UNIVERSIDADE DE LISBOA

FACULDADE DE CIÊNCIAS

DEPARTAMENTO DE INFORMÁTICA

Malware Analysis with Machine Learning

João Pedro Matias Cruz

Mestrado em Segurança Informática

Dissertação orientada por:
Professor Doutor Vinícius Vielmo Cogo, FCUL

2022

2022

Resumo

Desde o início dos anos 70, com o aparecimento do vírus Creeper, surge uma das maiores ameaças a indivíduos e as organizações por todo o Mundo. São diferentes as motivações que levam à criação de um *malware*, desde a demonstração de vulnerabilidades ou conceitos, aos com intuito malicioso onde o objetivo se foca no roubo de informação com a finalidade de obter um bem maior.

Os ataques de *malware* têm sido um dos maiores riscos na área da cibersegurança nos últimos anos. A cada semana, aumenta o número de vulnerabilidades reportadas na comunidade. Uma das principais causas do crescimento exponencial, é o fato dos atacantes introduzirem alterações por forma a evitar a deteção. O que leva a que ficheiros maliciosos de uma mesma família de *malware*, com o mesmo comportamento malicioso, sejam constantemente modificados ou ofuscados através de técnicas para que aparentem ser diferentes.

O e-mail tornou-se rapidamente num dos principais vetores de ataque contra as organizações, deste modo colocando o servidor de e-mail, como uma das partes integrantes mais críticas de qualquer organização. Estes, não foram desenhados na sua base tendo em conta a segurança, encontrado-se dependentes de outros softwares, tais como os antivírus ou *antispam*.

Atualmente, a detenção de software malicioso é efetuada com recurso a métodos baseados em assinaturas, heurísticos e também através da análise comportamental, no entanto incapazes de acompanhar o crescimento exponencial de *malware*. A aprendizagem automática é uma das áreas de tecnologias de informação em maior crescimento, tirando partido da estatística e inteligência artificial (IA), por forma a fornecer aos sistemas a capacidade de aprender e melhorar autonomamente. Sendo amplamente utilizada em aplicações científicas que requerem a análise de grandes volumes de dados.

A aprendizagem automática surge assim como nova abordagem na categorização de *malware*, através da análise de padrões presentes no código binário ou engenharia inversa. No entanto, a existência de múltiplos padrões dificulta o desenvolvimento de métodos genéricos, que abranjam a grande variedade de cenários possíveis.

Por forma a avaliar e categorizar toda a informação presente nos padrões, é necessário dividir em grupos e proceder à identificação de famílias com base no mesmo comportamento ou características. O código binário do software potencialmente malicioso, é convertido para uma representação em imagem monocromática, possibilitando a identificação de alterações subtis, sem que estas afetem a estrutura global da imagem. A identificação dos padrões é efetuada através de modelos de aprendizagem automática, estes compostos por três camadas, *Convolutional*, *Pooling* e *Fully Connected*. Os modelos tradicionais de aprendizagem automática como o ResNet-50, VGG16 e Google Inception V3, foram dimensionados e treinados tendo em consideração a classificação de imagens como um dos principais objetivos, utilizando bases de dados como a da ImageNet no seu treino e avaliação. As imagens que compõem estas bases de dados, apresentam principalmente objetos distintos e bem definidos. Ao contrário das imagens abstratas produzidas pela transformação do código binário, que apresentam um elevado nível de ruído devido ao processo de conversão.

Com base nos resultados obtidos pelas redes neuronais no desafio ImageNet, esta dissertação apresenta uma abordagem de aprendizagem automática agnóstica, na classificação eficaz de *malware* em famílias, com base na identificação dos padrões gerados nas imagens. Neste sentido, esta dissertação propõe um modelo com foco na avaliação de um elevado número de parâmetros nas imagens, a fim de capturar o máximo de semelhanças possíveis, ao mesmo tempo, tenta minimizar a captura de parâmetros que constituem o ruído causado pela transformação. Por forma a alcançar o melhor resultado possível com o menor número de camadas e o menor tempo de treino.

A base de dados utilizada no Microsoft Classification Challenge, foi usada na análise e avaliação dos modelos. O modelo proposto alcançou uma eficácia semelhante ao melhor modelo dos testes, atingindo-a em um terço do tempo de treino. Complementarmente ao estudo, as imagens foram cifradas homomorficamente e geradas as novas representações em imagens, com intuito de proceder à identificação dos mesmos padrões. Desta forma, a privacidade é garantida, mantendo os dados analisados pelas redes neuronais seguros contra terceiros. No entanto, os resultados demonstram a necessidade de substituir a camada de saída dos modelos tradicionais por funções que consigam avaliar polinômios.

O modelo proposto foi implementado, na solução Malwizard também apresentada nesta dissertação. O Malwizard consiste numa solução Python adaptável para empresas ou utilizadores finais, que tem como objetivo oferecer uma solução rápida na identificação automática de *malware*. Nós acreditamos até onde sabemos, ser a primeira implementação de uma ferramenta automatizada de deteção de *malware*, baseada em aprendizagem automática, direcionada para a análise de *malware* em e-mails. Esta, constituída por dois serviços, uma extensão para o Microsoft Outlook e um serviço de API para plataformas de Orquestração, Automação e Resposta de incidentes (SOAR). O Malwizard foi projetado tendo em vista dois requisitos funcionais e nove não funcionais. A solução disponibiliza dois modos de análise, o não cifrado onde a imagem em análise pode ser revertida para o ficheiro original, e o cifrado onde a imagem é cifrada homomorficamente garantindo a privacidade e a irreversibilidade da mesma para o ficheiro original.

Considerando a extensão para o Microsoft Outlook, esta pode ser instalada através da loja do Office, apartir do URL ou directamente do XML disponibilizado pelo Malwizard. Após configurada, ficará disponível se o Outlook detetar um anexo presente no e-mail que o utilizador esteja a consultar. O Malwizard converte o código binário dos anexos presentes no e-mail, numa representação em imagem e envia-a para a unidade de processamento de aprendizagem automática para ser analisada. Na implementação através da API, o utilizador tem a possibilidade de criar ou adaptar a solução para um propósito específico, como a implementação numa ferramenta de segurança como o TheHive. A API é de código aberto, permitindo deste modo adaptar-se às necessidades dos clientes ou de ferramentas especificas.

A unidade de processamento de aprendizagem automática, é onde o modelo de inteligência artificial se encontra. O servidor recebe a representação em imagem do ficheiro e entrega ao modelo para análise. Por fim, este, retorna o nível de confiança da análise, e se for maior que uma determinada percentagem a definir pelo utilizador, a amostra é considerada para análises futuras e identificada como *malware*. Caso contrário, é marcada como contendo sinais de *malware* e informa o utilizador que deverá proceder com cautela.

A solução como um todo foi alvo de testes e de análise de desempenho, comparando os dois modos de funcionamento diponíveis, cifrado e não cifrado. A criptografia aplicada a imagens é um enorme desafio computacional, uma vez que cada pixel da imagem é cifrado individualmente, aumentando o tempo de processamento. O tamanho das chaves de cifra é um dos fatores que influência diretamente o tempo de cifra, no entanto, este pode ser reduzido se existirem outros mecanismos de segurança adicionais. Desta forma, tornando possível a utilização do modo cifrado em tempo real, considerando a utilização de um modelo capaz de analisar imagens cifradas. A adoção da solução Malwizard aplicada na análise diária de e-mails, tanto de empresas como de utilizadores finais, apresenta-se como uma componente adicional na prevenção de *malware* e como um forte contributo para a comunidade da cibersegurança.

No fim desta dissertação, encontram-se as conclusões obtidas com este trabalho, possíveis trabalhos futuros, além do apêndice com as interfaces e tutoriais de utilização do Malwizard.

Palavras-chave: *Aprendizagem Automática, Visualização de malware, Deteção de malware, Redes neuronais convolucionais, Criptografia Homomórfica.*

Abstract

Malware attacks have been one of the most serious cyber risks in recent years. Almost every week, the number of vulnerability reports is increasing in the security communities. One of the key causes for the exponential growth is the fact that malware authors started introducing mutations to avoid detection. This means that malicious files from the same malware family, with the same malicious behaviour, are constantly modified or obfuscated using a variety of technics to make them appear to be different. Characteristics retrieved from raw binary files or disassembled code are used in existing machine learning-based malware categorization algorithms. The variety of such attributes has made it difficult to develop generic malware categorization methods that operate well in a variety of operating scenarios.

To be effective in evaluating and categorizing such enormous volumes of data, it is necessary to divide them into groups and identify their respective families based on their behaviour. Malicious software is converted to a greyscale image representation, due to the possibility to capture subtle changes while keeping the global structure helps to detect variations. Motivated by the Machine Learning results achieved in the ImageNet challenge, this dissertation proposes an agnostic deep learning solution, for efficiently classifying malware into families based on a collection of discriminant patterns retrieved from its visualization as images.

In this thesis, we present Malwizard, an adaptable Python solution suited for companies or end-users, that allows them to automatically obtain a fast malware analysis. The solution was implemented as an Outlook add-in and an API service for the SOAR platforms, as emails are the first vector for this type of attack, with companies being the most attractive targets.

The Microsoft Classification Challenge dataset was used in the evaluation of the noble approach. Therefore, its image representation was ciphered and generated the correspondent ciphered image to evaluate if the same patterns could be identified using traditional machine learning techniques. Thus, allowing the privacy concerns to be addressed, maintaining the data analysed by neural networks secure to unauthorized parties.

Experimental comparison demonstrates the noble approach performed close to the best analysed model on a plain text dataset, completing the task in one-third of the time. Regarding the encrypted dataset, classical techniques need to be adapted in order to be efficient.

Keywords: *Machine learning, Malware visualization, Malware detection, Convolutional neural network, Homomorphic Encryption.*

Agradecimentos

A realização desta dissertação deve-se ao contributo, apoio e confiança, de inúmeras pessoas. A todos em geral, gostaria de prestar o meu profundo agradecimento.

Agradeço ao meu coordenador e orientador, Engenheiro Valentim Vieira de Oliveira, por me ter aceite como seu aprendiz, pela disponibilidade prestada e interesse na condução deste estudo, pela paciência e amizade.

Ao Professor Doutor Vinícius Vielmo Cogo, meu orientador na Faculdade de Ciências da Universidade de Lisboa, pelo importante e determinante apoio, dedicação, estando sempre disponível, e pelo interesse em contribuir para um produto final melhor.

À Sociedade Interbancária de Serviços (SIBS), aos colegas Gonçalo Pereira, Jerónimo Ferreira, Luís Fonte. Colegas e Professores do Mestrado em Segurança Informática (MSI), com especial agradecimento ao Gonçalo Cordeiro.

Dedico esta dissertação à minha família, sem a qual, a realização desta não teria sido possível. À minha mãe, Teresa Matias, à minha avó, Celina Matias, e à minha companheira Susana Sousa. Em especial, ao meu avô, Luciano Ramos Matias, figura paternal, a quem agradeço o apoio constante e incondicional, que tornou todo o meu percurso possível e a quem devo o que sou hoje.

O meu sincero agradecimento a todos.

Contents

List of Figures

List of Tables

Abbreviations

API	Application Programming Interface
AUC	Area Under the ROC Curve
AVG	Average
CNN	Convolutional Neural Network (CNN / CNNs)
DKIM	Domain Keys Identified Mail
DMARC	Domain-based Message Authentication, Reporting, and Conformance
DNN	Deep neural network
ENISA	European Network and Information Security Agency
FHE	Fully Homomorphic Encryption
FN	False Negatives
FP	False Positives
HE	Homomorphic Encryption
IoT	Internet of Things
ML	Machine Learning
MSD	Mean Squared Deviation
MSE	Mean Squared Error
NN	Neural Network
PHE	Paillier Homomorphic Encryption
ROC	Receiver Operating Characteristics
SIEM	Security Information and Event Management
SOAR	Security Orchestration, Automation and Response
SPF	Sender Policy Framework
TL	Transfer learning
TN	True Negatives
TP	True Positives
URL	Uniform Resource Locator

Chapter 1
Introduction

Malicious software, commonly known as malware, is a continuous problem that has been growing considerably in the last few years. Almost every week, the number of vulnerability reports is increasing in the security communities, which may be seen as a failure. However, due to more advanced and specialized attacks, the current methods of detection are too slow or ineffective on real-time analysis. The huge development of Machine Learning (ML) technologies over different domains [1]-[4] appears as an innovative approach to the detection of more advanced threats through deep learning. This dissertation proposes to assess a faster and efficient method, utilizing ML and Convolutional Neural Networks (CNNs) on malware fingerprint analysis.

1.1 Context and Motivation

Malware is an expression to define different types of malicious software, such as adware, spyware, viruses, worms, trojans, and ransomware, designed to harm or exploit any programmable devices, service, or network. Since the early 1970s, when the Creeper Virus appeared, there has been a lingering threat to individuals and organizations all over the world. Different motivations may lead to malware creation, some are created to show vulnerabilities or concepts, while others have been used to steal information and take some kind of profit or used for blackmail extorsion.

According to European Network and Information Security Agency (ENISA) Threat Landscape 2020 report [5][6], email and phishing messages have become the primary malware infection vector, placing malware at the first position of the top 15 trending threats. Between 2019 and 2020, it is involved in 30 percent of all data breach incidents, from which 46.5 percent were detected in email messages found in Docx file type documents. Also, 71 percent of organizations experienced malware activity that spreads from one employee to another. A more recent study by Kaspersky [7] showed that at least 19.8 percent of computers in the world were subject to at least one malware attack over the last year.

The recent explosion of the Internet of Things (IoT) devices has led to the widespread implementation of a large quantity and variety of systems. Companies are pressured to try to be the first to launch new products over competitors, which justifies why it has been more common to see so many products with software flaws, resulting in more vulnerabilities and dramatically increasing the attack surface. Most of these devices have or allow to access a large amount of sensitive data, becoming an attractive target for malicious parties.

Nowadays, the detection of malicious software is done mainly with *Signature and Heuristic-based* methods, but also through *Behavioural* analysis. However, these methods struggle to keep up with the malware evolution. As opposed to *Signature-based* analysis, which utilizes signatures databases that uniquely identify known malware, *Heuristic-based* methods rely on rules and known patterns to detect known and new forms of malware. The *Behavioural* analysis helps to define and understand the malware capabilities, by tracking the behaviour and looking for suspicious activity in the system. Once the algorithm detects a malware, it places the suspect file in quarantine or subsequently deletes it. While viruses share common behaviours between their families, which allow the creation of a single generic signature for them, on the malware side, cybercriminals try to stay a step ahead of antivirus (AV) software, by developing polymorphic and metamorphic malware that does not match with the known signatures.

Therefore, AV vendors tend to use hybrid analysis, by combining both signature-based and heuristic-based methods to deal with unknown malware. But, they also perform Behavioural analysis through *Static Analysis* (by examining its code) or by *Dynamic Analysis* (through the execution in a safe environment).

The Static Analysis method is based on the analysis of code or patterns to detect signatures, byte-sequences, or operation code's frequency distribution, without executing the program. However, it is dependent on reverse engineering to obtain the original code. The Dynamic Analysis is based on the behaviour analysis during the execution of the malicious program in a controlled and secure environment (e.g., virtual machine, emulator, sandbox, etc.). Compared to the static analysis, it is more precise and does not need code analysis. However, it needs more time and resources when performing on a large scale. Besides, the controlled environment is different from the real environment, where the behaviour of the malware may be only triggered under certain conditions, such as a specific system, command, or sequence of actions. Also, the attackers have implemented protections to detect the environment where it is running, showing a different behaviour and making it hard to detect in a virtual environment.

Machine learning is one of the fastest-growing areas of computer science, using statistics and artificial intelligence (AI) to provide the systems with the ability to automatically learn and improve. It has become widely used in scientific applications that require big volumes of data analysis. People are surrounded by machine learning technologies, such as personal assistance applications on smartphones with voice recognition or face identification, search engines to learn how to provide the best results, antispam software to filter email messages, or on credit card transactions secured by software that learns to detect frauds. In recent years, due to the cheaper computing power, researchers are able to study more complex models and apply them to larger datasets. As a result, AV vendors start to adopt machine learning classifiers to address problems related to logistic regression [8], neural networks [9], and decision trees [10]. In order, to accomplish a new faster reliable method independent of the code or secure environment analysis, but able to detect, analyse and classify malware by family according to the content and behaviour.

1.2 Background of the Problem

An email server is a critical part of any organization, affected by the increasing security issues associated with business email compromise (BEC), phishing, data leakage, and privacy protection, among others. These risks motivate organizations to ensure a secure email environment.

Email servers were not designed to have security as their main concern. Apart from the combined utilization of antispam and antivirus applications, they are limited to grey lists [11], to the Sender Policy Framework (SPF) [12] / Domain Keys Identified Mail (DKIM) [13], and to the Domain-based Message Authentication, Reporting, and Conformance (DMARC) [14] mechanisms. These mechanisms are very effective in detecting forged sender addresses and protecting the domains from unauthorized use, although they are ineffective in detecting malicious content inside the email messages.

Antispam and antivirus applications can perform content analysis. However, if an email message has nothing suspicious, (e.g., a known signature or triggered a heuristic rule), it ends in the inbox after a quarantine analysis. The quarantine period is specified by the organization, taking into account the delay in the message analysis until receiving them into the inbox, which most of the time is not enough to classify malicious content with a high level of confidence. Increasing the quarantine time is not a valid option since there is a huge gap between the detection and when the signatures are available.

Hence, the Human factor is one of the biggest challenges for any organization. Cybercriminals take advantage of people’s curiosity to generate phishing emails that raise interest among users.

A more effective method, capable of detecting unknown malware present in zero-day attacks and able to minimize the dependence on the Human factor to identify malicious content in emails, represents some of the challenges that can be improved with a machine learning solution.

1.3 Problem Statement

Performing and managing security at the email server level, dependent on third-party apps to perform malware analysis, creates a dependency on the security levels, as well as the level of awareness in information security of the user. All these dependency factors, which are also points of failure, represent exposure to several threats that pose different risks to the people and organizations.

Cases of malware attacks targeting large companies and demanding huge payments are becoming more common in the news. Despite all the implemented methods, they are still not enough to prevent some attacks from succeeding, leading to permanent data loss, as well as reputational and financial consequences.

1.4 Aims and Objectives

The purpose of this dissertation is to present a novel approach model and an application service to deal with the malware classification problem. Inspired by experiments and conclusions of Nataraj *et al.* [15] work, in which, through the analysis of the greyscale images arising from the binary code transformation of known malware samples, it was inferred that the images from the same malware family were similar.

With the introduction of Convolutional Neural Networks by Gibert *et al.* [16] to the Nataraj *et al.* approach, it was possible to derive local and invariant features from the image, finding the patterns independently of their position. Thus, allowing the network to detect patterns of known malware present on an image.

This thesis aims to better identify the presence of unknown malware in email content at runtime, decreasing the reliance on known signatures and reducing the assessment and quarantine times. By applying and improving the findings of Gibert *et al.* [16], through the analysis and integration of a high-power predictive model in a similar CNN infrastructure.

Additionally, the development of an application that, as far as we know, is the first implementation of an automated email malware detection tool for the Microsoft Outlook add-in, as well as an API service, that can be used by risk-based Security Orchestration, Automation, and Response (SOAR) platforms such as TheHive [17].

1.5 Contributions

This dissertation's contribution is to identify, define, and assess the characteristics for developing a novel ML model with a set of criteria that will allow a CNN to quickly and accurately recognize and categorize unknown malware. In addition, the CNN will be integrated into a Microsoft Outlook add-in and an API service for SOAR systems, reducing dependency on known signatures and shortening assessment and quarantine times in the email content evaluation.

1.6 Research Method

The research method followed in this dissertation is the Design Science Research methodology, which comprises six activities [18]:

1. *Identification of the Problem and Motivation:* The goal of this dissertation is to propose a solution for automatically and quickly analyse files, such as email attachments, to detect the presence of malware. This is accomplished through the application of machine learning.

2. *Define Objectives for a Solution:* The objective is to find an ML model that gives the best overall performance, taking into account the balance between training time and prediction results. It will be implemented as a Microsoft Outlook add-in and API service solution.

3. *Design and Development:* A novel ML model will be proposed and evaluated in comparison to other existing community models that will be implemented using the Scikit-learn and Keras ML libraries. Simultaneously, the Microsoft Outlook add-in and API service, as well as the supporting infrastructure, will be developed.

4. *Demonstration:* To determine which model provides the highest overall performance, the unique suggested model will be compared to the community models. In addition, the Proof-of-Concept for the Microsoft Outlook add-in and API service solution will be presented in this thesis.

5. *Evaluation:* The performance of the developed model, as well as the community models, will be evaluated using machine learning metrics such as F1-Score, K-Fold tests and Receiver Operating Characteristics (ROC) curves.

6. *Communication:* The mentioned objectives will be accomplished through the completion of this dissertation, which contains the discussion and report of the findings and difficulties encountered during the study and implementation of the solution.

1.7 Description of Thesis Organisation

This dissertation is organized as follows to represent the various stages of the work:

The following chapter, "Chapter 2: Background on Deep Learning", will dive deep into Machine Learning concepts such as Artificial Neural Networks and Convolutional Neural Networks, as well as pre-trained models and most common evaluation mechanisms and metrics.

In “Chapter 3: State of the Art”, the literature is described, starting with the background on malware analysis as images and how to ensure privacy, to the frameworks used with Neural Networks.

"Chapter 4: Malware Classification" describes the dataset utilized throughout the study's execution and conclusion, the architecture of the proposed model, and compares the results to other machine learning algorithms that were assessed to decide which produced the best results. In addition to model testing, the experimental work detailed in this chapter aims to assess if it is possible to evaluate encrypted malware images with a traditional Neural Network approach while maintaining privacy during the evaluation process.

"Chapter 5: Malwizard" presents the implementation of the models in Chapter 4 in an Outlook add-in and an adaptable API backend service. Furthermore, it includes the requirements analysis, specifications, design, development, and components for the solution.

Finally, “Chapter 6: Conclusions” summarises the study and draws final conclusions and recommendations for future research.

Chapter 2

Background on Deep Learning

Neural networks mimic human brain function, allowing computer programs to discover patterns and solve common challenges in AI, ML, and deep learning. Deep Learning is the leading topic in the AI and ML communities and refers to the set of techniques used for learning in Neural Networks. This chapter describes the essential concepts behind deep learning.

2.1 Artificial Neural Networks

Artificial Neural Networks (ANN) emerged as a computer-based technique inspired by the concept of human biological neural networks, such as the nervous system and the brain, which process information and allow a machine to learn from data. There are various types of NNs, but only two of them are covered in this thesis: Feedforward Neural Network (FNN) and Convolutional Neural Network (CNN).

The fundamental unit of every neural network is called a *neuron* or a *processing unit*. These are organized into layers, where each neuron acts based on the local information, and transmits its output to the neurons at the same layer (intralayer connections), to another layer (interlayer connections), or to both intralayer and interlayer, to accomplish recognition tasks.

Human beings are able to make mental patterns on their biological neural networks from different input data (e.g. text, pictures, sounds), through the sensory mechanisms of vision, sound, touch, smell and taste. Similarly, neural networks recognize data patterns in the data features of the input.

ANNs architecture is based on interconnected neurons, where each neuron consists of a summing unit followed by an output unit. Like Humans, the summing unit receives the input data from the sensory mechanisms (unit), weighs each value, and computes a weighted sum. The sum result also called the activation value, is passed to the output unit to produce a signal.

