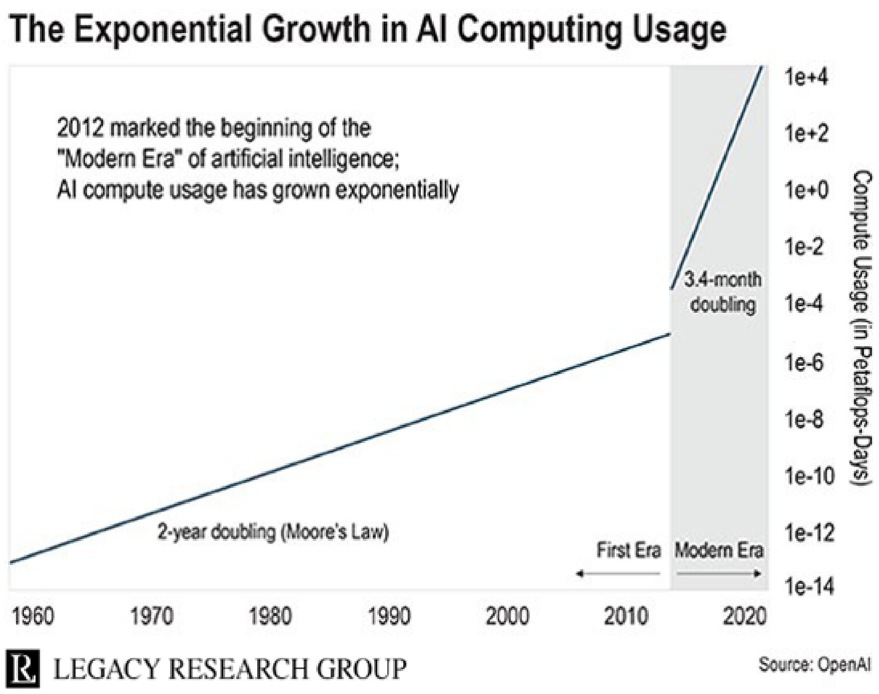


Figure 1.2. Machine Learning growth over the years [32]

This thesis will present a targeted work to address the challenge of building and testing a model, known in machine learning as a Recurrent Neural Network, capable of recognizing the degree of maliciousness of any file solely by judging its behavior.

The remainder of the document is organized as follows. In Chapter 2, a brief mention regarding all the knowledge needed to understand the state of the art and the techniques used, starting from the definition of malware, proceeding to machine learning basic knowledge. Chapter 3 will present the state of the art, including the most commonly used analysis protocols and techniques in malware analysis. Chapter 4 will describe in details how the test was conducted, from the starting point of the information gathering until the last block used to compose the model. Chapter 5 the results will be illustrated to present the reader with metrics and performance obtained so that conclusions can be drawn regarding its effectiveness. Finally, Chapter 6 draws the conclusions and will present some possible future works.

Chapter 2

Background

Malware is any software intentionally designed to cause disruption to a computer, server, client, or computer network, leak private information, gain unauthorized access to information or systems, deprive access to information, or which unknowingly interferes with the user's computer security and privacy [76]. *"In the past three decades almost everything has changed in the field of malware and malware analysis. From malware created as a proof of some security concept and malware created for financial gain to malware created to sabotage infrastructure."* [47].

2.1 What is a Malware

When the Internet was not yet as advanced and complex as it is today, it was still called ARPANET, which stands for The Advanced Research Projects Agency Network. It was a project initiated by the United States Department of Defense and Advanced Research Project Agency (ARPA), with the aim to enable resource sharing between remote computers [43]. The creation of such a system provided the basis for the genesis of experiments aimed at testing its destruction.

It was not many years before "Creeper", the first malware ever created, began to operate. Creeper copied itself from one computer to the next, after which it erased itself from the first infected computer so that it could move between systems [66]. Although it was not created with malicious intent, it was able to demonstrate to the founders of ARPANET the importance of devoting sufficient attention to the security measures to be taken to prevent similar situation from occurring. Creeper originated in the early 1970s, but with the advent of the Internet it was only that the beginning. At that time, no one could have guessed that this type of malware would later be referred to as "worm", unwanted software with the ability to replicate itself independently within a network. Subsequently, came increasingly complex patterns of malicious software: suffice it to say that in the 1990s most viruses used email as an attack vector, Windows as the main target, and possessed the characteristic of polymorphism, that is, the ability to evade the defense of the most popular antivirus by changing its very form.

Later, new Worm models were created, but this time with malicious intent. In 2000, a 24-year-old resident of the Philippines, unable to afford Internet service, built a worm based on a macro command system to steal the credentials needed to access the service De Guzman (the worm creator), called it ILOVEYOU. This was the first case of outlaw malware, a perfect example of phishing and social engineering. The Attack Vector was in fact an email attachment representing a fake love letter [67].

It is not news that the history of malware is dictated by a crescendo of economic damage: the ensuing two decades will be disastrous precisely because of the enormous impact malware will have on businesses. One of the most malicious kind of malware that impacts on companies is Ransomware (from Ransom and malware). It exploits the infection to encrypt valuable resources on the infected system to ask for a ransom, usually paid over cryptocurrency to make payments impossible to trace.

In 2013 the first type of Ransomware was released: CryptoLocker. Known for its rapid and powerful asymmetric encryption capabilities [67]. That time, the requested ransom was only of 2 bitcoins, of around $745.

Although defenses to contain new ransomware have become more efficient over the years, between 2019 and 2022 this type of malware represents one of the main daily threats that security experts have to deal with every day. Suffice it to say that in 2021 the leading U.S. gasoline and jet fuel supplier, Colonial Pipeline, was attacked by ransomware that blocked both public and private service for several days. The incident was so severe that it convinced the current President of the United States, Joe Biden, to declare a state of emergency.

According to Health and Human Services Cyber Security Program, “The average bill for rectifying a ransomware attack – considering downtime, people time, device cost, network cost, lost opportunity, ransom paid, etc. – was $1.27 million” [33].

2.1.1 Malware Classification

Malware, or malicious software, includes a variety of digital threats that compromise the security and functionality of computer systems. Their classification is determined in the distinct ways they behave and the objectives they pursue. One primary category is viruses, which embed themselves within legitimate programs and replicate when those programs are executed. The first distinction that can be demarcated between the various categories of malware is the execution environment, as a key factor necessary for the functionality of much malware is the need for the infected system to be connected to a network.

Viruses can spread from one system to another, causing widespread damage. Worms, on the other hand, are self-replicating entities that exploit vulnerabilities in networks to propagate. They don’t require user interaction to spread, making them particularly powerful in rapidly infecting multiple systems. Trojans, named after the ancient Greek tale of the wooden horse, disguise themselves as benign software but, once installed, grant unauthorized access to attackers. Unlike viruses and worms, Trojans rely on social engineering to deceive users into willingly installing them. Ransomware represents a more direct and financially motivated threat. This type of malware encrypts files on a victim’s system and demands payment, often in cryptocurrency, for the decryption key. Failure to comply can result in permanent data loss, making ransomware a significant concern for individuals and organizations alike. Spyware operates covertly, infiltrating systems to monitor and collect user information without their knowledge. This type of malware poses a substantial threat to privacy, as it can capture sensitive data, such as login credentials and personal details, leading to identity theft or unauthorized access to confidential information.

Understanding these classifications (see Table 2.1) is crucial for developing effective cybersecurity strategies. By recognizing the diverse tactics employed by malware, individuals and organizations can better protect their systems through proactive measures, including robust antivirus programs, regular software updates, and user education on safe online practices.

Malware Purpose	**Description**	**Examples**
Spyware	Gathers user information.	Keyloggers, Spyware Trojans
Adware	Displays unwanted ads.	Pop-up ads, Invasive banners
Ransomware	Encrypts files, demands ransom.	WannaCry, Locky, CryptoLocker
Virus	Infects and spreads.	Melissa, Sasser, CIH
Worm	Spreads autonomously.	Conficker, Slammer, Mydoom
Trojan Horse	Pretends to be legitimate.	Zeus, SpyEye, Backdoor.Rustock
Botnet	Controls remote computers.	Zeus Botnet, Mirai, Conficker
Rootkit	Conceals presence in the system.	Stuxnet, Sony Rootkit, HackerDefender

Table 2.1. Malware Classes Description

Another risk comes from the exponential growth of infected devices that remain connected to a network: individual machines, referred to as bots, that remotely commanded form a botnet. *"With a botnet on board, your device is no longer just a victim, but also a perpetrator of cyber crime. The botnet receives commands and executes them without any intervention or control from you. The personal data stored on your PC or smartphone isn't safe either"* [36].
Experts estimate that approximately **16% to 25%** of all the computers connected to the Internet are members of botnets [63].

2.1.2 Why Malware are Created

There can be multiple motivations behind the creation of malware: since usually behind the programming of a tool that can have a huge impact are people and groups with high capabilities from every branch of IT, one can easily assume that the targets of these attacks are entities with extremely sensitive data or companies with substantial wealth. The truth is that this is not a correct assumption, because to understand the targets, it is necessary to understand the motivation.

We can resume the most common motivations in the following list:

- **Financial Gain**: creating malware to steal valuable information like credit card details or to demand ransom payments.
- **Espionage and Cyber Warfare**: developing malware for spying on rivals or disrupting the operations of other nations.
- **Hacktivism**: using malware to further political or ideological causes by disrupting services or stealing and leaking sensitive information.
- **Information Theft**: creating malware to steal trade secrets, intellectual property, or other valuable data.
- **Botnets and DDoS Attacks**: using malware to create networks of compromised computers for various malicious activities, including overwhelming websites with traffic.
- **Cyber Sabotage**: creating and deploying malware to harm specific organizations or critical infrastructure.
- **Personal Vendettas**: occasionally, individuals may create malware out of personal animosity towards specific targets.

Wanting to focus on the most advanced forms of malware, it is worth addressing one case, codename **Stuxnet**, known to the world to be one of the most advanced forms of Cyber Warfare.

The following is taken from 2017 Stuxnet Report [7]. The discovery of Stuxnet occurred at a time of extreme tension between Iran and the U.S., a time when Iran announced that it wanted to produce nuclear energy, and, possibly, nuclear weapons. It happened when an antivirus company based in Belarus, VirusBlockAda, received a worm sample found in an Iranian computer that rebooted himself continually. Only years later, following numerous analyses, did people begin to understand the true nature of malware that was initially confused as spyware. Reports on the incident pointed in one direction: The malware analyzed was intended to disrupt an air-gapped SCADA-based industrial control system, and in particular, that of a Uranium enrichment plant located in Natanz. Since the target facilities are an air-gapped system (disconnected from any external network) the real strength of this attack lies in the attack vector (see Figure 2.1) and the worm's ability to replicate itself in other similar neighboring system. The attack led to the destruction of about 1,000 centrifuges used by the plant.

Therefore, the first public example of the use of malware in a conflict between nations deserves to be analyzed, primarily on the basis of the effects caused on the attacked country, which are of three types:

- **Social And Political Effect**: Iranian population felt defenseless in terms of cybersecurity measures.
- **Economic Effect**: since Iran could not rely on an international market, most of the centrifuges were by them themselves. The delay in the production led to a long-term economic repercussion.

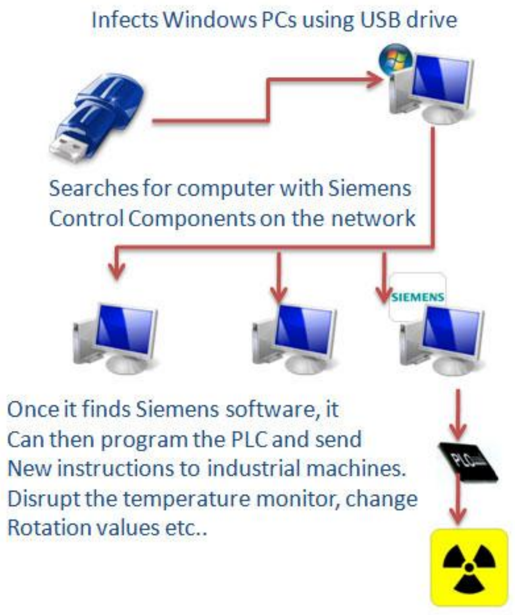

Infects Windows PCs using USB drive
Searches for computer with Siemens Control Components on the network
SIEMENS
Once it finds Siemens software, it Can then program the PLC and send New instructions to industrial machines. Disrupt the temperature monitor, change Rotation values etc..
PLC

Figure 2.1. Stuxnet Infection Mechanism [1]

- **Technological Effect**: long-term technological consequences of Stuxnet can be seen in the mistrust that Iranians people had towards technical malfunctions.
- **International Effects**: wealthiest States learned that they had to invest in specific Cybersecurity Units to increase security standard.

The incident described may be a good way to describe what impact such an attack can have in today's world. In any case, it is common for cyber criminals to have as their primary goal obtaining money as well as invalidating a system and causing harm, to such an extent that, as recounted by TrollEye Security in *The Rise of Ransomware as a Service (RaaS)* [68], the emergence of ransomware has led to the emergence of a new phenomenon: ransomware became a mercenary service, a product that could be purchased on a black market. According to *The Ransomware-as-a-Service economy within the darknet* [44] *"On the darknet markets, Ransomware-as-a-Service (RaaS) is being offered as a franchise model that allows people without programming skills to become active attackers and take part in the ransomware economy"*.

In the context of Cyber Warfare a new term has been coined to define an advanced threat that can maintain unauthorized access and are capable of conducting large-scale intrusions causing immense damage: Advanced Persistent Threat (APT).

As Cisco discussed in its article *What Is an Advanced Persistent Threat (APT)?*, *"An advanced persistent threat (APT) is a covert cyber-attack on a computer network where the attacker gains and maintains unauthorized access to the targeted network and remains undetected for a significant period. During the time between infection and remediation the hacker will often monitor, intercept, and relay information and sensitive data. The* *intention of an APT is to exfiltrate or steal data rather than cause a network outage, denial of service or infect systems with malware."* [74]. Stuxnet can be seen as an example of APT. Usually the cybercriminals behind APTs are adversaries well-funded, Information Technology Security Professionals that are able to target high-value organizations [73]. Most of the these teams are nation-state actors, and CrowdStrike has been able to track over 150 adversaries, including nation-states, eCriminals and hacktivists.

2.2 Malware Structure and Behaviour

Since malware are complex forms of code, it is not easy to explain a general pattern of operation among all existing families. Despite this, it is possible to make a classification, presented in Table 2.2, based on common components:

Malware Component	**Description/Details**
Crypter	Conceals malware existence, encrypts original binary code.
Downloader	Downloads malware from the Internet, facilitates installation.
Dropper	Camouflages malware payloads, initiates covert installation.
Exploit	Breaches system security via software vulnerabilities for malware installation or information theft.
Injector	Injects code into vulnerable processes, alters execution to resist removal.
Obfuscator	Conceals code and purpose using various techniques to evade detection.
Payload	Performs desired activities upon activation, compromising system security.
Wrapper (Packer)	Bundles files using compression techniques to create a single executable file.

Table 2.2. Malware Components Overview [14]

The structure helps to understand how behaviors are delineated, but they can also exhibit common patterns. Most of them are represented in the MITRE ATT&CK Matrix, a useful resource to investigate the most common categories in behavioural analysis of the incidents [48].

- **Persistence**: the ability of a Malware to maintain a foothold inside the target environment, using tools like backdoors. In order keep the activity on the target usually it requires to install a new entry in the Startup Application via Registry Modifications [55].

- **Monitor and Capture Data**: common in spywares, malicious programs can often work to capture sensitive informations from the victim by intercepting keystrokes, screen captures and other valuable data.

- **Privilege Escalation**: malware seeks privilege escalation to gain more control over a system, allowing it to persistently run, avoid detection, and perform malicious actions like data manipulation or spreading to other machines.
- **Command And Control Communication**: in a C&C Attack, attacker and compromised target machine can exchange data from and to specific endpoints. Also known as C2, this kind of attack is the most used when the goal is to form and control a botnet [41].
- **Lateral Movement :** Lateral movement enables the attacker to gain information and move between system inside the same environment of the victim target.

2.2.1 Obfuscation and Polymorphism

One of the first defensive maneuvers offered by antiviruses is the interception of programs by analyzing their fingerprint. The fingerprint is usually the result of an operation called Hashing, which can summarize the entire contents of a file into a string of a fixed length of characters [75]. In this way, a malicious file can be easily recognized if a form of it is always spread over the network in the same way (see Figure 2.2).

Nowadays, online services like VirusTotal offer the possibility to look for the fingerprint of a file and analyze previous feedback given to it. And this is where *Polymorphism* comes into play: by Polymorphism we mean the ability of a malware to change shape while keeping its behavior unchanged [77]. With this operation, which can be done by simple substitutions, but also by very complex patterns, a malware is able to evade the defensive measures of fingerprint-based antiviruses.

Another typology of shape-shifting malware lies in *Metamorphism*: it is an advanced technique that allows a program to self-modify its code whenever it propagates or is deployed.

In 2008, WetStone Technologies Inc. brought a speech at BlackHat Convention to talk about the evolution of Polymorphic and Metamorphic Malware to raise awareness among IT professionals about the impact this kind of threat would have in the future. According to that article, [18], the impact of these phenomenons on law enforcement would bring a slower incident response and a more difficult determination of the Attack Source.

More generally, these behaviors fall under the term **Obfuscation** which outlines a technique that makes a program more difficult to understand. Obfuscation is originally used to protect a developer's intellectual property [80]. However, one of the most widely used techniques to analyze the functionality of a malware since the beginning of this practice is static analysis. This technique makes use of a decompiler (such as Ghidra [3] or IDA [57]), which can reconstruct the source code of a program from the compiled executable.

According to "Malware Obfuscation Techniques: A Brief Survey" [80]:

- **Dead Code Insertion**: instruction lines may be inserted into the code (Instructions such as `nop`), used to code an operation without instructions to execute [60].
- **Register Reassignment**: switches registers used in machine language operations.

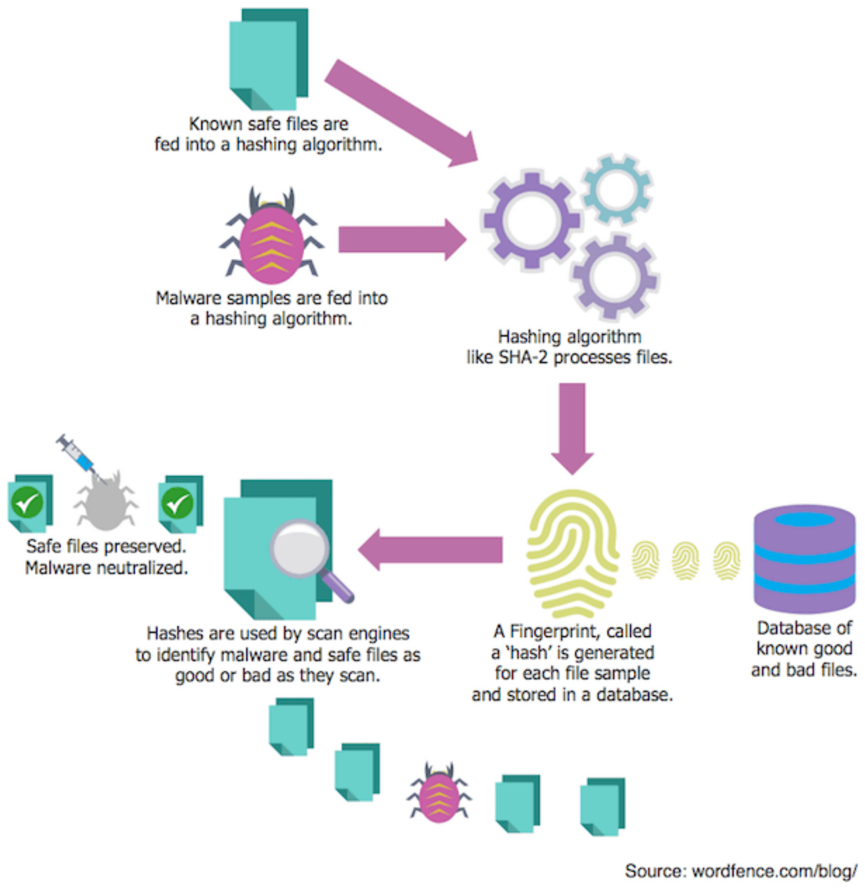

Known safe files are fed into a hashing algorithm.
Malware samples are fed into a hashing algorithm.
Hashing algorithm like SHA-2 processes files.
Safe files preserved. Malware neutralized.
Hashes are used by scan engines to identify malware and safe files as good or bad as they scan.
A Fingerprint, called a 'hash' is generated for each file sample and stored in a database.
Database of known good and bad files.
Source: wordfence.com/blog/

Figure 2.2. How Scanner Use Hasing

- **Subroutine Reordering**: it works alternating subroutines randomly.
- **Instruction Substitution**: substitutes specific instructions with equivalent ones.
- **Code Transposition**: sequence of instructions are reordered in random shuffles or exchanged only if they are independent between themselves.
- **Code Integration**: malware code is reassembled inside the code of a target program.

Another common form of obfuscation is what is done by special programs called *Packers*. A Packer *"is a program that runs on an already compiled binary, compresses it, and turns it into an executable that will decompress itself at runtime"* [64].

