MALWARE CLASSIFICATION SYSTEM USING WEIGHTLESS NEURAL NETWORKS

Luiz Claudio Sampaio Ramos

Dissertação de Mestrado apresentada ao Programa de Pós-graduação em Engenharia de Sistemas e Computação, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Mestre em Engenharia de Sistemas e Computação.

Orientadores: Felipe Maia Galvão França
Priscila Machado Vieira Lima

Rio de Janeiro
Agosto de 2021

MALWARE CLASSIFICATION SYSTEM USING WEIGHTLESS NEURAL NETWORKS

Luiz Claudio Sampaio Ramos

DISSERTAÇÃO SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Orientadores: Felipe Maia Galvão França
Priscila Machado Vieira Lima

Aprovada por: Prof. Felipe Maia Galvão França
Profa. Priscila Machado Vieira Lima
Prof. Diego Leonel Cadette Dutra
Prof. Mauricio Breternitz Jr.

RIO DE JANEIRO, RJ – BRASIL
AGOSTO DE 2021

Ramos, Luiz Claudio Sampaio

Malware classification system using weightless neural networks/Luiz Claudio Sampaio Ramos. – Rio de Janeiro: UFRJ/COPPE, 2021.

XII, 45 p.: il.; 29, 7cm.

Orientadores: Felipe Maia Galvão França
Priscila Machado Vieira Lima

Dissertação (mestrado) – UFRJ/COPPE/Programa de Engenharia de Sistemas e Computação, 2021.

Referências Bibliográficas: p. 39 – 43.

1. Malware Classification. 2. Weightless Neural Networks. 3. Decision Trees. I. França, Felipe Maia Galvão *et al.* II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia de Sistemas e Computação. III. Título.

Ao Nero, meu irmão pequeno que
me ensinou o que os livros não
podem ensinar.

Agradecimentos

Agradeço em primeiro lugar aos meus pais, Sonia e Mauri, que não se contentaram em me dar toda a estrutura necessária para a vida, mas também o fizeram com muita dedicação, amor e compreensão.

Aos meus irmãos, Stefano e Patricia, por serem exemplos a serem seguidos e por sempre estarem ao meu lado.

Aos meus orientadores, Felipe e Priscila, pela paciência e compreensão sempre que precisei, nunca medindo esforços para me ajudar em qualquer situação.

Ao coordenador Daniel Ratton e ao corpo administrativo do PESC, pela humanidade no tratamento com as pessoas.

À Camila, minha melhor amiga e companheira, que nunca me abandonou nos momentos mais difíceis da minha vida.

Aos meus amigos mais próximos: Santanelli, Braun e Nicolas, que me ajudaram a superar as adversidades com muito bom humor.

Aos amigos Bacelar, Caio e Escorcio que, apesar da distância, pude contar nos momentos complicados.

Aos colegas de pós-graduação Leopoldo e Brunno, pela ajuda e paciência com as questões de programação desse trabalho.

Ao professor Amarildo, por ter me apresentado a oportunidade de estudar no PESC.

À empresa LACE Engenharia e ao Luiz Renault, pela compreensão nos momentos difíceis da pós-graduação no tocante às relações profissionais.

Ao PESC, por ter me proporcionado participar de um congresso durante o mestrado.

Resumo da Dissertação apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

SISTEMA DE CLASSIFICAÇÃO DE MALWARES UTILIZANDO REDES NEURAIS SEM PESO

Luiz Claudio Sampaio Ramos

Agosto/2021

Orientadores: Felipe Maia Galvão França
Priscila Machado Vieira Lima

Programa: Engenharia de Sistemas e Computação

Apresenta-se, nesta dissertação, o sistema de classificação de programas maliciosos MalWiSARD, um sistema baseado em redes neurais sem peso. Com a crescente preocupação com segurança e integridade dos dados em sistemas de computação, faz-se necessário uma detecção rápida e eficaz para evitar o comprometimento de sistemas, o que pode ter consequências drásticas. O trabalho mostra MalWiSARD e suas variantes aplicadas em um banco de dados de 26 tipos de programas, incluindo 25 tipos maliciosos. A variante em árvore da MalWiSARD utilizando a análise de *ensemble* para classificação de vírus alcançou uma acurácia de 98,5693%, com tempos de treinamento em torno de 11 minutos para treinar 10582 imagens geradas diretamente a partir de executáveis de programa, obtendo um resultado melhor que o estado-da-arte do banco de dados tanto na acurácia como nos tempos de classificação.

Abstract of Dissertation presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

MALWARE CLASSIFICATION SYSTEM USING WEIGHTLESS NEURAL NETWORKS

Luiz Claudio Sampaio Ramos

August/2021

Advisors: Felipe Maia Galvão França
Priscila Machado Vieira Lima

Department: Systems Engineering and Computer Science

This dissertation presents the MalWiSARD classification system for classification of malicious computer programs (malware), a system based on weightless neural networks. With the growing concern about security and data integrity in computing systems, rapid and effective detection of malware is needed to prevent systems from being compromised, which can have drastic consequences. The work shows MalWiSARD and its variants applied to a database of 26 types of programs, including 25 malicious types. MalWiSARD tree variant using ensemble analysis for virus classification achieved an accuracy of 98.5693%, with training time around 11 minutes to train 10582 images generated directly from binary program executables, obtaining a better result than the state-of-the-art for this database related to both accuracy and classification times.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

As computer systems are more integrated into the everyday life of citizens and companies, including important areas like Internet of Things and Industry 4.0, the concern with privacy, security, and integrity of data is constantly increasing. Computer systems are used to control, for instance, power plants, nuclear plants and industries. A malware, or a malicious software, is a program whose main goal is to access and modify sensible information in an unauthorized way, commonly to achieve financial profit. This concern has lead to great research in the field of malware detection and classification, not only in the academic field but also inside companies which are afraid of exposing compromising information which could lead the firm to bankruptcy, for instance. One example is the use of Ransomware, a type of malware that steals sensitive information and asks for money in order to not expose it.