In order to understand how a neural network operates, it is important to understand what neurons are and how they learn and transfer knowledge to each other. Sections 2.1.1 and 2.1.2 present two main representations of a neuron, the Perceptron and the Sigmoid model.

2.1.1 Perceptrons

In 1958, Rosenblatt proposed an artificial neuron representation, the perceptron model. It was the earliest supervised learning[1] algorithm of binary classifiers, being the main structure for the Artificial Neural Networks (ANN).

The perceptron model introduces the adjustable weights (w_1-w_m) into the ANNs inputs to minimize the error from the actual binary output to the desired output, Figure 1. The weights of a neuron implicitly determine which features are more significant, and if a feature is more relevant, the output will be influenced by it. Each neuron connection has its own weight, which starts at a fixed or random

[1] The type of ML used to learn models from training data, allowing output prediction for future or unseen data.

amount and is then updated during the model's training. The output is also influenced by the *bias term (θ)*, which defines the sensitivity of the sensory inputs. Some types of data may have a large dynamic range. For example, an object in an image has different reflections depending on the exposure to dim light or bright light. It is necessary to define the sensitivity of the neuron. If it is too sensitive to smaller values of inputs, its output signal will saturate for large input values. However, if it is too sensitive to large values of the input, its activation value becomes insensitive to small values of the input.

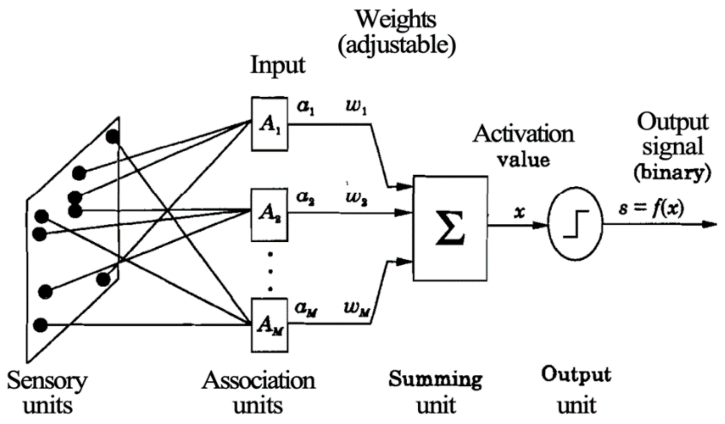


Figure 1: Rosenblatt's perceptron model of a neuron - Basics of Artificial Neural Networks [19].

The binary perceptron output that leads to a signal is the *f(x)* result, which can be 0 or 1, determined by the weighted sum $x = \sum_{i=1}^{M} w_i\alpha_i + \theta$ is less or greater than 0.

$$Output = f(x)\begin{cases}1, if\ x > 0\\0, if\ x \leq 0\end{cases}$$

Due to the binary output, a single layer of perceptron only handles linearly separable features in space. To perform any pattern classification task, a multi perceptron layer is necessary. In fact, Artificial Neural Networks (ANNs) are composed of multiple layers of perceptron's, while a perceptron is a single ANN layer representation.

2.1.2 Sigmoid

The main difficulty with a multilayer perceptron network is that it is very difficult to adjust the weights and bias. Small changes in them can drastically change the output, switching from 0 to 1 or vice versa, which will influence the behaviour of the network. With the introduction of Sigmoid neurons, the problem was solved. Based on the perceptron model, the output function is much smoother. Instead of returning a binary output, it returns a real value between 0 and 1 that can be interpreted as a probability. The sigmoid function is defined as:

$$f(x) = \frac{1}{1 + e^{-x}}$$

As a result, the difference between the perceptron and sigmoid neuron is the activation function, in which the output of the sigmoid neuron is:

$$Output = f(x) = \frac{1}{1 + e^{-(\sum_{i=1}^{M} w_i\alpha_i + \theta)}}$$

Thus, activation functions are used for standardizing the output values of each neuron. The most commonly used activation functions are the Sigmoid, and the Rectified Linear Unit (ReLu) [20] described in Section 2.5.1.

2.1.3 Cost / Loss Function

A loss function is used to measure the performance of a neural network with a single training sample, while a cost function, is the average over the entire training dataset. The Cost Function quantifies the error between the algorithm's prediction and the expected values, in a real number. To minimize the cost/loss function, it is necessary to find the balance between weights and biases using an optimization algorithm.

2.1.4 Backpropagation

Feed-forward networks flow in only one direction, from input to output. The backpropagation algorithm, inverts the flow, changing each weight in the network. It takes into account the proportion of the overall error, which allows to adjust and fit the algorithm appropriately. Each iteration results in a reduction of weight's error, which will eventually lead to weights that produce good predictions.

2.2 Optimization Algorithms

Optimization algorithms are used during the training process, to iteratively estimate the network parameters that lead to the minimum cost function value. Gradient Descent, Root Mean Square Propagation (RMSProp), and Adam are some of the optimization methods discussed in this section. Gradient Descent is the most basic method for optimizing a neural network, while RMSProp and Adam variants are generally quicker to converge to the global minimum of a function.

2.2.1 Gradient Descent

Gradient Descent (GD) is a popular optimization approach for finding the local minimum or the smallest value of a loss function whose gradient is null [21]. The algorithm starts with random initialization of weight and bias in the NN. Then, it iteratively derives the gradient descent at each step until it finds the bottom of the graph. Hopefully, it finds the minimum, where the error is the lowest (Figure 2 *Left*). This means the model was tuned for the data and predictions are more accurate.

Sizing the step is called the learning rate, shown in Figure 2 *Right*. A low rate is more precise since it recalculates gradient frequently, but it is more time-consuming, which will take more time to get to the bottom. High learning rates cover more steps, but there is a risk of overshooting the lowest point since the slope of the hill is constantly changing, getting stuck at worse values of loss over epochs[2].

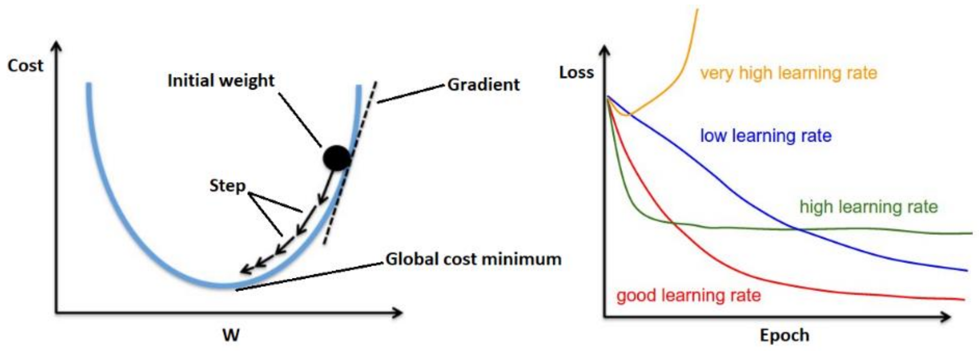


*Figure 2: **Left:** Representation of the gradient, defined as the slope of the curve and the derivative of the activation function. **Right:** Effects of different learning rates.*

[2] Dataset cycles with training data, to teach the neural network.

2.2.2 Root Mean Square Propagation

Root Mean Square (RMSProp) propagation is another optimization method widely used in machine learning (ML). Its behaviour is based on the Gradient Descent [21]. It restricts the oscillations in the vertical direction, which increases the learning rate and the algorithm can take larger steps in the horizontal direction converging faster to the local minimum [22].

The minimum or global optimum is represented by the Local Optima point in Figure 3. A global minimum is the best local minimum/optima that can be achieved. When starting the GD, a line that began at point 'A,' may wind up at position 'B,' the other side of the ellipse, after one iteration. Then another GD step could get stopped at point 'C.' As a result, getting to the minimum will take a long time, increasing the chances of Gradient Descent becoming stuck in a local optimum rather than reaching the global optimum.

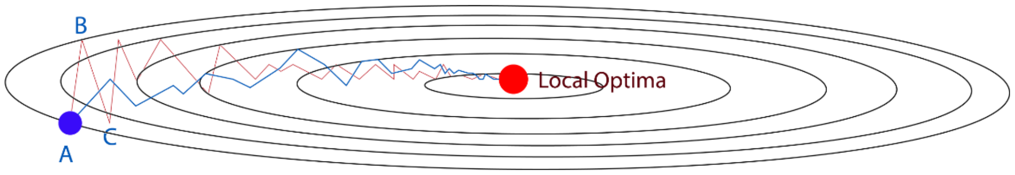


Figure 3: Comparison of Gradient Descent and RMSProp [22].

Slower learning in the vertical direction and faster learning in the horizontal direction are necessary to avoid becoming stuck in a local optimum. RMSProp uses the idea of the Exponentially Weighted Average (EWA) of the gradients, which allows the algorithm to forget early gradients and focus on the most recently observed partial gradients found during the search progress.

As demonstrated in Figure 3, extracted from [22], learning in the vertical direction has been slowed and the accomplishment of the local minimum is quicker.

2.2.3 Batch, Stochastic and Mini-batch

Depending on the amount of data, it may be better to have an accurate weight update instead of a shorter time to perform an update. There are three gradient descent variants, depending on the amount of data that is used to measure the gradient of the objective function.

Batch Gradient Descent

Computes the average of the gradients for the whole training dataset and then uses this mean gradient to update the weights, performing just *one* update (step) in one epoch. However, it is unreliable for big datasets, where it can be slow and intractable if it does not fit in the memory.

Stochastic Gradient Descent

Computes the gradient for each example in the training dataset and updates the weights. Since it considers just one example at a time, the cost can fluctuate over the training examples, which can complicate to convergence to the exact local minimum.

Mini-batch Gradient Descent

While Batch Gradient Descent converges directly to the local minimum and Stochastic Gradient Descent converges faster for larger datasets, although both have limitations, the only suitable for small datasets or computes one example at a time, respectively. The Mini-batch takes advantage of both approaches bases the computation on a batch with a fixed number of training examples, called mini-batch. It calculates the mean gradient of the mini-batch and updates the weights frequently, also allowing a vectorised implementation for faster computations.

2.2.4 Adam

Adam combines all the previous techniques into a single, efficient algorithm. As one might expect, this algorithm has grown in popularity as one of the most reliable and effective optimization algorithms for deep learning. It is computationally efficient, requires low memory, is invariant to gradient diagonal rescaling, and is well suited for problems with a lot of data or parameters. The method is also suitable for non-stationary objectives and problems involving very noisy or sparse gradients, where the network is unable to tune its weights due to a lack of strong signals [23].

2.3 Convolutional Neural Networks

Convolutional Neural Networks (CNN) is a type of feedforward NN, very effective in image recognition and similar tasks. The main difference between them is the neuron arrangements in the layer. CNNs have neurons arranged in three dimensions (width, height, and depth). Also, each neuron inside the convolutional layer is connected to only a small region on the next layer. This is arranged in three different types (Convolutional, Pooling and Fully Connected), which is also distinct from feedforward neural networks.

In general, CNN implementations can be defined as involving the following process [24], Figure 4:

1. Convolve several small filters on the image and apply ReLU activation to the matrix.
2. Perform pooling to subsample and reduce the dimensionality size.
3. Repeat steps 1 and 2 until as many convolution layers as enough to have high-level features.
4. Flush the output and feed it into a Fully Connected Layer.
5. Output the class using an activation function such as sigmoid or softmax (described in Section 2.5.2) to classify the images.

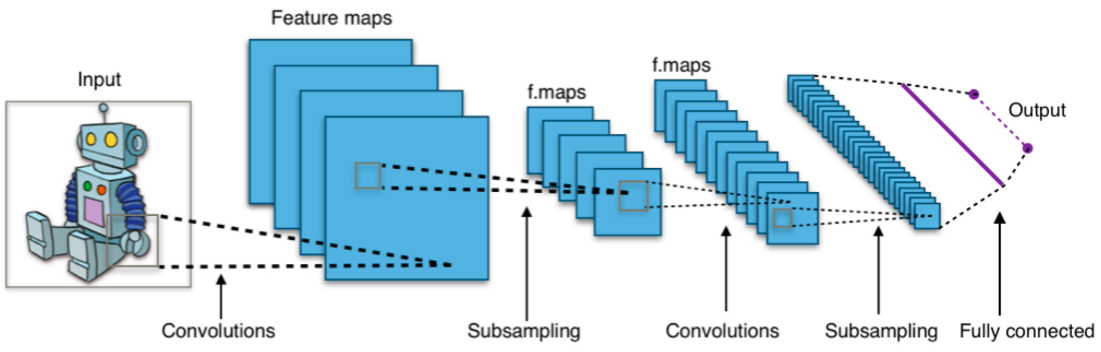


Figure 4: Typical CNN architecture [24].

2.3.1 Convolution Layer

The Convolution layer is the core of the CNN and is the first layer to extract features from an input image [25]. The layer consists of a set of learnable filters, which learn the image features using small squares of input data, producing a 2-dimensional activation map of the filter, Figure 5. As a result, the activation map triggers when it detects some criteria learned through the filters on some spatial position in the input.

Stacking multiple layers of filters allows to perform operations such as edge detection, blur, sharpen, colour, gradient orientation, etc. It also increases the capturing for low levels of detail, although at the cost of more computational power.

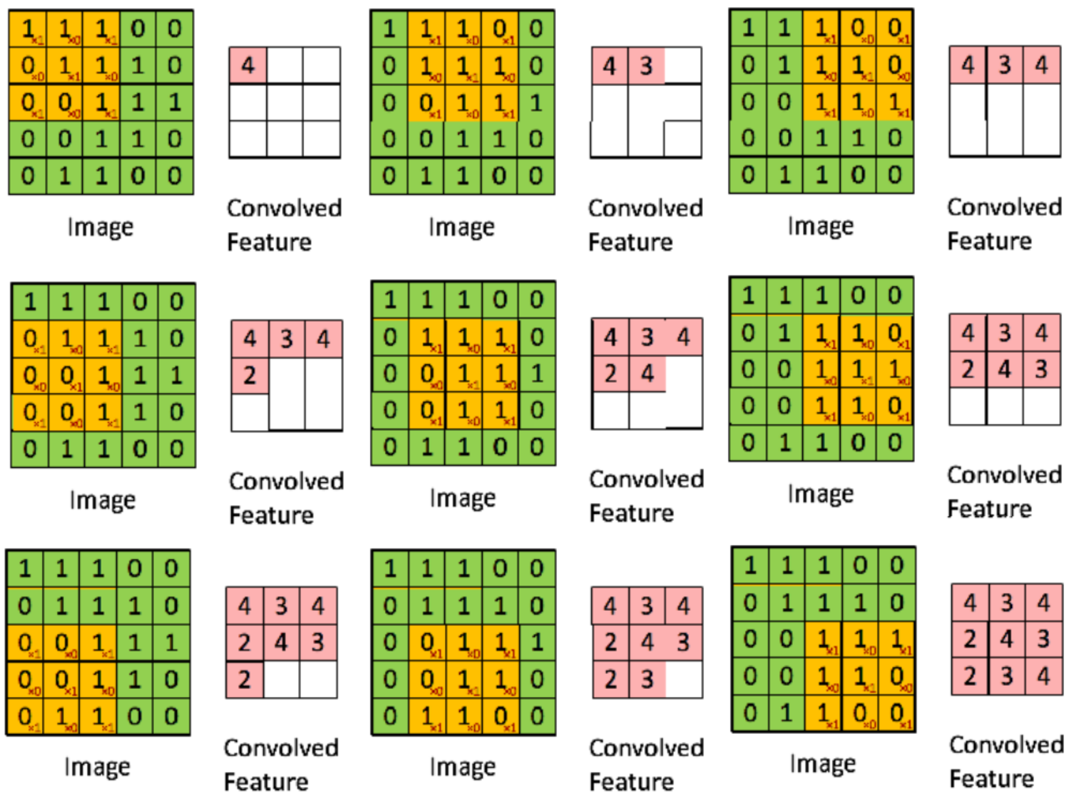


Figure 5: Feature Map Output [25].

2.3.2 Pooling Layer

A Pooling layer is responsible for subsampling or downsampling the spatial size of the convolved feature [25]. This dimensional reduction of each map, retaining important information, decreases the amount of computation in the network and controls the overfitting. There are a few non-linear functions to implement pooling, such as the minimum, maximum and average, whoever, the most common is the maximum or max pooling.

A maximum or maximum pooling is a pooling operation, which returns the largest element of the rectified resource map portion and uses it to create a downsampled (pooled) resource, which implicitly reduces the data and calculations in the network, Figure 6.

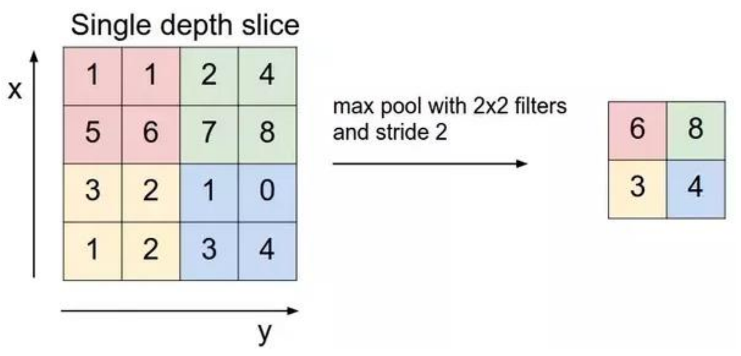


Figure 6: Max pooling [25].

2.3.3 Fully Connected Layer

The convolution layers output high-level features data, while the fully connected layer learns non-linear[3] combinations of these features, which flattens and connects to the output layers. Therefore, the flattened output is fed into a feed-forward neural network and backpropagation is applied to every iteration of training. Over a succession of times, the model is able to distinguish and classify between high and low-level characteristics in the pictures.

2.4 Overfitting

Overfitting occurs when a model seeks to predict a pattern in too noisy data, which leads to an inaccurate model since the trend does not reflect the reality present in the data. A model that produces good results on a training dataset but performs poorly on the unseen data, is a sign that the model is overfitting. Making correct assumptions from a set of training data to any data from the domain problem, which was never seen before, is the main goal of any ML model. This section describes the most common techniques to prevent overfitting while training large networks.

2.4.1 Data Augmentation

The data augmentation method consists of increasing the size of training data. Some of the easy techniques to artificially expand data are flipping, translation, rotation, scaling and transposing. These techniques are used on the most common training datasets for CNNs, such as CIFAR-10 [26]. Adding more data, forces the model to generalize, becoming unable to overfit all the samples.

2.4.2 Regularization

Regularization is a technique that adds an extra term to the loss function, in a way to penalise or impose a cost on weights, making the network choose smaller weight matrices leads to simpler models, which reduces the overfitting.

2.4.3 Dropout

Dropout is based on the idea of modifying the network into different networks, through randomly neuron drops from NN in each interaction during the training, preventing neurons from co-adapting. The different networks will be overfitted in different ways, hopefully, resulting in an overfitting reduction as an effect on the network.

In each iteration of the process, a percentage of neurons are randomly and temporally disconnected, except those who belong to the input and output layer. Then, the input is propagated through the modified network and the result as well. The weights and biases are updated, and the process is repeated by restoring the dropped neurons and choosing a new random subset of neurons to disconnect.

Finally, by reducing neurons co-adaptations, a neuron cannot rely on another neuron, being forced to learn more features in detail, which is useful in combination with other random subsets of neurons. The output result of the Dropout process can be seen as the average of a large number of networks.

[3] Not straight-line or direct relationship

2.5 Deep Learning

Deep Learning, as described at the beginning of the chapter, refers to the set of techniques used for learning in multi-layer neural networks (Deep Neural Networks (DNN)). Therefore, the “deep” in deep learning is associated with the depth of layers in a network and is a subset of ML algorithms.

While supervised ML is dependent on human interaction to label the datasets to understand the differences between data inputs, Deep learning does not need labelled datasets. Instead, it can learn through unstructured or unlabelled data. By observing patterns in the data, a deep learning model will need more data points to improve its accuracy, which means more computer power, whereas an ML model depends on less data due to the underlying data structure.

In DNN, the *Vanishing Gradients* are one of the biggest problems that can lead to unpredictable behaviour during the training phase. Which is defined by the inability of NN to propagate gradient information from the output back to the lower layers of the model. The gradients control how much the network learns during the training phase. If the gradients are close to zero, the model improves very slowly and it is also possible that training stops. This may cause the inability of models with many layers to learn on a given dataset or converge to poor predictive performance.

2.5.1 ReLU units

The Vanishing Gradients problem can be solved by using rectified linear units Rectified Linear Unit (ReLU) as the activation function. The function returns 0 if it receives any negative input, and the input itself for any positive input, f(x) = max(0, x) (Figure 7 (n) - “relu”).

The gradient of the sigmoid function described in Section 2.1.2, squishes a large input space into a small input space between 0 and 1 (Figure 7 (a) – “sigmoid”). Therefore, a large change in the input of the sigmoid function will cause a small change in the output. Hence, the derivative becomes small (Figure 7 (a) – “grad of sigmoid”), contributing to the vanishing gradient problem.

The ReLU gradient can only be 0 or 1, which after many layers tend to stabilize. So, the overall gradient is not too small or not too high (Figure 7 (b) – “grad of relu”), thus controlling the learning rate. Furthermore, it is simpler to calculate and its derivative consumes less computational resources than the sigmoid function [27].

a) Sigmoid activation b) ReLU activation

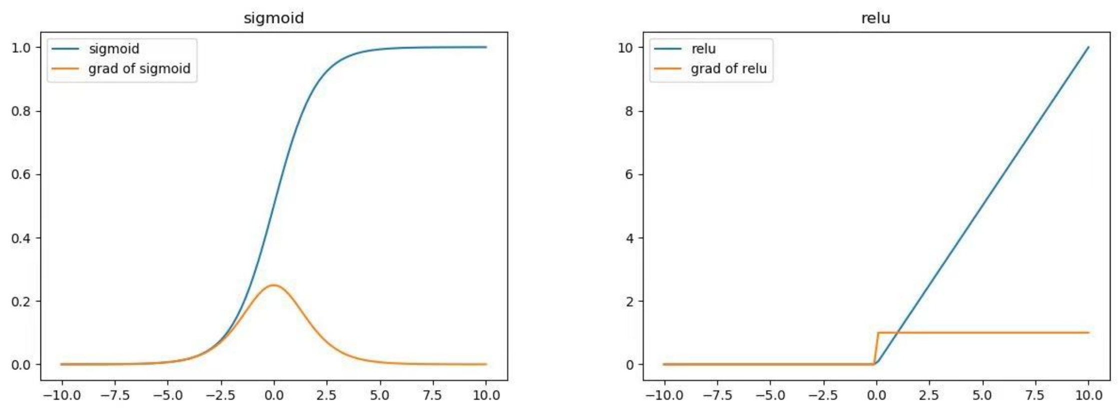


Figure 7: Sigmoid and ReLU comparison of activation function and derivative [27].

2.5.2 Softmax

The sigmoid activation function is used for the two-class logistic regression, whereas the softmax function is used for the multiclass logistic regression. Softmax normalizes an input value into a vector of values that follow a probability distribution with a total sum of one. It is used as an activation function similar to ReLU (Section 2.5.1) but applied to the last layers rather than the middle layers. To normalize the output of NN models for multiclass classification problems, where a class association is required in more than two class labels [28].

A standard structure of an NN model with softmax as output is shown in Figure 8.