2.3 Malware Analysis

Malware Analysis can be seen as the art of perceive and understand intentions and risks regarding a specific Malware Threat. Because of their complexity, these analyses can be decidedly labor-intensive and time-consuming, but sometimes it is necessary to learn new ways to defend one's systems and to learn more about digital threats [25].

In terms of software analysis, this branch should be understood as a *black-box analysis*, a term used to refer to the study of a program without any prior knowledge of the sample under examination. Two are the main kinds of analysis used, **Static** and **Dynamic** Analysis, but others were later outlined based on the tools used, such as Automated Analysis, or based on the sample behavior, called Behavioural Analysis (also a kind of Dynamic Analysis). Later, in this thesis, the above techniques will be explained in more detail.

With static analysis the intent is to analyze a program without executing it, useful to extract information such are File Properties, Strings, Fingerprint (via Hash Functions), Opcode (instruction codes), or Control Flow Graphs [30].

Dynamic Analysis instead represent a bigger group of methods, that have in common between themselves the execution of the sample. Running the Portable Executable is useful to monitor how the system behaves, what processes are spawned, what connections are established and so on.

With the advent of new Machine Learning capabilities, everyday new methods are developed, since these capabilities are able to define a model out of input data extracted directly from Malware (see Figure 2.3). These models are defined on the base of measurable property extracted from the PE (Portable Executable) of the programs.

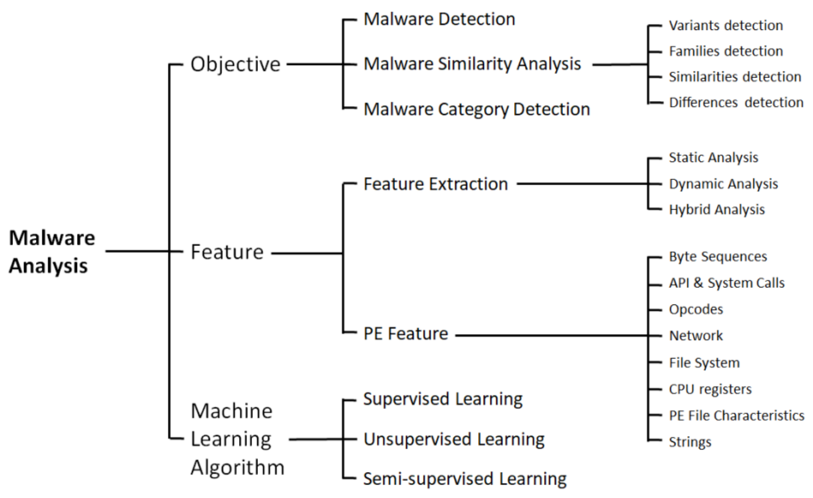


Figure 2.3. Modern Malware Analysis [70]

2.3.1 Sandboxing

Dynamic Analysis requires the execution of the malicious program, and so, exposing the testing environment to the infection of that specific sample. In order to test a sample in a safe environment are available various kinds of Malware Sandbox, that are virtual environments where malware can be analyzed without causing damage to the system. The Sandbox can full system emulations, or virtual machines, ad-hoc operating systems running in a fenced space of the host system regulated by a Hypervisor (a control mechanism to ensure that the operating systems are interacting only in their spaces and between them only following specific rules) [72]. Sandbox furthermore enables the execution of automated analysis systems like **Cuckoo** [21], **FlareVM** [42] and **CAPE**[61].

Inside the survey called ***An Inside Look into the Practice of Malware Analysis***[79] twenty-one analysts from 18 different companies were consulted in order to observe the main targets that experts focus on when studying a malware. Six main information was the focus of the analysts (Hashes, IP Addresses, Domain Names, Network and Host Artifacts, Tools, TTPs). The goals behind the research of these details are:

- **Blacklist**: communicate the Hash, IP Addresses and Domain Names used by the malware to block communications and further infections.
- **Extract Malicious Behaviour**: used to understand how malware works by capturing Network and Host Artifacts, IP Addresses and Domain Names through a Sandbox analysis.
- **Label Malware Families**: classify malicious programs into families to avoid in depth analysis on known samples.
- **Generate Report**: involves both static and dynamic analysis to triage the malware incident and generate a report, usually requested in companies to maintain a record of what happened.
- **Track Tactics, Techniques and Procedures**: used to define a bigger picture of how the malware authors operate, using a standard created by NIST (National Institute of Standards and Technology), that describes tactics as the best description of behavior.

2.4 Machine Learning

One of the main and the simplest form of Artificial Intelligence is Machine Learning, which regards the specific sets of algorithms and analysis that are able to extrapolate models and insights from training data [53]. In today's world, everyone is racing to keep up and invest in machine learning: according to statista.com [69] the 79% of all companies now uses AI for cybersecurity.

The broadest categories in terms of techniques are based on the type of training that the algorithm receives:

- **Supervised Learning:**

  - Involves machines learning from labeled data.
  - Tries to predict or infer a feature based on known data.
  - Examples include Bayesian networks, Decision trees, k-Nearest Neighbour (KNN), Support Vector Machine (SVM), and Artificial Neural Networks (ANN).

- **Unsupervised Learning:**
  - Deals with discovering hidden patterns or structures in unlabeled data.
  - Often referred to as clustering.
  - Common algorithms include k-means clustering, hierarchical clustering, and neighborhood-based methods like Self-organizing map (SoM).

- **Reinforcement Learning:**
  - Focuses on training machines to interact with a dynamic environment to achieve a specific goal.
  - Involves learning through trial and error.

Machine Learning is especially useful to classify (assign a label to a sample to associate it with a family), to cluster (group similar data into points based on their characteristics), to predict (given future or trends based on past values) and to optimize (using data to find the best solution or one of).

As simple as the operations may seem, it is not as simple when dealing with very large data sets characterized by a multitude of features. When it comes to large datasets, Machine Learning is able to perform Dimensionality Reduction (reduce the number of features keeping the most important information), and to grant Interpretability (analyzing the correlations within a dataset insight information can be found).

In this Thesis we will discuss a specific field of Machine Learning, consisting of Neural Networks, and in detail, Recurrent Neural Networks.

2.4.1 Neural Networks

Neural networks are an example of learning based on a system inspired by the functioning of the human brain (see Figure 2.4) and the connection between neurons [8].

In a human brain neurons are nerve cells interconnected between themselves, with the duty of transmitting chemical and electrical signals [8]. The branches of neurons are called dendrites and receive the information from them, and are connected to other neurons via long cables called axons and synapses that work as interface between the axon and the neuron. In a neural network the dendrites represent the *inputs*, the synapses operate as *weights* and the axon is referred as the *output*. To keep trace of the similarities, these are displayed in the Table 2.3.

Since the term "neural network" represents a very large family of algorithms, it is necessary to make a distinction.

1. **Perceptrons**: basic artificial neurons for binary classification.

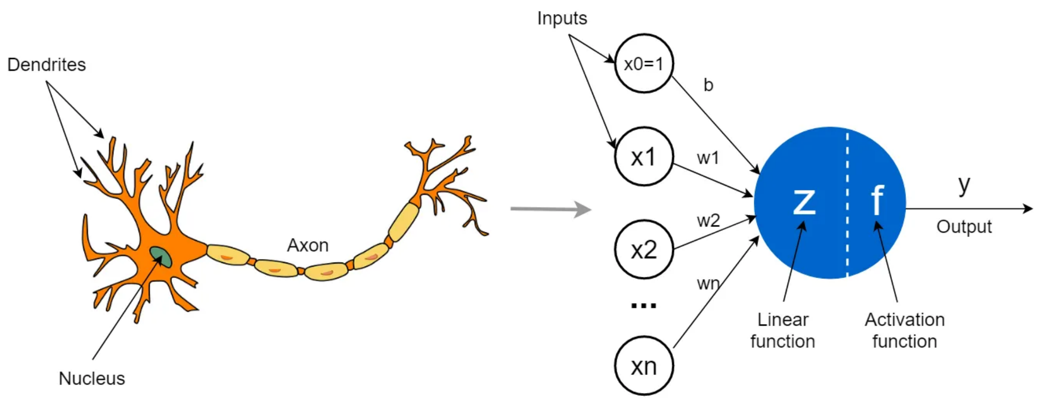


Figure 2.4. Neurons and Neural Networks [56]

Biological Neuron	**Artificial Neuron**
Cell Nucleus (Soma)	Node
Dendrites	Input
Synapse	Weights or interconnections
Axon	Output

Table 2.3. Neuron and Perceptron Similarities [8]

2. **Multi-Layer Perceptrons (MLP)**: neural networks with layers, capable of non-linear tasks.

3. **Convolutional Neural Networks (CNN)**: designed for grid-like data, especially images.

4. **Recurrent Neural Networks (RNN)**: processes sequential data with feedback loops.

5. **Autoencoders**: unsupervised learning for data reconstruction.

6. **Deep Learning**: broad term encompassing neural networks with many layers.

Nobody knows why Neural networks are able to perform well where others algorithms will not ever reach, but is not so difficult to understand how they work.

The main characteristic of this type of Machine Learning is without any doubt the so-called dense layers [23], in which each neuron of a specific layer is connected to every other neuron of the next layer, and so, using the output of the first as input for the second. What influences the transmitted information are called weights and biases: the weights decide how much of the input is passed to the next layer, whilst biases are constants that are added between a layer and its next one. So, to compute the output of a single neuron, it's sufficient to use the formula 2.1.

$$output = (inputs * weights) + bias \tag{2.1}$$

The inner beauty of this type of algorithm is that weight and biases are all tuned by the algorithm itself during its training. One last component is always necessary when it comes to Neural Network, and it's called Activation Function, and as the name suggest is a function in charge of simulating the role of the synapses, adding non-linearity to the system.

This new field opens up a world of possibilities for so many applications in cybersecurity. The simplest form of machine learning application are the handling of automated threat detection and response, and the possibility to provide to analysts data that are easier to understand and interpret. Secondarily it finds a purpose in Vulnerability Management, Behavioral Analysis and Sandbox Malware Analysis. The goal of this thesis is to apply a specific form of Neural Network to a whole dataset extracted from a Sandbox Analysis of over 40,000+ samples (both safe and malicious).

Chapter 3

State of the Art

The central part of this thesis will focus on how Recurrent Neural Networks can be used to enhance malware classifications after a behavioral analysis. This chapter will provide an overview of related works and the topics discussed will be the following:

- How professionals conduct Malware Analysis
- How sandbox and automated analysis work
- How to construct a feature dataset to train an RNN algorithm
- Which algorithms could be used to perform the analysis
- Analyzing the results and interpret them

3.1 Malware Analysis

Today's networks are constantly polluted by files, archives, documents and executables that, until proven otherwise, must be analyzed due to the doubt of their maliciousness. Each one of them can represent an attack vector, a starting point for the infection of a malware (using a Stager), or directly reveal a malware itself. Precisely for this reason it is necessary to develop robust and efficient analysis measures so that feedback can be provided and the "fingerprint" of the file reported on sites such as VirusTotal[1].

The goal of conducting an analysis on a particular file are almost always the same: determine whether it is malicious or benign. This operation is called *Binary Classification* (called binary because the possible outcomes are only two). The analysis consists in looking for evidences to demonstrate the presence or the absence of common pattern and characteristics found in malicious programs. Examples could be multiples:

- Hidden text inside the file that could be links to dangerous websites and server.
- The presence of a packer.

[1] https://www.virustotal.com

- The use of particular APIs that interfere with the system in which they are executed.
- IP Addresses related to known malicious endpoints.
- File metadata.
- Obfuscation Methods.

The classification then is done by assigning a score to each sample. If it exceeds a threshold, then the samples is labelled as malware, otherwise as safe. Hence, the classification is not immediate, and it can lead to classification errors referred to as *False Positives* (malware labelled as safe files) and *False Negatives* (safe programs identified as malware).

Watching a malware as part of a family is useful when it comes to understand the relationship behind multiple threats, and doing so, to have insight on which group hides behind the attack: this relation is found almost in each family because *"malware authors, like most software developers, tend to reuse their own code, share code and ideas with collaborators, pull code found on the internet into their projects, and follow trends set by competitors"* [46]. All this set of information is part of a branch called **Threat Intelligence**, and it is used to describe the knowledge associated to adversaries and their motivations, intentions and method.

3.1.1 Static Analysis

As mentioned earlier, analyzing malicious software without running it is called static analysis, but the steps are quite linear, due to the limited room for maneuver that such an analysis offers. Previously in this Thesis, some obfuscation methods were named, yet the first obstacle in running a static analysis is represented by packing mechanisms, that compress the executable and make them unintelligible when looking for information in the executable headers. To understand how packers work, a brief explanation of the Executable Structure is needed. In Windows executables are called PE (Portable Executable), and it represents a data structure that contains the information used by the OS loader to run the wrapped code (see Figure 3.1).

The most valuable information (when using a static analysis approach) are usually contained in the sections: the sections usually have the same reserved names, and varies between different Operative Systems, however Microsoft describes in depths how these work in Microsoft Docs [45]:

These sections are inspectable through software such as **Strings** [2], a tool to look for any combinations of three or more ASCII unicode characters, or **PeID** [3], used to determine if a packer has been used and identify the signature of the Executable, and more others. As one of the parts containing the most information, the role of packing becomes clear, as hiding what could be identified within minutes is the first step among anti-analysis techniques. The other trivial part of this kind of analysis put the focus on the Import Tables: a list of all the functions needed by the executable to work, as well as linked libraries. Indeed, a program has 3 main ways to have its libraries linked [5]:

[2] https://www.ibm.com/docs/en/aix/7.2?topic=s-strings-command

[3] https://www.aldeid.com/wiki/PEiD

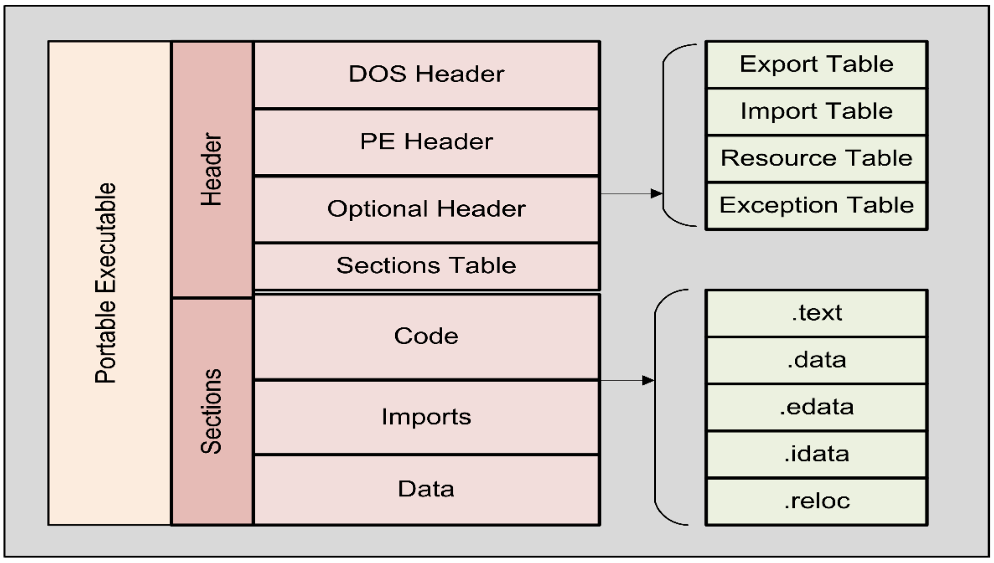
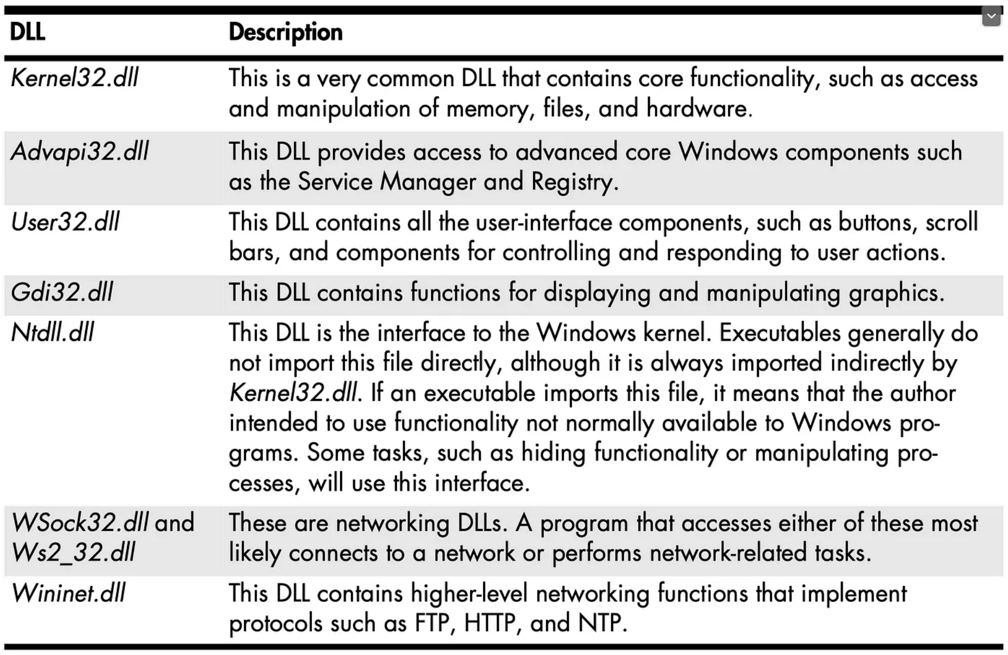


Figure 3.1. Portable Executable Format [62]

- **Static Linking:** the less-used type of linking, found in most of Linux and Unix programs. It works by merging the libraries needed with the author's code, expanding the size of the whole executable and making it harder to understand the executable's own code.
- **Dynamic Linking:** the de-facto standard when it comes to generate a malware, because when libraries are dynamically linked the PE includes automatically all the functions needed in the Header and delegates the role of executing them to the libraries themselves.
- **Runtime Linking:** the libraries and functions are loaded only when needed. In this case the import table does not show all the functions used by the executable.

When working with dynamically linked Malware, the content of the import table makes it easier to understand what will the malware behavior be, because each library has a specific subset of functions to observe the same role (see Figure 3.2).

In order to proceed with a static analysis on a particular malware, a previous step is needed. As mentioned before, a strong anti-analysis resource is represented by Packing mechanism. These techniques generate a new Portable Executable with a new header, using a compression algorithm, a cryptographic operation or both [12]. The possibility to use a random key, moreover, makes each sample unique making fingerprint checking useless. The code, now packed, incomprehensible by any analyst, contains only a runtime created by the packer and used to unpack the malware in runtime (see Figure 3.3).



DLL	Description
Kernel32.dll	This is a very common DLL that contains core functionality, such as access and manipulation of memory, files, and hardware.
Advapi32.dll	This DLL provides access to advanced core Windows components such as the Service Manager and Registry.
User32.dll	This DLL contains all the user-interface components, such as buttons, scroll bars, and components for controlling and responding to user actions.
Gdi32.dll	This DLL contains functions for displaying and manipulating graphics.
Ntdll.dll	This DLL is the interface to the Windows kernel. Executables generally do not import this file directly, although it is always imported indirectly by *Kernel32.dll*. If an executable imports this file, it means that the author intended to use functionality not normally available to Windows programs. Some tasks, such as hiding functionality or manipulating processes, will use this interface.
WSock32.dll and *Ws2_32.dll*	These are networking DLLs. A program that accesses either of these most likely connects to a network or performs network-related tasks.
Wininet.dll	This DLL contains higher-level networking functions that implement protocols such as FTP, HTTP, and NTP.

Figure 3.2. Portable Executable Format [5]

3.1.2 Dynamic Analysis

Dynamic analysis requires extensive attention due to the risk of infecting the machine on which the analyst is running the sample, and for this reason, according to [51], characteristics necessary to address the analysis must be outlined:

- *"It must be trusted":* insights coming out of the analysis environment must not be influenced by the malware.
- *"It must be undetectable by the analyzed file":* malware could behave differently if the system is recognized as analysis systems.
- *"It must collect as much relevant data as possible about actions performed by the malware":* the more data it can collect, the better the description of the behavior.
- *"It must meet the malware's expectations to expose its full behavior":* a malware is always created for a specific target, so the environment should appear in the condition that the malware expects, in order to unleash its full potential.
- *"It must limit/emulate network access by the malware":* malicious programs such as worms are capable of moving through the network, so a proper control must be enabled to prevent infections.
- *"It must generate a coherent and concise report":* a report must be generated, and it must be comprehensible by an analyst or a machine learning algorithm.