1.2 Goals and Contribution

This work aims to present a malware classification system based on weightless neural networks, the MalWiSARD. As detailed in this work, weightless neural networks present relatively low training and classification times and are based in random access memories; therefore, the simplicity of these systems can be employed to scale to big computer systems and cyber-physical systems.

The main contribution is the MalWiSARD, a malware classification system which achieves more than 98% accuracy classifying among 26 classes of software, including 25 malware. This was done in more than 10 times faster than the last work in the same dataset [1], and with higher accuracy. Therefore, this work shows the feasibility of employing weightless neural networks in malware classification, which can lead to fast response times in real time systems and also protect large computer systems

with small memory footprint components, such as for applications in Internet of Things.

1.3 Dissertation Organization

This work is organized as follows. Initially, Chapter 2 presents the definition of malware and the most common types of malware, and also further explores the importance of detecting and classifying malware. Furthermore, it presents the main classification and detection techniques and their categories, showing related works which employed these techniques.

Subsequentialy, Chapter 3 presents MalWiSARD theoretical background, explaining the main techniques employed in the method; it discusses the weightless neural network model and its training and testing phase and also variations which can be used to increase accuracy. Finally, it discusses decision trees, which were also employed in the MalWiSARD.

Afterwards, Chapter 4 illustrates the MalWiSARD method, presenting all the phases involved from the program executable binary to the classification, clarifying all the possible variants of the method. Then, Chapter 5 presents the experiments done in order to validate the method and discusses the results.

Finally, Chapter 6 concludes the work summarizing the results obtained and proposing future work in the malware classification field using weightless neural networks. In Appendix A, it is presented the papers published relating to this work.

Chapter 2

Classification and Detection of Malware

2.1 Definition

The term *malware*, short for **mal**icious soft**ware**, is a generic term which can be describe a program whose objetive is malevolent [2], that is, code added, changed or removed from a software system in order to intentionally cause harm or subvert the intended function of the system [3].

2.1.1 Types of malware

Based on the different purposes and proliferation ways, malware can be broadly classified into the following categories.

Adware

Adware or advertising-supported software automatically plays, displays, or downloads advertisements to a computer after malicious software is installed or application is used. This piece of code is generally embedded into free software [4]. The objective of adware is to gain financial profit for their author. Adware are not harmful by nature, but they can be in the form of a pop-up window which can interrupt users thinking. Some adware may come with integrated spyware such as key loggers and other privacy-invasive software [5] [6].

Trojan

Trojan horses emulate behavior of an authentic program such as login shell and hijacks user password to gain control of system remotely. Other malicious activities may include monitoring of system, damages system resources such as files or disk

data, denies specific services [4]. The embedded malware could also be a time bomb [7].

Virus

A computer virus is code that replicates by inserting itself into other programs. A program that a virus has inserted itself into is infected, and is referred to as the virus's host. An important caveat is that viruses, in order to function, require their hosts, that is, a virus needs an existing host program in order to cause harm. For example, in order to get into a computer system, a virus may attach itself to some software utility (e.g. a word processing application). Launching the word processing application could then activate the virus that may, for example, duplicate itself and disable malware detectors enabled on the computer system [6].

Worm

Worms are self replicating programs. It uses network to send copies of itself to other systems invisibly without user authorization. Worms may cause harm to network by consuming the bandwidth. Unlike virus the worms do not need the support of any file. It might delete files, encrypt files in as crypto viral extortion attack or send junk email [4].

Backdoor

Backdoors, also called trap doors, are malcode written into an applications or operating systems with the intention of granting programmers access to the system without requiring them to go through ordinary methods of authentications. They're written by experts or specialized developers for friendly usage. The security problem with trapdoors is the full access, getting in without authentication, because these programs can be used remotely by enemies to make attacks [5].

2.1.2 Background

As presented in a survey conducted by the FBI (Federal Bureau of Investigation) [8], this section discusses some major issues which can be caused by malware, allowing analysis of important computer security trends. The key findings from the survey participants are presented. In a shift from previous years, both virus attacks and denial of service outpaced the former top cost, the theft of proprietary information. Virus costs jumped to $55 million [8]. Thus, companies began to invest in cybersecurity adopting actions such as increasing the financial investiment in security and promoting security awareness training for the colaborators. In this

manner, over 80 percent of the organizations conduct security audits and do not outsource computer security activies, such that the company has full control of the integrity and security of data [8]. Furthermore, it can be seen that the percentage of organizations reporting computer intrusions to law enforcement is on the decline and most organizations conduct some form of economic evaluation of their security expenditures, which shows that with more than 55 percent returning on investment. Thereby, unauthorized use of computer systems is on the decline, as is the reported dollar amount of annual financial losses resulting from security breaches [8].