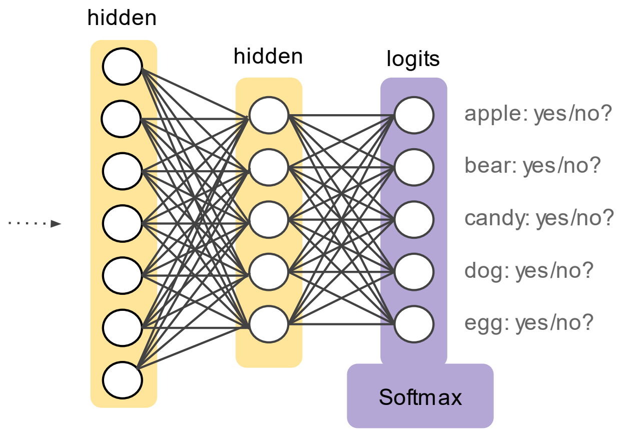


Figure 8: Softmax layer within a Neural network.

2.6 Pre-trained Models

Pre-trained models can be applied to neural networks, pre-trained networks, which includes layers, features, weights, and biases that have been adjusted to the feature values of the dataset for which they were trained [29]. The vast majority of pre-trained networks are trained on a subset of the ImageNet database [30], which is used in the ImageNet Large-Scale Visual Recognition Challenge (ILSVRC) [31]. These networks have been trained on over a million images and can classify images into 1000 different object categories. Using a pre-trained network with transfer learning is typically much faster and easier than training a network from the ground up, which makes them widely used.

This section goes over ResNet-50, VGG16, and Google's Inception, three pre-trained CNN models that are frequently used for transfer learning. Transfer learning (TL) is a machine learning (ML) technique that focuses on storing and transferring knowledge gained while solving one similar problem to another.

Pre-trained models, like the ones mentioned above, can be adapted to a new problem by swapping their fully connected network layers while keeping the convolutional and pooling layers. As a result, in a transfer learning process, only fully connected network layers must be trained, which can save significant time during network training.

2.6.1 ResNet-50

The Residual Network, also known as ResNet, is a model that uses the residual module rather than attempting to learn features. ResNet has numerous variations that employ the same concept but varies in the amount of layers in its architecture. ResNet-50 refers to the variant with 50 neural network layers.

It was created by Microsoft researchers [32] and has since become extensively used for image classification [33].

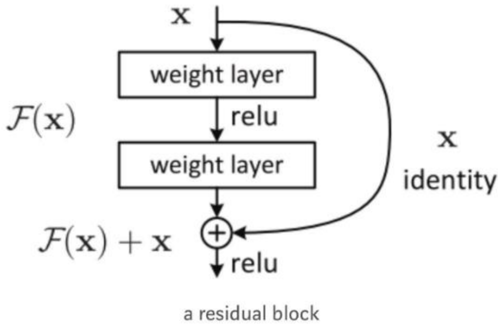

x
weight layer
relu
weight layer
$\mathcal{F}(\mathbf{x})$
x
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu
a residual block

Figure 9: Residual Network [33].

ResNet skip connections to propagate information over layers. In Figure 9 from [33], the link labelled as "identity" allows the network to learn the identity function and sends the input through the block without going through the other weight layers. The output value of *F(x)* + *x* is obtained by adding the value *x* from the previous layer to the output of the layer ahead, *F(x)*. This is advantageous because it allows creating a deeper network, which can pass over layers that the model considers to be less relevant.

2.6.2 VGG16

VGG16 is a convolutional neural network model created by K. Simonyan and A. Zisserman that scored 92.7 percent in ImageNet's top-5 accuracy test [34]. The network's input is limited to 224 × 224 RGB pictures that are subjected to a stack of layers, with the filters set to an extremely tiny receptive field size of 3 × 3.

The network employs three completely linked layers, which are common in this sort of network, with the softmax layer serving as the final layer. Figure 10, presents the architecture of the VGG16.

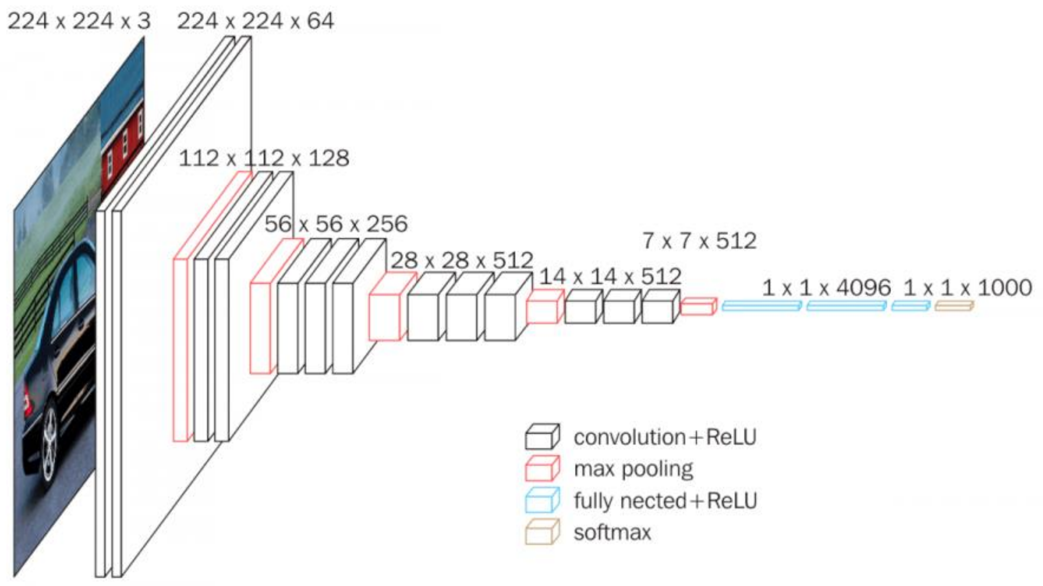


Figure 10: VGG16 Architecture [34].

2.6.3 Google's Inception

GoogLeNet (or Inception V1) is a 22-layer deep convolutional neural network that is a variant of the Inception Network, a DNN developed by Google in 2014 [35], to help with image processing and object identification. Inception V3 is the third iteration of GoogLeNet NN, which achieved 78.1 percent accuracy on the ImageNet dataset during the ILSVRC [36].

Based on the original work "*Rethinking the Inception Architecture for Computer Vision*" by Szegedy, *et al.* [37], the model represents the result of numerous concepts explored by various researchers over the years. It was designed to allow for deeper networks while still limiting the number of parameters to around 25 million, as compared to 138 million for VGG16 [29]. Convolutions, average pooling, max pooling, concats, dropouts, and fully connected layers are among the symmetric and asymmetric building blocks in the architecture (Figure 11).

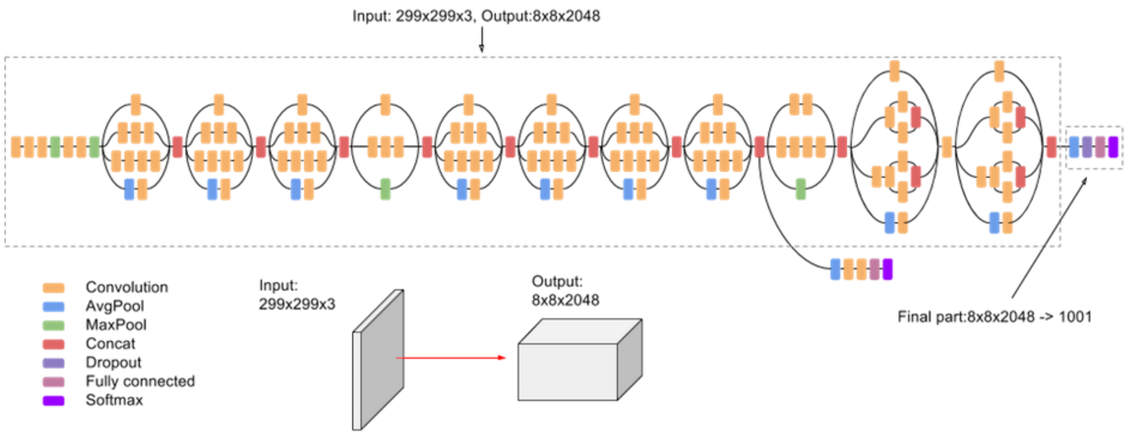


Figure 11: Inception v3 Architecture [37].

Chapter 3

State of the Art

Recently, ML methods have been used as an approach to the detection and analysis of malware. The growth in the number of cases of malware attacks, the decrease in the cost of processing power and the advances made in the field contributed to new proposals in the literature to improve malware analysis. Meanwhile, privacy has emerged as a big concern. Its preservation in the context of ML is significantly different from traditional data privacy protection. Therefore, most existing solutions only address privacy problems during the ML process. As a result, it is important to continually invest in research on privacy issues and ML.

3.1 Malware as an Image

L. Nataraj *et al.* [15], proposed to characterize and evaluate malware based on its visualization as greyscale images. Through the transformation of the malware binary to be interpreted as an 8-bit array. The bit is the most fundamental unit of information in computers and digital communications since it represents a binary integer. A group of 8 bits, 1 byte, contains 2^8 different values, where the range of integer values that can be recorded in 8 bits varying depending on the integer format selected. The two most popular representations are the ranges -128 (-1×2^7), to 127 ($2^7 - 1$), for representation as two's complement, and from 0 to 255 ($2^8 - 1$), for representation as unsigned, which is the same value range that a pixel is represented in a greyscale image, Figure 12.

By reshaping the 8-bit array into a matrix and vieweing it as a greyscale image, revealed important visual correlations in the image texture of malware belonging to the same family. It can be a consequence of the widespread method of reusing the code to build new malware variants.

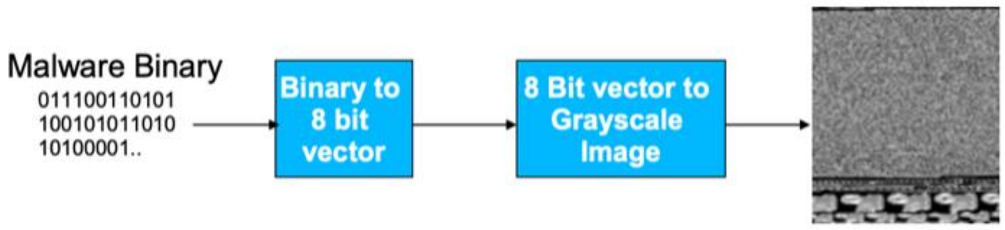
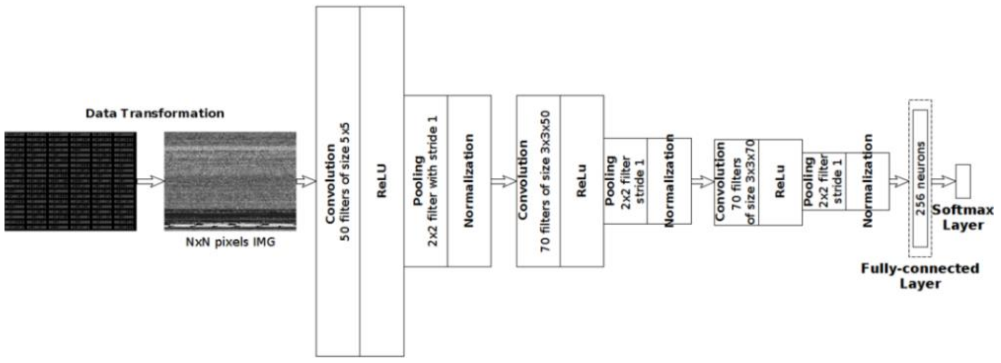


Figure 12: Visualizing Malware as an Image[15].

To compute texture features in the malware images, it was used GIST [38], which uses a wavelet decomposition, to extract features from the global structure of the image. These elements are used to perform a comparison against previously identified malicious patterns. Finally, the *k-nearest neighbors* with Euclidean distance carried out the malware classification, achieving a classification accuracy of 97.18 percent for a dataset composed of 25 malware families.

Although an image rendered based on a global structure features is vulnerable to structural changes, cybercriminals that are aware of this technique can avoid detection, by relocating sections of code or adding dummy data.

Gibert *et al.* [16] suggest a different approach to solve the countermeasure employed by cybercriminals. Using Convolutional Neural Networks (CNNs) to extract local and invariant features from an image, finding the patterns independently of their position. Thus, it allows the network to detect patterns of known malware present on an image. Figure 13 shows the structure of Gibert's CNN.



Data Transformation
NxN pixels IMG
Convolution 50 filters of size 5x5
ReLU
Pooling 2x2 filter with stride 1
Normalization
Convolution 70 filters of size 3x3x50
ReLu
Pooling 2x2 filter stride 1
Normalization
Convolution 70 filters of size 3x3x70
ReLu
Pooling 2x2 filter stride 1
Normalization
256 neurons
Softmax Layer
Fully-connected Layer

Figure 13: Gibert's CNN for classification of malware represented as greyscale images [16].

To assess CNNs approach, two publicly available data sets were used: the same one used by Nataraj *et al.* [15], which included 9,458 samples from 25 different malware families, and a second from the Microsoft Malware Classification Challenge (presented in Section 4.1).

In the following images of malware representation, it is possible to detect small changes, while maintaining the overall structure of samples belonging to the same family. However, they are distinct from the malware images of other families, Figure 14.

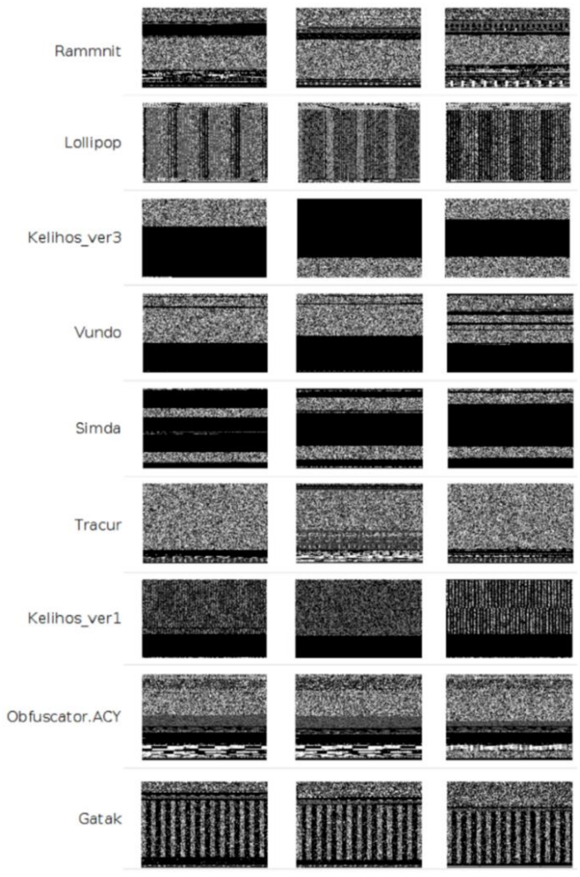


Figure 14: Greyscale images of malicious software belonging to various families – Microsoft Dataset [16].

The overall classification accuracy achieved by CNN (Figure 13) to the dataset with the same 25 malware families was higher than the approach of Nataraj *et al.* [15], achieving 98.48 percent of accuracy. For the Microsoft dataset, it achieved 97.3 and 97.5 percent accuracy, and 92.7 and 94 percent of F1-Score macro average, in the 5-fold and 10-fold cross validation tests, respectively.

Although the proposed solution has a range of benefits that enable malicious programs to be detected in a real-time context, this strategy has issues for certain samples that have been compressed or encrypted, which may have a completely different overall structure.

3.2 Image Privacy

The article "*When Machine Learning Meets Privacy*" [39] tried to find answers to the topic "*What are the privacy challenges and solutions associated with ML*". This study attempts to provide the first comprehensive survey on privacy in ML, by investigating different scenarios/applications of privacy and ML, such as private machine learning, machine learning aided privacy protection, and machine learning-based privacy attack and corresponding protection schemes.

With a focus on private ML, the study presents several private ML schemes, such as encryption, obfuscation, and aggregation. The encryption or cryptography-based methods applied to the CNN data, take advantage of homomorphic cryptography, where calculations are represented as Boolean or arithmetic circuits, which can be preserved after encryption, making it possible to compute over the encrypted data without knowing the secret key. Another method proposed is to encrypt the ML model by using homomorphic encryption on the gradients.

The two main demonstrations of the use of neural networks over data encrypted with homomorphic encryption (HE) are presented by Dowlin *et al.* [40], who proposed CryptoNets, and Hesamifard *et al.* [41], who presented CryptoDL. Both show how to efficiently convert learned neural networks to make them applicable to encrypted input data.

Overall, training neural networks, particularly DNNs, over encrypted data remains computationally challenging. Even when trained in plain text, the network is slow. When HE is added to a process, it slows it down by at least an order of magnitude.

Lastly, obfuscation strategies can be used for both the data and the model, reducing the accuracy of both. This is accomplished by adding noise to the model parameters or the original dataset.

CryptoNets

Dowlin *et al.* [40] propose a method for converting trained neural networks into CryptoNets, which are neural networks that can be applied to encrypted data. This enables a data owner to communicate their data to a server in an encrypted format. Because the server does not have access to the keys needed to decode the data, encryption ensures that it stays private. Furthermore, the neural network's encrypted predictions can be transmitted back to the owner of the secret key, who is the only person capable of decrypting them.

YASHE [42] is a fully homomorphic encryption (FHE) scheme used to improve the efficiency of the regular schemes, where the encryption scheme's parameters are selected such that arithmetic circuits of a specified depth may be assessed. It implies knowing the topology of the neural network, including the activation functions. As a result, it was possible to achieve 99.9 percent accuracy for optical character recognition tasks using the MNIST [43] dataset.

CryptoDL
Hesamifard *et al.* [41] approach focuses on operations inside neurons. All other operations in a neural network, aside from activation functions, are additions and multiplications, which can be performed on homomorphically encrypted data. Although activation functions cannot be utilized in HE schemes, it is necessary to firstly design methods for approximation of the activation functions commonly used in CNNs (i.e. Sigmoid, ReLU, and Softmax) with low degree polynomials which is essential for efficient homomorphic encryption schemes. Then, it trains convolutional neural networks with the approximation polynomials instead of original activation functions.

Their proposed solution consists of a polynomial approximation, based on the derivative of the activation function ReLU due to its impact on error calculation and weight updating. It was applied to several neural networks, and the models' performance was evaluated using approximation polynomials rather than actual activation functions. Figure 15 shows the structure of CNN, which was used to accomplish 99.52 percent on the MNIST optical character recognition challenges, proving that CryptoDL provides efficient, accurate, and scalable privacy-preserving predictions.

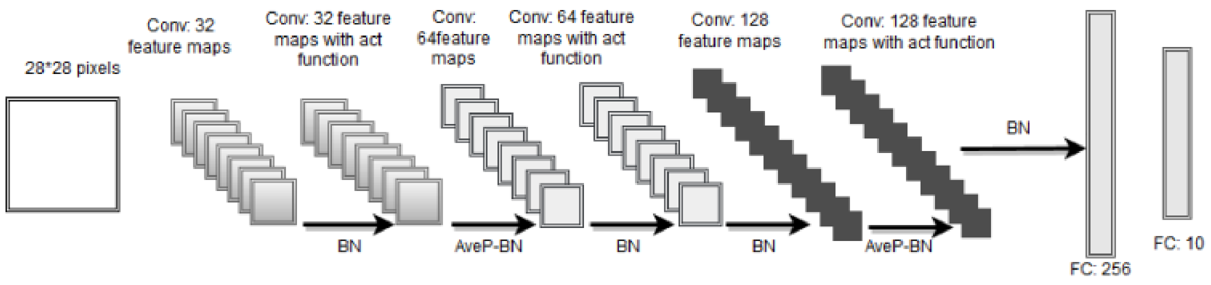

28*28 pixels
Conv: 32 feature maps
Conv: 32 feature maps with act function
Conv: 64feature maps
Conv: 64 feature maps with act function
Conv: 128 feature maps
Conv: 128 feature maps with act function
BN
AveP-BN
BN
BN
AveP-BN
BN
FC: 256
FC: 10

Figure 15: CNN model with Polynomial Activation Function [41].

3.3 Deep Learning Frameworks

The problem of malware identification and recognition is a very challenging task with no perfect solution. This challenge culminated in several framework implementations with distinct benefits in specific sub-areas of deep learning [44]. Some of the well-known frameworks are listed below:

TensorFlow
Developed by Google Brain Team, is an open-source machine learning framework written in Python on top of a C/C++ engine, offering high performance with large datasets [44]. As a result, it is well-suited for developing and experimenting models with deep learning architectures. Furthermore, it supports mobile platforms, such as iOS and Android.

Keras
Built on top of TensorFlow with an emphasis on user experience, it is a framework that easily scales up, through data parallelism and hence, it can process massive volumes of data while accelerating the training time for the models [44]. It is considered one of the most popular frameworks, offering consistent and straightforward APIs while reducing the number of user actions required for typical use cases, providing clear and actionable error messages.

PyTorch

Based on the Torch library and developed by Facebook with the primary purpose of speeding up the entire process from research and prototyping to production deployment [44]. It operates with a dynamically updated graph, which allows making improvements to the model design during the training phase. It is ideal for preparing, developing, and launching small projects and prototypes.

Sonnet

DeepMind's Sonnet is a Tensorflow-based system for integrating neural networks [44]. It is based on the development and creation of Python objects to describe components of the neural network. Encapsulating neural network elements, such as models, allows being combined multiple times into a data flow chart while helping to simplify the design of high-level architectures.

MXNet

Apache MXNet is an open-source deep learning framework, although similar to TensorFlow, which achieves a faster training speed and offers better performance on data-intensive tasks [44]. While MXNet performs better on processing tasks, it offers fast context switching and optimized computing. Compared to TensorFlow, MXNet also has a smaller open-source community.

3.4 Class Balancing

Imbalanced data is a common challenge problem when training Machine Learning models, which means the dataset classes are not equally represented. There are two popular approaches to deal with it, in order to establish the balance, and each class starts to have the same weight consideration to the model.

Class Weighting

Class weights directly affect the loss function by imposing a greater (or lower) penalty on classes with more (or fewer) weights. In effect, by intentionally adjusting the model bias to favour more accurate predictions in the higher weight class, losing some ability to predict the lower weight class (the majority class in unbalanced datasets) [45].

Oversampling and Undersampling

Oversampling methods essentially give more weight to specific classes, through the duplication of observations, giving them more influence in the model fit. However, this might lead to the model being overfitted. Undersampling methods, on the other hand, remove samples from the majority class, which can lead to the loss of crucial information for a model [45].

Chapter 4

Malware Classification

The problem of malware identification and recognition is a very challenging task with no perfect solution. The malware industry has become a well-organized market involving large amounts of money. It has led to massive investments in technologies and resources built to evade traditional protections, representing one of the biggest challenges to the security community.

In order to explore the problem, this chapter introduces the Microsoft Malware Classification Challenge, which has one of the largest and newest publicly available datasets for the Big Data Innovators Gathering Cup (BIG 2015) [46][47]. Additionally, the VirusShare.com repository, considered one of the main sources of malware in providing samples, to security researchers, incident responders, forensic analysts, or merely curious people, access to samples of live malicious code [48].

The use of a robust dataset, such as Microsoft or Virusshare, composed of a large and diverse number of samples, is insufficient for accurate malware detection. To ensure that the ML model is accurate, metrics must be identified and established to evaluate its performance. This chapter describes some of the most frequent assessment measures used to evaluate deep learning architectures.