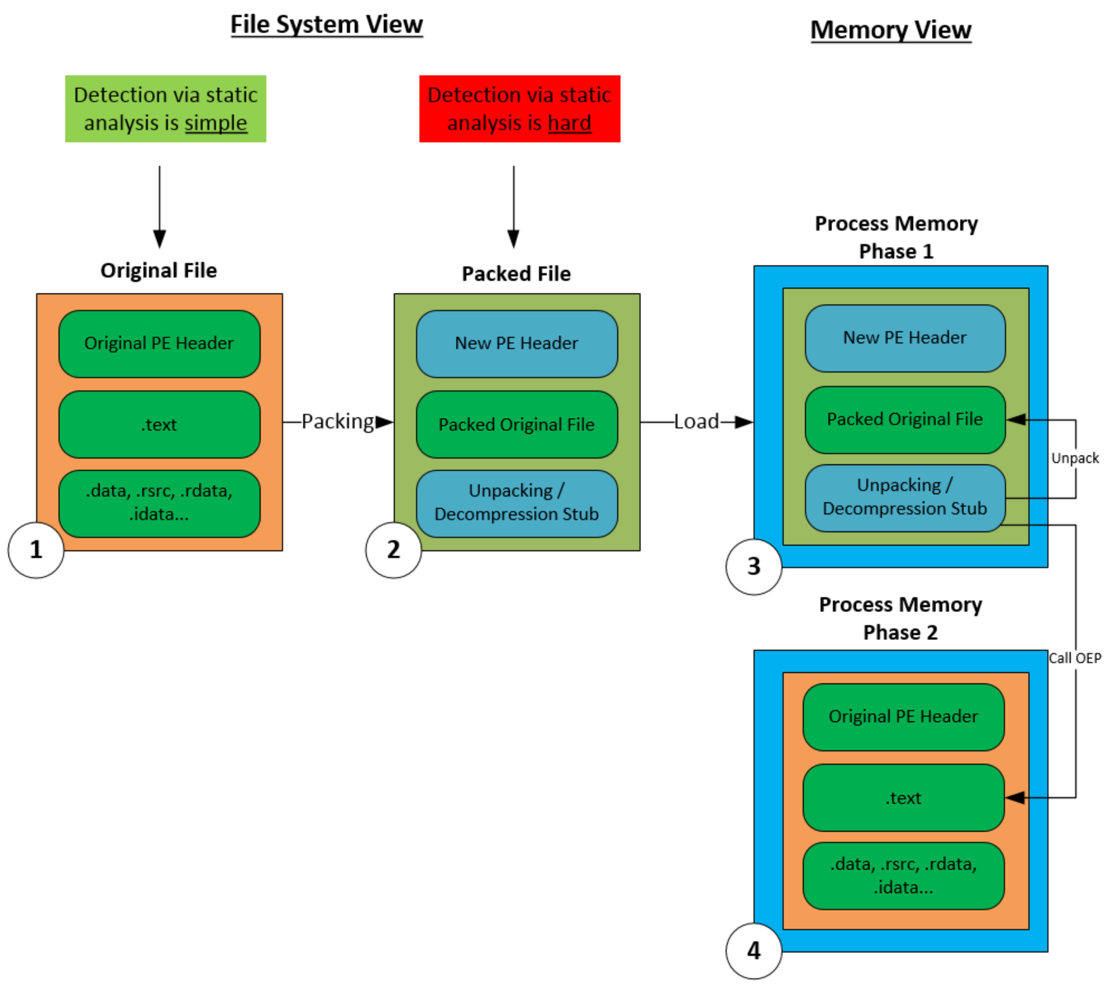


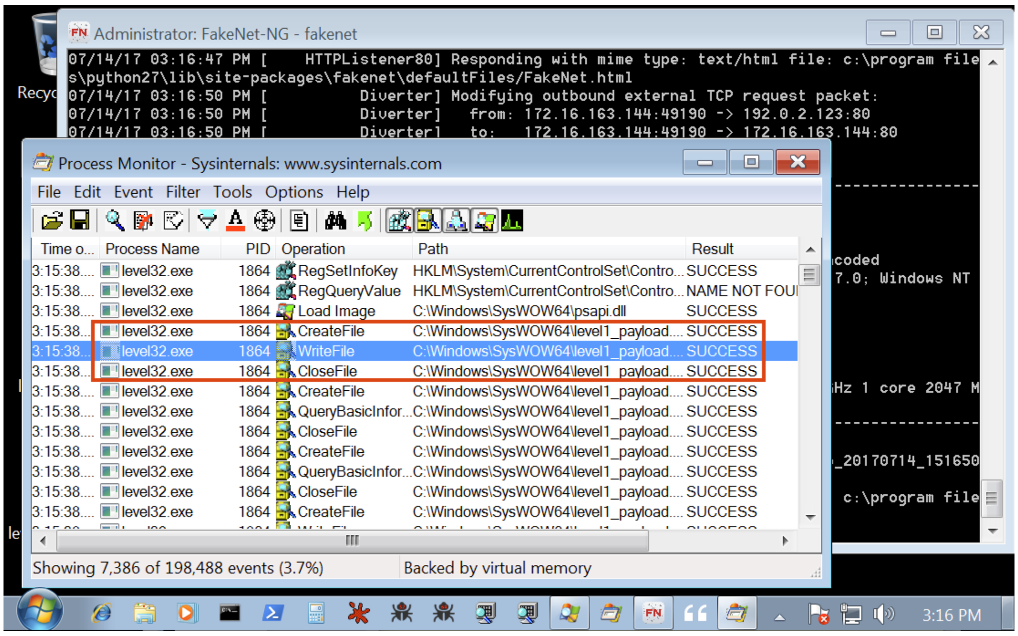
Figure 3.3. How a Packer works[12]

With these issues being said, the goal is to let the malware infect only part of the system, that was previously prepared with a correct environment to prevent the evasion of the sample. This can be done by using a virtual machine, a hypervisor or an emulator. In such cases, some programs may have the ability to detect if the infected host is a virtual machine or not, and in that case the second point of the above list is not true anymore. When it comes to decide the type of framework, a distinction is needed between *bare-metal* and *host-guest model*.
Bare-Metal approach relies on the construction of a full physical machine, with the only goal of analyzing a sample without any virtualization model. This approach guarantees the best growing soil for the malware to infect and take actions. However, after the analysis is finished, the problem of the bare-metal solution lies in the extremely difficult operation of reverting, because the only methods available are formatting and changing the hardware

used, that can lead to huge costs in terms of time and money.
Host-Guest model instead offers a simpler solution, more scalable and affordable by anyone, with the only contraries aspects of evasions or the analysis environment to be discovered by the sample. In this last situation, reverting the machine to a clean state is immediate, because the state of the OS can be saved to a file (a Snapshot), that will later be used as a baseline to revert the system settings, RAM and file systems.

The two most used types of Host-Guest model are **REMnux**[58] and **FLARE VM** [42]. These two were born for the exact same reason: conduct malware analysis and reverse engineer in a safe analysis framework. They differ only on the operative system used as victim, since the first is used to analyze Linux malware, while the second for Windows samples. For the purpose of this Thesis, only Flare VM will be presented.

Flare VM Developed by Mandiant as a collection of useful tools to aid malware researchers and Reverse Engineers, FLARE VM was born with the idea of deploying an open-source extension to Windows to speed up the process of initializing a Windows analysis framework. The entire workflow becomes now easier, because it offers process monitors (see Figure 3.4) network emulators to deceive the sample, Decompilers, Unpackers, multiple Reverse Engineering software and the whole Sysinternals Suite (a collection of Microsoft diagnostic tools), in addition to the possibility to revert the state of the machine to a clean state.


Administrator: FakeNet-NG - fakenet
Process Monitor - Sysinternals: www.sysinternals.com
File Edit Event Filter Tools Options Help
3:16 PM

```
07/14/17 03:16:47 PM [    HTTPListener80] Responding with mime type: text/html file: c:\program file
s\python27\lib\site-packages\fakenet\defaultFiles/FakeNet.html
07/14/17 03:16:50 PM [          Diverter] Modifying outbound external TCP request packet:
07/14/17 03:16:50 PM [          Diverter]   from: 172.16.163.144:49190 -> 192.0.2.123:80
07/14/17 03:16:50 PM [          Diverter]   to:   172.16.163.144:49190 -> 172.16.163.144:80
```

Time o...	Process Name	PID	Operation	Path	Result
3:15:38....	level32.exe	1864	RegSetInfoKey	HKLM\System\CurrentControlSet\Contro...	SUCCESS
3:15:38....	level32.exe	1864	RegQueryValue	HKLM\System\CurrentControlSet\Contro...	NAME NOT FOU
3:15:38....	level32.exe	1864	Load Image	C:\Windows\SysWOW64\psapi.dll	SUCCESS
3:15:38....	level32.exe	1864	CreateFile	C:\Windows\SysWOW64\level1_payload....	SUCCESS
3:15:38....	level32.exe	1864	WriteFile	C:\Windows\SysWOW64\level1_payload....	SUCCESS
3:15:38....	level32.exe	1864	CloseFile	C:\Windows\SysWOW64\level1_payload....	SUCCESS
3:15:38....	level32.exe	1864	CreateFile	C:\Windows\SysWOW64\level1_payload....	SUCCESS
3:15:38....	level32.exe	1864	QueryBasicInfor...	C:\Windows\SysWOW64\level1_payload....	SUCCESS
3:15:38....	level32.exe	1864	CloseFile	C:\Windows\SysWOW64\level1_payload....	SUCCESS
3:15:38....	level32.exe	1864	CreateFile	C:\Windows\SysWOW64\level1_payload....	SUCCESS
3:15:38....	level32.exe	1864	QueryBasicInfor...	C:\Windows\SysWOW64\level1_payload....	SUCCESS
3:15:38....	level32.exe	1864	CloseFile	C:\Windows\SysWOW64\level1_payload....	SUCCESS
3:15:38....	level32.exe	1864	CreateFile	C:\Windows\SysWOW64\level1_payload....	SUCCESS

Showing 7,386 of 198,488 events (3.7%) Backed by virtual memory

Figure 3.4. Process Monitor in FLARE VM [42]

But Dynamic Analysis Framework are not risk-free. As mentioned before, when a malicious software is busy doing malicious activity, the analysis work as expected. However, multiple avoiding and detection techniques have been developed by malware authors to evade the analysis system and eventually escape from the system to infect new victims.

Bare Metal could be vulnerable to hardware infection and to the detection of the analysis framework. Host-Guest model could be vulnerable, instead, to Rootkits and Malicious Hypervisors.

Dynamic Analysis Techniques

The main techniques used in dynamic analysis are three: **Function Call Monitoring**, **Function Parameter Analysis** and **Information Flow Tracking** [24].

Function Call Monitoring A function, in terms of programming, is a block of code that perform a specific action, that may elaborate input values and produce output values. With that given role, a function has in its advantages the re-usability, since it may be used multiple times in the same program or the same function could be used in multiple programs. So starting with understanding how to analyze the functions used in the program, it is possible to abstract its behavior: that process of intercepting function calls is called *hooking*. According to Kaspersky definition, Hooking is a *"Techniques used to alter or augment the behavior of an operating system, applications or of other software components by intercepting function calls or messages or events passed between software components. Code that handles such intercepted function calls, events or messages is called a hook"* [40]. Basically it works by modifying the analyzed program in order to implement the required analysis functionality. To understand what can be hooked, the following definitions are given [24]:

- **Application Programming Interface (API):** groups of functions that enable a common functionality, that can be shared between different programs, such as manipulating files and communicating over the network. Usually each operating system provide its own set of API.
- **Function Calls:** programs have two modes of requesting the access of an operating system environment. General applications usually to request access to kernel-mode, that enables them to manipulate what they need. User-mode however has no right to modify the system state directly, so the operating system provide a set of well-defined API, called *system-call*. Some malware have shown the ability to perform a privilege escalation, gaining the kernel-mode privileges.
- **Windows Native API:** the Windows Native API are placed between the system call interface and the Windows API. While Windows API remains stable and coherent between all the versions, Native API may vary. These are called by higher-level APIs, to invoke the system calls and act as a wrapper for the functions. Malicious programs that use Native API are mostly written by people with a deep knowledge of Windows Internals, since these APIs don't have a complete documentation.

The results of Function Hooking provide a better description on the abstraction level of the respective set of functions. Semantically, the best observations are found by hooking Windows API functions, while a more detailed view can be seen in Native API hooking. To enable this type of analysis, multiple approaches can be used, such as inserting hook functions into the source code, or enabling compiler flags.
One of the most used techniques is called Binary Rewriting, and is used if the program is

only available in binary form, and it works by modifying the monitored function so that this call invokes the hook, or, alternatively, find all the call sites and invoke the monitored function to call the hook. In both cases, the hook function can analyze all the optional argument on the stack, and if the function is invoked through a pointer, this value can be changed to point to a hook function instead.
Otherwise, to modify an already-running binary, it may be useful to exploit the DLL injection: by forcing another process to load a dynamic-link library, it is possible to run code within its address space. Debugging techniques are also used to hook into invocations of specific functions, by inserting breakpoints that trigger and delegate the control of the process to the debugger, so that it can have full access to memory contents and CPU state of the process.

Function Parameter Analysis In dynamic analysis, differently from the role that has in static analysis, *function parameter analysis* focuses on the actual values that are passed to a function when is called. This technique can be useful to determine correlations between functions into logically coherent sets to give a *"different, object-centric, point-of-view"* [24].

Information Flow Tracking Information flow tracking was born to understand how the data propagate and find useful resource such as endpoints by using **taint analysis**: the data that should be controlled while flowing are labelled (tainted). Such operation make sense when the concept of *Taint Sources and Sinks* is introduced: taint source are endpoints used by the programs to generate new taint labels into the system, whereas a sink is a part of the system that acts with a specific routine when the tainted input is modified, such as conditional branches that react to a different value of the tainted value. The value of information flow tracking in malware analysis, as described in [16], could have a great impact, but with the advent of this technique newer malware could be generated in such ways to evade this analysis technique.

Unlike static analysis, dynamic analysis has a significant weakness: since it works by running the sample and analyze its functioning, it can't cover all the possible execution flows, because it covers only the executed path.

3.1.3 Challenges in Malware Analysis

The history of malware analysis is constituted by an evolving path of technics to uncover what is hide behind the mask of a false friend such as a Trojan or a worm, intelligible information that somehow have to be retrieved to enhance defense systems and to sharpen analysis tools. Each discovery in defensive counter-measures was later faced by an anti-analysis approach and led to new features, and vice versa. The ACM Computing Surveys published in [24] this issue calling it "Malware Analysis Arm Race". In the previously cited paper, a specific chapter is dedicated to this challenge, to present all the critical point in malware analysis approaches of those years, remained then topical.

- **Self-Modifying Code and Packers:** Signatures cannot be used anymore to assess a threat, due to polymorphic code and packers.
- **Detection of Analysis Environment:** Malware are able to detect Virtual Machines, Monitor Tools and debuggers.

- **Logic Bombs:** Logic bombs trigger only in specific conditions. Some analysis may lead to inefficient results because the logic bomb didn't exhibit its behavior.
- **Analysis Performance:** Excess time spent due to the performance requirements of the analysis may lead to timeouts.

3.1.4 Automated Analysis

To conduct a behavior-based large-scale analysis with a reasonable amount of resource, thinking of analyzing each malware one-by-one is an idea to be discarded due to its time requirements. Merging the concept of Sandboxing, Static and Dynamic Analysis, a new type of analysis can be defined: the *Automated Analysis*. This concept is generated by the aim to find the shared patterns behind malware families and classify these based on their common behavioral pattern. The workflow of an automated analysis proceeds by clustering the samples considering their features and use the clusters generated to form prototypes to use as class labels (see Figure 3.5)

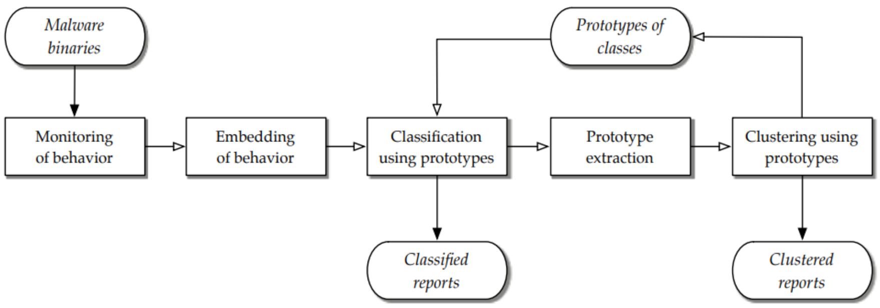


Figure 3.5. Workflow of Automated Analysis model [59]

In this Thesis, the purpose is to analyze the results of an automated analysis system to constitute a dataset for training a Recurrent Neural Network. The network will be tasked with generating a binary classification model (determining whether a sample belongs to one of two categories) to determine the degree of maliciousness of an ambiguous file by analyzing its sequence of calls to system APIs (specifically, the APIs of a Windows operating system).

Cuckoo Sandbox The open-source automated analysis tool used is called Cuckoo Sandbox [21]. Cuckoo Sandbox was born as a project in Google Summer Code in 2010, and was originally created to run and analyze suspicious or malicious files and generate a set of results [22]:

- Traces of calls performed by all processes spawned by the malware.
- Files being created, deleted, and downloaded by the malware during its execution.
- Memory dumps of the malware processes.
- Network traffic trace in PCAP format.

- Screenshots taken during the execution of the malware.
- Full memory dumps of the machines.

Its architecture is composed by a central management framework that handles executions and analysis, with each one of them running in an isolated and clean virtual machine (see Figure 3.6). The host runs the core of the sandbox, and may run a web-interface to interact with a graphical interface on which upload each sample. Cuckoo is equipped with a network simulator (Inetsim in Cuckoo Rooter) that enables the analysis environment to interact with different modalities of network:

- **None Routing:** all the communications are disabled.
- **Drop Routing:** completely drops all the cuckoo non-related traffic.
- **Internet Routing:** full internet access.
- **Inetsim Routing:** routes all the requests to multiple fake servers running on the host.
- **Tor Routing::** all the traffic is routed to TOR (The Onion Router [13]).
- **VPN Routing:** all the communications must pass through a VPN predefined endpoint.

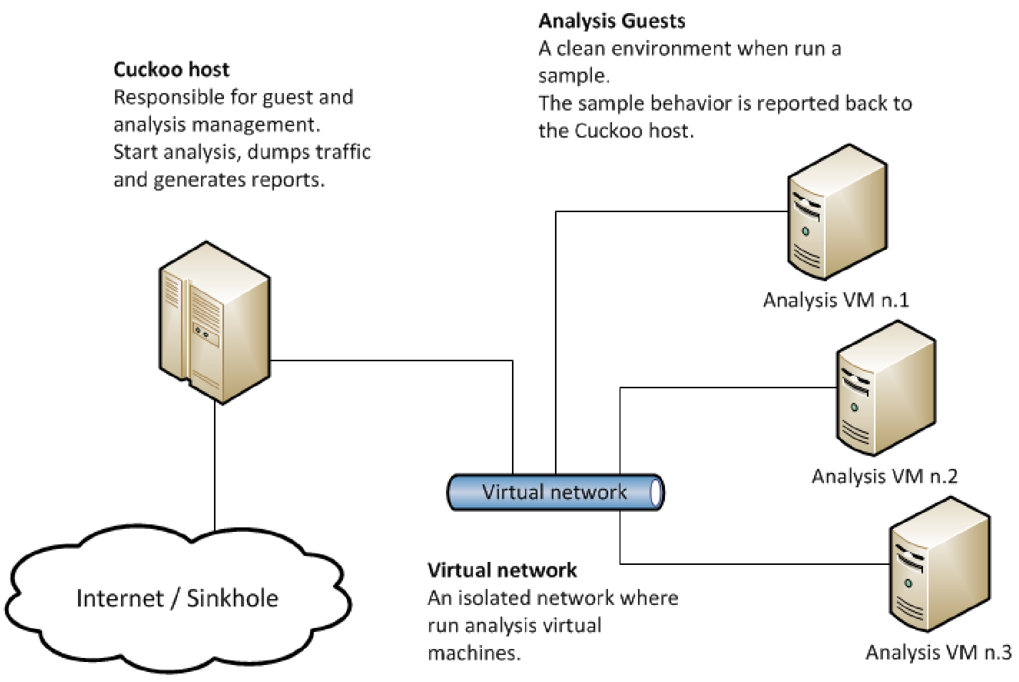

Analysis Guests
A clean environment when run a sample.
The sample behavior is reported back to the Cuckoo host.
Cuckoo host
Responsible for guest and analysis management.
Start analysis, dumps traffic and generates reports.
Analysis VM n.1
Analysis VM n.2
Analysis VM n.3
Virtual network
Internet / Sinkhole
Virtual network
An isolated network where run analysis virtual machines.

Figure 3.6. Cuckoo Sandbox Architecture [22]

3.2 Feature Extraction

As written in [11], a **feature** is an individual measurable property or characteristic of a phenomenon, that represents a crucial element in pattern recognition algorithms and machine learning. To better describe what a feature is and what does it mean to perform feature extraction, a standard dataset will be introduced to understand this concepts: **the Iris Dataset** [26].
The Iris Dataset is *"one of the earliest dataset used in the literature on classification methods and widely used in statistics and machine learning"* [26]. It contains 3 types of Iris plant, with 50 instances per Flower, all described by four variables as features (sepal length, sepal width, petal length, petal width), and a class label that indicates to which class of iris plant the sample belong (Setosa, Versicolour, Virginica as in figure 3.7).