Given the dimension of the damage malware can cause, several works have been conducted in the last years in order to detect and classify malware in computer systems. Besides, thousands of new malware appear very quickly, as the creation of new malware can be done from existing malware with little modifications [9].

2.2 Classification and Detection Techniques

As the number of new malware samples has been increasing, anti-malware vendors are confronted with millions of potential malware samples per year. In this manner, there is an urgent need to develop intelligent methods for effective and efficient malware detection from the daily sample collection. A general structure of the progress in malware detection in the last years is depicted in Figure 2.1 [10].

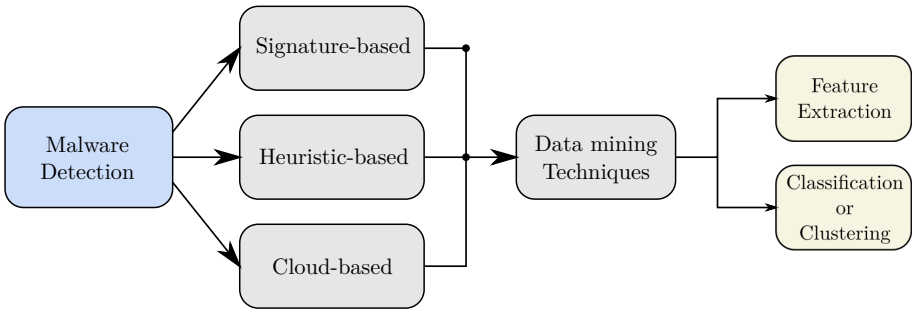


Figure 2.1: Progress in malware detection techniques.

2.2.1 Signature-Based

A *signature-based method* uses what is called a signature, which is a short sequence of bytes unique to each known malware. The goal is that this signature can help identifying newly encountered files with a small error rate [11]. As new malware can be created from existing ones, they will have a known signature, and can be detected, but there are several techniques to bypass these methods, such as encryption, obfuscation and polymorphism. The signature bases are manually generated, update and disseminated, such that the process is very time consuming. Furthermore, anti-malware tools will be less responsive to new threats [12].

2.2.2 Heuristic-Based

An *heuristic-based method* is based on rules determined by experts to discriminate malware samples and benign files. These rules should be generic enough to be consistent with variants of the same malware threat, but not falsely matched on bening files [13]. As the analysis done by experts can be erroneous is very consuming, this makes the speed of malware creation faster than the manual pattern development and the necessity of an automated process of analysis [10].

2.2.3 Cloud-Based

In order to produce a quicker response and development of rules to identify malware, anti-malware vendors have used *cloud-based detection*, which consists of having a client-server architecture such that files that cannot be classified offline (it is unknown by the offline system) will have its features sent to a server in which they can be classified. The veredict will spread to all client users in the system, such that all users will have up-to-date security solutions [11].

2.2.4 Data Mining Techniques

With great interest and research in machine learning, the last techniques became to rely on data mining techniques. The idea consists of classifying previously unseen malware samples, identifying malware type and family. In general, these techniques consist of two steps: feature extraction and classification/clustering [10]. In the first step, there are several techniques that try to extract features that can be used to predict the software behavior, such that API calls, opcodes and binary sequences. This can be done statically, which extracts patterns from the code executable; or dinamicaly, which extracts patterns from the program in runtime. In the second step, in the case of labelled data, supervised learning can be employed, using neural networks, decision trees or support vector machines, for instance; if there is no label or previous information about the software, clustering techniques are used, in order to group malware samples that share similar features.

2.3 Related Works

This section explores previous research employing the discussed techniques.

ELLIS *et al.* [14] used characteristic patterns of worm behavior in network traffic and studied how the network application architecture can influence how a work can impact a network. ILGUN *et al.* [15] created a technique called state transition analysis which models penetrations as a series of state changes that lead from an

secure state to a target compromised state. Both are examples of dynamic signature-based methods.

Several examples of static signature-based methods have been developed. In CHRISTODORESCU e JHA [16], it is shown a tool called SAFE, which is resilient to common obfuscation transformations. In order to create platform-independent tools, KUMAR e SPAFFORD [17] implemented a virus detection tool in the C++ language, which extract features in viruses and use them to detect malware. Finally, SULAIMAN *et al.* [18] presents a robust assembly language detection technique for obfuscation detection.

As obfuscation became a popular technique to create new malware, TREADWELL e ZHOU [19] presents a heuristic detection approach performing a series of static checks on binary files portable executable structure for common traces of obfuscation. As of using data mining and machine learning techniques, ELOVICI *et al.* [20] presents the ensemble of neural networks, decision trees and bayesian networks based on byte sequences and portable executable headers to outperform each individual classifier. WANG *et al.* [21] uses a hybrid approach to feature extraction (that is, both static and dynamic features) to classify spyware using a support vector machine. Finally, ANDERSON *et al.* [22] presents multiple kernel learning employing Markov chains from dynamically collected instruction traces.