Finally, a novel CNN model will be introduced for the malware email analysis component in order to attempt better results compared to the work done by Nataraj *et al.* [15] and enhanced by Gibert *et al.* Following the performance analysis and evaluation of the novel model, along with a comparison to pre-trained models (Section 2.6).

4.1 Microsoft Malware Classification Challenge

In 2015, Microsoft conducted a Kaggle competition [46][47] with the aim of classifying malware into families based on its content and characteristics. Microsoft provided a dataset of 21741 samples for this competition divided into 10868 samples for training and 10873 for testing purposes, totalling almost half a terabyte of uncompressed data. The dataset is organized into 9 different families of malware. Each sample contains an identifier, a 20-character hash value that uniquely identifies the file, and a class label, which is an integer representation of the malware family name to which the malware belongs: (1) *Ramnit [Worm]*, (2) *Lollipop [Adware]*, (3) *Kelihos_ver3 [Backdoor]*, (4) *Vundo [Trojan]*, (5) *Simda [Backdoor]*, (6) *Tracur [TrojanDownloader]*, (7) *Kelihos_ver1 [Backdoor]*, (8) *Obfuscator.ACY [Any kind of obfuscated malware]*, (9) *Gatak [Backdoor]*. The distribution of classes in the training data is not standardized, and the number of instances of certain families outnumbers other families, Figure 16.

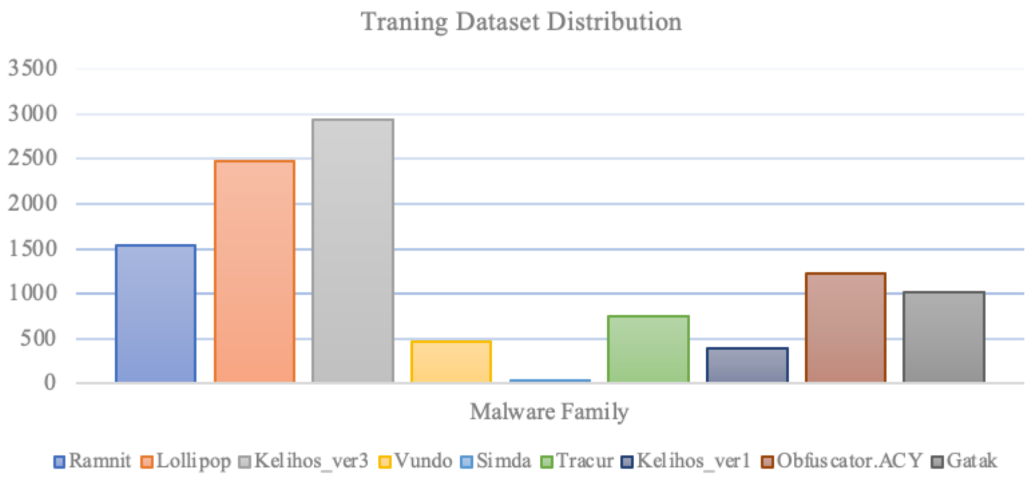
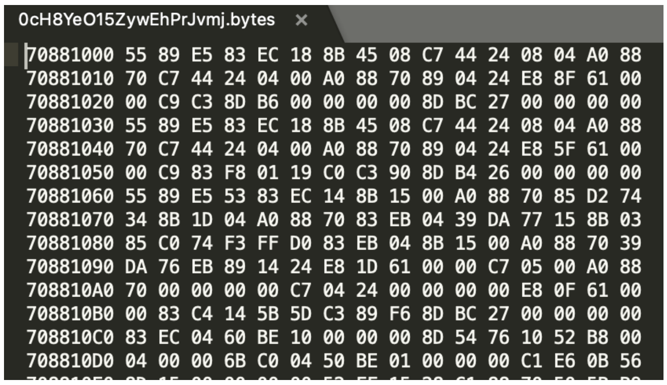


Figure 16: Malware Classification Challenge – Dataset.

Each malware sample is provided with a file containing the hexadecimal of the file's binary content, and a metadata file generated by the IDA disassembler tool with extracted data from binary, such as assembly instructions, function calls, arguments, variables and registers used, etc.

4.1.1 Bytes File

The bytes file contains the malware's binary content in its raw hexadecimal form, Figure 17. The following image shows a snapshot of a *Rammit* malware sample bytes file.



```
70881000 55 89 E5 83 EC 18 8B 45 08 C7 44 24 08 04 A0 88
70881010 70 C7 44 24 04 00 A0 88 70 89 04 24 E8 8F 61 00
70881020 00 C9 C3 8D B6 00 00 00 00 8D BC 27 00 00 00 00
70881030 55 89 E5 83 EC 18 8B 45 08 C7 44 24 08 04 A0 88
70881040 70 C7 44 24 04 00 A0 88 70 89 04 24 E8 5F 61 00
70881050 00 C9 83 F8 01 19 C0 C3 90 8D B4 26 00 00 00 00
70881060 55 89 E5 53 83 EC 14 8B 15 00 A0 88 70 85 D2 74
70881070 34 8B 1D 04 A0 88 70 83 EB 04 39 DA 77 15 8B 03
70881080 85 C0 74 F3 FF D0 83 EB 04 8B 15 00 A0 88 70 39
70881090 DA 76 EB 89 14 24 E8 1D 61 00 00 C7 05 00 A0 88
708810A0 70 00 00 00 00 C7 04 24 00 00 00 00 E8 0F 61 00
708810B0 00 83 C4 14 5B 5D C3 89 F6 8D BC 27 00 00 00 00
708810C0 83 EC 04 60 BE 10 00 00 00 8D 54 76 10 52 B8 00
708810D0 04 00 00 6B C0 04 50 BE 01 00 00 00 C1 E6 0B 56
```

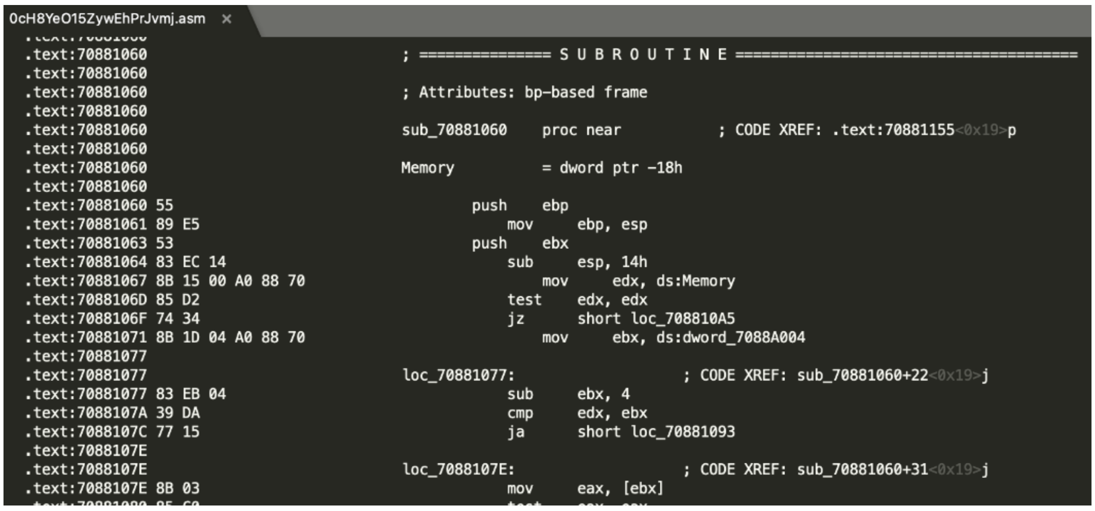
Figure 17: Byte's file of a Rammit sample.

A hexadecimal record is composed of six fields:

- ***Byte Count***, two hex digits which indicate the number of bytes in the data field.
- ***Address***, four hex digits to represent the data's 16-bit beginning memory address offset.
- ***Record Type***, two hex digits to define the meaning of the data field, from 00 to 05 representation (Data, End Of File or Extended Segment, Start Segment, Extended Linear, Start Linear Address).
- ***Data***, a data sequence of *n* bytes represented by *2n* hex digits.
- ***Checksum***, two hex digits, a calculated value used to ensure the record is error-free.

4.1.2 ASM File

The ASM file is a metadata manifest, which is a log containing metadata information, such as function calls, memory allocation, and variable manipulation (Figure 18). The following image shows the content of the ASM file snapshot of the previous *Rammit* bytes file.



```
0cH8YeO15ZywEhPrJvmj.asm
.text:70881060                     ; =============== S U B R O U T I N E =======================================
.text:70881060
.text:70881060                     ; Attributes: bp-based frame
.text:70881060
.text:70881060                     sub_70881060    proc near           ; CODE XREF: .text:70881155<0x19>p
.text:70881060
.text:70881060                     Memory          = dword ptr -18h
.text:70881060
.text:70881060 55                          push    ebp
.text:70881061 89 E5                           mov     ebp, esp
.text:70881063 53                          push    ebx
.text:70881064 83 EC 14                        sub     esp, 14h
.text:70881067 8B 15 00 A0 88 70                   mov     edx, ds:Memory
.text:7088106D 85 D2                           test    edx, edx
.text:7088106F 74 34                           jz      short loc_708810A5
.text:70881071 8B 1D 04 A0 88 70                   mov     ebx, ds:dword_7088A004
.text:70881077
.text:70881077                     loc_70881077:                     ; CODE XREF: sub_70881060+22<0x19>j
.text:70881077 83 EB 04                        sub     ebx, 4
.text:7088107A 39 DA                           cmp     edx, ebx
.text:7088107C 77 15                           ja      short loc_70881093
.text:7088107E
.text:7088107E                     loc_7088107E:                     ; CODE XREF: sub_70881060+31<0x19>j
.text:7088107E 8B 03                           mov     eax, [ebx]
```

Figure 18: Assembly file of a Rammit sample.

An assembly program has three main sections:

- The ***data*** section is used to declare initialized data or constants that do not change during runtime, such as constant values, file names, or buffer size.
- The ***bss*** section is used for declaring variables, such as uninitialized data.
- The ***text*** section is where the actual code is placed.

Aside from the previous sections, there may be additional sections such as:

- The ***rsrc*** section contains all the resources for the program.
- The ***rdata*** section is used to store data that does not fall into the ***data*** or ***bss*** section. This is also only readable data, which contains literal strings, constants and the debug directory information.
- The ***idata*** section contains information about the imports, such DLLs of the program, including the Import Directory and Import Address Table.
- The ***edata*** section contains the information about the names and addresses of exported functions, includes the export directory that provides the address and offset of the functions to programs that import the DLL.
- The ***reloc*** section holds a table of base relocations. A base relocation is a change to an instruction or an initialized variable value that is required if the loader is unable to load the program.

Other parts can also appear as a result of using polymorphic or metamorphic techniques to hide the actual code. In some malware transformations, the different binary fragments can be seen, and the assembly section of the malware can be identified by the different textures in the images. The following Figure 19 shows a *Rammit* sample transformation.

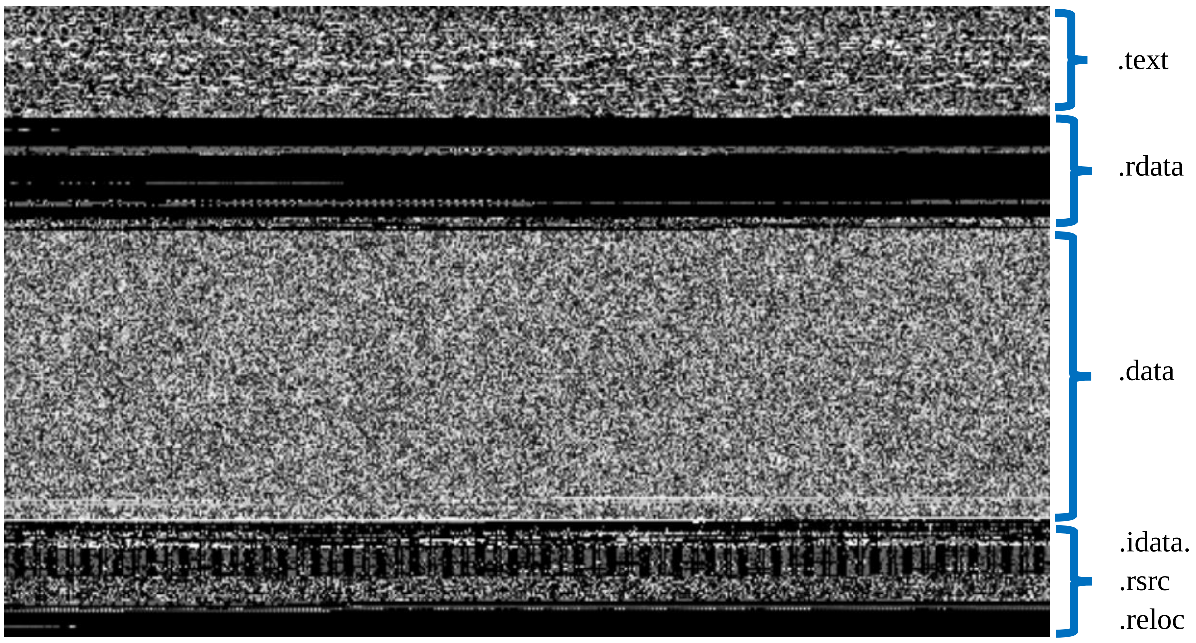


Figure 19: Image of binary Fragments of a Rammit sample.

4.2 VirusShare Repository

VirusShare repository [48] is a service hosted and maintained by Corvus Forensics, which provides digital forensics, data breach and cyber-crime incident response, malware analysis, and strategic advisory services. It is considered one of the main sources of malware samples, providing them globally.

It has joined efforts and successfully contributed to the global research security community, with more than 35 million live malware samples daily update. It allows searches by malware families, retrieving files where they are presented, which could include different MIME types, such as (pdf, exe, xlsx, etc..). The result is presented, with each sample containing the following Indicators of Compromise (IoC): MD5, SHA1, and SHA256 hashes, as well as ExIF data and whether or not the malware was found by some AV.

4.3 Tools and Libraries

This section briefly describes the primary machine learning tools and libraries used for the work on this dissertation: the deep learning framework, the Scikit-learn Python library, and the assessment metrics.

As described in Section 3.3, TensorFlow has been widely adopted, mostly because it has a Python API, which was recently enhanced in TensorFlow 2.0. It is well known and endorsed by a broad community that it is constantly developing it. However, Keras was chosen because it is built on TensorFlow and is a high-level framework that allows the abstraction of TensorFlow complexity, making it more user-friendly. It was designed to scale up, and its support for distributed computing makes it ideal for analysing new malware types emerging on a daily basis.

4.3.1 Scikit-learn

Scikit-learn (Sklearn) is a Python library that provides simple and effective prediction implementation methods, such as statistical modelling, classification, regression, clustering, and dimensionality reduction [49]. Also, it comprises the implementation of numerous traditional machine learning methods. This mostly Python-written package is based on NumPy, SciPy, and Matplotlib.

4.3.2 Assessment Metrics

The performance of machine learning algorithms can be measured through various techniques. An assessment metric function is used to evaluate the performance of a model. This section goes over a few of them.

Mean Square Error

Mean squared error (MSE) is the average sum of the squared difference between the actual value and the predicted or estimated value. It is also termed as mean squared deviation (MSD). The squaring is necessary to remove any negative signs, which also gives more weight to larger differences. It is called mean square error, as it is the process of finding the average of a set of errors. The smaller the MSE, the narrower the regression line (Figure 20) and the better the prediction. Figure 20 shows the best regression / data-fitted line [50].

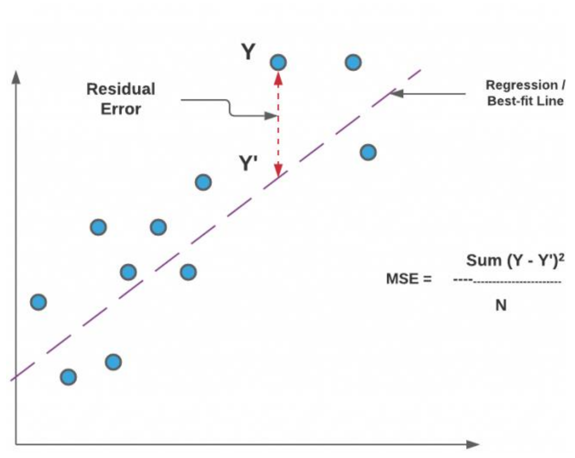
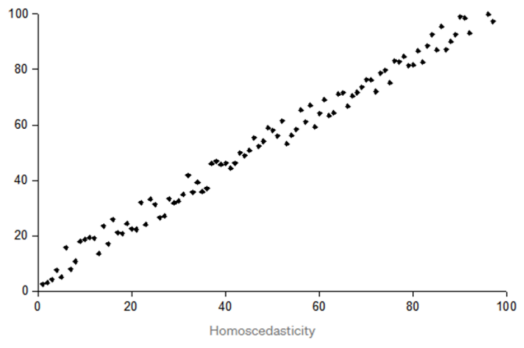


Figure 20: Mean Square Error [50].

Pearson Correlation Coefficient

Pearson Correlation Coefficient measures the intensity of connection and the direction of the association between two variables. The correlation coefficient has a value between -1 and +1 depending on the strength of the association. A value of 1 shows that the two variables are perfectly associated. As the correlation coefficient value approaches zero, the link between the two variables weakens. The sign of the coefficient shows the direction of the relationship; a + sign indicates a positive relationship and a - sign indicates a negative link. Figure 21 (a) illustrates a situation in which the variables are highly correlated (homoscedasticity) and Figure 21 (b) shows a case in which the correlation is low (heteroscedasticity) [51].

a) Homoscedasticit



b) Heteroscedasticity

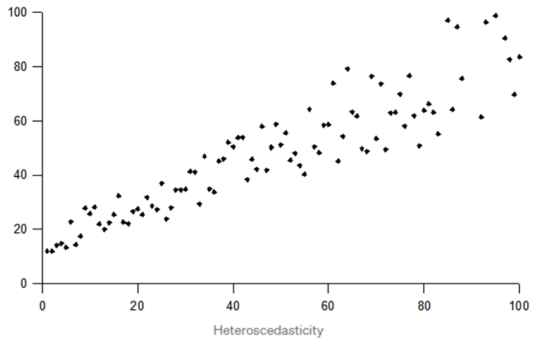


Figure 21: Heteroscedasticity vs homoscedasticity [51].

Accuracy, Precision and Recall

Accuracy, Precision and Recall are well known metrics used on classification problems in machine learning. They are extremely important when it comes to statistical hypothesis testing. When dealing with classification problems, we are attempting to predict a binary outcome, like "*Is it malware or not?*" and "*Does the malware belong to this family or not?*". Where it is important to consider the number of predictions that have been falsely classified as positive and falsely classified as negative, especially considering the context of what is intended to be predicted. Therefore, correctly classified samples are designated by True Positives (TP) or True Negatives (TN), depending on the class to which they belong. Misclassified samples are referred to as False Positives (FP) if the actual class is negative, but the predicted class is positive, and False Negatives (FN) otherwise [52].

The accuracy is the ratio of correct predictions divided by the total number of executed predictions, computed as:

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

Precision is the ratio between the correct predictions for a given class and its number of samples, which represents the percentage of relevant results:

$$Precision = \frac{TP}{Actual\ Results}\ or\ \frac{TP}{TP + FP}$$

Recall is characterized as the percentage of relevant results that are correctly classified by the model:

$$Recall = \frac{TP}{Predicted\ Results}\ or\ \frac{TP}{TP + FN}$$

F1-Score

The F1-Score metric takes both precision and recall into account to measure the accuracy of the model. False positives and false negatives can be critical depending on what is being predicted, but true negatives are frequently less important. The F1-Score attempts to adjust for this by assigning more weight to false negatives and false positives while excluding large numbers of true negatives from the prediction.

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

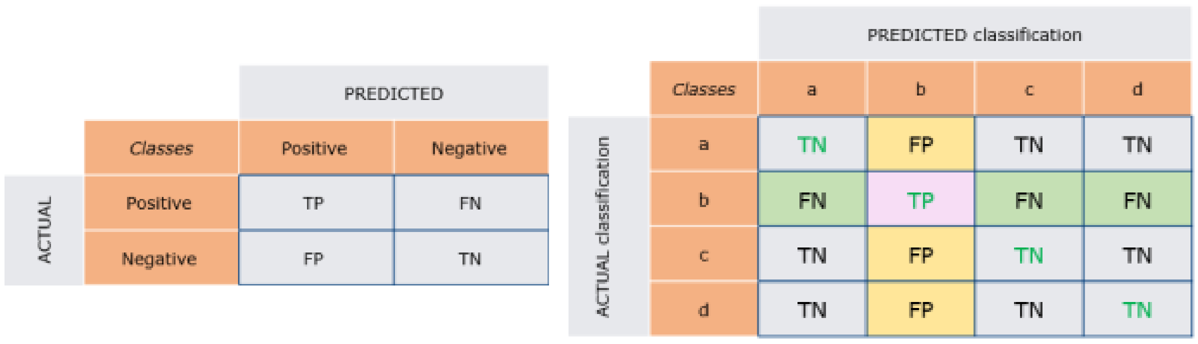
Confusion Matrix

A confusion matrix is a technique for describing the performance of a statistical classification model. Classification accuracy alone can be misleading if the number of observations in each class is not the same (e.g., imbalanced dataset), or if the dataset has more than two classes. Calculating a confusion matrix can provide a better understanding of the classification model's correct behaviour and help to identify the errors that can be made.

The number of accurate and erroneous predictions is summarized with count values and divided by class. A binary confusion matrix representation is illustrated in Figure 22 (a), while a multiclass confusion matrix is shown in Figure 22 (b) [53].

a) Binary Confusion Matrix

b) Multiclass Confusion Matrix



		PREDICTED	
	Classes	Positive	Negative
ACTUAL	Positive	TP	FN
	Negative	FP	TN

		PREDICTED classification			
	Classes	a	b	c	d
ACTUAL classification	a	TN	FP	TN	TN
	b	FN	TP	FN	FN
	c	TN	FP	TN	TN
	d	TN	FP	TN	TN

Figure 22: Binary / Two-class(Left) and Multi class Confusion Matrix [53].

4.4 CNN Architecture

Malware identification and recognition are incredibly complex topics, where there is no one-size-fits-all solution. As a result, AV vendors focus on hybrid approaches that combine conventional signature-based, heuristic-based, and machine learning methods with human analysis.