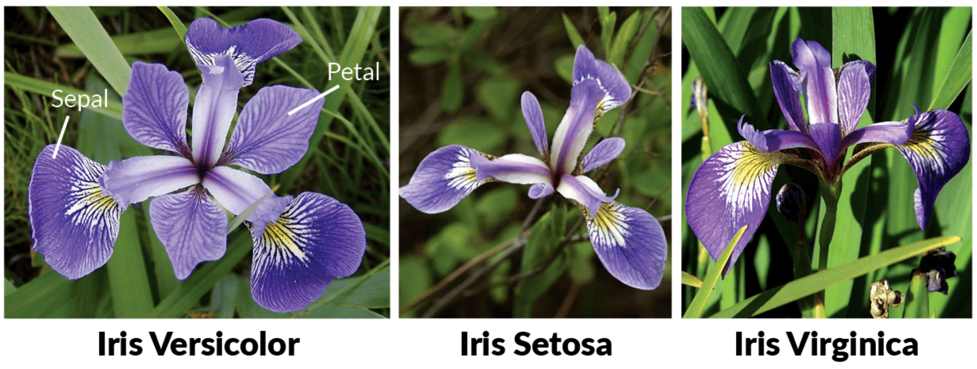


Figure 3.7. Iris families used for the Iris Dataset [78]

Of course, the data extracted from a single plant could be more than 4 types, but these represent the most significant in terms of classification. The act of extracting information from a sample, pre-process them, and select the most significant, is called **Feature Extraction**. All the data used as a feature must overcome a preliminary phase to check their correctness, since they could be redundant or big enough to make the optimization part ineffective.
Features can be of 2 types: categorical, including discrete values that can be grouped into categories that necessitate to receive a specific encoding before the analysis, or numerical, such as continuous values that can be measured on a scale and do not need any conversion before the elaboration.

Within the context of malware analysis how are the features extracted? The following sources of features are taken in account [4]:

- **Application features:** from the static analysis API calls, PE headers, string hashes, import address table. From the dynamic instead, API call sequences.
- **Network features:** the traffic generated by the sample is captured and analyzed to extract IP addresses.

- **Features from external sources:** antivirus engine signatures and reports used as features [17].

3.3 Machine Learning goals and algorithms

As introduced in Chapter 2, Neural Networks represent a branch of Machine Learning, a family of algorithms having the ability to improve and learn on their own. These algorithms are able to predict values, to detect anomalies, classify groups of samples and find patterns where people cannot see them. Using neural networks, is possible to implement speech and image recognition tasks in terms of minutes, instead of hours.

3.3.1 Recurrent Neural Networks and LSTM

Among all branches of Deep Learning, the type of neural network to be addressed in this Thesis is the Recurrent Neural Network (RNN), defined in [31] as *"is any network whose neurons send feedback signals to each other"*. In the past these have been involved in the analysis of sequence of events or characters in words, but later were used to enhance motion detection and music synthesis. To better address the functioning of RNN, the working schema must be explained. Differently from basic Neural Networks where inputs, weights and biases are independent of one another, Recurrent Neural Networks parameters are not independent, because they are modeled by each input of the sequence due to the feedback model (see Figure 3.8). The feedback loop acts as a memory, that is used by the algorithm to alter the next evaluation based on the previous inputs, and this creates a hidden state, an abstract component used to remember specific data about a sequence.

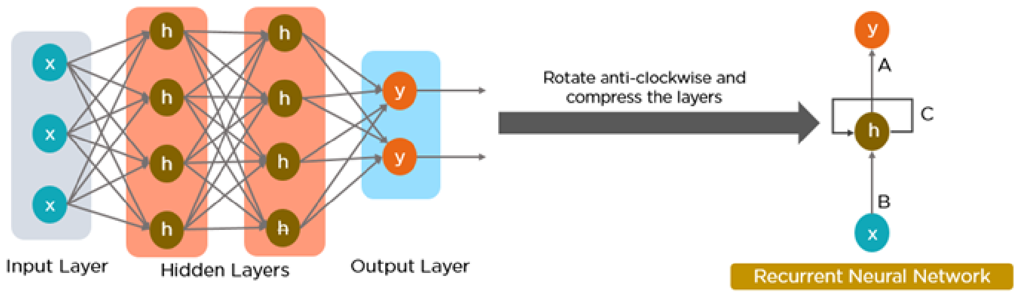


Figure 3.8. Difference between Neural Networks schema and RNN schema [39]

Given the number of inputs and outputs Recurrent Neural Networks are classified in 4 types, that are used to determine which model is the best fit for a specific application [29]:

- One-to-One: the simplest one, it acts as a standard neural network, used mostly in Image Classification
- One-to-Many: the input is fixed and gives a series of data outputs, found in Music Generation and Image Captioning

- Many-to-One: given a sequence of inputs, the hidden layer are converted into a single output, found in Sentiment Analysis
- Many-to-Many: multiple inputs to multiple outputs, the more general, found in Machine Translation

As in neural networks, in RNNs the output value is calculated with the linear equation (see equation2.1), but when dealing with feedback loops this formula leads to a common problem known as **"Exploding and Vanishing Gradient"**. Basically, it is possible to depict the first rounds of this algorithm on a random sequence, by looking at what would the first different rounds act considering each consecutive round as a simple neural network with one input and one output: this technique is called **"unfolding"** (see Figure 3.9).

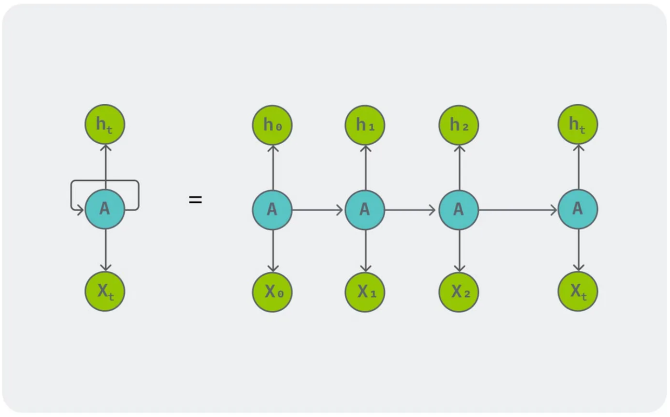


Figure 3.9. Unfolding of a Recurrent Neural Network [29]

The problem of the "Exploding and Vanishing Gradient" was first highlighted in 1994 by Bengio in [10], and it refers to a propagating large scale increase in the norm of the gradient during training [54]. During backpropagation, a common step in RNNs used to update the weights of the network in each iteration, weights are updated proportionally to the gradient value after each iteration, and this propagating error increases exponentially by each training iteration (epoch) [9]. It can be seen by assuming a relatively small (less than 1) or a relatively big (bigger than 1) value as initial weight: the first case led to vanishing gradients, the second case to exploding gradients. Each output of the function used in a single iteration, in RNN, become the weight for the next iteration. By considering starting values less than 1, after many iteration the propagation of the weight brings the latest decision to have no influence on the results. On the other hand, by choosing values bigger than 1, the power elevation of the weight brings it to grow exponentially. This act of deciding what gradient should be taken is referred as part of the optimization of the Recurrent Neural Network, and can be achieved by analyzing gradients with a Gradient Descent algorithm, used to find the minimum of a differentiable function (or to find parameters values to minimize the loss function).

Multiple solutions are available to overcome the problem of the "Exploding and Vanishing Gradient": the one used in this Thesis and the most popular is called **LSTM**, or **Long Short-Term Memory**.

LSTM - Long Short-Term Memory

LSTM, or Long Short-Term Memory, is the most popular solution to the problem of the *"Exploding and Vanishing Gradient"*. This solution transformed a huge part of the machine learning fields, and has been used to improve Google's speech, Amazon's Alexa answers and Facebook reached 4 billion LSTM-based translations per day in 2017 [34].
On first look it has a complex structure (see figure 3.10), but can be understood by splitting the concept into pieces:

- The upper line works as a propagator for the long-term memory, and it is influenced only by the so-called **forget-gate** and **input-gate**. The forget-gate is needed to decide what information to discard by a previous state, by assigning it a value between 0 and 1 (by mean of the sigmoid activation function).
- The bottom line instead is needed to elaborate the short-term memory in multiple operations, in the forget gate to decide what part of the input should be remembered in long-term, in the input gate, to decide which percentage of the short-term memory should be remembered and added to long-term memory, and a last one called **output-gate**, that make use of the long-term memory to influence short-term memory values and produce an output.

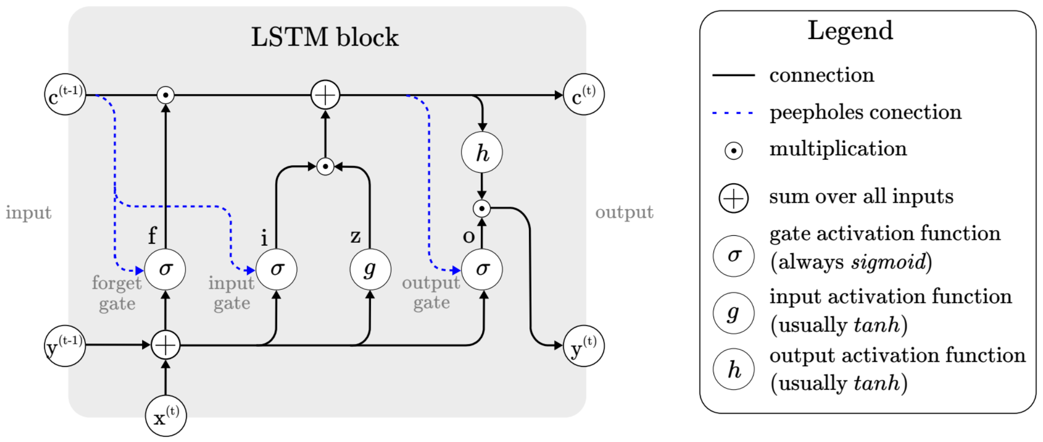


Figure 3.10. A LSTM block [34]

As presented in [34], between all the types of LSTM variants, the most efficient and easiest one is the so-called *vanilla LSTM*. The Vanilla LSTM is *"interpreted as the original LSTM block with the addition of the forget gate and peephole connections. In total, eight variants were identified for experimentation. In a nutshell, the vanilla architecture performs well on a number of tasks, and none of the eight investigated variants significantly outperforms the remaining ones. This justified most applications found in the literature to employ the vanilla LSTM."* [34].

Chapter 4

Implementations

The core of this thesis is explained in the Implementation chapter, in which it is demonstrated how the starting idea was was applied to produce a Proof of Concept. Leveraging the theoretical knowledge described in the chapter 3, the workflow was well defined. With the purpose of extrapolating an inherent characteristic of an unknown file, it was planned to collect a set of samples, both malicious and safe, from which it was then possible to extract the necessary values to build a dataset. A tool capable of automating the extraction process was therefore chosen, combining it with an ad-hoc script capable of collecting a sufficient number of samples from a reliable source on the Internet.

The third block is the construction of the actual machine learning classification model, which is initially made in a rough way and then refined so that it can achieve a fair trade-off between optimal results and training cost.

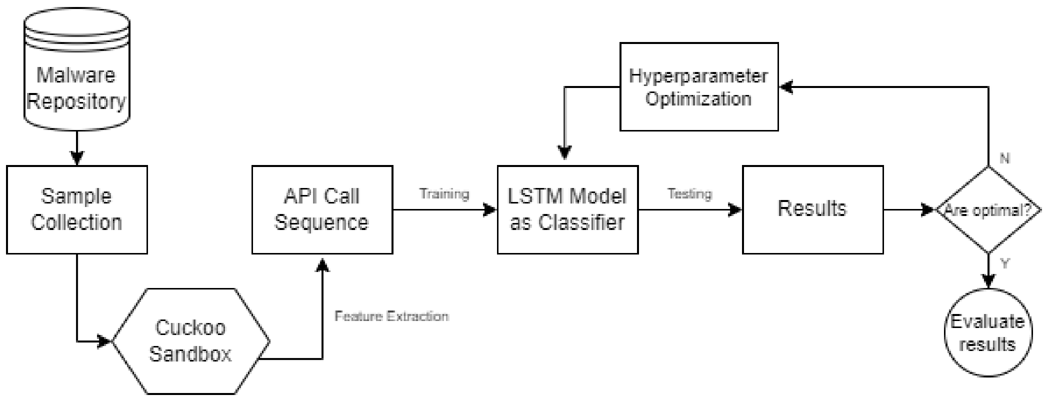


Figure 4.1. Implementation Schema

Finally, after the first results are obtained from the scratch, the model can be tuned in order to achieve better results and evaluate it the approach was correct. Through specific classification metrics, indeed, the parameters are tested with different values to choose the best solution.

4.1 Analysis Environment

When faced with an unknown file whose content and behavior one wants to learn, the first step is to have a laboratory available that can contain the analysis context and isolate it from valuable assets, whether sensitive information or valuable content. As explained in Chapter 3, there are many versions of ecosystems already set up to handle the analysis of hazardous content, so it may be useful to delve into the available choices to evaluate the most efficient solution that fits the type of analysis proposed in this Thesis. The key values sought are as follows:

- *Sandbox robustness*: sandboxing is a technique, not a de-facto invulnerable defensive system. Sandboxing does not represent the perfect form of computational fencing, because even hypervisors may have vulnerabilities. Since multiple malware have shown the ability to evade and infect environments regulated and protected by hypervisors, the sandbox must be robust and constantly updated whenever a vulnerability is found, to prevent the inefficiency of this defensive counter-measure.
- *Ability to extract malware configurations*: the environment should provide enough tools and programs to elaborate each sample.
- *Automated functions*: once the analysis routine is determined, the laboratory should provide a system to automate the function for the multitude of samples.
- *Analysis limits*: not all systems are free or affordable in terms of time or money.
- *Virtualization limits*: virtualization, if present, let the guest system to use only part of the available resources on the host.

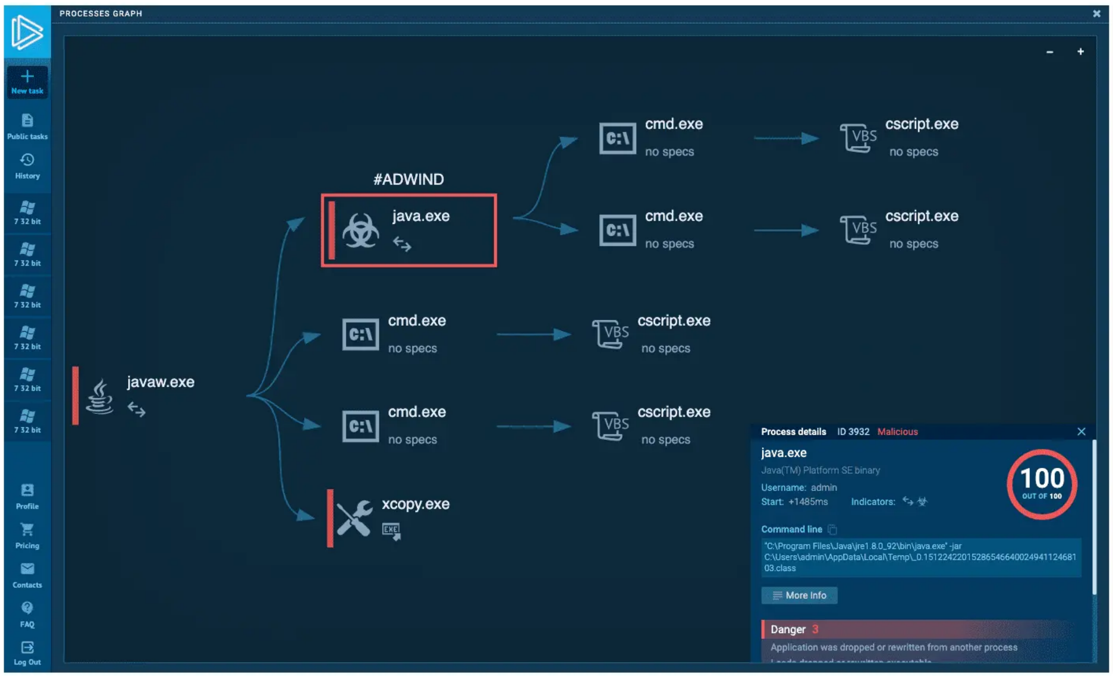
What follows is the description of pros and cons of the three most suggested sandbox systems to better understand the one that has been chosen:

ANY.RUN ANY.RUN [1] is a cloud-based sandbox with full access to the environment and able to track in real-time DNS resolutions, IP connections, and file spawned. At the end of the analysis a report is generated, and it is possible to customize the analysis configuration choosing a timeout, various types of routing, and human-activity simulations. Without any doubt, this could represent the best solution to limit any damage and extract malware configurations, but each task requires a fair amount of time. It offers a wide range of solutions to assess the behavior and the maliciousness of a file (see Figure 4.2), website or address, but makes it harder to extrapolate information such as API call sequences. Therefore, ANY.RUN is an idea discarded from the beginning as it does not align with the needs of the work required in this Thesis.

Flare VM Flare VM 3.1.2 characteristics can be summarized as:

- **Comprehensive Toolset**: FLARE VM provides a comprehensive set of tools for malware analysis and reverse engineering.

[1] https://any.run/



Process details ID 3932 Malicious

java.exe

Java(TM) Platform SE binary

Username: admin

Start: +1485ms Indicators:

Command line

```
"C:\Program Files\Java\jre1.8.0_92\bin\java.exe" -jar
C:\Users\admin\AppData\Local\Temp\_0.151224220152865466400249411246810
3.class
```

Danger 3

Application was dropped or rewritten from another process

Figure 4.2. Visualization of malware behavior in ANY.RUN

- **Windows-Based Environment**: It is a Windows-based virtual machine, suitable for analyzing Windows-specific malware and threats.
- **Frequent Updates**: FLARE VM is regularly updated with the latest tools and plugins, ensuring access to the most current resources.
- **Community Support**: Developed by FireEye, FLARE VM benefits from community support and collaboration, enhancing its effectiveness and reliability.
- **Pre-configured Environment**: The VM comes pre-configured with a variety of tools, saving time and effort in setting up the analysis environment.

Although Flare VM may seem a viable choice, defining an analysis routine that can generate a report to be used to take valuable information from it is actually quite complicated.

Cuckoo Sandbox Cuckoo Sandbox 3.1.4 characteristics can be summarized as:

- **Dynamic Analysis**: Cuckoo Sandbox allows for dynamic analysis of malware by running it in a controlled environment, enabling observation of its behavior.
- **Automated Execution**: Cuckoo automates the execution of suspicious files, making it efficient for analysts to analyze a many samples.

- **Network Traffic Monitoring**: Cuckoo Sandbox monitors the network traffic generated by the malware during execution, helping analysts understand communication patterns and potential threats.
- **API Support**: Cuckoo provides an API for integration with other security tools and systems, allowing seamless incorporation into existing workflows.
- **Screenshots and Memory Dump Analysis**: Cuckoo captures screenshots and extracts memory dumps during malware execution, aiding in visual and memory-based analysis.
- **Behavioral Signatures**: The observed behavior of malware in the sandbox can be used to create behavioral signatures for identifying similar patterns in other samples.
- **Open Source and Community Support**: Cuckoo is an open-source project with a community of security professionals contributing to its development, ensuring continuous improvement.

Newer systems are available to perform automated malware analysis but Cuckoo sandbox was chosen because through the year it proved to be the most consistent and coherent in each part of the analysis, and it represents the best low-budget solution to produce a proof-of-concept.

4.1.1 Malware Repositories

Once the analysis environment is deployed and ready to work on new samples, the next step is to decide where find malware samples. The most popular solution resides in MalwareBazaar [2], a project ideated by abuse.ch with the intent to create a place where freely share malware (see Figure 4.3), *“helping IT-security researchers and threat analysts protecting their constituency and customers from cyber threats”* [6]. Due to its properties, this kind of malware repository is always up-to-date, thanks to all the professionals and societies that help by sharing their own samples. The service is accessible via a website user interface and via API, providing its own endpoints to retrieve the daily, or even the monthly, batch of malware.

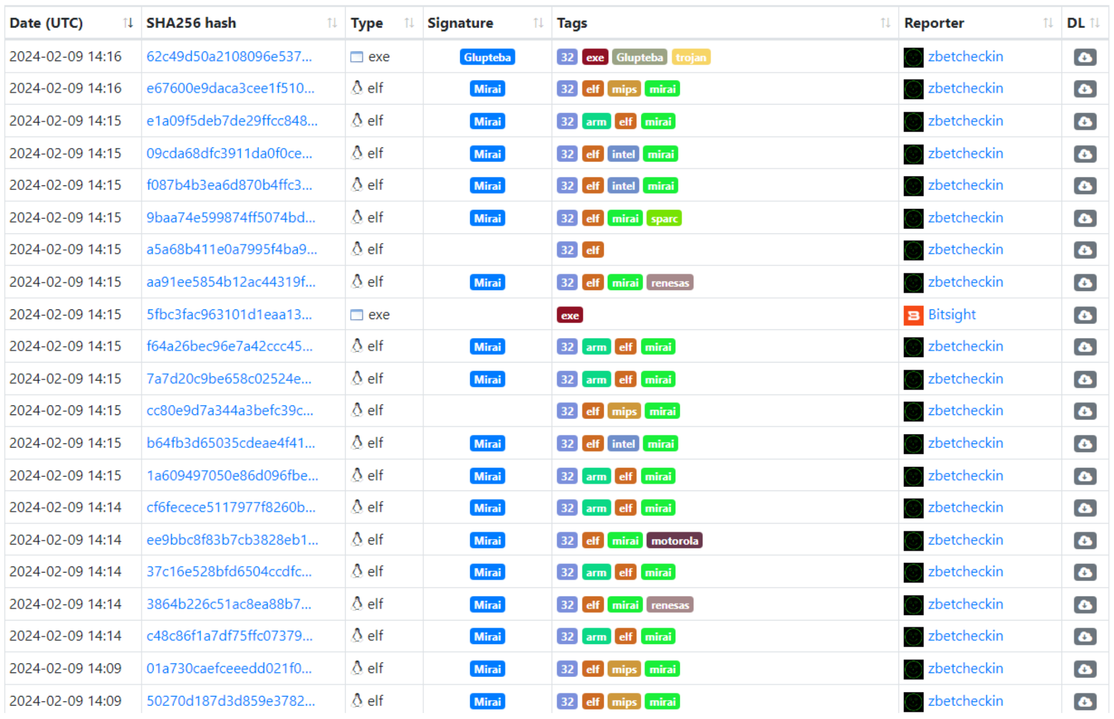
4.1.2 Automating the Analysis

A diverse, scalable, and well-performing programming language was sought to coordinate calls to the Cuckoo API, downloading samples from Malware-Bazaar, and pre-processing reports from Cuckoo. The choice fell on Python.