NATARAJ [23] shows that the program's binary files can be treated as images by direct conversion and analyzed. In this manner, NATARAJ *et al.* [24] presented SARVAM, a search system for malware based on the converted image and on the Nearest-Neighbors technique. BOZKIR *et al.* [1] employed various contemporary convolutional neural networks, such as Resnet, Inception, DenseNet, VCG and AlexNet, that have proven success in image classification, to show that classification via the use of computer vision and machine learning methods over byte-level images extracted from malware is an effective static solution. Moreover, GARCIA e MUGA II [25] employed a Random Forest to classify the images. Finally, besides image processing techniques, FARROKHMANESH e HAMZEH [26] uses audio processing techniques and pattern findind methods in music to classify malware.

Besides, techniques which follow different terminology have been focus of research, such as: anomaly-based in GARCÍA-TEODORO *et al.* [27], which uses statistical analysis in order to detect unknown malware; specification-based in CHAUGULE *et al.* [28], which utilizes keypad or touch screen information to differentiate between malware and human activity; rule-based in BLOUNT *et al.* [29], which employs a combination of a rule system and a evolutionary algorithm-based reinforcement learning; and network-bases in AHMED *et al.* [30], which uses a sniffer in each network segment to prevent external attack to a network via a malware.

Chapter 3

Classification Techniques

This chapter explores the classification techniques employed in this work. The main topics are Weightless Neural Networks, which is the core technique used in the proposed method, and Decision Trees, which is compared with the last in some cases.

3.1 Weightless Neural Networks

Originally known as *n-tuple classifier* and developed by BLEDSOE e BROWNING [31], weightless neural networks are also inspired by the human nervous system, but provides a different approach when compared to traditional neural networks. In this type of network, the emulation of the topology of the connections between dendrites and axons is prioritized, that is, the dendritic tree [32]. The information learned is stored in RAM memories, which work like artificial neurons.

3.1.1 RAM Discriminators

The *n-tuple classificators* consist of n-tuples used to address initially flushed RAM memories, that is, memories filled with zeroes. These classificatiors were first used for manuscript character classification, as shown in Figure 3.1. The classificator randomly chooses groups of n binary positions in the image and use them to address what is called *Address Groups*, each tuple addressing one group. The value of the binary positions of the tuple determines the bit which to address in the memory, and this position is set to the value 1. It is important to note that each possible label of incoming data has memory zones which represents them in each address groups; in other words, in the example, every character has a memory zone which corresponds to it and, therefore, the system can remember which character activated each positions in the address groups.

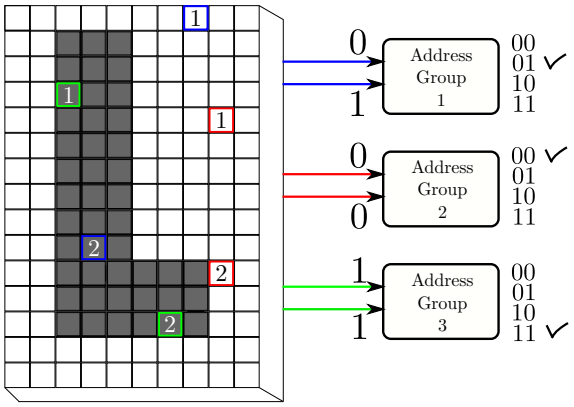


Figure 3.1: Discriminator example using 2-tuples.

After training with some labeled examples, in order to classify a new input, the same mapping used in the training session is used, again, to address the memories. Using an adder, the system can determine the number of memory cells which returned the value 1 for each label, and the sum is used as a measurement of sensibility against data used to train. The memory zone corresponding to a certain character which has the larger sum is considered as the classified label for that input, as depicted in Figure 3.2.

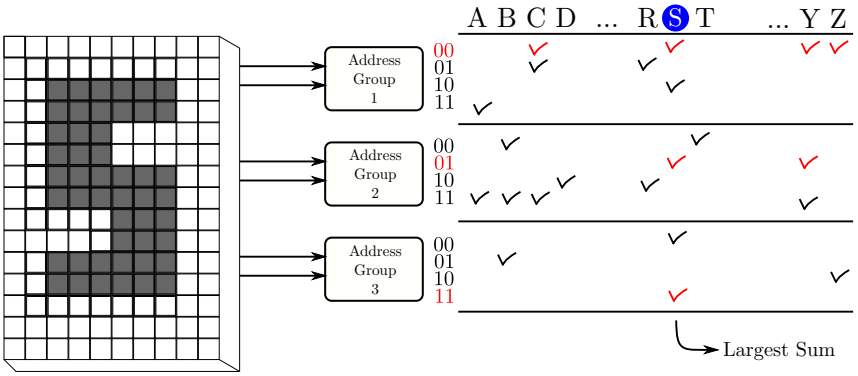


Figure 3.2: Example of classification session; the block which corresponds to each character is added.

As each tuple represents a limited part of the input data, it is not capable of generalizing the information trained. For instance, if a 2-tuple is used, there are four possible addressable positions for each character, so that there are 16 possible outcomes for each block in each address group. In this manner, it is possible that the memory (and the whole system in general) will saturate quickly and will not be able to recognize similar patterns in the input data.

3.1.2 WiSARD

In order to surpass the saturation problem, the Wilkie, Stoneham and Aleksander's Recognition Device, also known as WiSARD [33], expands the concept explored in

Section 3.1.1. The system consists of one discriminator for each data label and, therefore, it is able to capture the patterns of each class in a separate discriminator. In the example of character recognition, there would be 26 discriminators, one for each letter of the latin alphabet. In this case, during the training session, only the specific discriminator which represents the class will be trained, whereas in the classification session each discriminator will output a different value for the input data. In order to classify, the system gives the highest sum among the discriminators, as shown in Figure 3.3; besides, the confidence of the result can be defined as the relative difference between the response of the predicted discriminator and the second highest valued discriminator.