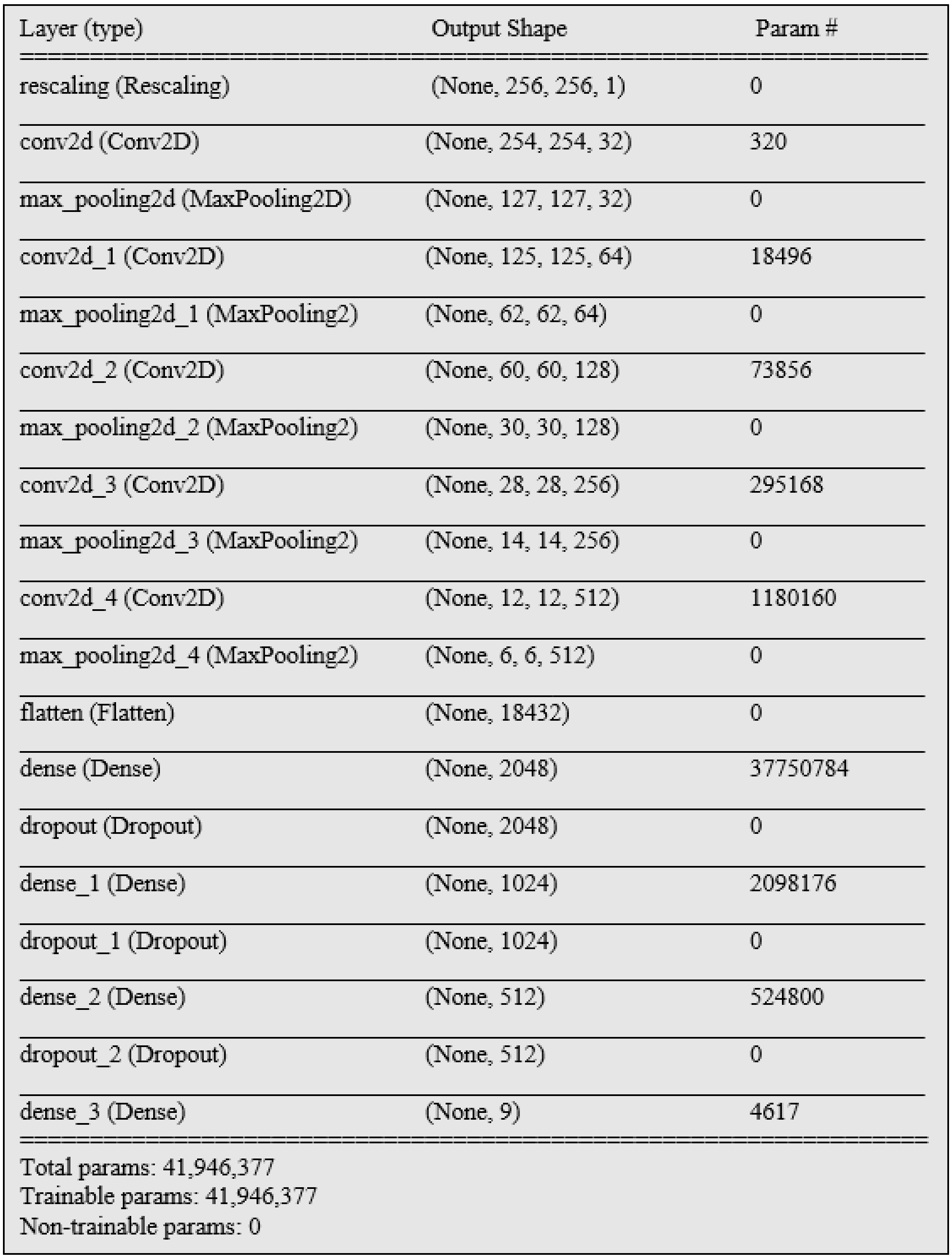
In this thesis, we propose a novel approach to classify malicious software present in emails attachments through their transformation into greyscale images and taking advantage of Convolutional Neural Networks to learn a features hierarchy from pixels to the layers of the classifier. Furthermore, we adapted the proposed method to preserve users privacy by encrypting their email attachment images representation with homomorphic encryption, which enables classifying images without decrypting data [40], [41]. Finally, the performance and evaluation of the novel model will be compared to the more complex models mentioned in Section 2.6 and the Gibert *et al.* [16] model described in Section 3.1.

4.4.1 Neural Network

The traditional ML models (Section 2.6) were dimensioned and trained considering image classification from the ImageNet database [30] as one of the key goals. The images that comprise it are mainly composed of distinct and well-defined objects. Unlike abstract pictures produced by the transformation of binary code, which features a lot of noise due to the conversion process. In this sense, the proposed model considers a large number of features (Section 2.3) in order to catch as many similarities as possible, while attempting to eliminate noise caused by the transformation through dropouts, to achieve the best possible result with the fewest number of layers and at a lower training time.

The proposed model has not been subjected to the rigorous tests that have evaluated traditional models from the ML community over the years, but it is intended to test a different model structure, more focused on abstract image analysis and compare it with traditional models already tested and proven. An NN model based on Gibert *et al.* [16] work was developed, containing an input shape form of 256 × 256 pixels images, five convolutional layers of 32, 64, 128, 256, and 512 neurons each with ReLU activation functions and a learning rate of 0.001 deployed on the Adam optimizer. Finally, a Categorical Crossentropy loss function was applied, used in multiclass classification problems to find the class with the highest correspondence among all potential classes.

We propose an NN model containing four dense layers, four dropouts, and an output layer with softmax activation function. The dropout layer was utilized with a factor of 0.5, which means that half of the neuronal connections are discarded and neurons that enter the dropdown leave with half of the connections. Figure 23 shows the architecture of the NN, where it achieves 41,946,377 trainable parameters with the Microsoft dataset (Section 4.1).



```
Layer (type)                        Output Shape              Param #
=====================================================================
rescaling (Rescaling)               (None, 256, 256, 1)       0
_____________________________________________________________________
conv2d (Conv2D)                     (None, 254, 254, 32)      320
_____________________________________________________________________
max_pooling2d (MaxPooling2D)        (None, 127, 127, 32)      0
_____________________________________________________________________
conv2d_1 (Conv2D)                   (None, 125, 125, 64)      18496
_____________________________________________________________________
max_pooling2d_1 (MaxPooling2)       (None, 62, 62, 64)        0
_____________________________________________________________________
conv2d_2 (Conv2D)                   (None, 60, 60, 128)       73856
_____________________________________________________________________
max_pooling2d_2 (MaxPooling2)       (None, 30, 30, 128)       0
_____________________________________________________________________
conv2d_3 (Conv2D)                   (None, 28, 28, 256)       295168
_____________________________________________________________________
max_pooling2d_3 (MaxPooling2)       (None, 14, 14, 256)       0
_____________________________________________________________________
conv2d_4 (Conv2D)                   (None, 12, 12, 512)       1180160
_____________________________________________________________________
max_pooling2d_4 (MaxPooling2)       (None, 6, 6, 512)         0
_____________________________________________________________________
flatten (Flatten)                   (None, 18432)             0
_____________________________________________________________________
dense (Dense)                       (None, 2048)              37750784
_____________________________________________________________________
dropout (Dropout)                   (None, 2048)              0
_____________________________________________________________________
dense_1 (Dense)                     (None, 1024)              2098176
_____________________________________________________________________
dropout_1 (Dropout)                 (None, 1024)              0
_____________________________________________________________________
dense_2 (Dense)                     (None, 512)               524800
_____________________________________________________________________
dropout_2 (Dropout)                 (None, 512)               0
_____________________________________________________________________
dense_3 (Dense)                     (None, 9)                 4617
=====================================================================
Total params: 41,946,377
Trainable params: 41,946,377
Non-trainable params: 0
```

Figure 23: Proposed CNN model architecture.

4.4.2 Model's Performance and Evaluation

The architecture of the proposed model was described in the previous section. Using the Microsoft dataset, this section presents an empirical evaluation of the proposed model's performance in comparison to popular models (Section 2.6) and the Gibert *et al.* (Section 3.1) model. The goal of this performance analysis is to find the best overall model for the Malwizard solution by achieving the highest accuracy percentage while minimizing training time. All aspects about Malwizard will be covered in Chapter 5, including the solution performance.

To minimize possible interferences in the tests, all experiments were performed in the same environment, a single computer with the following hardware specification:

- CPU: AMD Ryzen 9 3900x (12 Cores)
- Memory: 32 GB RAM
- GPU: AMD Vega 56 (8GB VRAM)
- GPU API: ROCm 4.1.1
- OS: Ubuntu 20.04 LTS (5.4.0-54 kernel)
- ML Implementation: Keras and Tensorflow 2.4.2

Due to the highly unbalanced classes in the Microsoft dataset, where the number of samples in one class far outnumbers the number of instances in another, it is required to uniformize the model such that all classes are taken into account equally. For this approach, Class Weighting (Section 3.4) was chosen to balance the nine classes on the Microsoft dataset due to the large sample discrepancy between the classes. The class with the highest number of samples has 2942 entries, while the one with the lowest number has 42.

The model was trained and verified using a data random split of 80 percent for training and 20 percent for testing. Before the test, all the images were resampled to 256 × 256 pixels using *area interpolation*, which resamples based on the relation of surrounding pixels [54]. The training phase was performed for 200 epochs, which represents the number of times the model was trained using the training data, with the default Keras batch size of 32. To maintain a uniform distribution, the weights for balancing the sample were derived using the following formula:

$$Class\ Weight = \frac{1}{Number\ of\ Samples\ in\ Class} \times \frac{Total\ of\ Samples}{Number\ of\ Families}$$

Figure 24 presents the results during the training (Train) and validation processes (Val). The train line in the Recall graph (c) does not alter significantly from epoch 50 in the training curve, and the validation curve remains in the 0.9 recall range. Furthermore, the loss ((a) loss graph) in the validation curve grows, implying that the longer the model is trained, the greater the loss in the validation is. Contrarily, the loss in the training curve tends to decrease. The model also appears to be over-fitting in training since the accuracy (b), precision (d), and recall (c) curves are unstable and all tend to one. On the other side, this also might occur as a result of the excessively imbalanced data.

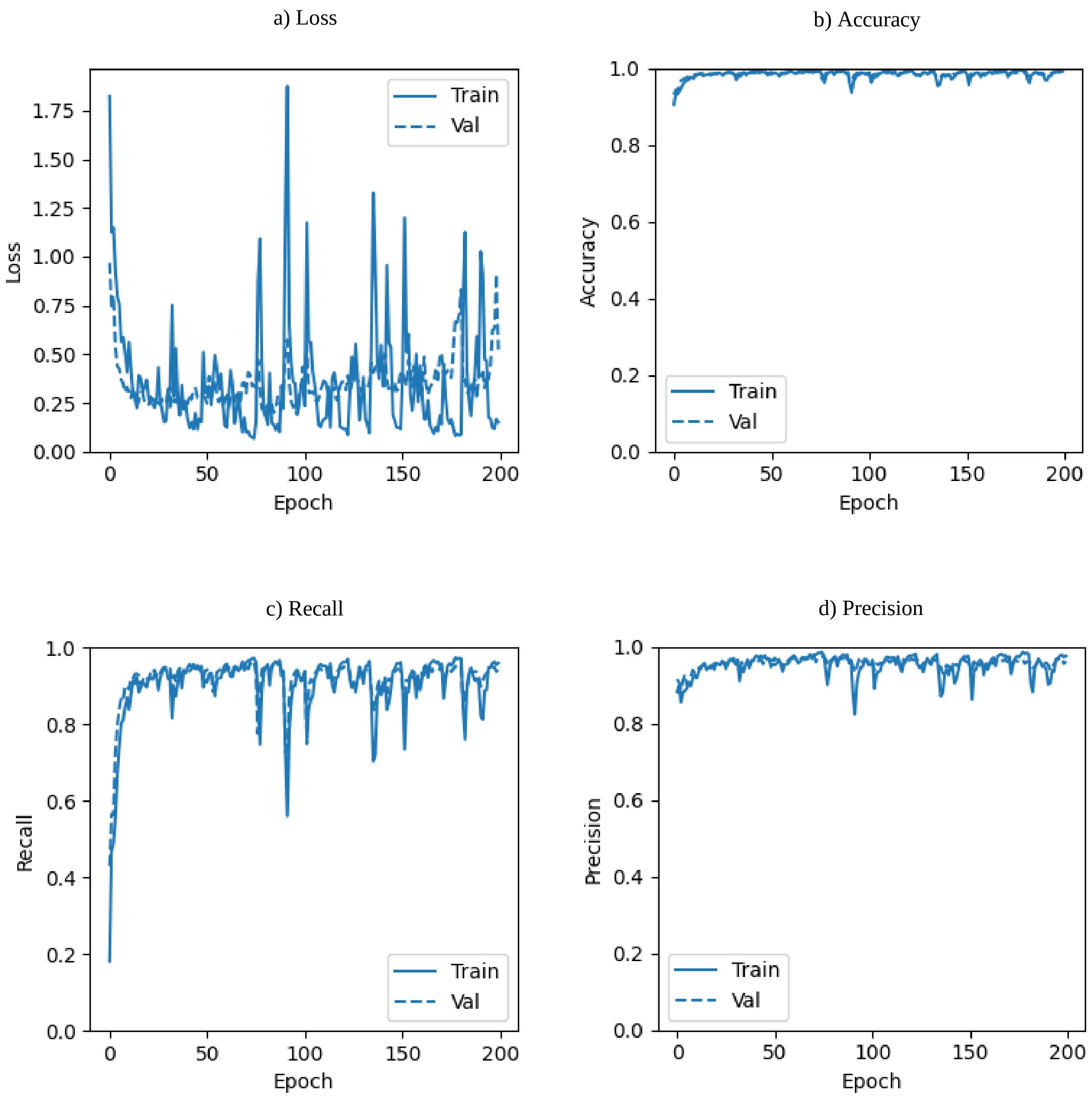


Figure 24: Variation of the main metrics for evaluating the model as the number of epochs grows.

Finally, the learned model was used to make predictions on the training data, achieving an F1-Score average (avg) of 96.19 percent. The model's confusion matrix, shown in Table 1, shows how this NN reacts in the predicted malware classes. It can accurately predict all classes, except the ones that have fewer samples. Even with weight balancing, increasing samples in these classes or reducing the gap to a more manageable level with the aid of class balancing is critical for increasing and uniformizing accuracy.

Malware Family	Precision	Recall	F1-Score	Support
Gatak	0.917874	0.989583	0.952381	192
Kelihos_ver1	0.908046	0.975309	0.940476	81
Kelihos_ver3	0.991843	0.996721	0.994276	610
Lollipop	0.983299	0.971134	0.977178	485
Obfuscator.ACY	0.974249	0.900794	0.936082	252
Ramnit	0.955479	0.914754	0.934673	305
Simda	0.666667	0.888889	0.761905	9
Tracur	0.890909	0.960784	0.924528	153
Vundo	0.952941	0.941860	0.947368	86
Accuracy			**0.961804 (96.18%)**	2173
Macro avg	0.915701	0.948870	**0.929874 (92.99%)**	2173
Weighted avg	0.963139	0.961804	**0.961909 (96.19%)**	2173

Table 1: Confusion Matrix of the Proposed Model.

K-fold Cross Validation

K-fold cross validation is a method used to measure the generalization performance, which helps to validate the overall results obtained in Table 1. The data is divided into K folds of equal size. A single subsample from the K subsamples is kept as validation data for testing the model, this being 20 percent, while the others are utilized as training data, 80 percent. This technique is done as many times as the number of folds, with each of the K folders serving as validation data exactly once [55].

The results of the 5-fold and 10-fold tests for the proposed model are presented in Table 2 and Table 3.

5-Fold Test	Accuracy	Macro AVG F1-Score	Weighted AVG F1-Score (min)
1	0.940230 (94.02%)	0.853941 (85.39%)	0.939520 (93.95%)
2	0.956322 (95.63%)	0.841059 (84.10%)	0.953014 (95.30%)
3	0.940092 (94.00%)	0.808700 (80.87%)	0.938738 (93.87%)
4	0.928571 (92.86%)	0.876935 (87.69%)	0.928945 (92.89%)
5	0.933180 (93.31%)	0.811034 (81.10%)	0.932164 (93.22%)
Total (avg)	**0.939679 (93.97%)**	**0.838334 (83.83%)**	**0.938476 (93.85%)**

Table 2: Performance 5-Fold test for classification of Microsoft dataset.

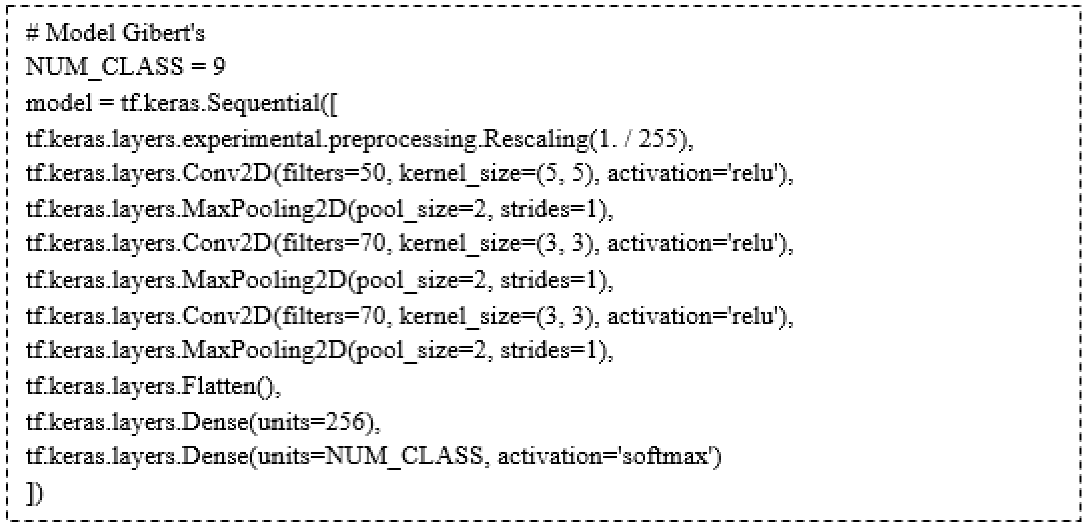
10-Fold Test	Accuracy	Macro AVG F1-Score	Weighted AVG F1-Score (min)
1	0.940367 (94.04%)	0.902099 (90.21%)	0.940637 (94.06%)
2	0.912844 (91.28%)	0.779254 (77.93%)	0.910203 (91.02%)
3	0.930876 (93.09%)	0.810115 (81.01%)	0.928611 (92.86%)
4	0.912442 (91.24%)	0.791727 (79.17%)	0.912438 (91.24%)
5	0.912442 (91.24%)	0.792789 (79.28%)	0.911164 (91.11%)
6	0.926267 (92.63%)	0.902182 (90.22%)	0.926554 (92.66%)
7	0.936111 (98.61%)	0.903654 (90.37%)	0.936424 (93.64 %)
8	0.917051 (91.71%)	0.774705 (77.47%)	0.919697 (91.97%)
9	0.912037 (91.20%)	0.784986 (78.50%)	0.907264 (90.73%)
10	0.921296 (92.13%)	0.775749 (77.58%)	0.918941 (91.89%)
Total (avg)	**0.922173 (92.22%)**	**0.821726 (82.17%)**	**0.921193 (92.12%)**

Table 3: Performance 10-Fold test for classification of Microsoft dataset.

As can be seen in Table 2 and Table 3, the overall weighted average of the F1-Score on the k-fold tests does not differ much from the overall result (96.1 percent). The existing discrepancy occurs due to the fact that the dataset was split, which results in a lesser number of samples in each test, decreasing the final result. This confirms that the model is accurate, and the precision of 96.19 is valid.

Before proceeding with model comparisons, the Gibert model in Figure 13 must be generated in Keras, the same framework that was used to generate the traditional models and to develop the novel model. Despite the findings of the Gibert et al. model were made public, the test environment is not the same. Furthermore, the F1-Score was not implemented in Tensorflow 1.9, which was utilized in their study, since it was introduced in Tensorflow 2.0 [56]. This shows that a non-standard F1-Score function, which is not included in the public code [57], was used to evaluate the model. To obtain the best results with the least amount of interference, all models must be tested under the same conditions.

The following code snippet (Figure 25) represents the implementation of Figure 13 in Keras:



```
# Model Gibert's
NUM_CLASS = 9
model = tf.keras.Sequential([
tf.keras.layers.experimental.preprocessing.Rescaling(1. / 255),
tf.keras.layers.Conv2D(filters=50, kernel_size=(5, 5), activation='relu'),
tf.keras.layers.MaxPooling2D(pool_size=2, strides=1),
tf.keras.layers.Conv2D(filters=70, kernel_size=(3, 3), activation='relu'),
tf.keras.layers.MaxPooling2D(pool_size=2, strides=1),
tf.keras.layers.Conv2D(filters=70, kernel_size=(3, 3), activation='relu'),
tf.keras.layers.MaxPooling2D(pool_size=2, strides=1),
tf.keras.layers.Flatten(),
tf.keras.layers.Dense(units=256),
tf.keras.layers.Dense(units=NUM_CLASS, activation='softmax')
])
```

Figure 25: Gibert's Keras implementation.

Table 4 presents the findings of the empirical performance evaluation for the different models, comparing the trainable parameters, weighted F1-Score attained, *epoch average time* and *overall training time*. The epoch average time represents the average time in each training iteration over the 200 epochs, while overall training time is the total duration of the process.

Model	Params	Accuracy	F1-Score	Epoch Time (sec/avg)	Time to Train (min)
ResNet-50	23 546 761	97.52%	97.52%	1350s	4509.538
VGG16	165 753 545	93.56%	93.68%	130s	434.983
InceptionV3	21 786 217	97.79%	97.80%	58s	189.333
Gibert's Model	245 386 489	81.36%	80.95%	29s	99.440
Proposed Model	41 946 377	96.18%	96.19%	16s	54.387

Table 4: Models performance comparison for classification of Microsoft dataset.

The VGG16 and Gibert models had the lowest accuracy in the tests, with one of the key reasons being the large number of parameters assessed, which are considered a lot of the noise present in the images making it difficult to identify the most important features. The ResNet-50, on the other hand, has one of the best F1-score. However, the time to train is 23 percent higher than InceptionV3 developed by Google. Although Google's NN achieved the best F1-Score (97.80 percent) of all four models, the proposed model achieved a close value on F1-Score at less than one third of the training time with double of the training parameters.

Keras results for Gibert's model are vastly different compared with what they published in the study. The total accuracy attained in Keras implementation was 81.36 percent (Table 4). Reaching 52.26 (Table 6) and 54.59 (Table 7) percent for the 5-fold and 10-fold cross validation tests respectively, much lower than the 97.3 and 97.5 percent published [16]. Because the dataset must be split, the accuracy in the k-fold tests tends to be lower than overall accuracy, as a consequence of the lower number of samples in each test. Table 5 shows the model confusion matrix for Gibert's Keras implementation.

Malware Family	Precision	Recall	F1-Score	Support
Gatak	0.589286	0.515625	0.550000	192
Kelihos_ver1	0.931507	0.839506	0.883117	81
Kelihos_ver3	0.986928	0.990164	0.988543	610
Lollipop	0.746141	0.896907	0.814607	485
Obfuscator.ACY	0.872510	0.869048	0.870775	252
Ramnit	0.743772	0.685246	0.713311	305
Simda	0.000000	0.000000	0.000000	9
Tracur	0.582734	0.529412	0.554795	153
Vundo	0.868852	0.616279	0.721088	86
Accuracy			**0.813622 (81.36%)**	2173
Macro avg	0.702414	0.660243	**0.677360 (67.74%)**	2173
Weighted avg	0.811368	0.813622	**0.809535 (80.95%)**	2173

Table 5: Confusion Matrix of the Gibert Model.

The results of the 5-fold and 10-fold tests for the Gibert's model are presented in Table 6 and Table 7.

5-Fold Test	Accuracy	Macro AVG F1-Score	Weighted AVG F1-Score (min)
1	0.528736 (52.87%)	0.432769 (43.28%)	0.486894 (48.69%)
2	0.526437 (52.64%)	0.419545 (41.95%)	0.484018 (48.40%)
3	0.532258 (53.23%)	0.422431 (42.24%)	0.499228 (49.92%)
4	0.529954 (53.00%)	0.401976 (40.20%)	0.484921 (48.49%)
5	0.495392 (49.54%)	0.381608 (38.16%)	0.458928 (45.89%)
Total (avg)	**0.522555 (52.26%)**	**0.411666 (41.17%)**	**0.482798 (48.28%)**

Table 6: Gibert's model performance 5-Fold test for classification of Microsoft dataset.