During the script development process, the **requests**[3] module was used, because it offers a wide-range solution to interact with Cuckoo API, using REST, and to perform downloads from the malware repository.

[2] `https://bazaar.abuse.ch/`

[3] `https://requests.readthedocs.io/en/latest/`



Date (UTC)	SHA256 hash	Type	Signature	Tags	Reporter	DL
2024-02-09 14:16	62c49d50a2108096e537...	exe	Glupteba	32 exe Glupteba trojan	zbetcheckin	
2024-02-09 14:16	e67600e9daca3cee1f510...	elf	Mirai	32 elf mips mirai	zbetcheckin	
2024-02-09 14:15	e1a09f5deb7de29ffcc848...	elf	Mirai	32 arm elf mirai	zbetcheckin	
2024-02-09 14:15	09cda68dfc3911da0f0ce...	elf	Mirai	32 elf intel mirai	zbetcheckin	
2024-02-09 14:15	f087b4b3ea6d870b4ffc3...	elf	Mirai	32 elf intel mirai	zbetcheckin	
2024-02-09 14:15	9baa74e599874ff5074bd...	elf	Mirai	32 elf mirai sparc	zbetcheckin	
2024-02-09 14:15	a5a68b411e0a7995f4ba9...	elf		32 elf	zbetcheckin	
2024-02-09 14:15	aa91ee5854b12ac44319f...	elf	Mirai	32 elf mirai renesas	zbetcheckin	
2024-02-09 14:15	5fbc3fac963101d1eaa13...	exe		exe	Bitsight	
2024-02-09 14:15	f64a26bec96e7a42ccc45...	elf	Mirai	32 arm elf mirai	zbetcheckin	
2024-02-09 14:15	7a7d20c9be658c02524e...	elf	Mirai	32 arm elf mirai	zbetcheckin	
2024-02-09 14:15	cc80e9d7a344a3befc39c...	elf		32 elf mips mirai	zbetcheckin	
2024-02-09 14:15	b64fb3d65035cdeae4f41...	elf	Mirai	32 elf intel mirai	zbetcheckin	
2024-02-09 14:15	1a609497050e86d096fbe...	elf	Mirai	32 arm elf mirai	zbetcheckin	
2024-02-09 14:14	cf6fecece5117977f8260b...	elf	Mirai	32 arm elf mirai	zbetcheckin	
2024-02-09 14:14	ee9bbc8f83b7cb3828eb1...	elf	Mirai	32 elf mirai motorola	zbetcheckin	
2024-02-09 14:14	37c16e528bfd6504ccdfc...	elf	Mirai	32 arm elf mirai	zbetcheckin	
2024-02-09 14:14	3864b226c51ac8ea88b7...	elf	Mirai	32 elf mirai renesas	zbetcheckin	
2024-02-09 14:14	c48c86f1a7df75ffc07379...	elf	Mirai	32 arm elf mirai	zbetcheckin	
2024-02-09 14:09	01a730caefceeedd021f0...	elf	Mirai	32 elf mips mirai	zbetcheckin	
2024-02-09 14:09	50270d187d3d859e3782...	elf	Mirai	32 elf mips mirai	zbetcheckin	

Figure 4.3. Malware-Bazaar repository

Cuckoo Sandbox provide a simple REST API server to interact with the environment remotely or without using the Graphical User Interface [20]. To operate with REST API an Authentication Token must be set, in order to admit to the analysis environment only samples submitted by those who have access to the token.

The server runs on the port 8090, and can be configured to use HTTPS, to tune server performance and enable other protocols. Multiple endpoints are available, but basically, the most useful are few (according to Official Cuckoo API Documentation4.1):

The requests are customizable through various parameters: it is possible to set a timeout, to modify the analysis options, add tags, a priority or request a memory dump. The only mandatory field is the file to analyze.

The analysis and report-processing routine is developed as follows:

1. A Malware batch is retrieved by Malware-Bazaar, using a specific endpoint provided by the same platform.
2. Using the **ZipFile** module, each batch is extracted.
3. All the executable files extracted are sent, one by one, to Cuckoo via POST request. The **request** module enables python to construct Headers and Payload of a request and send it to a specific URL. This action starts a new task with the submitted sample.

Request Method	Resource	Description
POST	/tasks/create/file	Adds a file to the list of pending tasks to be processed and analyzed.
GET	/tasks/report	Returns the report generated out of the analysis of the task associated with the specified ID.
GET	/tasks/view	Returns the details on the task assigned to the specified ID.
GET	/tasks/list	Returns the list of tasks stored in the internal Cuckoo database. You can optionally specify a limit of entries to return.

Table 4.1. Description of Cuckoo API Endpoints

4. The sample analysis, set up with a given timeout, is appended to the cuckoo workflow and the script is set in sleep mode to wait for the report.
5. After the report has been produced, the script retrieve the report in JSON format.
6. From the text report, the fingerprint and the list of API is extracted and elaborated. Each consecutive repeated call is replaced by a single occurrence. Only the first 100 are collected.
7. The extracted data is then appended to a **csv** file, that would later form the dataset.

Since this is a very expensive process to implement on a large scale on a single device, it was decided to use an existing dataset [4] to produce a proof-of-concept.

4.2 Dataset Insights

The dataset [15] represents perfectly the target of the used analysis model: it was created starting from 49,797 malware and 1,079 goodware, passed through the analysis offered by Cuckoo sandbox, to extrapolate the API list of each sample. As can be seen from the amounts of information available, the dataset is heavily unbalanced, as the number of goodware represents less than 2 percent of all total samples. However, since the weight of classification has more importance in recognizing true positives instead of true negatives, the expected model can be applied by accepting a reasonably high percentage of false positives.

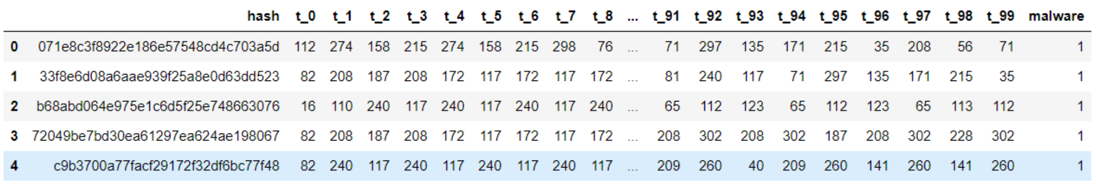
Here follows a brief description of the dataset structure:

To better comprehend each field, could be useful to look at Figure 4.4. Each one of the values contained in the middle columns, refers to a particular API Call. API calls are

[4]https://www.kaggle.com/datasets/ang3loliveira/malware-analysis-datasets-api-call-sequences/data

Feature	Description
hash	Represents the fingerprint(using MD5 hashing) of the sample analyzed
t_0 to t_99	Contains the ID of a Windows API at a specific timestamp
malware	The label(0 or 1) to identify if the sample is malicious or not

Table 4.2. Dataset feature description



	hash	t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	...	t_91	t_92	t_93	t_94	t_95	t_96	t_97	t_98	t_99	malware
0	071e8c3f8922e186e57548cd4c703a5d	112	274	158	215	274	158	215	298	76	...	71	297	135	171	215	35	208	56	71	1
1	33f8e6d08a6aae939f25a8e0d63dd523	82	208	187	208	172	117	172	117	172	...	81	240	117	71	297	135	171	215	35	1
2	b68abd064e975e1c6d5f25e748663076	16	110	240	117	240	117	240	117	240	...	65	112	123	65	112	123	65	113	112	1
3	72049be7bd30ea61297ea624ae198067	82	208	187	208	172	117	172	117	172	...	208	302	208	302	187	208	302	228	302	1
4	c9b3700a77facf29172f32df6bc77f48	82	240	117	240	117	240	117	240	117	...	209	260	40	209	260	141	260	141	260	1

Figure 4.4. First five samples of the dataset used

reported in the order they were called from the sample, not repeated and truncated to 100 values: all identical contiguous values are considered as a single call, since it is usual for a call to be made dozens of times in succession. Before the building of a classification model, it is possible to see that the information in the dataset is already valuable in its own right. As reported in the Section 3.1.1, each class of API and system library can be traced to a type of required function, and as will show in the following summary, many APIs recur often.

For example, **LdrGetProcedureAddress**, **LdrLoadDll** and **LdrGetDllHandle** are mostly present due to the dynamic linking of the libraries and DLL injection techniques.

The second most present type of API in malware are related to Registry Keys: malware use these functions to ensure persistence and make the infection harder to eradicate. This justifies the presence of APIs such as **RegCreateKeyEx**, **RegOpenKeyEx** and **RegGetValue**. Secondarily other functions required could relate to Encryption functionalities, such as all the API including **Crypt** in the name, or Network functionalities such as the APIs with **Internet** in the name.

The presented set of functions also include shared words, that imply shared behavior. **Ldr** refers to all Loader functions, **Nt** functions are used to run system calls, or **Rtl**, that is instead used for C Run-time library functions. When a call ends with the letter A, means that the output produce will be in an ASCII format. With the letter W, a Unicode output is intended.

Over all the malware samples the following APIs usage were found (see Table 4.3):

In the goodware subset instead, some differences appear almost immediately (see Table 4.4):

Conclusions can be drawn from the above tables :

- Loader functions and DLL functions are present almost with the same frequency.

API Name	**# of occurrences**	**% of usage per sample**
LdrGetProcedureAddress	722904	16.89%
LdrLoadDll	406631	9.50%
RegOpenKeyExW	256080	5.98%
CryptAcquireContextW	12043	0.28%
NtCreateFile	37479	0.87%
InternetOpenA	274	0.006%
InternetSetOptionA	230	0.005%
InternetConnectA	213	0.005%

Table 4.3. Number of occurrence per API in the malware subset

API Name	**# of occurrences**	**% of usage per sample**
LdrGetProcedureAddress	14371	13.31%
LdrLoadDll	7651	7.09%
RegOpenKeyExW	8495	7.87%
CryptAcquireContextW	32	0.02%
NtCreateFile	2177	2.01%
InternetOpenA	1	0.0001%
InternetSetOptionA	1	0.0001%
InternetConnectA	0	0%

Table 4.4. Number of occurrence per API in the goodware subset

- Goodware show less occurrences of Cryptographic operations.
- In goodware, almost no Internet function is present.

These differences are better explained by looking at the heatmap in figure 4.5

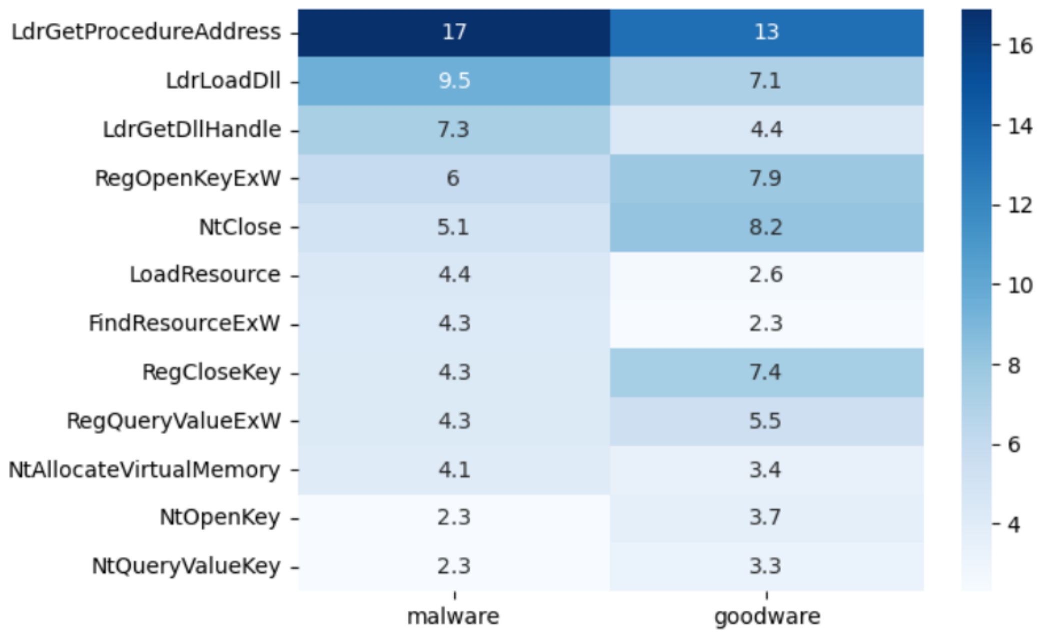


Figure 4.5. Differences between Most common API usage

4.3 LSTM Application

With a dataset complete, and ready to be used, model creation can begin. Given the complexity of the algorithms used in the model study, it was decided to use implementations already provided by well-known libraries set up for neural network generation in Python and data analysis:

- **Keras** [5]: a simple, but powerful, deep-learning API written in Python.
- **pandas**[6]: a necessary data analysis library to handle matrices as data-frame.
- **numpy** [7]: similar to pandas, but used to perform conversions and to handle matrices manipulations.
- **matplotlib** [8]: used as visualization tool to create charts and plots.
- **Scikit-learn** [9]: the most versatile library, considered as a “Jack-of-all-trades” of machine learning.

[5] https://keras.io/

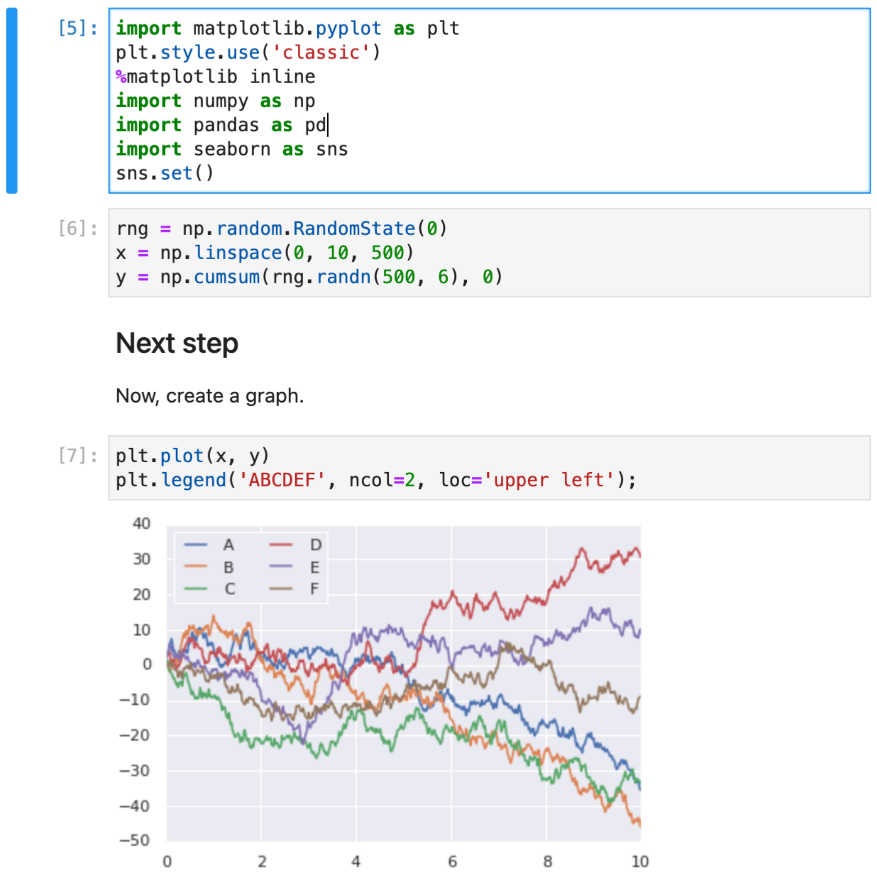
[6] https://pandas.pydata.org/

[7] https://numpy.org/

[8] https://matplotlib.org/

[9] https://scikit-learn.org/stable/

Instead of writing a single script to contain all the execution-flow of the neural network application, it was opted to use Jupyter Notebook [10], a web application local-hosted used to create and share computational documents. Jupyter has been a fundamental part of the research, because the feature included in the notebook helped to "backtrack" rapidly when encountering errors, fix parameters where they were found to be inadequate and visualize the results in the same windows.



```
[5]: import matplotlib.pyplot as plt
     plt.style.use('classic')
     %matplotlib inline
     import numpy as np
     import pandas as pd
     import seaborn as sns
     sns.set()
```

```
[6]: rng = np.random.RandomState(0)
     x = np.linspace(0, 10, 500)
     y = np.cumsum(rng.randn(500, 6), 0)
```

Next step

Now, create a graph.

```
[7]: plt.plot(x, y)
     plt.legend('ABCDEF', ncol=2, loc='upper left');
```

Figure 4.6. A common Jupyter Notebook page [38]

As shown in the Figure 4.6, the notebook provide a cell-oriented programming interface with the possibility to run single cell at a time and the charts are illustrated directly below the cell.

These characteristics make Jupyter one of the first choice when dealing with machine-learning applications.

[10]https://jupyter.org/

Algorithm Initial Settings Before starting to work with a new dataset, it is recommended to extract a percentage of it to form two groups: the two groups will be used to train the neural network and to test its performance.

The action of splitting the database in 2 partitions, sampling a given percentage of rows randomly, is done by using the function *train_test_split*. The percentage of sampled elements can be tuned to enhance the training phase or to test the neural network in a more accurate way. It was decided to start with a standard ratio used in splitting the dataset, 80% for the training set and 20% for the test set. The second step that must be addressed, regards the mean and the variance of the features.

When a neural network operates on unscaled data, it is possible for large inputs to slow down the learning process and could prevent the effectiveness of the network. The most used types of scalers are two: Standardizer or Normalizer.

- **Standard Scaler:** Mostly used for features characterized by negative values, useful to arrange data in a standard normal distribution. Should generally be preferred in classification models.
- **Normalizer:** Recommended for positive values with high magnitude and range, especially in regressive models.

For what has been said, the first was chosen to produce a Proof-of-Concept.

As the reader will observe in this chapter, all the tweakable parameters are chosen from an objective analysis of performance. But generally, what are the parameters?

- **Batch size**: groups of samples that are processed at a time.
- **Epochs**: an epoch is an iteration in which the dataset has passed through the LSTM entirely.
- **Optimizers**: families of algorithms used to find the best parameters and weights.

Epochs are necessary because this type of algorithm work better by using multiple iterations and updating weights with multiple passes. Using a single epoch, the model would lead to a phenomenon called "underfitting".

As it is possible to notice from the figure 4.7, underfitting and overfitting phenomenons may happen whenever a predictive model is outlined. The first regards an overly simplistic model, where both training and testing underperforms and results present errors in both phases. Overfitting, on the other hand, happens when the model fits too closely to training data, and underperforms in testing data because of its high variance.

Given the batch size, the number of iterations is represented by the number of batches needed to complete a single epoch. This information will prove important when the algorithm is to be set to improve its efficiency.

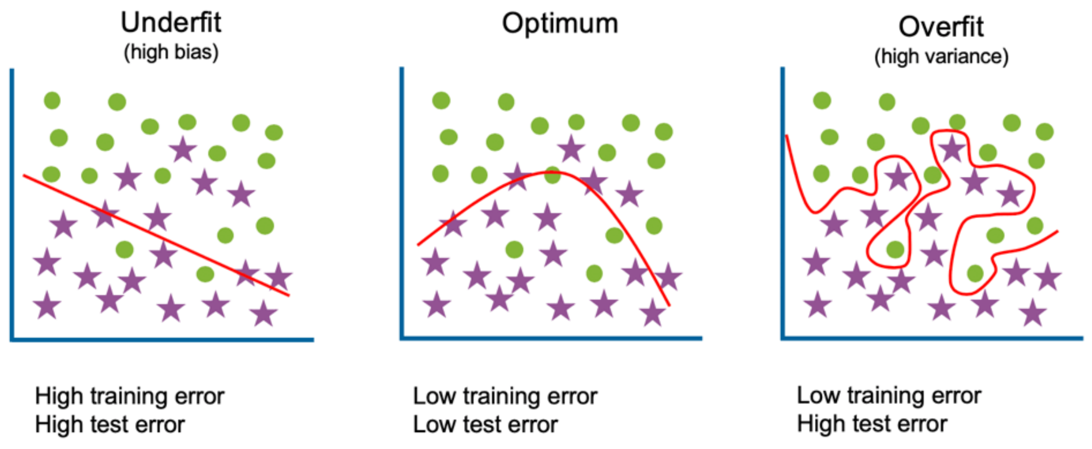


Figure 4.7. Visualization of Underfitting and Overfitting Phenomenon [52]

LSTM Optimizers In terms of improving the performance of a machine-learning algorithm, Optimizers play a crucial part. Optimizers are families of algorithms used to find the best parameters and weights that minimize the error when evaluating the results between inputs and outputs, and doing so, to minimize the loss function (see Figure 4.8).