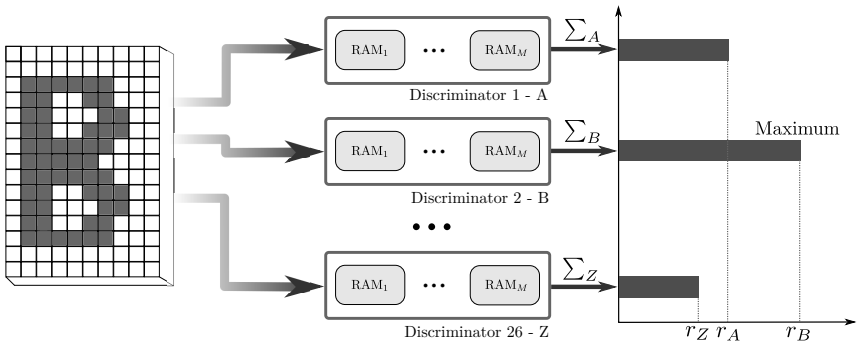


Figure 3.3: Example of WiSARD classification session; the discriminator related to the letter *b* has the maximum activation rate.

Besides, a system can be built to have more discriminators than classes, in order to capture different patterns of the same class. As described, as the WiSARD has at least one discriminator for each class, it has higher tolerance to noise in the data and can achieve generalization, because the system no longer has the saturation problem as the n-tuple classificator. One problem that can still arise is the possibility of a tie; in other words, the confidence of the result can be zero, meaning that two classes have the same output response. In this case, the WiSARD chooses randomly a class between the two, and a criteria is needed to overcome this issue.

3.1.3 Bleaching

A technique named *bleaching* was elaborated in such a manner to mitigate the tie problem of WiSARDs, and consists of adding an access counter to each position in memory. This way, throughout the classification, the response of each discriminator will add up only the positions in memory which has a value in the counter bigger than a threshold, whose initial value is zero. As long as there are ties between two discriminators, the threshold keeps increasing and new responses are calculated based on the new threshold value, until there is a winner. In the training session, each time a pattern is presented to the network, the position accessed in memory will

have its counter incremented. However, in the classification session, this value, called *bleaching value*, can delay the classification compared to the traditional WiSARD, as the threshold may need to be increased. If the threshold becomes large as to give zero response to both discriminators, only in this case the network will choose one randomly [34].

3.1.4 Mental Images

In order to understand what has been learnt by a WiSARD, a DRASiW model can be employed to recall learnt patterns, which are called *mental images* [35]. In order to implement a DRASiW model, during training, it is necessary to store, for each memory location addressed, how many times it was accessed. This can be done using a counter, which increments every time a memory location is accessed during training. In this manner, the mental image produced by a discriminator represents how many times each input bit was presented to the network. Therefore, an image in which the channel levels are proportional to the counter values can be produced, which can be used to represent the class the discriminant learned [32].

Consider the example presented in Figure 3.4. Figures 3.4a, 3.4b and 3.4c represent training data for the character X, which is fed to a discriminator. The DRASiW will increment each bit counter and recall the image presented in Figure 3.4d, which has gray tones proportional to the number of accesses during the training phase.