10-Fold Test	Accuracy	Macro AVG F1-Score	Weighted AVG F1-Score (min)
1	0.660550 (66.06%)	0.457428 (45.74%)	0.595371 (59.54%)
2	0.536697 (53.67%)	0.446229 (44.62%)	0.483920 (48.39%)
3	0.456221 (45.62%)	0.425299 (42.53%)	0.448255 (44.83%)
4	0.493088 (49.31%)	0.411959 (41.20%)	0.443503 (44.35%)
5	0.506912 (50.69%)	0.428766 (42.88%)	0.436272 (43.63%)
6	0.493088 (49.31%)	0.414958 (41.50%)	0.432534 (43.25%)
7	0.490741 (49.07%)	0.392032 (39.20%)	0.440053 (44.01%)
8	0.488479 (48.85%)	0.436586 (43.66%)	0.457958 (45.80%)
9	0.657407 (65.74%)	0.451765 (45.18%)	0.585097 (58.51%)
10	0.675926 (67.59%)	0.469577 (46.96%)	0.601491 (60.15%)
Total (avg)	**0.545911 (54.59%)**	**0.433446 (43.34%)**	**0.492445 (49.24%)**

Table 7: Gibert's model performance 10-Fold test for classification of Microsoft dataset.

Another important metric is the weighted average of F1-Score values, which takes dataset balancing into account and is the most commonly used metric for evaluating model accuracy with imbalanced datasets. However, the Gibert's study only presents the macro average, which yields the average without taking into account the proportion for each class in the dataset. Additionally, they did not stated if it was used any approach to balance the dataset that allows the macro average metric to be considered.

Since it is not indicated, comparing the macro average value in Table 5, 67.74 percent is considerably lower than what was achieved in the Gibert's study: 92.7 percent for the 5-fold test and 94 percent for the 10-fold test [16]. Additional metrics such as epoch time, training time, and ROC assessment, could be used to evaluate the performance of the model and help to find the best model, but they were not made public. Taking into consideration the findings and information supplied by *Gibert et al.* [16], the novel model outperforms theirs in all metrics.

By observing Figure 26, which depicts the F1-Score variation for the complex three models (ResNet-50, VGG16, InceptionV3) and the novel model, it can be seen that all of the models tend to overfit, indicating that they do not improve the accuracy over 100 epochs. Thus, it allows to establish the training boundary and fit the models to avoid overfitting during the training phase.

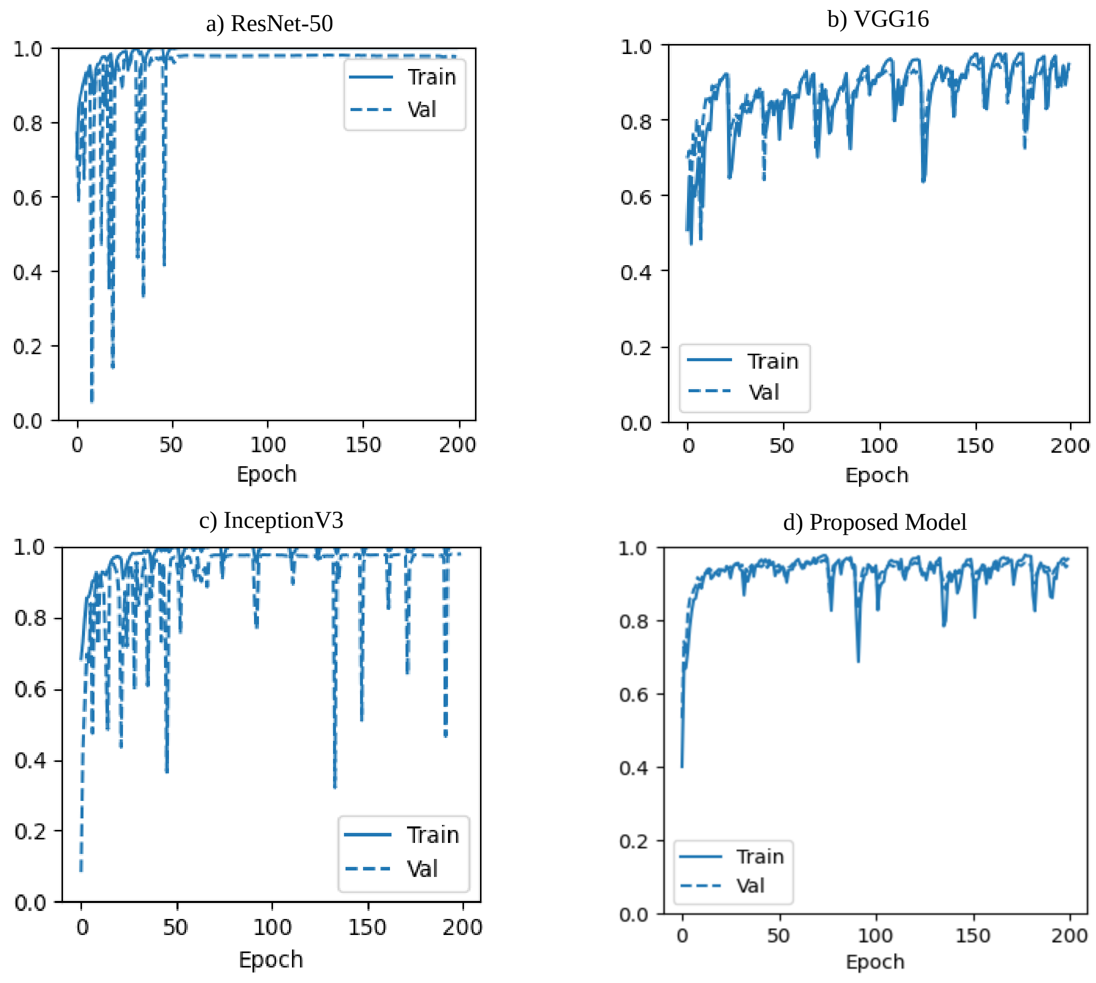


Figure 26: Models variation of the F1-Score.

Receiver Operating Characteristics

The Receiver Operating Characteristics, also known as ROC graph, is often used in binary classification to analyse the output of a classifier. It is another common approach to measure the model's performance. To apply the ROC curve to multi-label classification, the result must be binarized [58]. Each class is represented by a distinct curve in the ROC graph, as well as two additional curves: macro-average and micro-average. A macro-average curve computes the metric separately for each class and then averages it (therefore treating all classes equally), whereas a micro-average curve aggregate all class contributions to computing the average metric.

Figure 27 shows the proposed model findings. The Area Under the ROC Curve (AUC) value varies between 0 and 1, 0 and 100 percent respectively, with the Lollipop curve being the poorest, 46 percent. AUC provides an aggregate measure of performance across all possible classes, reflecting the likelihood that the network will identify one class over the others. In this situation, the network had a harder time to successfully identify the Lollipop malware samples than the other types, which can lead to misclassifying the samples as Lollipop instead of the correct class.

To evaluate the obtained AUC values of the proposed model, it was compared with the model that hit the best F1-Score value in the tests. Figure 28 shows the ROC curves for the InceptionV3 model, where the same behaviour is observed in the Lollipop class (56 percent) having similar results in the other classes. As a result, it demonstrates that the proposed model produced comparable results while taking less time during the training phase.

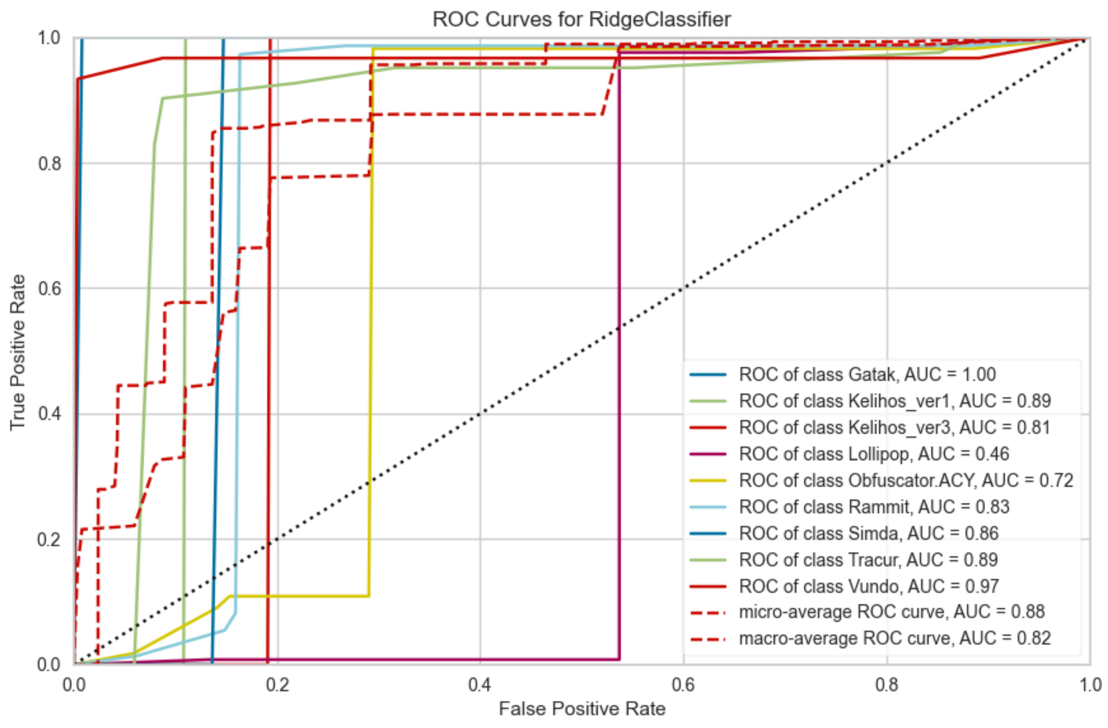


Figure 27: Proposed Model ROC curve all classes.

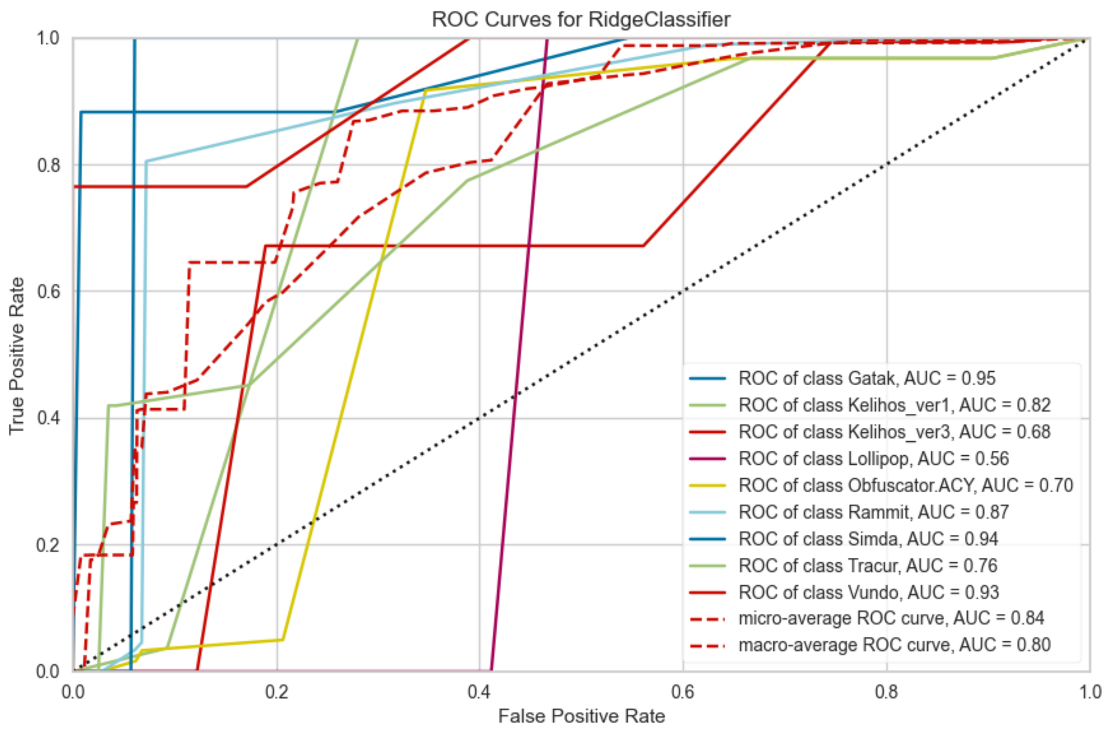


Figure 28: InceptionV3 ROC curve all classes.

Encrypted Dataset

Machine learning's popularity and success have made it possible to utilize ML models in a range of online applications. Companies are making use of machine learning's ability to handle enormous volumes of their customers' data, forcing the need to ensure that the data stays secure and private.

Homomorphic cryptography (HE) appears as a solution to the ML privacy problem, allowing computation on encrypted data. The training data is encrypted and then ML uses the encrypted data to directly produce an encrypted prediction that only the owner could decrypt. Since the most common NNs and the proposed model were shown to be successful in a malware classification, an HE implementation called Paillier Homomorphic Encryption (PHE) was used to protect the privacy of malware images generated from the Microsoft dataset.

The combination of the PHE and CNN is depicted in the process shown in Figure 29. Data is encrypted using a public key during pre-processing and cannot be decrypted without knowing the private key. As a result, the CNN-based model can only access encrypted data (cipher-images). Instead of training malware images, the focus is now on cypher-images data, which is made possible by the Paillier encryption scheme's partly homomorphic characteristic [59]. The process involves encrypting each pixel in an image and applying a modulo 256 operation to return values between 0 and 255, allowing a ciphertext to be represented as an image.

Following the training phase, the model is tested using the new cypher-images data that has been encrypted with the same public key as the training procedure. As a result, data privacy is protected during both training and testing, ensuring that image data processing is secure and unauthorized parties are unable to decrypt data.

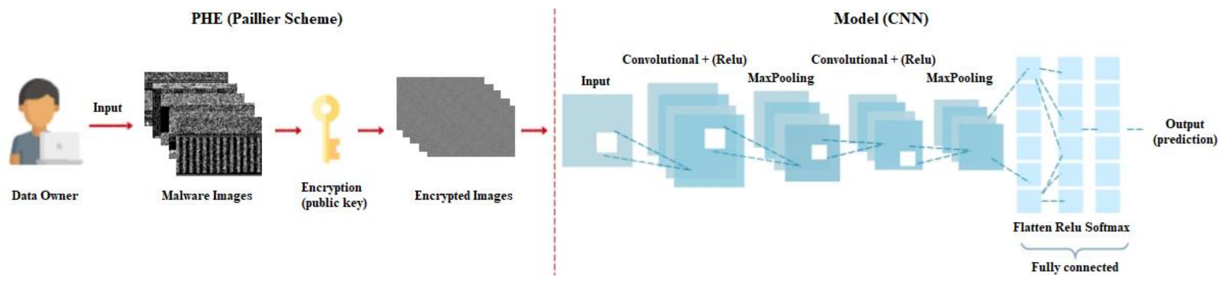


Figure 29: Paillier Homomorphic Encryption CNN workflow.

When an image is homomorphically encrypted, noise is produced in the ciphertext that is directly proportional to the cypher key size, which is often bigger than the input data. Security Best Practices [60] recommend 2048-bit keys for asymmetric encryption. However, in order to reduce noise during the tests and test the performance, 128-bit keys were used to determine if it is feasible to analyze encrypted data using the models examined in Section 4.4.2. Note that weak (e.g., 128-bit) keys are instantly breakable, making their use unacceptable in a real-world environment.

The images were resampled to a 256 × 256 shape and encrypted with the 128 bits keys. Table 8, shows the results obtained for the four models under analysis, based on the training of 50 epochs:

Model	Params	F1-Score	Epoch Time (sec/avg)	Time to Train (min)
ResNet-50	23 546 761	15.88%	1350s	1127.31
VGG16	165 753 545	0.3%	130s	109.15
InceptionV3	21 786 217	1.2%	58s	49.28
Proposed Model	41 946 377	0.3%	16s	13.30

Table 8: Models performance comparison for classification of Microsoft encrypted dataset.

The F1-Score values obtained in Table 8 highlight the problem that as expected, it is not the best practice to utilize the traditional CNN approaches to analyse ciphered datasets. Aside from activation functions, all operations in a NN are additions and multiplications, which can be performed on homomorphically encrypted data. However, to analyse the output, the last layer, the softmax function, should be replaced with one that can evaluate polynomials, as suggested by Hesamifard *et al.* [41].

In a future study, it would be interesting to be able to evaluate and compare the behaviour of polynomial functions applied to the analysis of encrypted malware. To determine which factors can impact due to noise present in images and try to get better performance.

Chapter 5

Malwizard

Malwizard is an adaptable Python solution suited for companies or end-users, that allows them to automatically obtain a fast malware analysis. This chapter describes the solution specifications, as well as its design, development, and components. Its security mechanisms are presented, and Malwizard's performance is evaluated with and without the use of cryptography in the sample to be analysed, regardless of the model used. A tutorial on how to use Malwizard is available in Appendix A.

5.1 Requirement Analysis

5.1.1 Functional Analysis

The biggest challenge for Malwizard is to achieve a high level of confidence in malware identification at a reduced evaluation cost. The first functional requirement, *the automated extraction of suspicious malware files from the emails and convert them into greyscale images*, could be solved by an add-in for Microsoft Outlook and an API service in a way to adapt into different email applications or any incident response platforms. Then, images are sent to the ML processing unit of the Malwizard to be evaluated.

The second functional requirement is the *malware identification process with Malwizard*, which receives the image from the add-in or API, extracts the features from the image, and test the potential malicious intent. In order to solve this requirement, a CNN model with regular updates and a diverse training set is essential. As a result, a good foundation for identifying malware is built to detect threats in advance and help to prevent zero-day attacks.

In terms of information management, Malwizard must only access the email attachments in order to convert their binary code into images. A privacy option is also provided for testing purposes as it needs to be tested and adapted so that in the future models that apply polynomial functions can be used. It applies homomorphic encryption into greyscale images on the client-side, before sending them to the ML processing unit, which prevents a reverse engineer attack on the images to obtain the original file. Furthermore, the ML processing unit stores the image analysis to enhance the model, as well as a small amount of mandatory data. This information data is part of the ML processing unit, related to the communication layer and services execution.

5.1.2 Non-functional Analysis

Compliance. In order to comply with the General Data Protection Regulation (GDPR) requirements, no sensitive information is collected, and all gathered information is subject to the bare minimum requirements.

Correctness. A correct CNN model can be characterized as given a set of training data with respective classification classes, identifies the correct class with a high level of confidence, when an unknown sample is presented. Therefore, the correctness of malware identification is directly correlated to the training sample. If users do not have a diverse and reasonable size training set, the Malwizard cannot guarantee an accurate malware identification.

Documentation. All classes in Malwizard code must be documented in order to help users to understand the methods, as well as encourage them to develop and improve the work.

Extensibility. Malwizard provides an API, which receives the image to be analysed. As a result, users can create new middleware layers to connect with various applications, such as email applications or incident response frameworks, without having to change the ML processing unit.

Licensing. Malwizard source code is proprietary. However, it offers an open-source API that allows the development of customized middleware for specialized applications.

Maintainability. In order to detect new types of malware, high maintainability is a mandatory function. Malwizard must support continuous sample updates, which can be with data gathered in each analysis or through sample modules, in order to improve the detection performance over time.

Scalability. The ML processing unit must maintain the minimum amount of necessary data collected in each training sample in order to remain scalable in the face of the large amounts of data that are needed to achieve a high level of confidence.

Security. An SSL certificate provides security by authenticating the HTML/XML Malwizard pages and the API identification, as well as enabling an encrypted connection. Also, when the privacy option is enabled, the greyscale photos are ciphered on the client-side before being transferred to the ML processing unit, preventing reverse engineering on the image to obtain the original file. To address external attacks, the backend server includes mechanisms that lock traffic if there are more than 500 requests per day or hour and more than 100 requests per minute from the same origin.

Usability. Malwizard must focus on being user friendly, taking into account being the least intrusive while presenting the data succinctly to the end-user.

5.1.3 Architectural Analysis

Malwizard is positioned in the architectural analysis as a component that can be implemented into any email client or email server that conducts message processing, as well as any SOAR platform via the API supplied. Although AVs are generally responsible for detecting malware in attachments, basing detection on fingerprints can lead to long quarantine periods, especially when considered zero-day attacks and, in addition, the large number of emails that companies receive every day. Such analysis can be optimized, reducing the impact times through ML malware analysis. Malwizard must be an add-in/API that could be instantiated by clients, sysadmins into the email servers or incident response platforms to become responsible for the emails attachments analysis. Therefore, it must be responsible for generating alerts according to the result of the analysis and enabling them to be exported to a SIEM platform. Diagram (b) in Figure 30 shows a potential Malwizard placement to client software and malware detection. Malwizard's internal components are described and discussed in the following sections.

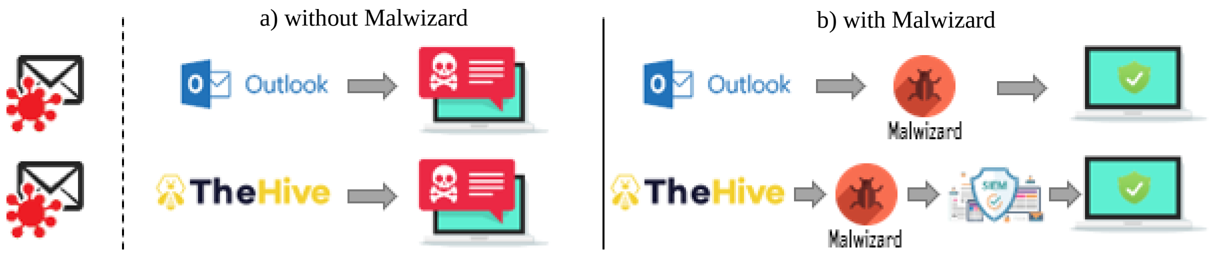


Figure 30:Malwizard positioning.

5.2 Implementation

All the components present in Figure 31 will be presented in detail, which includes the two types of implementations suggested in this thesis, the Microsoft Outlook Add-in developed using HTML and JavaScript languages, and an API/backend Python service, which can be used by the risk-based Security Orchestration, Automation and Response (SOAR) platforms, like TheHive [17].