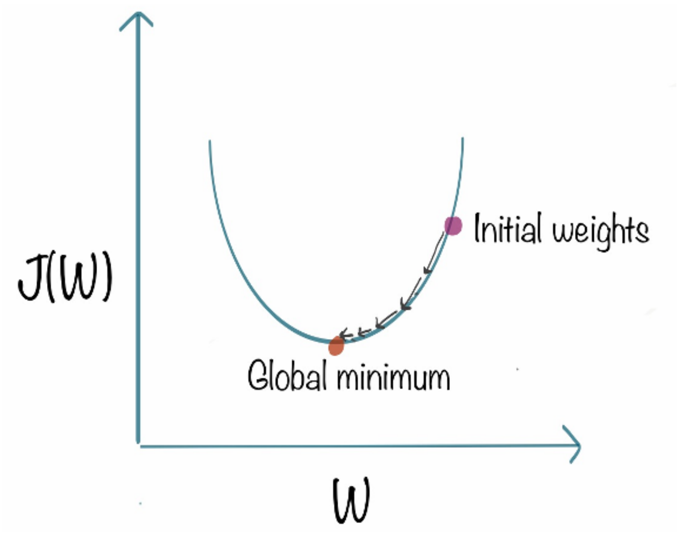


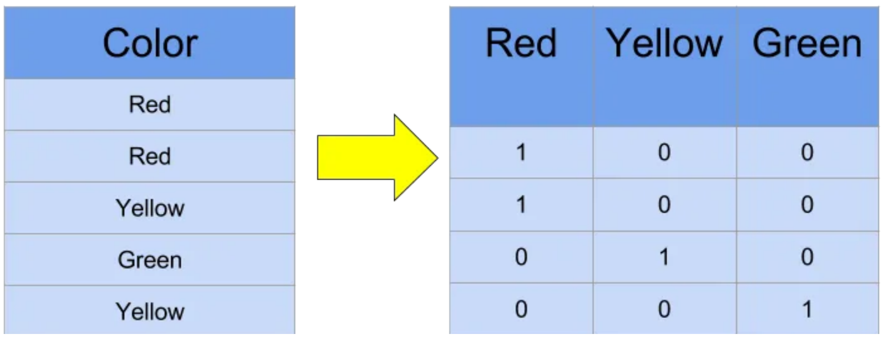
Figure 4.8. Plot of a Loss Function [2]

Minimizing the loss function relies on the process of quantifying the error between a prediction and a target value. The most basic types of loss function are based on simple statistical formulas, like MSE (Mean Squared Error) or MAE (Mean Absolute Error), or involving mathematical operations such as logarithms. However, the family of optimizers used in conjunction with an LSTM model are more complicated. The main optimizers are listed below:

- **Gradient Descent**: works by calculating gradients and represents the best choice for most of the purposes, but it becomes more expensive proportionally to the size of the dataset.
- **Stochastic Gradient Descent**: born as remedy to the gradient descend scalability problem, it exploits randomness to calculate gradients over a set of samples instead of using the whole dataset. Still one of the most used.
- **Adagrad (Adaptive Gradient Descent)**: used to adopt an optimizer without the need to define the learning rate, but which can equally exploit the efficiency of the gradient descend family.
- **AdaDelta**: a more robust adaptation of Adagrad. Since in Adagrad the learning rate becomes infinitesimally small, AdaDelta is able to mantain an efficient ability to learn.
- **Adam (Adaptive Moment Estimation)**: extension of Stochastic Gradient Descent that is able to dynamically set learning rates. Similar to Adagrad.

Finally, it is necessary to prepare the labels so that the algorithm can process them to evaluate the training phase and to test the predictive model. The labels then, represented by a unique value that can be 0 or 1 depending on the class they belong to, must be coded in such a way that they become categorical data. One can accomplish this by using the *to_categorical* function offered by Keras as an extension to Numpy, which exploits a type of encoding called "One Hot Encoding".

This technique (see Figure 4.9) relies on the conversion between a single value to a categorical variable as binary vectors. Each integer value is represented as a binary vector with all zero values except the matching index of the integer, that is instead marked with a 1.



Color
Red
Red
Yellow
Green
Yellow

Red	Yellow	Green
1	0	0
1	0	0
0	1	0
0	0	1

Figure 4.9. How "One Hot Encoding" works [49]

4.3.1 Description of the Model

The model, written in Python using Keras as API, was developed with the goal to produce a binary classification algorithm based on the Long Short-Term Memory concept. The first step to be conducted in the decision-making process leading to the construction of the above model concerns the analysis of the cardinality of the input and output:

- The algorithm expects to receive a three-dimension matrix as input for the training phase, and the respective labels must be represented as categorical data.
- Concerning the outputs instead, the model produce a categorical result of cardinality 2.

Given the cardinality of inputs and outputs, it is possible to begin creating the model from the *Sequential* API, provided by Keras, to build a model that by definition must accept a sequence of values. The Sequential model is based on the idea of connecting a defined number of layers, each with a single input and a single output, and only then it is possible to set a proper block on each layer. In the specific case of the work related to this Thesis, each block was represented by a single LSTM cell. Each output then is elaborated by a common block, called Dense. The Dense layer is intended to process a defined number of inputs and produce an output of a lower cardinality.

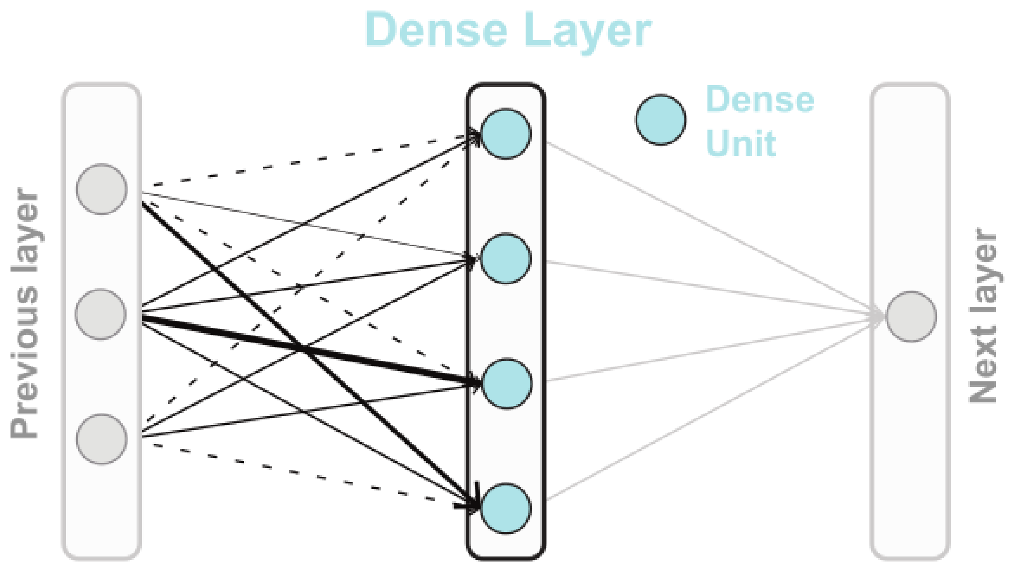


Figure 4.10. Dense Layer schema[28]

Dropout Optionally, it is possible to another layer to prevent the effect of the overfitting behavior that this model could cause. This is called *Dropout*, and it is used to cancel the effect of specific neurons that are excessively trained over the input data.

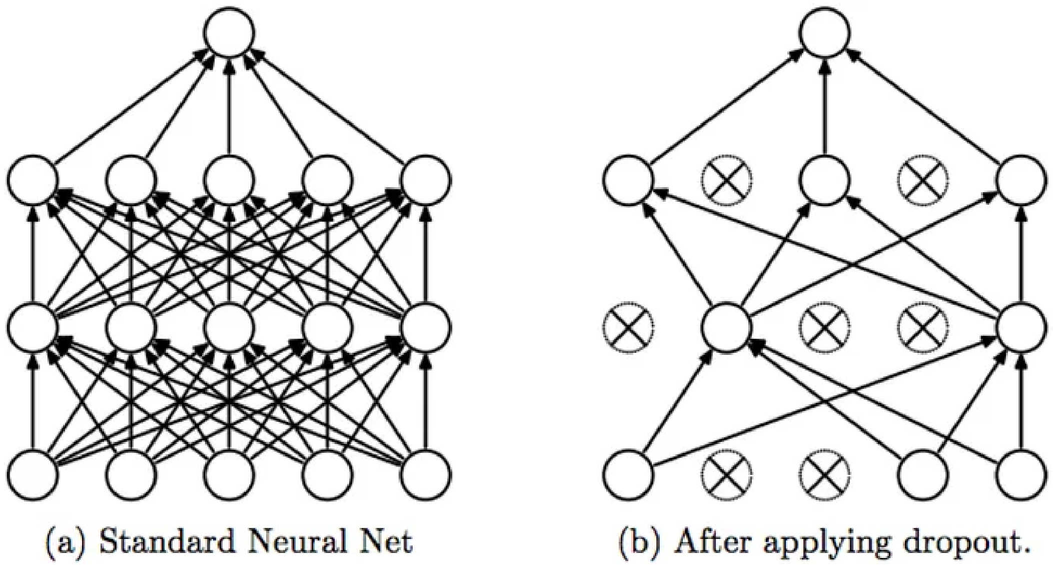


Figure 4.11. Dropout schema[28]

4.3.2 Evaluation of the performance

A key factor in parameters decision is represented by the possibility to try different configurations. The first launch can be done with a default configuration, and then the parameters can be changed for better performance. From an analysis one of the information that can be derived are measurement information called metrics, which similar to loss functions serve to test the model, but unlike the latter are not used directly in training.

To understand how metrics work, a new concept must be defined, called Confusion Matrix (see Figure 4.12). Recalling the definitions of false negatives and false positives introduced in the Section 3.1, Confusion Matrices represent the easiest way to visualize the results of a classification model. The values on the main diagonal are the correctly classified values, while the misclassified (false prediction), are on the anti-diagonal.

Several metrics can be considered in evaluating the model, and all can be derived from one of the four pieces that make up the Confusion Matrix.

Acronym	Full Name
TP	True Positive
TN	True Negative
FP	False Positive
FN	False Negative

Table 4.5. Confusion Matrix Acronyms and Full Names

Considering the acronyms in the table 4.5, the following definitions can be given:

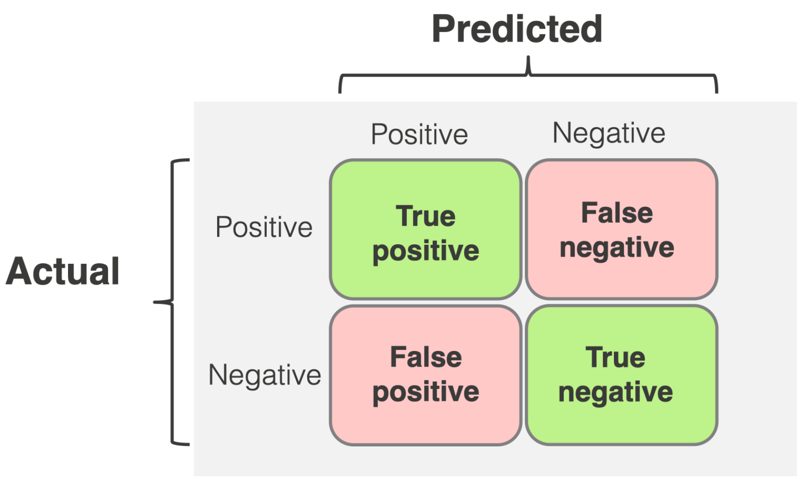


Figure 4.12. Confusion Matrix Schema [35]

Accuracy The Accuracy represents the amount of correct predictions, divided by the sum of the total number of cases. If over a total of 1000 values, the model predicts correctly 900 of them, then the accuracy is 90%.

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

As discussed previously, since the dataset shows a strong imbalance in the distribution of classes, accuracy may not be the optimal metric to evaluate the model. With an unbalanced dataset, accuracy is an inadequate way of evaluating results because it does not give importance to the weight of false positives and/or false negatives, which might be of greater or lesser importance.

Precision Precision is a kind of metric that relies entirely on the rate of the positive class prediction.

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

Since it is based on the results relevant for the positive class, it works well with imbalanced classes when the cost of false positive is high. The higher the precision, the lower are the positive errors.

Recall Also called True Positive Rate, it is used to assess the ability to correctly classify all instances of a particular class.

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

It has a great impact whenever the cost of missing positive samples is high, and acts as a complementary part of Precision since it also assesses the weight of false negatives. Due to this characteristic, increasing Recall might lead to a decrease in Precision.

F1 Score The F1 Score is a metric that combine the previous 2 definitions: Precision and Recall.

$$F1\_Score = \frac{Precision \cdot Recall}{Precision + Recall}$$

Being a formula that combines Precision and Recall, it fits well in problems where the weight of False Positives and False Negatives is equal. As it is possible to see, the above presented formula ignores totally the weight of True Negatives.

ROC and AUC Before describing what a ROC (Receiver Operating Characteristic) curve is, two rates must be introduced: TPR (True Positive Rate) and FPR (False Positive Rate).

$$TPR = \frac{TP}{TP + FN}$$

$$FPR = \frac{FP}{FP + TN}$$

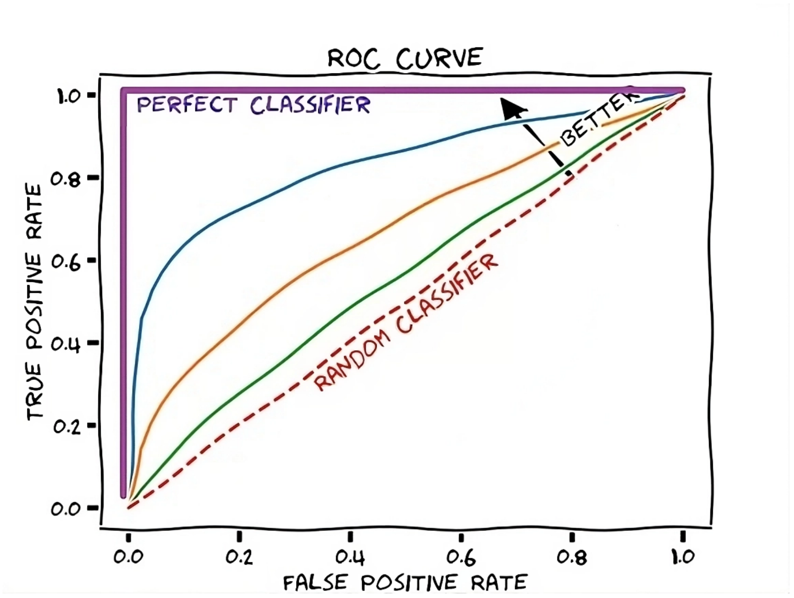


Figure 4.13. ROC Curve [71]

The previous rates can be used as a metric by assigning the FPR to a horizontal axis, the TPR to a vertical axis, and then plot a graph (see Figure 4.13) to test the performance of a classification model and its parameters (such as classification threshold).

The ROC is then used to compute the AUC value. AUC stands for **Area Under ROC Curve**, and as the name suggests, it is obtained by the integral of the curve, to compute the area underneath it. It represents a valuable resource, but it is not optimal when used alone, and moreover works only for binary classification models.

Once all the parameters are explicated, the results can be interpreted and the model can be optimized. The next chapter will explore in depth the achieved results to assess the performances of the algorithm in the presented context.

Chapter 5

Experimental Results

The investigation conducted began with the aim of finding out whether information inherent in unknown files could be used to understand their degree of maliciousness with as little error as possible. Before the advent of machine learning in the context of malware analysis, the techniques were shown to be functional but with major flaws:

- Poor adaptability to polymorphism mechanisms
- Insufficient extraction of information from samples
- Low degree of automation in the analysis process

In the following section evidence and results necessary to understand whether Recurrent Neural Networks can bring valuable results will be exhibited.

As presented in the previous chapter, the metrics used to evaluate the performance will be used to generate an estimation of the underfitting/overfitting grade. Keras *fit* method, provide the functionality to test fitting performances over a fixed set of samples for each epoch, and so, generating two tuple.

Given a certain metrics, for example Precision, fit history will provide a first tuple containing $(precision, val\_precision)$, and $loss, val\_loss$. Independently by the used metric, a loss tuple will be generated, to estimate how the loss evolves, while the first tuple describes how the chosen metric varies.

To ensure that the optimal parameters are chosen for model operation, a special library will be used to optimize these values. The library in question is Optuna.

Optuna [1] enables the ability to iterate the behavior of the model with parameters tested in a particular range, and once finished, extract the optimal value to use. The presented behavior fills the part of the optimization needed for a machine learning algorithm to draw the best conclusions.

This branch falls under the name **Hyperparameter Optimization**.

[1]https://optuna.org/

5.1 Hyperparameter Optimization

As explained previously, 4 main parameters need to be tuned to ensure the best performance (see Table 5.1).

Parameter	**Range**
Epochs	5-40
Batch Size	10-128
LSTM Depth level	1-3
Dropout	0, 0.1, 0.2, 0.4

Table 5.1. Hyperparameters in the scope

Thanks to Optuna, it is possible to conduct a runtime analysis to decide which value could be chosen, and it works by using 3 components:

- **Study**: The Object containing the studied case, the results and the weights used.
- **Trial**: The Instance of the case, with the configuration values used to run the analysis.
- **Objective**: The Objective represents the brain of the Optuna test, it contains the code used to build the model and an iterable value that is chosen by the module itself.

The library becomes useful expecially because of its implementation. The Objective function is sent to Optuna APIs (see Figure 5.1), that create abstract workers in a multithreaded environment, and each worker is able to comprehend if the current path needs to be pruned (skipped) or evaluated.

To draw a better overview about model performance, it may be useful to run some brief tests to illustrate the obtained results, even if the model can still be seen as a draft.

5.1.1 No Optimization

The first attempt will see the model composed by only a single LSTM layer, a single Dense layer, a batch size of 128 and 15 epochs.

		Predicted Values		
		Goodware	Malware	Total
Actual Values	Goodware	74	124	198
	Malware	15	8563	8579
	Total	89	8687	8776

Table 5.2. Confusion Matrix with no optimization

The classification results reported in table 5.2 show a high accuracy, with a 98.8% of total accuracy, but as it is shown, the false negatives value is high, while the false positive is low.

An important aspect to take into account when reviewing false positives and false negatives is their value. Depending on the context, a false positive can carry enormously

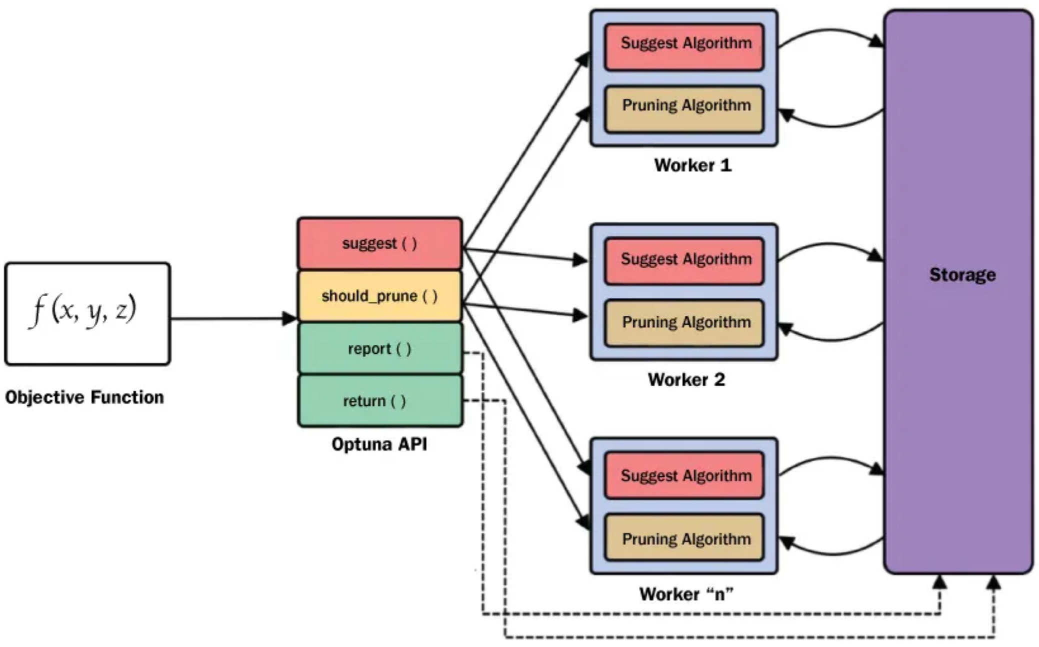


Figure 5.1. Optuna Schema [50]

more weight than a false negative and vice versa. Given the context, it would be more appropriate to consider a decrease in misclassified malware more valuable than misclassified goodware.

Metric	**Malware Class %**	**Goodware Class %**
F1-Score	0.98	0.52
Precision	0.98	0.83
Recall	0.99	0.37

Table 5.3. Result metrics with no optimization

Metrics values shown in Table 5.3, represent the significant imbalance in the obtained results, which, however, results not only from a lack of optimization but also from a lack of sufficient goodware samples to balance the dataset. In any case, even without considering these factors, the results look promising, not because of overall accuracy but taking into account goodware class precision and weighted accuracy.