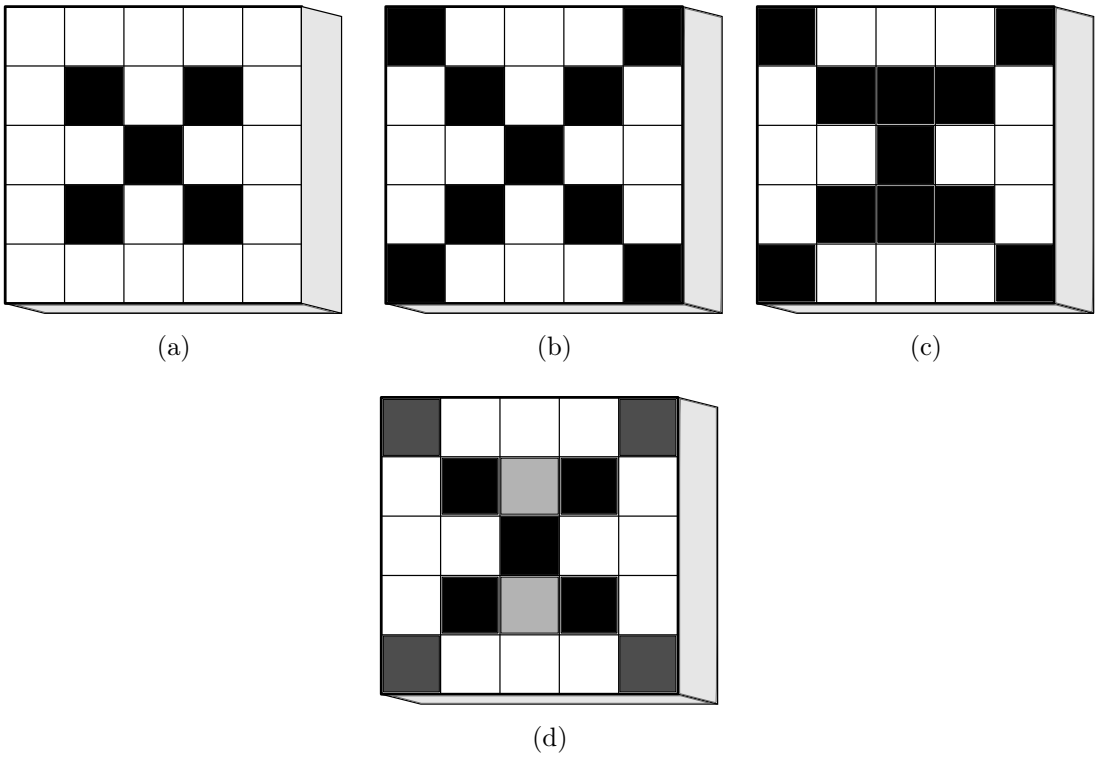


Figure 3.4: Training examples for the letter X and its correspondent mental image.

As can be seen, the DRASiW model can be used to extract the learned rules from the training set, which is not a straightforward procedure in the study of neural networks [36]. With this, it is possible to identify and understand the learned patterns and study how the WiSARD was able to learn the patterns presented in the training data.

3.2 Decision Trees

3.2.1 Decision Tree Model

A decision tree can be represented as a direct graph $G = (V, E), E \subset V^2$, with set of nodes V split into three disjoint sets of decision and terminal nodes. In a decision node, the decision maker selects an action; that is, one of the nodes stemming from this node. Terminal nodes represent the end of a sequence of actions of a decision problem [37]. For example, the decision three in Figure 3.5 can be used to classify characters amongst A, B and C based on the number of loops in the drawing.

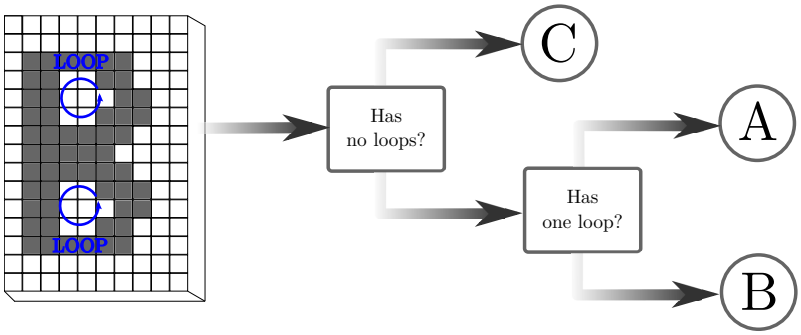


Figure 3.5: An example of a decision tree to classify amongst the characters A, B and C.

This decision model is a simple yet widely-used classification technique, as they provide high accuracy even when the size of the dataset increases [38]. Besides the applications, which ranges from detection spam e-mails to classifying galaxies based on their shapes, it can also be implemented in hardware, leading to a speedup of more than 5 when compared to software implementations [39].

3.2.2 Classification Tree

The first step in order to use a decision tree as a classifier is to train the model using data which the class is known, as in supervised learning. Later, the decision tree can be used to predict the classes of other data. In order to achieve this, there has to be an algorithm, based on a metric, for construction the decision tree, i.e, choose how the tree will branch at each decision node.

One of the first and most famous algorithms introduced was used by the CART (classification and regression tree) algorithm [40], which used the Gini impurity as a metric. This approach is based on evaluating the ability of each attribute to generate pure partitions, that is, partitions in which each branch is homogeneous with respect to the class distribution of its example. That is why the metric is called an impurity, as it measures how impure each branch is [41]. If $\mathbf{u} = (u_1, \cdots, u_n)$ is a vector counting how many examples of each class there are in a node, the Gini impurity is defined as

$$i(\mathbf{u}) = \sum_{i=1}^{n} \frac{u_i}{||\mathbf{u}||} \left(1 - \frac{u_i}{||\mathbf{u}||}.\right)$$

In this manner, the algorithm aims to find the values for each attribute in each decision node that induces a partition of the set of examples with minimum weighted impurity, i.e, in which the weights are given by the number of examples in each of the branches [41].

Chapter 4

Proposed Method

In this chapter, the proposed method for malware classification will be discussed. Section 4.1 will scrutinize the general structure of MalWiSARD, a malware classification system based on weightless neural networks. After that, Section 4.2 illustrates the preprocessing techniques used before a program can be used as input to a WiSARD. Furthermore, Sections 4.3 and 4.4 provide WiSARD systems that can be used within MalWiSARD. Finally, Section 4.5 give further details on virus classification.

4.1 General Structure of MalWiSARD

The general structure of MalWiSARD is depicted in Figure 4.1, which provides a high level overview of the steps involved in the system. The method can be used to classify among 25 malware plus 1 bening software classes. These classes permeate different types of malware, including adware, trojan, virus, worm and backdoor. The system input is a binary executable program and the output is the predicted class for the program. Table 4.1 shows the types and classes of programs that can be analyzed by means of MalWiSARD. The classes were chosen based on the open dataset MaleVis [1], which was used in this work to compare the MalWiSARD with the previous technique.