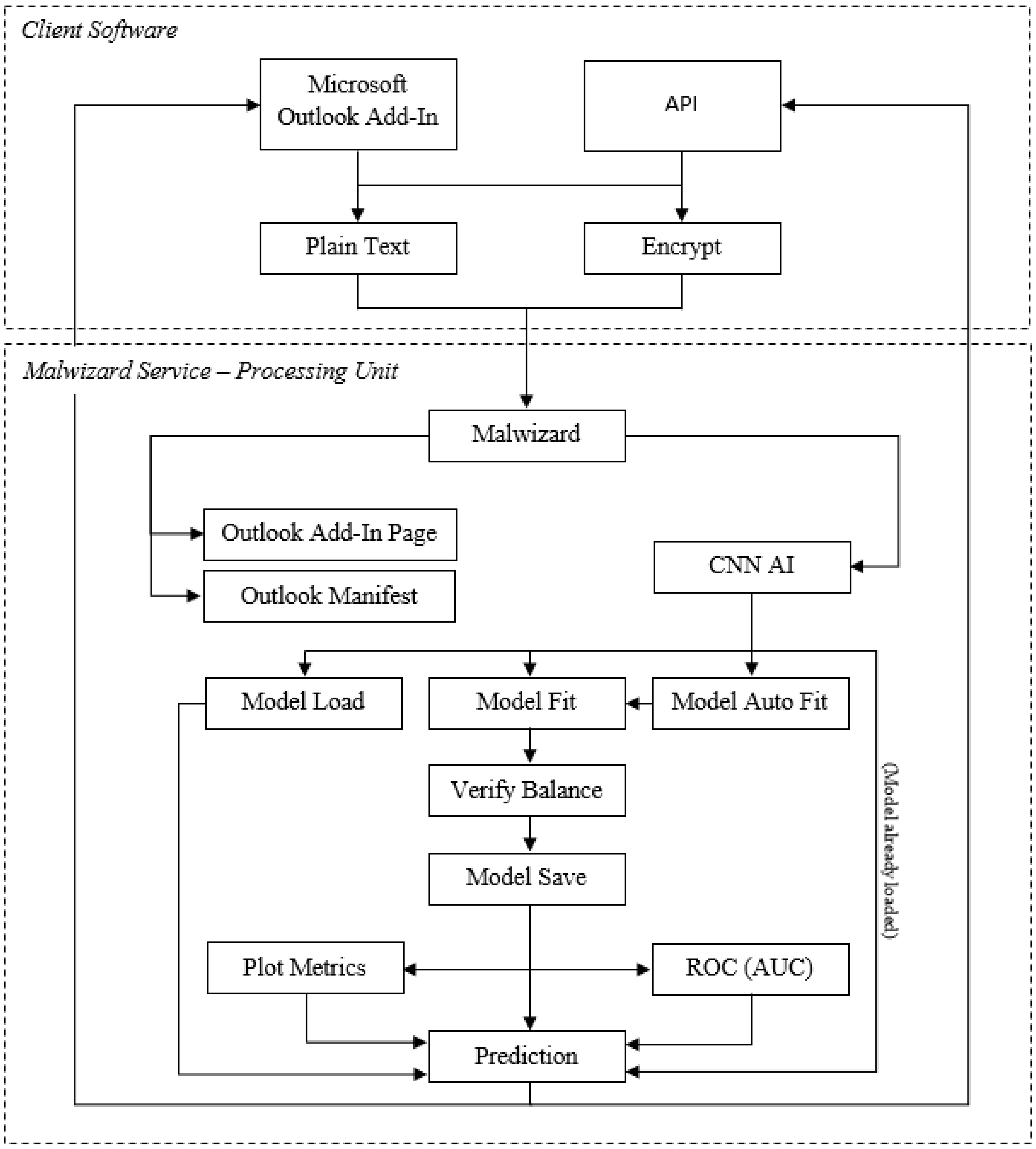


Figure 31:Malwizard flow diagram.

5.2.1 How does it work?

User-friendly and Simplicity are the words that best describe Malwizard. The process to analyse the intended malware samples is composed of three parts: Microsoft Outlook Add-In, API, and ML processing unit (containing the machine learning model).

Malwizard workflow is directly correlated with client configuration, which may be either an end-user or a backend implementation.

Starting with Microsoft Outlook (an end-user application), clients are responsible to install the add-in, through the Office store by adding the Outlook manifest.xml URL, hosted on the Malwizard. Once, the initial configuration is finished, Malwizard appears if the Outlook detects an attachment present in the email that the user is viewing. Malwizard converts it into a binary image representation and sends it to the ML processing unit to be analysed in one of the five machine learning models addressed in Section 4.4.

In an API implementation like to TheHive, clients are responsible for creating calls or adapting the middleware developed for the TheHive to communicate with the ML processing unit. Since the middleware is open source, it can be customized to suit the needs of the clients or the applications.

The ML processing unit, where the machine learning model is running, receives the image file representation and applies the convolution process, which extracts the features and exposes them to trigger filters. After all filters have been applied and the fully connected layer has returned the confidence identification level, if this is greater than 95 percent, the sample will be considered for future analysis and reported to the middleware as malware. Otherwise, it will be marked as containing signs of malware, and caution should be taken. This percentage can be changed by the user to suit its needs.

5.2.2 Design and Components

A diagram with Malwizard's flow is presented in Figure 31. The following descriptions are dedicated to explaining each step, which in the source code are represented by Python functions.

Microsoft Outlook Add-In and API

Microsoft Outlook Add-In and API are the interfaces from which Malwizard ML processing unit will receive the potential malicious files to be analysed. It contains the component responsible to convert the files into image representations and generate encrypted images based on these representations if the privacy flag is instantiated. Then the base64 of the images, a binary-to-text encoding scheme that represents binary data in an ASCII string format [61], with the flag are sent to Malwizard.

Microsoft Outlook Add-In Page

Microsoft Outlook Add-In Page is the front-end page of the Malwizard for Outlook, the interface where the user can select the privacy option, start an attachment analysis, and view the ML conclusion of the files in analysis.

Microsoft Outlook Manifest

Microsoft Outlook Manifest is the XML manifest file that allows the add-in to be installed through the Office store, through a Malwizard URL or by loading this file locally.

Plain Text and Encrypt

Plain Text and Encrypt options are available to the user by a toggle button on the Microsoft Outlook Add-In and by a parameter on the API call, which tells the ML processing unit which type of image needs to be analysed.

Malwizard

Malwizard is the backend server of the solution. It can be deployed in a Cloud Service or On-Premises, provides all public interfaces for the add-in and API clients, and reports the file analysis results. This is also where the ML processing unit, CNN AI, is located.

CNN AI

CNN AI or ML processing unit is responsible for calling the loading, training, compiling and fitting functions that interact with the machine learning model.

Model Load

Model Load loads the ML model progress when the server is initiated or migrate to a new one. Additionally, allows loading external models from the community.

Model Fit

Model Fit trains the ML model for a predetermined number of epochs (default is 100). This is also where the Class Balancing weights are passed to the model.

Model Auto Fit

Model Auto Fit is responsible for starting the Model Fit to retrain the ML model after collecting a determined number of samples which achieved a determined level of accuracy on the running model, where both are defined by the user.

Verify Balance

Verify Balance verifies if the training dataset is balanced. If it is not, it applies the class weighting and calculates them to pass to the Model Fit.

Model Save

Model Save saves the ML model progress after training, avoiding long training times when restarting or migrating the server.

Plot Metrics

Plot Metrics generates the ML model training report which includes metrics like accuracy, precision, recall, and F1-Score. In addition, it calls the Scikit-learn Python module to generate the confusion matrix.

ROC (AUC)

ROC (AUC) generates the ML model ROC curve graph, which allows understanding the likelihood that the network will identify one class over the others.

5.2.3 Performance and Evaluation

Malwizard performance and rating are presented in this section, to simulate normal usage comparing the encrypted and plain text modes.

To evaluate the performance, the proposed model was used due to the best overall results obtained in Section 4.4.2. In the model comparison, the softmax activation function was utilized in the last layer, since the Microsoft database contains only malware and each one belongs to only one class. However, on daily basis, some files are legitimate and cannot be classified as malware. In this sense, it is necessary to have an extra class that represents the legitimate files to apply the softmax function, since it normalizes the output and attempts to discover the class that has the most similarities with the sample under analysis. Another alternative is to use the sigmoid activation function in the last layer, which allows the probability to be calculated individually and represents the maximum probability for each class. Using the sigmoid function can reduce the degree of accuracy, but it does not require a large and diversified set of legitimate file samples.

Table 9 shows the model confusion matrix for the proposed model with Sigmoid output.

Malware Family	Precision	Recall	F1-Score	Support
Gatak	0.509485	0.979167	0.670232	192
Kelihos_ver1	0.896552	0.962963	0.928571	81
Kelihos_ver3	0.996522	0.939344	0.967089	610
Lollipop	0.957606	0.791753	0.866817	485
Obfuscator.ACY	0.715789	0.539683	0.615385	252
Ramnit	0.970464	0.754098	0.848708	305
Simda	0.380952	0.888889	0.533333	9
Tracur	0.616114	0.849673	0.714286	153
Vundo	0.951220	0.906977	0.928571	86
Accuracy			**0.830649 (83.07%)**	2173
Macro avg	0.777189	0.845838	**0.785888 (78.59%)**	2173
Weighted avg	0.873736	0.830649	**0.838520 (83.85%)**	2173

Table 9: Confusion Matrix of the Proposed Model with Sigmoid Output.

As expected, the weighted mean of F1-Score 83.85 percent (Table 9) is lower than with the softmax function, 96.19 percent (Table 1). The Simda class is the worst due to the lowest available samples. However, the other classes achieve a relatively high F1-Score value, enough for the model to be able to distinguish between malware and non-malware samples, without knowing what a legitimate file is. As a result, the sigmoid activation function calculates the highest probability of identical characteristics in the sample under consideration.

The performance tests were carried out ten times each with a file size of 20 MB, which is the maximum attachment default size authorized in Microsoft Outlook [62]. Table 10 depicts the baseline timings when no encryption type was used, showing the request processing time ("Plain Text Time") and the overall time to obtain the prediction from the ML model ("Overall Request Time").

	Plain Text Time (sec)	**Overall Time Request (sec)**
Microsoft Outlook Add-In	1.192	1.502
Middleware - API	0.070	1.160

Table 10: Plain Text performance comparison.

The Plain Text Time (Table 10) includes the resampling process of images made by the Open Source Computer Vision (OpenCV) library developed in C / C ++ [63]. For the Microsoft Outlook add-in, we used OpenCV.js, which extends the OpenCV language binding by providing a JavaScript interface. However, as it runs at a higher level of abstraction, due to the Javascript language, it tends to be slower, 1.192 seconds. Unlike the Python OpenCV implementation, which consists of a wrapper around the original C/C++ code, which makes it faster (0.070 seconds) due to C/C++ being a lower-level programming language than javascript, running closer to the machine level instruction set.

Table 11 compares the HE performance between Javascript (Microsoft Outlook Add-In) and Python (API) implementation.

	Key Size	**Encrypt Time (sec)**	**Overall Time Request (sec)**
Microsoft Outlook Add-In	128-bits	83.975 (≈ 1.40 min)	86.062 (≈ 1.43 min)
Middleware - API	128-bits	21.186 (≈ 0.35 min)	23.294 (≈ 0.39 min)
Microsoft Outlook Add-In	2048-bits	20 189.884 (≈ 336 min)	20 191.324 (≈ 336 min)
Middleware - API	2048-bits	5 093.693 (≈ 85 min)	5 095.798 (≈ 85 min)

Table 11: Encrypt Javascript and Python performance comparison.

Image encryption is a considerable computational challenge, as each pixel in the image must be encrypted, increasing processing time. Using a 128-bit key to process homomorphic encryption on 256 × 256 images takes about 83 seconds for the Javascript-based Microsoft Outlook add-in. If the key is increased to the recommended values, 2048 bits [60], it takes 336 minutes, which is not feasible for real-time processing. The same goes for the API implemented in Python, it takes 21 seconds with a 128-bit key, but 85 min for the 2048-bit key.

However, because of the 128-bit keys security weakness, strong keys (e.g., 2048-bit keys) are highly recommended, despite the fact that their poor performance prevents our solution from being used in production.

Chapter 6

Conclusions

6.1 Final Remarks

The main goal of this thesis was to provide a new approach for assessing malware, in comparison to existing ones that rely on known signatures and quarantine assessment. This broad goal resulted in two major contributions: a proposal for a novel ML model and its implementation in Malwizard, a malware analysis solution for Microsoft Outlook and an API for the Orchestration, Automation, and Security Response (SOAR) platforms using machine learning analysis.

Regarding the first contribution, a proposal for a novel ML model, it was analysed the transformation of malware files into greyscale images, which allows an ML to assess the patterns contained in them. A novel model was designed and customized to achieve the goal while being more efficient than traditional models known in the community. In addition, the idea of applying standard models in the evaluation of encrypted images was explored. However, the negative outcomes reinforce the necessity to replace the output layer of traditional models with functions that can evaluate polynomials, as Hesamifard *et al.* recommended [41].

The tests were carried out with the help of the Keras framework, which offered efficient implementations for the majority of traditional ML classification models. Then, it was used to create the novel ML model with the goal of outperforming the existing models in the malware analysis. To evaluate the performance of the models the Scikit-learn library was used, which validated the 96.18 percent accuracy attained by the novel model in one third of the time when compared to the best accurate model, which reaches 97.79 percent.

The Microsoft Malware Classification Challenge dataset, one of the largest and newest publicly accessible initiatives datasets, as part of the Big Data Innovators Gathering Cup (BIG 2015), was used to test the models presented in this thesis. The dataset comprises a variety of malware files organized into training and validation datasets classified into families. However, the large discrepancy in the number of samples in the classes was an important factor to take into consideration, due to easily leading to overfitting the ML models. Also, due to the high memory usage and runtime challenges caused by many of the tests requiring long training periods, several improvements were made to the final solution and some tests were executed again.

Concerning the second contribution, Malwizard's solution was developed, which consists of a Microsoft Outlook Add-in developed in Javascript, an API and an ML processing unit server developed in Python. These components were designed considering two functional requirements and nine non-functional. In addition, Malwizard internal composition is explained, as well as the Microsoft Outlook add-in and API service. Two analysis modes are provided, the sample into the analysis can be plain text where it is possible to revert to the original file or homomorphically encrypted which ensures that is not possible to revert to the original file. However, the encrypted mode has only been tested with conventional ML models, which have been shown not to be able to make inferences on encrypted images.

A performance evaluation analysis was provided, giving proof that the Malwizard implementation can be considered on a daily basis malware analysis, actively contributing to cybersecurity. We believe that many other ML models can be applied to malware analysis, each one depending on the intended analysis objective, which can influence the structure and analysis functions in the model. Our focus was to prove that it is possible to have an easy to use solution that makes quick automatic analysis in emails content, but also have the possibility to be applied to other tools through the Malwizard API.

Malwizard evolution and continuity were also this thesis concerns, since Malwizard operates independently of the ML model, malware assessment can be continually enhanced by employing more efficient models as well as models specialized to encrypted images analysis. We hope this solution may contribute to the detection and prevention of new malware forms, securing companies and end-users, and helping to fill some of the open problems in the ML malware analysis.

6.2 Future Work

In this section, we present some opportunities of continuity on Malwizard development and ML encrypted images analysis improvements. The first possible step is to develop new extensions using the Malwizard API in order to extend the integration with more security tools.

Regarding component extensions, the second possible future work is one of the most important, which is the use of models capable of analysing encrypted images, in order to maintain privacy during the malware analysis. The ML computing analysis is a growing market, where more players are entering the field. With this in mind, we identify the need to reduce the encryption time of the suspicious files in analysis, which can be accomplished by improving the homomorphic encryption methodology or by employing more efficient cypher algorithms that preserve the structure of the samples as homomorphic encryption does. To become a helpful solution as well as a reference for industry products.

The third possible future step encompasses the image resampling process, it is necessary to find a method to downsample the malware representation images. When applied, it is capable of identifying noise in the images and disregarding it, thereby keeping the vital information. This is extremely important for Malwizard, in order to downsize the image in analysis to reduce de file size, minimize the encrypted time when applying the privacy mode, and to better identify the presence of malware.

The last future work is identifying a legitimate file dataset to create the legitimate class, so that the ML model used in Malwizard can train beyond the malicious files. Having the class of legitimate files allows the use of functions that normalize the output and attempt to discover the class that has the most similarities with the sample under analysis, such as the softmax. Thus, improving the accuracy in identifying malicious files.

Appendix A

Malwizard public interfaces

The Malwizard public interfaces are explained in this appendix.

A.1 Initialising Malwizard Service

There are three main ways to initialize the Malwizard service. The first one is using the simplest command, without any argument, as follow:

```
python malwizard.py
```

The Malwizard starts and initiates the model training process, once completed, the service is ready to receive files for analysis.

The second is bypassing a “-l” flag with an argument to load a model located in the main “models” directory. This directory stores the progress of Malwizard models for when the server is started or migrated to a new instance. In addition, it allows uploading external models from the community. The command, in this case, is as follows:

```
python malwizard.py -l model_to_load
```

The third is related to the automatic relearning feature "-a", where it is necessary to pass an integer with the total of new samples (totalOfNewSamples) needed to start the new training process and the minimum precision (minAccuracy) for the samples to be considered for further analysis.

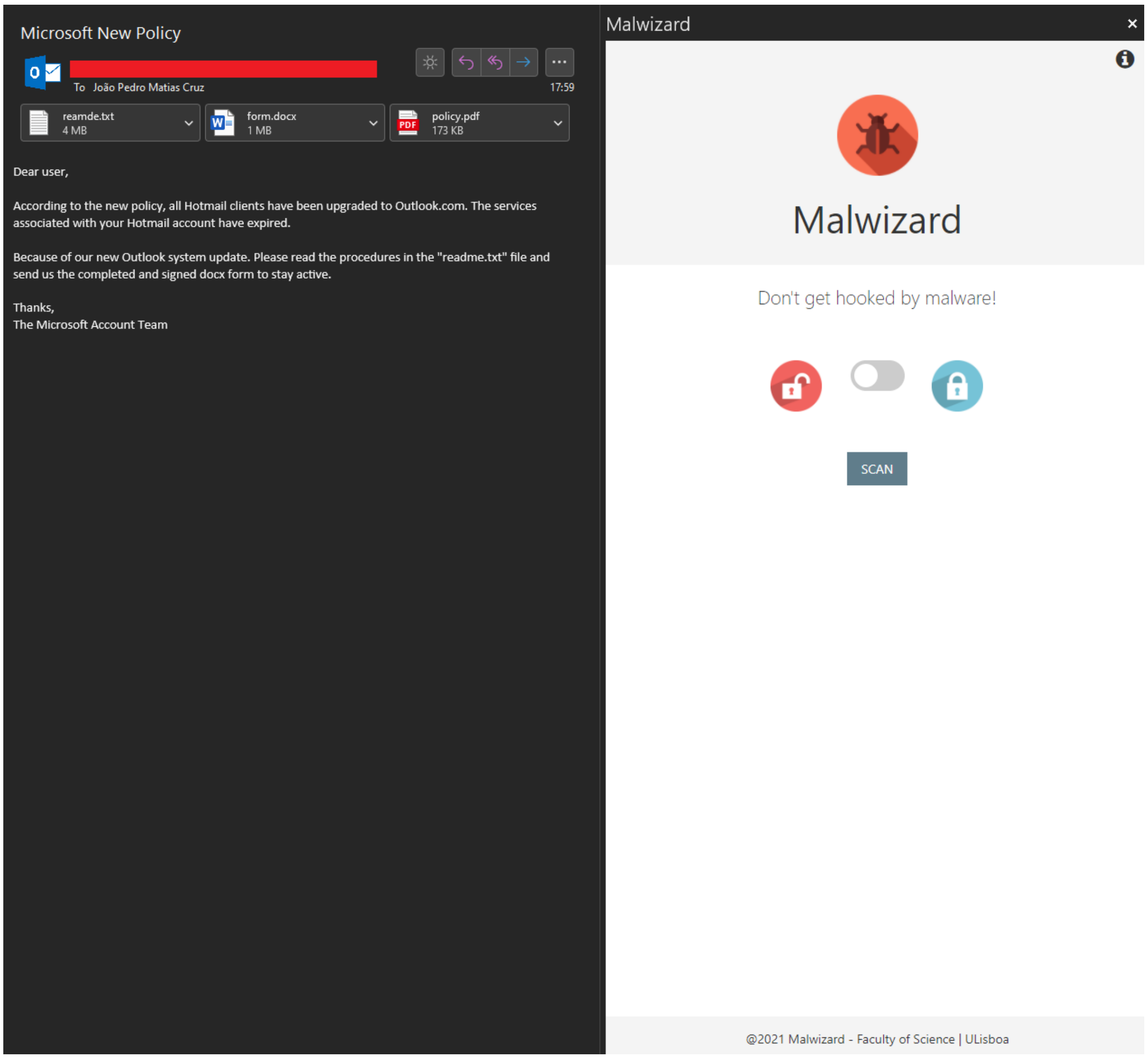
```
python malwizard.py -a totalOfNewSamples minAccuracy
```

After Malwizard initialisation, the Outlook Add-In or middlewares with API clients can start using it for malware analysis.

A.2 Malwizard Outlook Add-In

In the Outlook Add-In interface, there are two padlocks, one unlocked and one locked, which represent respectively the plain text and encrypted modes (Figure 32). Information about the modes and the percentages of the levels that Malwizard uses to identify (Figure 33) are available through the info icon in the upper right corner.

The “SCAN” button, starts the analysis for the files in the attachment, then the user is redirected to the waiting screen until the Malwizard processing unit predicts all the attachments, Figure 34. Finally, the analysis results are displayed in a table (Figure 35), with the name of the attachment, a shield to facilitate the identification of the Malwizard forecast and a preview with the transformation of the file into an image.



Microsoft New Policy

[redacted]

To João Pedro Matias Cruz

17:59

reamde.txt
4 MB

form.docx
1 MB

policy.pdf
173 KB

Dear user,

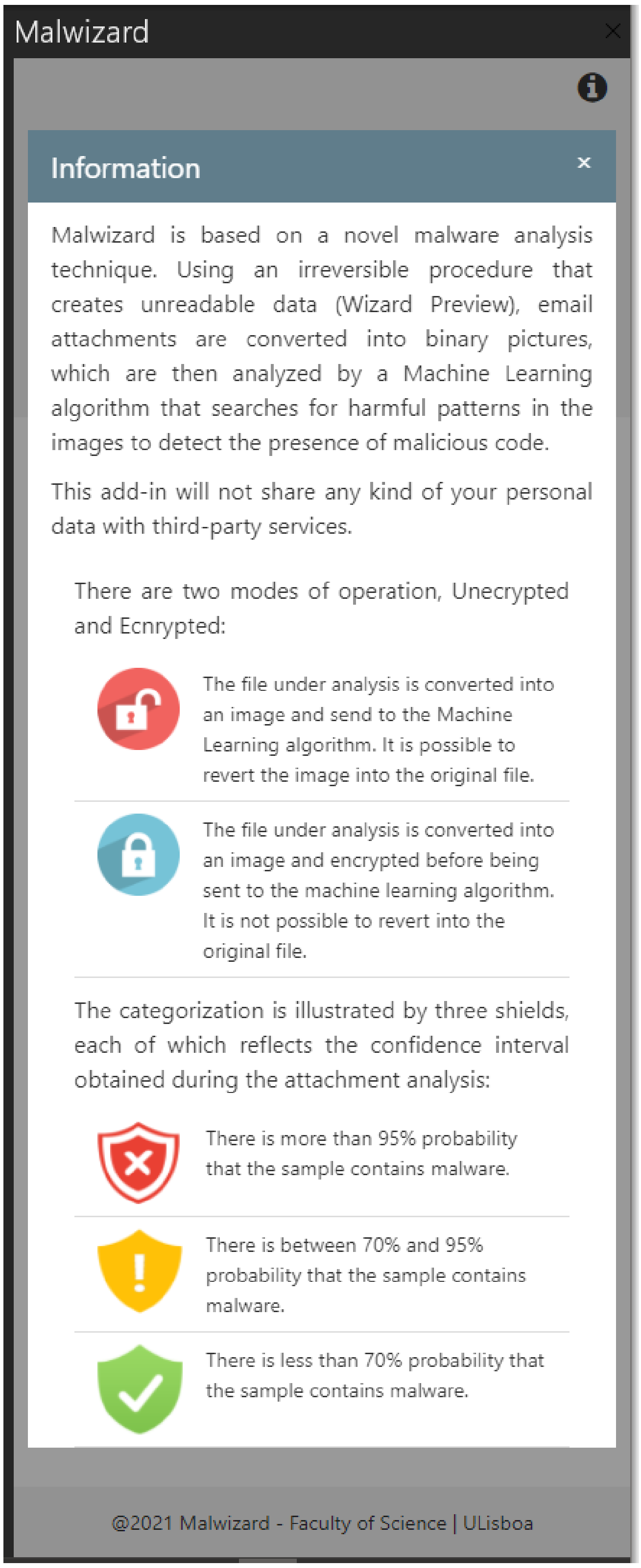
According to the new policy, all Hotmail clients have been upgraded to Outlook.com. The services associated with your Hotmail account have expired.