Looking at Goodware metrics indeed, it is possible to notice a high precision at the expense of a low recall. This means that the class is not well detected, but when it is, the model is reliable.

5.1.2 Epochs optimization

The first step in the optimization phase is to adjust the number of epochs needed to train the algorithm. Keeping the other parameters stable, different values should be tested to analyze the classification performance: therefore, it was chosen to keep the batch size at 128 samples, a single layer of LSTM and dropout at zero. Optuna is then asked to run Trials to achieve the goal with the greatest outcome, in this case, maximizing Precision. Given a timeout and a maximum number of trials, Optuna is able to produce an approximate value that should be used to improve the performance.

In Figure 5.2 is displayed the process of optimization for each trial.

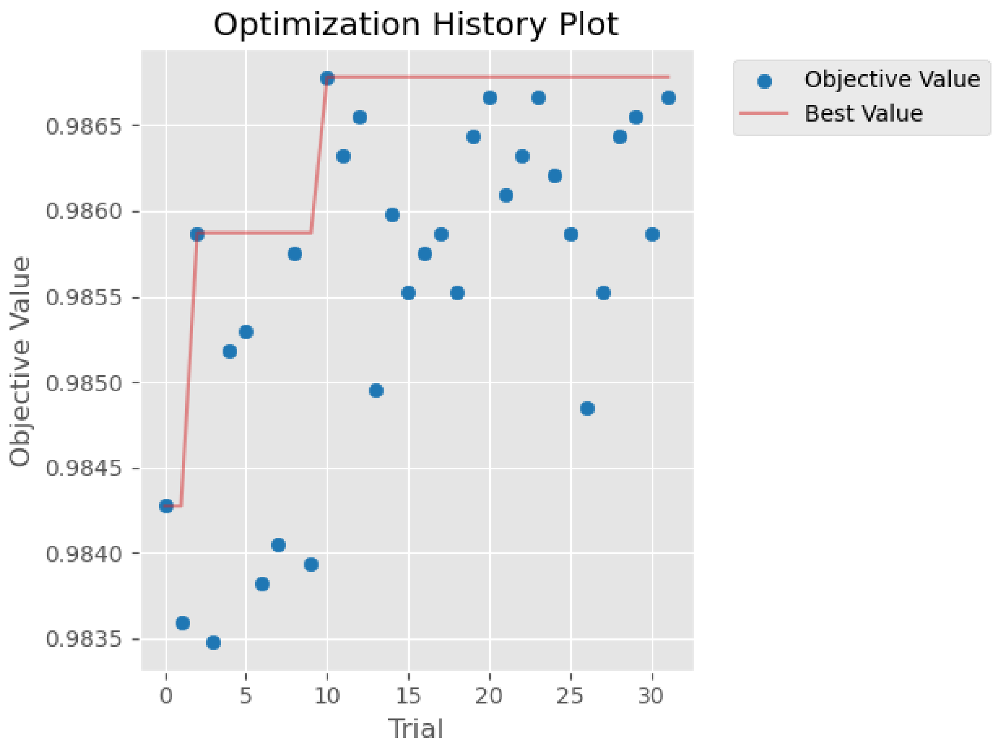


Figure 5.2. Optuna Optimization plot

This process will then result in the decision of the number of the epochs that will most probably enhance the obtained results. In Figure 5.3 is it possible to see the illustrated results of the decision.

The point with the maximum precision obtained is in line with the value 37, but each value nearby have a great impact on the performance. After each optimization, the algorithm can be tested to ensure that the classification method improved.

Table 5.4 present the results obtained by repeating the training with 37 epochs, 128 samples per batch, a single LSTM layer and no dropout.

Multiple considerations can be drawn from what emerged in the last test:

- F1-Score increased because of an improvement in Recall. Now the model is more

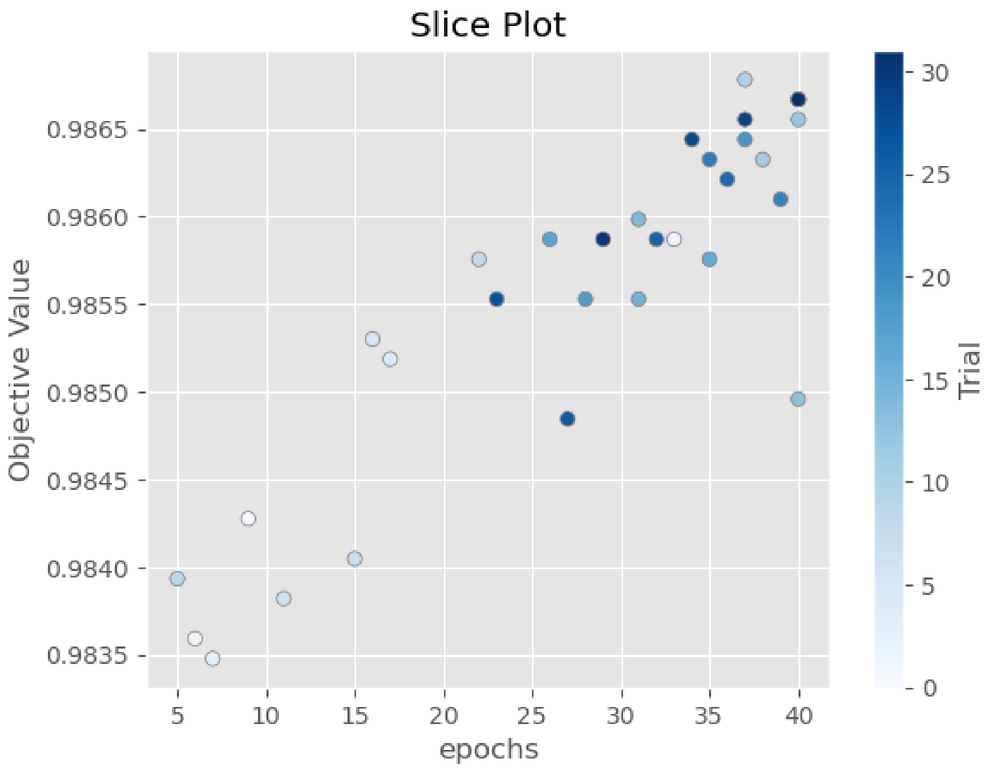


Figure 5.3. Optuna Epoch Slice plot

		Predicted Values: Goodware	Predicted Values: Malware	Total
Actual Values	Goodware	66	54	120
Actual Values	Malware	23	8633	8656
	Total	89	8687	8776

Table 5.4. Confusion Matrix with optimized epoch value

capable of distinguish between the two classes, returning most of the relevant results (Table 5.5).

- Goodware Class Precision decreased, but overall the Precision average increased (Table 5.5).
- The Hyperparameter optimization provided an enhancement of the performance, even if not all optimizations have been performed.
- The number of True Positives classifications increased (Table 5.4).
- True Negatives number decreased by an almost insignificant percentage factor (Table 5.4).

Metric	Malware Class %	Goodware Class %
F1-Score	0.99	0.63
Precision	0.99	0.74
Recall	0.99	0.55

Table 5.5. Result metrics with epoch number optimization

5.1.3 Batch size optimization

The second Trial tested in Optuna is dedicated to batch size optimization: as already presented, the batch size is used to divide the training set in batches that represent groups that are elaborated together. Differently from Epochs value, generally the lower the batch size, the more accurate is the classification model. However, this over-specialization, may lead to an overfitting behavior, and so, the improvement in the model could be seen only in training data and not in test labels.

Optuna has been set to select the best value as batch size to maximize the precision.

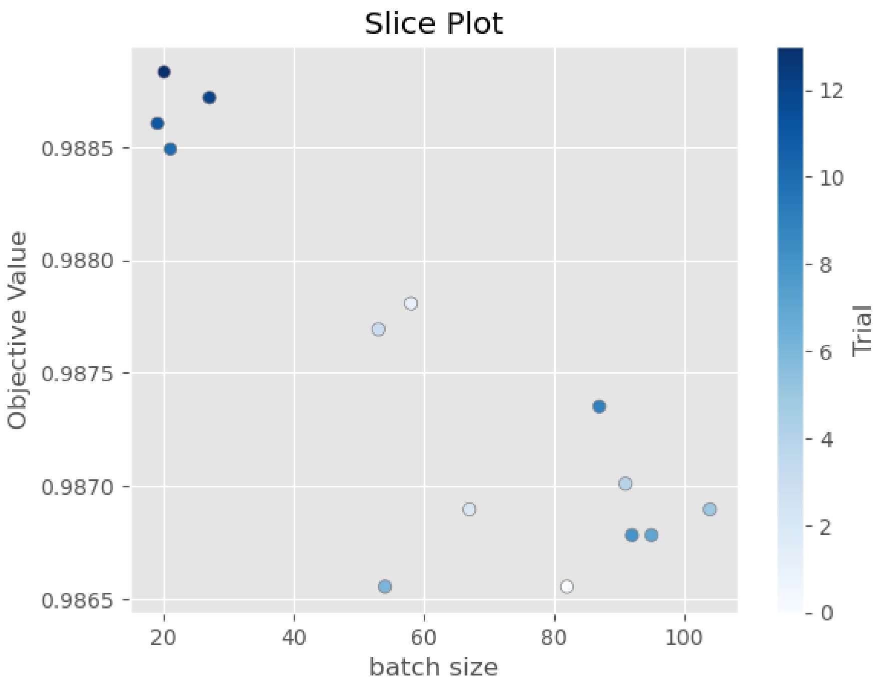


Figure 5.4. Optuna Batch Size Slice plot

The Figure 5.4 is used to visualize how the test parameters perform. In details, the slice plot has on its x-axis the range of values that are tested, and on the y-axis the precision values obtained. The higher the point, the better the performance.

		Predicted Values		
		Goodware	Malware	Total
Actual Values	Goodware	76	39	115
	Malware	13	8648	8661
	Total	89	8687	8776

Table 5.6. Confusion Matrix with optimized batch size

The newer confusion matrix (Table 5.6) depicts how much the results changed after implementing the change obtained from the test.

Metric	**Malware Class %**	**Goodware Class %**
F1-Score	0.99	0.74
Precision	0.99	0.85
Recall	0.99	0.66

Table 5.7. Result metrics with batch size optimization

From the previous Tables (5.6 and 5.7), that represent the classifications results and the metrics obtained from them, after a comparison with previous results, remarkable differences are shown. The Precision, that is the ratio between the number of correct predictions and the overall predictions for a specific class, generally increased by 0.11. Recall instead, is used to present the ratio between the number of correct predictions and the number of all the actual samples in a class. Finally, the F1-Score is a factor derived from both previous metrics to show the performance of both with a single value.

5.1.4 Depth optimization

Optuna represent an extremely useful tool when the parameter is a value that vary in a certain range, reason why it has been successfully used to find the optimal values in the previous cases, but which cannot be used to determine how many layers of LSTM are needed. Three main tests, using respectively one, two and three LSTM layers, were conducted to compare how the training curve changes and see if these changes transfer any improvement in the testing phase of the classification model. The main metric used to evaluate and compare the three cases is the Precision, to maintain consistent the analysis using the same metric used for the other parameters.

In Figure 5.5, 5.6 and 5.7 are presented the Precision obtained in the three cases.

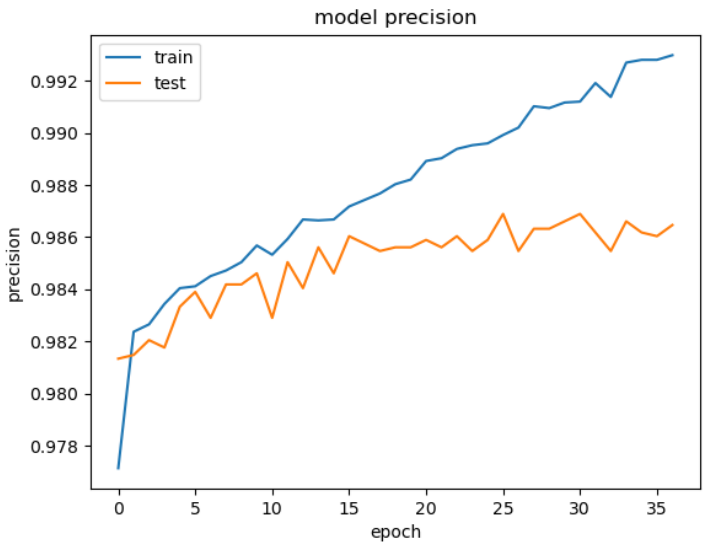


Figure 5.5. Precision obtained with a single LSTM layer

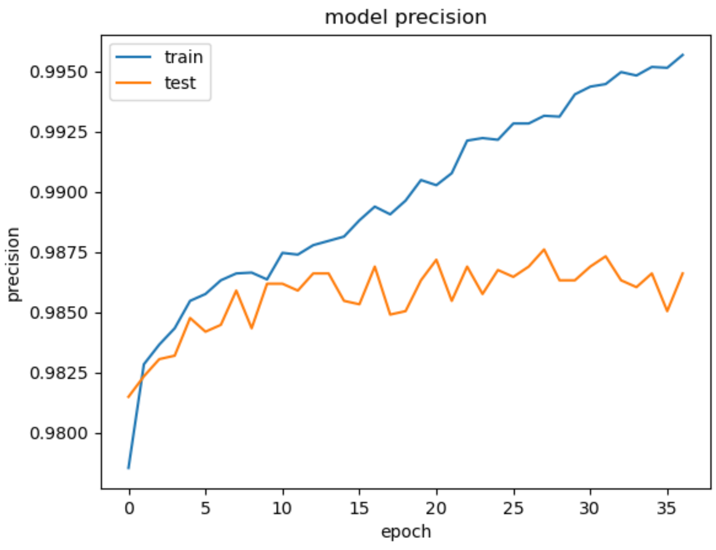


Figure 5.6. Precision obtained with two LSTM layers

As it is possible to see in Figures 5.5, 5.6 and 5.7, the test performance (represented in orange) seems to saturate at the same level in both three cases, meanwhile the estimated performance level in training phase reach almost the same value regardless of the case. Given the lack of substantial improvement in the results obtained and the significant increase in the computational complexity of the added layers, it was decided to maintain a

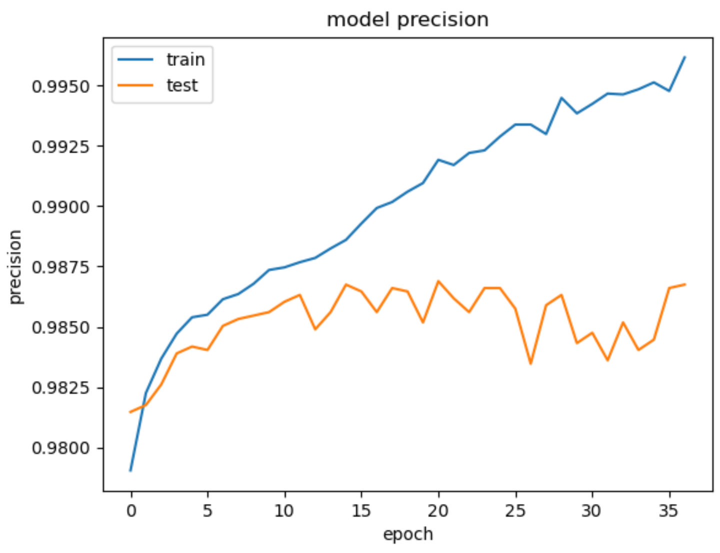


Figure 5.7. Precision obtained with three LSTM layer

single active layer.

5.1.5 Dropout optimization

The same process was repeated straightforward to evaluate the adoption of a Dropout threshold to adjust any possible damaged caused by an underlying overfitting phenomenon. The depth analysis of the LSTM layers, indeed, showed a maximum threshold of capacity growth in the algorithm in the testing phase that diverges from the increase instead obtained in the training phase. Dropout could force the algorithm not to rely on single, over-specialized, neurons and thus force it to find other weights and connections with which to learn.

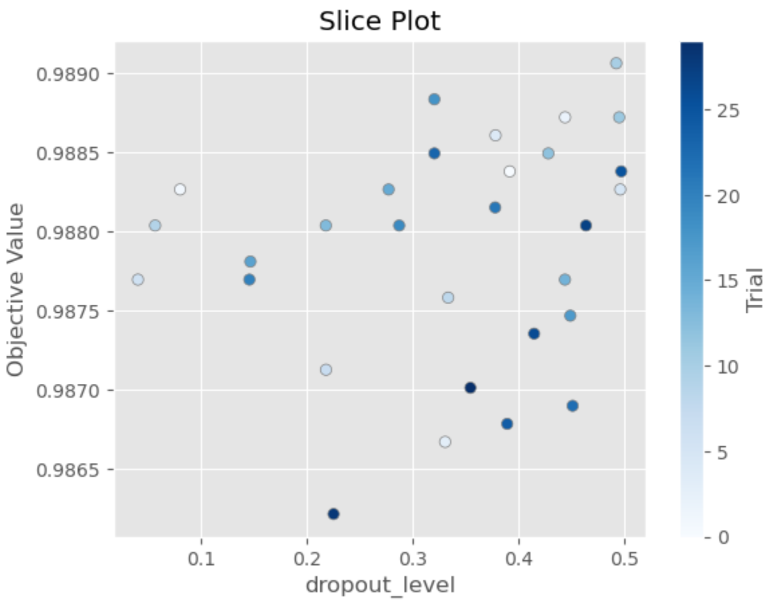


Figure 5.8. Optuna Dropout Slice plot

The selective process produced by the module used led to the choice of a Dropout value of 0.49. Once the dropout value is chosen by the Optimization algorithm (see Figure 5.8), it is possible to proceed and analyze in depth the results obtained.

5.2 Final Results

To recapitulate, the parameters chosen are listed in Table 5.8

Parameter	**Range**
Epochs	37
Batch Size	18
LSTM Depth level	1
Dropout	0.49

Table 5.8. Hyperparameters in the scope

These values should allow the model to reach its best classification performance in terms of training cost and prediction metrics. After repeating the training phase with the chosen parameters the results obtained are reported in Tables 5.9 and 5.10.

		Predicted Values		
		Goodware	Malware	Total
Actual Values	Goodware	76	37	113
	Malware	13	8650	8663
	Total	89	8687	8776

Table 5.9. Confusion Matrix with optimal parameters

Although the model presented excellent results even without optimizations, it is possible to see how much performance has improved, and what new considerations can be made.

- Initially, the model was so specialized in identifying the class of malware that it could not effectively recognize goodware. Goodware class recall indeed increased by 30%, passing from 37% to 67% (see Table 5.10). This means that the model learned to better recognize the difference between the two classes.
- The average Precision of previous attempts, before tuning all the parameters, was around 80% for the goodware class, and increased to 85%, meaning that less samples were missclassified as Malware.
- Due to the increase in both the previous metrics, the F1-Score increased coherently.
- Overall, the malware class had a lower increase in its metrics, but less malware were classified as goodware.

Generally the improvements and the achieved results seems interesting, but as can be seen from Table 5.10, the model still presents high loss in goodware's classification.

- With a relatively low recall value such as 0.67, it can be seen that the model is still unable to effectively recognize samples belonging to the goodware class.
- On the other hand, between all the samples labeled as goodware 85% were correctly classified, obtaining a wrong result only for 13 out of 89 cases (see Table 5.9).

Metric	**Malware Class %**	**Goodware Class %**
F1-Score	0.99	0.75
Precision	0.99	0.85
Recall	0.99	0.67

Table 5.10. Result metrics with optimal parameters

To summarize, the evolution of each metrics for the goodware class is shown in Table 5.11.

Goodware Class Opt	TP	TN	FP	FN	Precision	Recall	F1-Score
No-Optimization	8563	74	124	15	0.83	0.37	0.52
After Epochs opt.	8633	66	54	23	0.74	0.55	0.63
After Batch Size opt.	8648	76	39	13	0.85	0.66	0.74
Final Version	8650	76	37	13	0.85	0.67	0.75

Table 5.11. Metrics improvements for goodware Class

Although the results regarding the classification of the Malware class had marginal growth with respect to an already very good starting point, improvements were manifested, and these are presented in the following Table 5.12. The percentages did not increase as substantially as in the case of the other class, but it can be seen that from the first version to the optimized version, false positives decreased from 124 to 37, even if the false positive value remained unchanged.

Malware Class Opt	TP	TN	FP	FN	Precision	Recall	F1-Score
No-Optimization	8563	74	124	15	0.9858	0.9982	0.9920
After Epochs opt.	8633	66	54	23	0.9937	0.9973	0.9955
After Batch Size opt.	8648	76	39	13	0.9955	0.9984	0.9970
Final Version	8650	76	37	13	0.9957	0.9984	0.9971

Table 5.12. Metrics improvements for malware Class

Chapter 6

Conclusions and Future Work

This Thesis presents a comprehensive exploration into the use of advanced machine learning techniques, specifically Long Short-Term Memory (LSTM) networks, to improve the classification and analysis of malware, a critical concern in cybersecurity. As malware continues to evolve in complexity, traditional detection methods may be not sufficient, necessitating the development of more dynamic and adaptable solutions. The thesis addresses the escalating threat posed by malware, not only to individual users but also to large organizations and the infrastructure of the internet itself. It highlights significant incidents where malware has caused substantial damage, illustrating the need for more effective countermeasures.