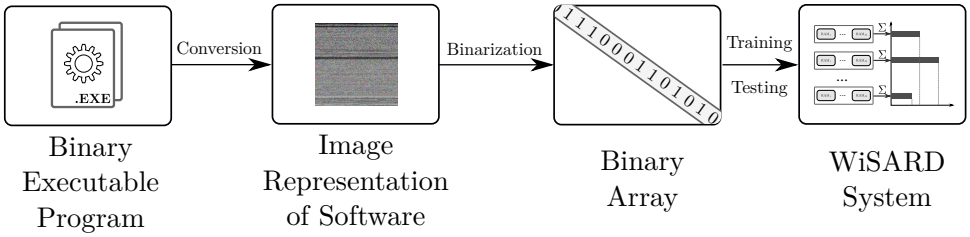


Figure 4.1: General structure of MalWiSARD.

Table 4.1: Types and classes of programs analyzed by MalWiSARD. These are the classes of the MaleVis dataset [1].

Goodness	**Type**	**Class**
Goodware	—	—
Malware	Adware	Adphoshel
		Amonetize
		BrowseFox
		InstallCore
		MultiPlug
		Neoreklami
	Trojan	Agent-fyi
		Dinwod
		Elex
		HackKMS
		Injector
		Regrun
		Snarasite
		VBKrypt
		Vilsel
	Virus	Neshta
		Sality
		Expiro
		VBA
	Worm	Allaple
		Autorun
		Fasong
		Hlux
	Backdoor	Androm
		Stantinko

The first step of MalWiSARD is a preprocessing phase, which consists of preparing a binary executable program to be able to serve as input to a WiSARD, whose input can only be a binary array. After preprocessing into a binary array, data can be classified by a WiSARD system, which consists of one or more weightless neural networks. In this method, there are two kinds of WiSARD system: the Simple WiSARD system and the Tree WiSARD System.

4.2 Preprocessing

The first step consists of preprocessing the input program file, which includes conversion and binarization. As previously has been shown, program's binary files can be treated as images by direct conversion [23], and this Section describes this conversion steps.

4.2.1 Conversion

Although is phase is not done in this work, the method used by the MaleVis dataset is presented. The software data available in the MaleVis dataset is already converted to RGB images. The conversion phase accepts a binary executable program and converts it directly to an 3-channel image. Each byte of the original program represents one channel, such that each pixel has three values each between 0 and 255 for the red, green and blue channels. This process is depicted in Figure 4.2. If the program has a non-multiple of 3 size, the final channels are padded with zeroes.

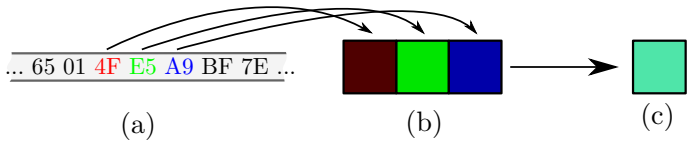


Figure 4.2: Conversion example. Bytes in (a) are converted in sequence to channels RGB in (b), which are combined for the final pixel in (c).

To finish the conversion, the image is resampled to a fixed square sized resolution using Lanczos resampling [42]. The effect that each input sample casts on the interpolated values is defined by the reconstruction Kernel $L(x)$ given by Equation 4.1 [43]:

$$L(x) = \begin{cases} \text{sinc}(x) \cdot \text{sinc}\left(\frac{x}{a}\right), & -a < x < a \\ 0 & \text{otherwise,} \end{cases} \tag{4.1}$$

where $\text{sinc}(x) = \frac{\sin \pi x}{\pi x}$. The value of a determines the size of the kernel. In two dimensions, the kernel is defined by the product of two one-dimensional kernels, as in Equation 4.2:

$$L(x, y) = L(x) \cdot L(y). \tag{4.2}$$

Considering a two dimensional signal s_{ij}, such as an image, the resampled version $S_{x,y}$ is given by Equation 4.3:

$$S(x,y) = \sum_{\lfloor x \rfloor - a + 1}^{\lfloor x \rfloor + 1} \sum_{\lfloor y \rfloor - a + 1}^{\lfloor y \rfloor + 1} s_{ij} \cdot L(x - i) \cdot L(y - j). \tag{4.3}$$

The idea of using a Lanczos resampling is to minimize the pixel loss in the resample phase [1].

4.2.2 Binarization

The binarization phase consists of transforming each channel into a sequence of zeroes and ones in order to use it as an input to a WiSARD, which can only accept binary values. Several techniques can be using within the MalWiSARD, which are described as follows.

Binary Threshold

In the Binary Threshold method, each channel value gets converted to 0 or 1 based on a fixed threshold value. The binarization function is given by Equation 4.4:

$$b_{BT}(x,y) = \begin{cases} 1, & \text{if } S(x,y) \geq T \\ 0 & \text{otherwise,} \end{cases} \tag{4.4}$$

where the fixed value T is chosen prior to the process.

Dynamic Binary Threshold

In the Dynamic Binary Threshold method, the process is still based on the Equation 4.4 but the threshold value is not fixed for every image. The threshold value is chosen as the median of all the channel values in the image, such as shown in Equation 4.5:

$$b_{DBT}(x,y) = \begin{cases} 1, & \text{if } S(x,y) \geq T(S) \\ 0 & \text{otherwise,} \end{cases} \tag{4.5}$$

where $T(S) = \text{median}(S(x,y))$.