Because of our new Outlook system update. Please read the procedures in the "readme.txt" file and send us the completed and signed docx form to stay active.

Thanks,
The Microsoft Account Team

Malwizard
Malwizard
Don't get hooked by malware!
SCAN
@2021 Malwizard - Faculty of Science | ULisboa

Figure 32: Malwizard Outlook Add-In.


Malwizard

Information

Malwizard is based on a novel malware analysis technique. Using an irreversible procedure that creates unreadable data (Wizard Preview), email attachments are converted into binary pictures, which are then analyzed by a Machine Learning algorithm that searches for harmful patterns in the images to detect the presence of malicious code.

This add-in will not share any kind of your personal data with third-party services.

There are two modes of operation, Unecrypted and Ecnrypted:

The file under analysis is converted into an image and send to the Machine Learning algorithm. It is possible to revert the image into the original file.

The file under analysis is converted into an image and encrypted before being sent to the machine learning algorithm. It is not possible to revert into the original file.

The categorization is illustrated by three shields, each of which reflects the confidence interval obtained during the attachment analysis:

There is more than 95% probability that the sample contains malware.

There is between 70% and 95% probability that the sample contains malware.

There is less than 70% probability that the sample contains malware.

@2021 Malwizard - Faculty of Science | ULisboa

Figure 33: Outlook Add-In information menu.

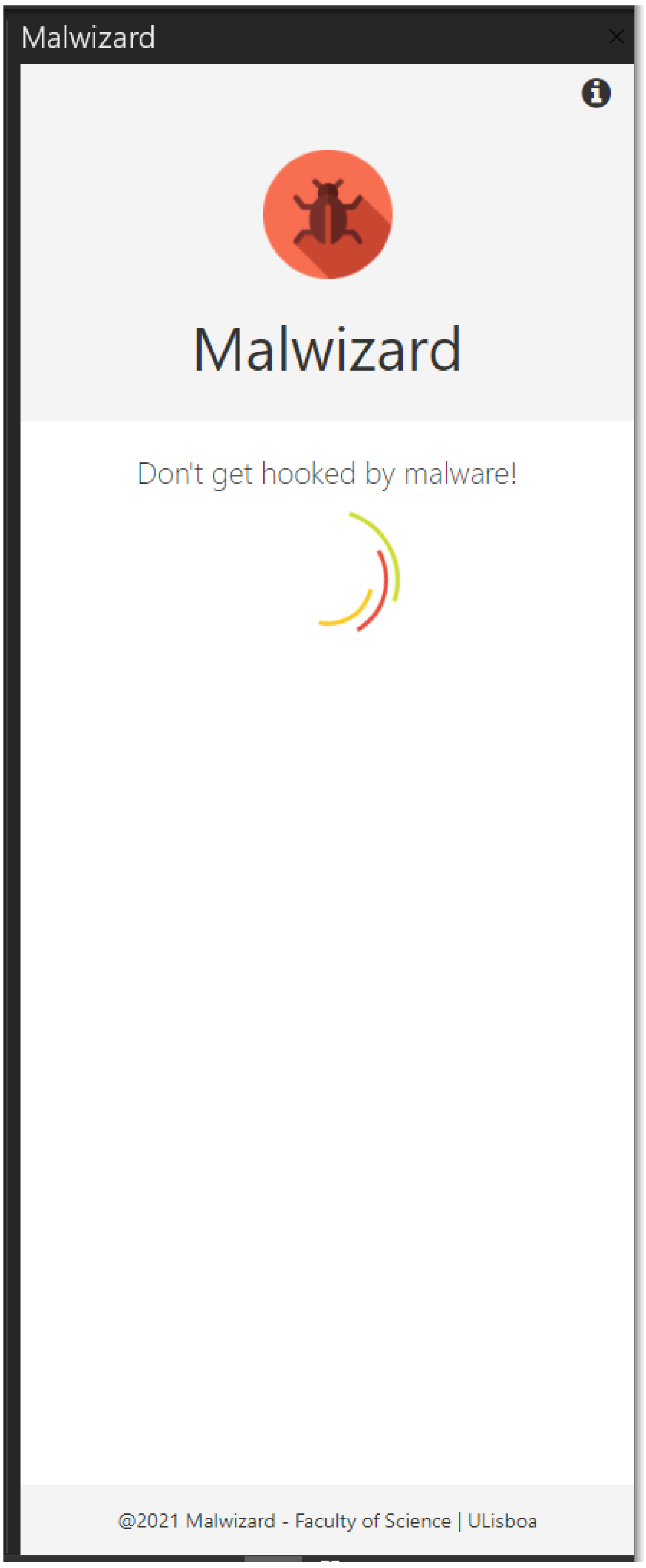
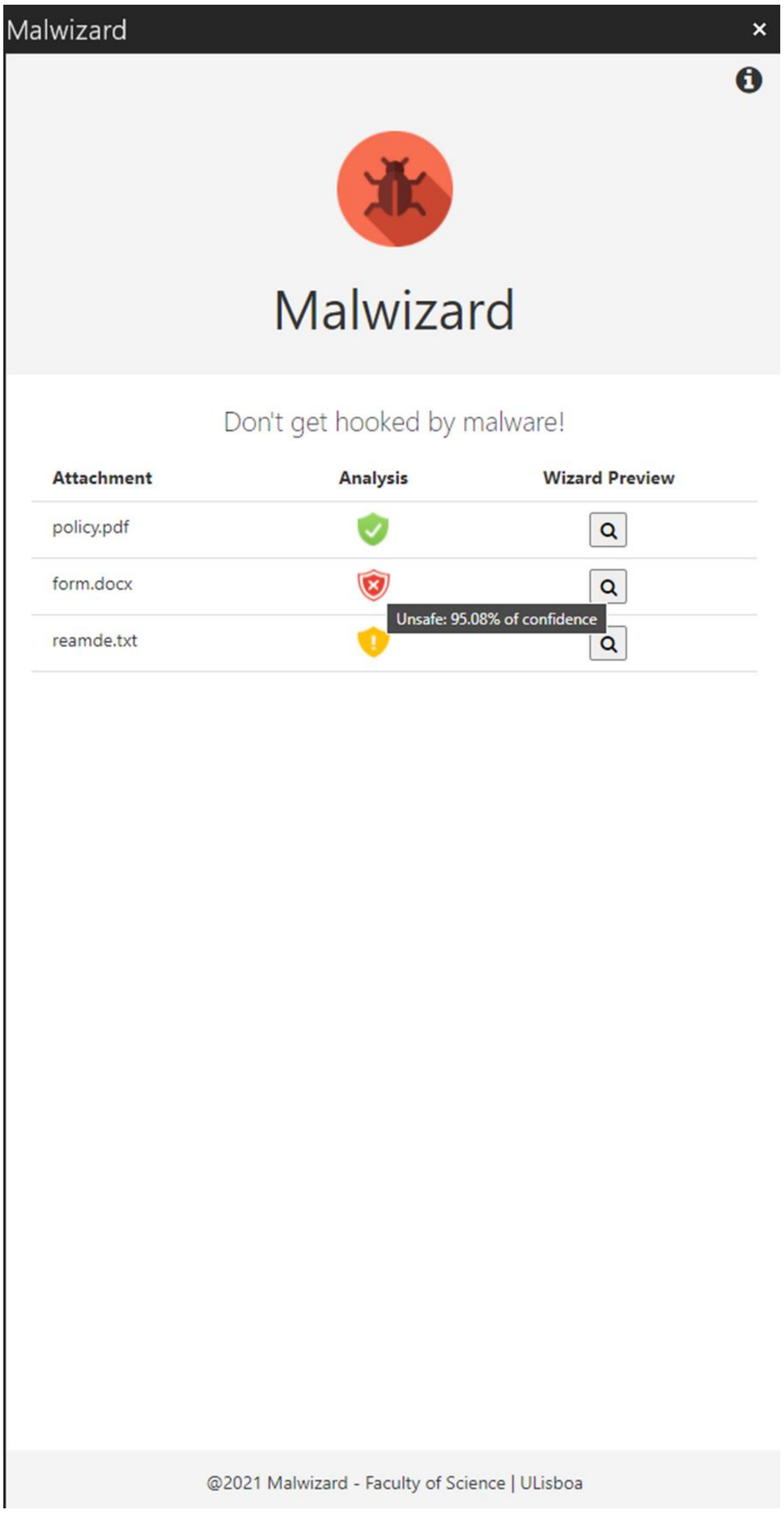

Malwizard
Malwizard
Don't get hooked by malware!
@2021 Malwizard - Faculty of Science | ULisboa

Figure 34: Outlook Add-In waiting screen.



Malwizard

Malwizard

Don't get hooked by malware!

Attachment	Analysis	Wizard Preview
policy.pdf	✓	🔍
form.docx	✗	🔍
reamde.txt	!	🔍

Unsafe: 95.08% of confidence

@2021 Malwizard - Faculty of Science | ULisboa

Figure 35: Outlook Add-In analysis results.

A.3 Malwizard Middleware - API

The Middleware with API integration can be initialized in two ways. In order to integrate with another tool, the tool must be able to make Python calls. The first option is to start in plain text mode, as follows:

```
python malwizard_api.py -a 127.0.0.1 -p 5000 -f malware
```

To enable the encrypted method it is necessary to add the "-e" flags to the command. This informs the Malwizard processing unit that the image under analysis is encrypted and an AI template that supports image encryption analysis must be applied.

```
python malwizard_api.py -a 127.0.0.1 -p 5000 -f malware -e
```

The results returned by the Middleware are in JSON, however, it can be adapted to return in another format if it is necessary.

```
python malwizard_api.py -a 127.0.0.1 -p 5000 -f malware

--- 0.09502148628234863 seconds ---
{"file": "0A32eTdBKayjCWhZqDOQ.bytes", "prediction": 95.69253921508789}
```

The Middleware with API integration is available at https://github.com/escamudo/Malwizard

References

[1] Alanazi SA, Kamruzzaman MM, Alruwaili M, Alshammari N, Alqahtani SA, Karime A. *Measuring and preventing COVID-19 using the SIR model and machine learning in smart health care*. Journal of healthcare engineering. 2020 October. Available: https://www.hindawi.com/journals/jhe/2020/8857346/

[2] Kang Z, Catal C, Tekinerdogan B. *Machine learning applications in production lines: A systematic literature review*. Computers & Industrial Engineering. 2020 November 1;149:106773. Available: https://www.sciencedirect.com/science/article/pii/S036083522030485X

[3] Turkson RE, Baagyere EY, Wenya GE. *A machine learning approach for predicting bank credit worthiness*. In 2016 Third International Conference on Artificial Intelligence and Pattern Recognition (AIPR) 2016 September (pp. 1-7). IEEE. Available: https://ieeexplore.ieee.org/document/7585216

[4] Roy R, George KT. *Detecting insurance claims fraud using machine learning techniques*. In 2017 International Conference on Circuit, Power and Computing Technologies (ICCPCT) 2017 April (pp. 1-6). IEEE. Available: https://ieeexplore.ieee.org/document/8074258

[5] European Union Agency For Cybersecurity. *List of top 15 threats*. In2020 ENISA Threat Landscape. Available: https://www.enisa.europa.eu/topics/threat-risk-management/threats-and-trends/etl-review-folder/etl-2020-enisas-list-of-top-15-threats

[6] European Union Agency For Cybersecurity.*The Year In Review*. In 2020 ENISA Threat Landscape. Available: https://www.enisa.europa.eu/publications/year-in-review/at_download/fullReport

[7] Kaspersky Security Bulletin 2019. (2018) [Online]. Available: https://securelist.com/kaspersky-security-bulletin-2019-statistics/95475/ (cited on page 1)

[8] Anil Thomas Nikos Karampatziakis, Jack Stokes and Mady Marinescu. *Using file relationships in malware classification*. Detection of Intrusions and Malware, and Vulnerability Assessment, 7591:1–20, 2013. Available: https://www.microsoft.com/en-us/research/wp-content/uploads/2012/07/UsingFileRelationshipsinMalwareClassification.pdf

[9] G. E. Dahl GE, Stokes JW, Deng L, Yu D. Large-scale malware classification using random projections and neural networks. In2013 IEEE International Conference on Acoustics, Speech and Signal Processing 2013 May (pp. 3422-3426). IEEE. Available: https://ieeexplore.ieee.org/abstract/document/6638293

[10] Ravi C, Manoharan R. Malware detection using windows api sequence and machine learning. International Journal of Computer Applications. 2012 April; 43(17):12-6. Available: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.736.9347&rep=rep1&type=pdf

[11] Create blocked sender lists in EOP (2021) [Online]. Available: https://docs.microsoft.com/en-us/microsoft-365/security/office-365-security/create-block-sender-lists-in-office-365?view=o365-worldwide

[12] Set up SPF to help prevent spoofing (2019) [Online]. Available: https://docs.microsoft.com/en-us/microsoft-365/security/office-365-security/set-up-spf-in-office-365-to-help-prevent-spoofing?view=o365-worldwide

[13] Use DKIM to validate outbound email sent from your custom domain (2021) [Online]. Available: https://docs.microsoft.com/en-us/microsoft-365/security/office-365-security/use-dkim-to-validate-outbound-email?view=o365-worldwide

[14] Use DMARC to validate email (2021) [Online]. Available: https://docs.microsoft.com/en-us/microsoft-365/security/office-365-security/use-dmarc-to-validate-email?view=o365-worldwide

[15] Nataraj L, Karthikeyan S, Jacob G, Manjunath BS. *Malware images: visualization and automatic classification*. InProceedings of the 8th international symposium on visualization for cyber security 2011 July (pp. 1-7). Available: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.228.1233&rep=rep1&type=pdf

[16] Gibert D, Mateu C, Planes J, Vicens R. Using convolutional neural networks for classification of malware represented as images. Journal of Computer Virology and Hacking Techniques. 2019 March;15(1):15-28. Available: https://link.springer.com/article/10.1007/s11416-018-0323-0

[17] TheHive Project. TheHive Project: Security Incident Response For The Masses [Online]. Available: https://thehive-project.org/

[18] Venable JR, Pries-Heje J, Baskerville RL. Choosing a design science research methodology.; 2017. Available: https://aisel.aisnet.org/acis2017/112/

[19] Yegnanarayana B. Artificial neural networks. PHI Learning Pvt. Ltd.; 2009 January. Available: http://cdn.iiit.ac.in/cdn/speech.iiit.ac.in/svlpubs/book/Yegna1999.pdf

[20] Nwankpa C, Ijomah W, Gachagan A, Marshall S. Activation functions: Comparison of trends in practice and research for deep learning. 2018 November. arXiv:1811.03378 Available: https://arxiv.org/abs/1811.03378

[21] Dauphin YN, De Vries H, Bengio Y. Equilibrated adaptive learning rates for non-convex optimization. 2015 April. arXiv:1502.04390. Available: https://arxiv.org/abs/1502.04390

[22] Sweta Shaw. RMSprop : A Better Way to Optimize Your Model. 2019. Available: https://medium.com/@shwetaka1988/rmsprop-a-better-way-to-optimize-your-model-bc4eaca33090

[23] Kingma DP, Ba J. Adam: A method for stochastic optimization. 2014 December. arXiv:1412.6980. Available: https://arxiv.org/abs/1412.6980

[24] Convolutional neural network [Online]. Available: https://en.wikipedia.org/wiki/Convolutional_neural_network (cited on page 1)

[25] Understanding of Convolutional Neural Network (CNN) — Deep Learning (2018) [Online]. Available: https://medium.com/@RaghavPrabhu/understanding-of-convolutional-neural-network-cnn-deep-learning-99760835f148 (cited on page 1)

[26] CIFAR-10. CIFAR-10 Dataset [Online]. https://www.cs.toronto.edu/~kriz/cifar.html

[27] Neural Network and Deep Learning - From Theory to Realization: Building L-Layer Neural Network from Zero (2019) [Online]. Available: https://programmer.help/blogs/building-l-layer-neural-network-from-zero.html

[28] Multi-Class Neural Networks: Softmax. 2020 March. Available: https://developers.google.com/machine-learning/crash-course/multi-class-neural-networks/softmax

[29] Keras. Keras Applications [Online]. Available: https://keras.io/api/applications/

[30] ImageNet [Online]. Available: https://image-net.org/

[31] Russakovsky O, Deng J, Su H, Krause J, Satheesh S, Ma S, Huang Z, Karpathy A, Khosla A, Bernstein M, Berg AC. Imagenet large scale visual recognition challenge. International journal of computer vision. 2015 December;115(3):211-52. Available: https://link.springer.com/article/10.1007/s11263-015-0816-y

[32] He K, Zhang X, Ren S, Sun J. Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition 2016 (pp. 770-778). Available: https://openaccess.thecvf.com/content_cvpr_2016/html/He_Deep_Residual_Learning_CVPR_2016_paper.html

[33] Quora.com. What is the deep neural network known as "ResNet-50"? - Quora. 2014. Available: https://www.quora.com/What-is-the-deep-neural-network-known-as-%E2%80%9CResNet-50%E2%80%9D

[34] Neurohive.io. VGG16 - Convolutional Network for Classification and Detection. 2018. Available: https://neurohive.io/en/popular-networks/vgg16

[35] Richmond Alake. Deep Learning: GoogLeNet Explained. 2020 December. Available: https://towardsdatascience.com/deep-learning-googlenet-explained-de8861c82765#:~:text=GoogLeNet%20is%20a%2022%2Dlayer,developed%20by%20researchers%20at%20Google.

[36] Advanced Guide to Inception v3 on Cloud TPU. 2021 November. Available: https://cloud.google.com/tpu/docs/inception-v3-advanced

[37] Szegedy C, Vanhoucke V, Ioffe S, Shlens J, Wojna Z. Rethinking the inception architecture for computer vision. InProceedings of the IEEE conference on computer vision and pattern recognition 2016 (pp. 2818-2826). Available: https://www.cv-foundation.org/openaccess/content_cvpr_2016/html/Szegedy_Rethinking_the_Inception_CVPR_2016_paper.html

[38] Oliva A, Torralba A. Modeling the shape of the scene: A holistic representation of the spatial envelope. International journal of computer vision. 2001 May;42(3):145-75. Available: https://link.springer.com/article/10.1023/A:1011139631724

[39] Liu B, Ding M, Shaham S, Rahayu W, Farokhi F, Lin Z. When machine learning meets privacy: A survey and outlook. ACM Computing Surveys (CSUR).2021 March 5;54(2):1-36. arXiv: 2011.11819. Available: https://arxiv.org/abs/2011.11819

[40] Gilad-Bachrach R, Dowlin N, Laine K, Lauter K, Naehrig M, Wernsing J. Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In International conference on machine learning 2016 June (pp. 201-210). PMLR. Available: http://proceedings.mlr.press/v48/gilad-bachrach16.html

[41] Hesamifard E, Takabi H, Ghasemi M. Cryptodl: Deep neural networks over encrypted data. 2017 November. arXiv:1711.05189. Available: https://arxiv.org/abs/1711.05189

[42] Bos, Joppe W, Lauter, Kristin, Loftus, Jake, and Naehrig, Michael. Improved security for a ring-based fully homomorphic encryption scheme. In Cryptography and Coding. 2013. Available: https://www.microsoft.com/en-us/research/publication/improved-security-for-a-ring-based-fully-homomorphic-encryption-scheme-2/

[43] Yann LeCun, Corinna Cortes, Christopher J.C. Burges: The MNIST database [Online]. Available: http://yann.lecun.com/exdb/mnist/

[44] Kechit Goyal. Top 10 Deep Learning Frameworks in 2021 You Can't Ignore. 2021 January. Available: https://www.upgrad.com/blog/top-deep-learning-frameworks/

[45] Classification on imbalanced data [Online]. 2021 November. Available: https://www.tensorflow.org/tutorials/structured_data/imbalanced_data

[46] Ronen R, Radu M, Feuerstein C, Yom-Tov E, Ahmadi M. Microsoft malware classification challenge. 2018 February. arXiv:1802.10135. Available: https://arxiv.org/abs/1802.10135

[47] Dataset: Microsoft Malware Classification Challenge (BIG 2015) [Online]. Available: https://www.kaggle.com/c/malware-classification/overview

[48] VirusShare.com - Because Sharing is Caring [Online]. Available: https://virusshare.com/

[49] Scikit-learn. Scikit-learn: Machine Learning in Python. Available: https://scikit-learn.org/stable/

[50] Ajitesh Kumar. Data Analytics: Mean Squared Error or R-Squared – Which one to use?. September 2020. Available: https://vitalflux.com/mean-square-error-r-squared-which-one-to-use/

[51] Joseph Magiy. Pearson Coefficient of Correlation Explained. 2019 May. Available: https://towardsdatascience.com/pearson-coefficient-of-correlation-explained-369991d93404

[52] Christopher Riggio. What's the deal with Accuracy, Precision, Recall and F1?. 2019 November. Available: https://towardsdatascience.com/whats-the-deal-with-accuracy-precision-recall-and-f1-f5d8b4db1021

[53] Grandini M, Bagli E, Visani G. Metrics for multi-class classification: an overview. 2020 August. arXiv:2008.05756. Available: https://arxiv.org/abs/2008.05756

[54] Chadrick. CV2 Resize Interpolation Methods. 2018 November. Available: https://chadrick-kwag.net/cv2-resize-interpolation-methods/

[55] Jason Brownlee. Machine Learning Mastery: A Gentle Introduction to k-fold Cross-Validation. 2020 August. Available: https://machinelearningmastery.com/k-fold-cross-validation/

[56] Tensorflow Addons. Github: F1 metric - micro/macro/weighted [Online]. Available: https://github.com/tensorflow/addons/pull/284

[57] Daniel Gibert. Github: Using Convolutional Neural Networks for Classification of Malware represented as Images [Online]. Available: https://github.com/danielgibert/mlw_classification_cnn_img

[58] Sarang Narkhede. Understanding AUC - ROC Curve. 2018 June. Available: https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5

[59] Mark A. Will, Ryan K.L. Ko. The Cloud Security Ecosystem, 2015 (pp. 109-111)

[60] Damien Giry. BlueKrypt: Cryptographic Keys Length Recommendation. 2020 May. Available: https://www.keylength.com/en/compare/

[61] Mozilla: MDN Web Docs [Online]. Available: https://developer.mozilla.org/en-US/docs/Glossary/Base64

[62] Outlook: Send large files with Outlook [Online]. Available: https://support.microsoft.com/en-us/office/send-large-files-with-outlook-8c698842-b462-4a4c-8d53-5c5dd04f77ef

[63] Open Source Computer Vision (OpenCV) [Online]. Available: https://docs.opencv.org/4.x/