The study provides an in-depth look at traditional malware analysis techniques, including static and dynamic analysis, outlining their methodologies and limitations. Static analysis involves examining the malware without executing it, focusing on code analysis and signature detection. However, it struggles with obfuscated or packed malware, which can hide its true nature. Dynamic analysis, on the other hand, involves executing malware in a controlled environment to observe its behavior, but it faces challenges such as detecting the analysis environment and dealing with sophisticated evasion techniques. The limitations of these traditional approaches highlight the need for more advanced methods capable of adapting to the evolving nature of malware.

This Thesis tries to explain that machine learning, particularly neural networks like LSTM, offers a promising solution to these challenges. LSTM networks, a type of RNN, are well-suited for analyzing sequential data, making them ideal for understanding the complex behaviors exhibited by malware. These networks can learn from large datasets of malware samples, identifying patterns and anomalies that might indicate malicious intent. The research outlines the process of implementing LSTM networks for malware classification, from setting up the analysis environment and selecting appropriate features to training and optimizing the model. The study emphasizes the importance of feature extraction in machine learning, where meaningful attributes of the malware samples are identified and used as inputs for the model. Features such as API calls, network traffic, and file characteristics are considered, as they provide valuable insights into the malware's behavior.

The experimental results of this Thesis demonstrate the effectiveness of LSTM networks in classifying malware. Through a series of optimizations, including adjusting hyperparameters like epochs, batch size, and dropout rates, the model's accuracy and reliability are significantly improved. The results show that LSTM networks can effectively distinguish between benign and malicious software based on behavioral analysis, outperforming traditional analysis methods. The research also addresses challenges encountered during the model's development, such as overfitting and the need for a balanced dataset to train the model effectively.

In conclusion, this work highlights the potential of LSTM networks to revolutionize malware classification and analysis. By leveraging the capabilities of machine learning, cybersecurity professionals can develop more resilient defenses against the ever-changing landscape of malware threats. The research acknowledges the limitations of the current study and suggests avenues for future work, including exploring different neural network architectures, incorporating more diverse datasets, and refining the model to improve its scalability and real-time detection capabilities. This thesis contributes to the ongoing effort to enhance cybersecurity measures and provides a foundation for further research in the field of machine learning-based malware analysis.

Multiple insights can be drawn from the work done in This thesis to outline future work: indeed, there are many deficient points that can be improved. The starting point probably concerns the dated analysis environment that performs the feature extraction part in virtual machines of operating systems that are no longer considered a target by the vast majority of today's malware, as they represent disused machines.

The search for automated analysis tools continues to move forward, and one might consider building a feature dataset from analysis on more complex machines. It would make sense to direct future research toward the creation of a more balanced dataset with respect to the one used to produce the Proof-Of-Concept of this Thesis: in fact, a heavily unbalanced dataset was used, given the large presence of malicious samples but the almost complete absence of benign samples (about 2 percent of the entire dataset). Indeed, by remedying this problem, one could assess how the classification model would behave while avoiding overspecialization in a particular class such as malware.

Finally, since machine learning is an exponentially growing field of research, it would be worthwhile to evaluate approaches other than the neural network used, which could adapt more naturally to the characteristics of the features used.

The research presented is also limited by the intended goal: there has been a focus on binary classification, whereas consideration could be given to shifting the focus to a multi-class type of classification, to distinguish samples based on the Malware family to which they belong.

Bibliography

[1] *21st Century Cyber Threats and the Middle East Dilemma*. Accessed on 11 Jan, 2024. URL: https://www.researchgate.net/figure/Stuxnet-infection-mechanism-using-USB-drive_fig2_259583202.

[2] *A Comprehensive Guide on Optimizers in Deep Learning*. Accessed on 15 Jan, 2024. URL: https://www.analyticsvidhya.com/blog/2021/10/a-comprehensive-guide-on-deep-learning-optimizers/.

[3] National Security Agency. *Ghidra*. [Online; accessed 11-January-2024]. URL: https://github.com/NationalSecurityAgency/ghidra.

[4] Abdelouahab Amira, Abdelouahid Derhab, Elmouatez Billah Karbab, and Omar Nouali. «A Survey of Malware Analysis Using Community Detection Algorithms». In: *ACM Comput. Surv.* 56.2 (2023). Accessed on 15 Jan, 2024. ISSN: 0360-0300. DOI: 10.1145/3610223. URL: https://doi.org/10.1145/3610223.

[5] Aditya Anand. *Malware Analysis 101 - Basic Static Analysis*. Accessed on 15 Jan, 2024. 2019. URL: https://infosecwriteups.com/malware-analysis-101-basic-static-analysis-db59119bc00a.

[6] *any.run*. Accessed on 15 Jan, 2024. URL: https://bazaar.abuse.ch/.

[7] Marie Baezner and Patrice Robin. *Stuxnet*. en. Report 4. Accessed on 11 Jan, 2024. Zurich, 2017-10-18. DOI: 10.3929/ethz-b-000200661.

[8] Mayank Banoula. *What is Perceptron: A Beginners Guide for Perceptron*. Accessed on 13 Jan, 2024. 2023. URL: https://www.simplilearn.com/tutorials/deep-learning-tutorial/perceptron.

[9] Sunitha Basodi, Chunyan Ji, Haiping Zhang, and Yi Pan. «Gradient amplification: An efficient way to train deep neural networks». In: *Big Data Mining and Analytics* 3.3 (2020). Accessed on 15 Jan, 2024, pp. 196–207. DOI: 10.26599/BDMA.2020.9020004.

[10] Y. Bengio, P. Simard, and P. Frasconi. «Learning long-term dependencies with gradient descent is difficult». In: *IEEE Transactions on Neural Networks* 5.2 (1994). Accessed on 15 Jan, 2024, pp. 157–166.

[11] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Accessed on 15 Jan, 2024. Berlin: Springer, 2006.

[12] Joe Security's Blog. *Generic Unpacking Detection*. Accessed on 15 Jan, 2024. 2018. URL: https://www.joesecurity.org/blog/8506317946374998489.

[13] *Browse Privately. Explore Freely.* Accessed on 11 Jan, 2024. URL: https://www.torproject.org/.

[14] Josué Carvajal. «An overview of malware components». In: (). Accessed on 11 Jan, 2024. URL: https://www.linkedin.com/pulse/overview-malware-components-josu%C3%A9-carvajal/.

[15] Ferhat Ozgur Catak, Ahmet Faruk Yazı, Ogerta Elezaj, and Javed Ahmed. «Deep learning based Sequential model for malware analysis using Windows exe API Calls». In: *PeerJ Computer Science* 6 (2020), e285. ISSN: 2376-5992. DOI: 10.7717/peerj-cs.285. URL: https://doi.org/10.7717/peerj-cs.285.

[16] Lorenzo Cavallaro, P. Saxena, and R. C. Sekar. «On the Limits of Information Flow Techniques for Malware Analysis and Containment». In: *International Conference on Detection of intrusions and malware, and vulnerability assessment.* Accessed on 15 Jan, 2024. 2008. URL: https://api.semanticscholar.org/CorpusID:7068075.

[17] Yihang Chen. «MalCommunity: A Graph-Based Evaluation Model for Malware Family Clustering». In: *Data Science.* Accessed on 15 Jan, 2024. Singapore: Springer Singapore, 2018, pp. 279–297. ISBN: 978-981-13-2203-7.

[18] Chet Hosmer. *Polymorphic & Metamorphic Malware.* [Online; accessed 11-January-2024]. 2008. URL: https://www.blackhat.com/presentations/bh-usa-08/Hosmer/BH_US_08_Hosmer_Polymorphic_Malware.pdf.

[19] CrowdStrike. «2023 GLOBAL THREAT REPORT». In: *CrowdStrike* (2023). [Online; accessed 9-January-2024], p. 42.

[20] *Cuckoo REST API.* Accessed on 15 Jan, 2024. URL: https://cuckoo.readthedocs.io/en/latest/usage/api/.

[21] *Cuckoo Sandbox.* Accessed on 11 Jan, 2024. URL: https://github.com/cuckoosandbox/cuckoo.

[22] *Cuckoo Sandbox.* Accessed on 11 Jan, 2024. URL: https://cuckoo.readthedocs.io/en/latest/.

[23] Andrea D'Agostino. *Introduction to neural networks — weights, biases and activation.* Accessed on 13 Jan, 2024. 2021. URL: https://medium.com/mlearning-ai/introduction-to-neural-networks-weights-biases-and-activation-270ebf2545aa.

[24] Manuel Egele, Theodoor Scholte, Engin Kirda, and Christopher Kruegel. «A survey on automated dynamic malware-analysis techniques and tools». In: *ACM Comput. Surv.* 44.2 (2008). Accessed on 15 Jan, 2024. ISSN: 0360-0300. DOI: 10.1145/2089125.2089126. URL: https://doi.org/10.1145/2089125.2089126.

[25] Abdullah Al Fayaz. *Uncovering the Threat: The Importance of Malware Analysis in Cybersecurity.* Accessed on 11 Jan, 2024. 2023. URL: https://www.linkedin.com/pulse/uncovering-threat-importance-malware-analysis-abdullah-al-fayaz/.

[26] R. A. Fisher. *Iris.* UCI Machine Learning Repository. Accessed on 15 Jan, 2024. 1988.

[27] *FROM UKRAINE TO THE WHOLE OF EUROPE:CYBER CONFLICT REACHES A TURNING POINT*. Accessed on 15 Jan, 2024. URL: https://www.thalesgroup.com/en/worldwide/security/press_release/ukraine-whole-europecyber-conflict-reaches-turning-point.

[28] *Fully Connected*. Accessed on 15 Jan, 2024. URL: https://epynn.net/Dense.html.

[29] Vijay Gadre. Accessed on 15 Jan, 2024. URL: https://vijaygadre.medium.com/recurrent-neural-networks-a-beginners-guide-16333bd2eeb1.

[30] Ekta Gandotra, Divya Bansal, and Sanjeev Sofat. «Malware Analysis and Classification: A Survey». In: *Journal of Information Security* 05 (Jan. 2014). Accessed on 11 Jan, 2024, pp. 56–64. DOI: 10.4236/jis.2014.52006.

[31] S. Grossberg. «Recurrent neural networks». In: *Scholarpedia* 8.2 (2013). Accessed on 15 Jan, 2024, p. 1888. DOI: 10.4249/scholarpedia.1888.

[32] *Growth in Artificial Intelligence Is Beyond Exponential*. Accessed on 15 Jan, 2024. URL: https://www.legacyresearch.com/the-daily-cut/growth-in-artificial-intelligence-is-beyond-exponential/.

[33] Health and Human Services Cyber Security Program. «Ransomware Trends 2021». In: (2021). [Online; accessed 9-January-2024].

[34] Greg Van Houdt, Carlos Mosquera, and Gonzalo Nápoles. «A review on the long short-term memory model». In: *Artificial Intelligence Review* (2020), pp. 1–27. URL: https://api.semanticscholar.org/CorpusID:218605692.

[35] *How to interpret a confusion matrix for a machine learning model*. Accessed on 15 Jan, 2024. URL: https://www.evidentlyai.com/classification-metrics/confusion-matrix.

[36] Bundesamt für Sicherheit in der Informationstechnik. «Botnets - consequences and protective actions». In: (). [Online; accessed 9-January-2024].

[37] AAG IT. *Latest-Cyber-Crime-Statistics*. https://aag-it.com/the-latest-cyber-crime-statistics/. [Online; accessed 29-November-2023]. 2023.

[38] *Jupyter-Notebook image*. Accessed on 15 Jan, 2024. URL: https://matt-rickard.com/in-defense-of-the-jupyter-notebook.

[39] Debasish Kalita. Accessed on 15 Jan, 2024. URL: https://www.analyticsvidhya.com/blog/2022/03/a-brief-overview-of-recurrent-neural-networks-rnn/.

[40] Kaspersky. Accessed on 15 Jan, 2024. URL: https://encyclopedia.kaspersky.com/glossary/hooking/.

[41] Bart Lenaerts-Bergmans. *WHAT ARE COMMAND AND CONTROL (C&C) ATTACKS?* Accessed on 11 Jan, 2024. 2023. URL: https://www.crowdstrike.com/cybersecurity-101/cyberattacks/command-and-control/.

[42] Mandiant. *What is a malware sandbox?* Accessed on 11 Jan, 2024. URL: https://github.com/mandiant/flare-vm.

[43] John Markoff. *"An Internet Pioneer Ponders the Next Revolution"*. https://archive.nytimes.com/www.nytimes.com/library/tech/99/12/biztech/articles/122099outlook-bobb.html. [Online; accessed 9-January-2024]. 1999.

[44] Per Håkon Meland, Yara Fareed Fahmy Bayoumy, and Guttorm Sindre. «The Ransomware-as-a-Service economy within the darknet». In: *Computers & Security* 92 (2020). Accessed on 11 Jan, 2024, p. 101762. ISSN: 0167-4048. DOI: https://doi.org/10.1016/j.cose.2020.101762. URL: https://www.sciencedirect.com/science/article/pii/S0167404820300468.

[45] Microsoft. *PE Format*. Accessed on 13 Jan, 2024. 2023. URL: https://learn.microsoft.com/en-us/windows/win32/debug/pe-format.

[46] Craig Miles, Arun Lakhotia, Charles LeDoux, Aaron Newsom, and Vivek Notani. «VirusBattle: State-of-the-art malware analysis for better cyber threat intelligence». In: *2014 7th International Symposium on Resilient Control Systems (ISRCS)*. Accessed on 13 Jan, 2024. 2014, pp. 1–6. DOI: 10.1109/ISRCS.2014.6900103.

[47] Nikola Milosevic. «HISTORY OF MALWARE». In: *Digital Forensics* (Aug. 2013). [Online; accessed 9-January-2024].

[48] *MITRE ATT&CK Matrix*. Accessed on 11 Jan, 2024. URL: https://attack.mitre.org/.

[49] *ONE HOT ENCODING AND LABEL ENCODING*. Accessed on 15 Jan, 2024. URL: https://medium.com/@milanbhadja7932/one-hot-encoding-and-label-encoding-3a329481984e.

[50] *Optuna — An Automatic Hyperparameter Optimization Framework*. Accessed on 15 Jan, 2024. URL: https://medium.com/@publiciscommerce/optuna-an-automatic-hyperparameter-optimization-framework-f64638621ff7.

[51] Ori Or-Meir, Nir Nissim, Yuval Elovici, and Lior Rokach. «Dynamic Malware Analysis in the Modern Era—A State of the Art Survey». In: *ACM Comput. Surv.* 52.5 (Sept. 2019). Accessed on 15 Jan, 2024. ISSN: 0360-0300. DOI: 10.1145/3329786. URL: https://doi.org/10.1145/3329786.

[52] *Overfitting and Underfitting in Machine Learning*. Accessed on 15 Jan, 2024. URL: https://www.linkedin.com/pulse/overfitting-underfitting-machine-learning-ml-concepts-com/.

[53] Nagababu Pachhala, S. Jothilakshmi, and Bhanu Prakash Battula. «A Comprehensive Survey on Identification of Malware Types and Malware Classification Using Machine Learning Techniques». In: *2021 2nd International Conference on Smart Electronics and Communication (ICOSEC)*. Accessed on 13 Jan, 2024. 2021, pp. 1207–1214. DOI: 10.1109/ICOSEC51865.2021.9591763.

[54] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. *On the difficulty of training Recurrent Neural Networks*. Accessed on 15 Jan, 2024. 2013. arXiv: 1211.5063 [cs.LG].

[55] Asaf Perlmany. *The Art of Persistence*. [Online; accessed 11-January-2024]. URL: https://www.cynet.com/attack-techniques-hands-on/the-art-of-persistence/.

[56] Rukshan Pramoditha. *The Concept of Artificial Neurons (Perceptrons) in Neural Networks*. Accessed on 13 Jan, 2024. 2021. URL: https://towardsdatascience.com/the-concept-of-artificial-neurons-perceptrons-in-neural-networks-fab22249cbfc.

[57] hex rays. *IDA*. [Online; accessed 11-January-2024]. URL: https://hex-rays.com/ida-free/.

[58] *REMnux: A Linux Toolkit for Malware Analysis*. Accessed on 11 Jan, 2024. URL: https://remnux.org/.

[59] Konrad Rieck, Philipp Trinius, Carsten Willems, and Thorsten Holz. «Automatic analysis of malware behavior using machine learning». In: *Journal of Computer Security* 19 (June 2011). Accessed on 15 Jan, 2024, pp. 639–668. DOI: 10.3233/JCS-2010-0410.

[60] Marc Schiffman. *A Brief History of Malware Obfuscation: Part 1 of 2*. [Online; accessed 11-January-2024]. 2010. URL: http://blogs.cisco.com/security.

[61] Context Information Security. *CAPE*. Accessed on 11 Jan, 2024. URL: https://github.com/ctxis/CAPE.

[62] Syed Shakir Hameed Shah, Abd Rahim Ahmad, Norziana Jamil, and Atta ur Rehman Khan. «Memory Forensics-Based Malware Detection Using Computer Vision and Machine Learning». In: *Electronics* 11.16 (2022). Accessed on 13 Jan, 2024. ISSN: 2079-9292. DOI: 10.3390/electronics11162579. URL: https://www.mdpi.com/2079-9292/11/16/2579.

[63] W Sturgeon. «Net pioneer predicts overwhelming botnet surge». In: *ZDNet News, January* 29 (2007). [Online; accessed 9-January-2024].

[64] Christopher Tarry. *A Brief Survey of Code Obfuscation Techniques*. [Online; accessed 11-January-2024]. 2021. URL: https://chris124567.github.io/2021-06-23-survey-obfuscation/.

[65] Petroc Taylor. *Amount of data created, consumed, and stored 2010-2020, with forecasts to 2025*. https://www.statista.com/statistics/871513/worldwide-data-created/. [Online; accessed 9-December-2023]. 2023.

[66] *"The History of Malware"*. https://advenica.com/en/blog/2023-03-02/the-history-of-malware. [Online; accessed 9-January-2024]. 2023.

[67] *"The history of malware: A primer on the evolution of cyber threats"*. https://www.ibm.com/blog/malware-history/. [Online; accessed 9-January-2024]. 2023.

[68] *The Rise of Ransomware as a Service (RaaS)*. Accessed on 11 Jan, 2024. URL: https://www.linkedin.com/pulse/rise-ransomware-service-raas-trolleyesecurity-biqdc/.

[69] Bergur Thormundsson. *Machine learning - statistics & facts*. Accessed on 13 Jan, 2024. 2024. URL: https://www.statista.com/topics/9583/machine-learning/#topicOverview.

[70] Daniele Ucci and Leonardo Aniello. «Survey on the Usage of Machine Learning Techniques for Malware Analysis». In: *Computers & Security* 81 (Oct. 2017). Accessed on 11 Jan, 2024. DOI: 10.1016/j.cose.2018.11.001.

[71] *Understanding AUC-ROC: Clearly explained*. Accessed on 15 Jan, 2024. URL: https://medium.datadriveninvestor.com/understanding-auc-roc-clearly-explained-74c53d292a02.

[72] VMRAY. *What is a malware sandbox?* Accessed on 11 Jan, 2024. 2023. URL: https://www.vmray.com/glossary/malware-sandbox/.

[73] «What is an Advanced Persistent Threat?» In: (). Accessed on 11 Jan, 2024. URL: https://www.crowdstrike.com/cybersecurity-101/advanced-persistent-threat-apt/.

[74] «What Is an Advanced Persistent Threat (APT)?» In: (). Accessed on 11 Jan, 2024. URL: https://www.cisco.com/c/en/us/products/security/advanced-persistent-threat.html.

[75] Wikipedia contributors. *Cryptographic hash function.* [Online; accessed 11-January-2024]. 2023. URL: https://en.wikipedia.org/w/index.php?title=Cryptographic_hash_function\&oldid=1192131939.

[76] Wikipedia contributors. *Malware — Wikipedia, The Free Encyclopedia.* [Online; accessed 11-January-2024]. 2024. URL: https://en.wikipedia.org/w/index.php?title=Malware&oldid=1194533699.

[77] Wikipedia contributors. *Polymorphism (computer science) — Wikipedia, The Free Encyclopedia.* [Online; accessed 11-January-2024]. 2024. URL: https://en.wikipedia.org/w/index.php?title=Polymorphism_(computer_science)\&oldid=1194747227.

[78] Karlijn Willems. *Machine Learning in R for beginners.* https://www.datacamp.com/. Accessed on 15 Jan, 2024. 2018.

[79] Miuyin Yong Wong, Matthew Landen, Manos Antonakakis, Douglas M. Blough, Elissa M. Redmiles, and Mustaque Ahamad. «An Inside Look into the Practice of Malware Analysis». In: *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security.* CCS '21. Accessed on 11 Jan, 2024. Association for Computing Machinery, 2021, 3053–3069. ISBN: 9781450384544. DOI: 10.1145/3460120.3484759. URL: https://doi.org/10.1145/3460120.3484759.

[80] Ilsun You and Kangbin Yim. «Malware Obfuscation Techniques: A Brief Survey». In: *2010 International Conference on Broadband, Wireless Computing, Communication and Applications.* [Online; accessed 11-January-2024]. 2010, pp. 297–300. DOI: 10.1109/BWCCA.2010.85.