One-hot encoding

In the One-hot encoding method, N intervals of values between 0 and 255 are evenly chosen to construct vectors in a one-hot enconding way. This means that each channel is analyzed and the output is a N-dimensional vector which contains one value of 1 representing the interval in which the channel is contained and 0 on the other entries. The binarization is represented by Equation 4.6:

$$b_T(x,y) = (0,0,\cdots,0,\overbrace{1}^{(k+1)^{th}\text{ position}},0,\cdots,0,0), \quad \text{for } S(x,y) \in \left[k\frac{256}{N},(k+1)\frac{256}{N}\right),\ k \in \mathbb{Z}_+. \tag{4.6}$$

For instance, if $N = 4$ is chosen, and the image has a channel with value $3F$, which corresponds to 63 in decimal representation, this value would fall in the interval $\left[0, \frac{256}{4}\right)$. Hence, its representation would be the vector $[1, 0, 0, 0]$.

Circular Representation

The circular representation works in a similar manner of the one-hot encoding, in which a channel value is represented by a vector of N bits and the positions of value 1 represent intervals of values between 0 and 255 that are evenly chosen. The difference lies in the number of value 1 bits to represent each interval. In the circular representation, half the bits always have value 1, and the binarization is represented by Equation 4.7.

$$b_{CR}(x,y) = (0,0,\cdots,0,\overbrace{1,1,\cdots,1,1}^{(k+1)^{th}\text{ position to }(k+\frac{N}{2})^{th}\text{ position}},0,\cdots,0,0), \quad \text{for } S(x,y) \in \left[k\frac{256}{N},(k+1)\frac{256}{N}\right),\ k \in \mathbb{Z}_+. \tag{4.7}$$

The positions which have value 1 are chosen modulo N. For instance, if $N = 4$ is chosen and the image has a channel with value $3F$, which corresponds to 63 in decimal representation, this value would fall in the interval $\left[0, \frac{256}{4}\right)$, and its representation would be the vector $[1, 1, 0, 0]$. However, if the value is CF, which corresponds to 207 in decimal representation, this value would fall in the interval $\left[3 \cdot \frac{256}{4}, 4 \cdot \frac{256}{4}\right)$; in this case, the representation would be the vector $[1, 0, 0, 1]$, because the positions 4 and 5 modulo $N = 4$ would be positions 4 and 1.

4.3 Simple WiSARD System

In the scope of the MalWiSARD, the Simple WiSARD System consists of a WiSARD which can classify programs among the 26 possible classes exposed by Table 4.1. In this manner, this system does not take into account the different types do malware.

4.4 Tree WiSARD System

In order to take into account the different types of malware, another system proposed is the Tree WiSARD System, which takes decisions based on the results of various neural networks. The general structure of the Tree WiSARD System is shown in Figure 4.3.

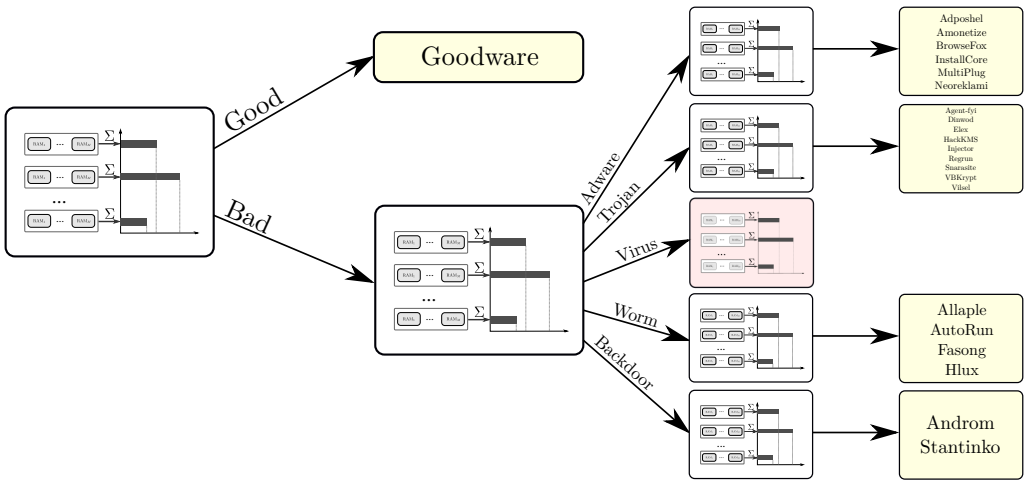


Figure 4.3: Structure of MalWiSARD-tree WiSARD system.

First, the system differentiates between goodware and malware, that is, between unharmful software and malware, through a first WiSARD. If the program is classified as a goodware, there is nothing left to do. Afterwards, if the program is classified as a malware, it goes through another WiSARD which classifies it in one of the five types of malware analyzed: adware, trojan, virus, worm or backdoor. Subsequently, each type has a WiSARD which can distinguish among the classes of malware.

As some virus can make use of software in the host computer in order to spread itself, there are several signatures which can describe virus. In this manner, within this type of malware, it can be difficult for a classification system to differentiate between two classes of virus. Taking this into account, the MalWiSARD can make use of different techniques when analyzing virus, as described in Section 4.5.

4.5 Virus Classification Analysis

Regarding to virus classification, the MalWiSARD can make use of the following techniques: the Simple Analysis, which consists of a WiSARD to differentiate between the classes; the Ensemble Analysis, which employs several WiSARD with different number of addressing bits and use a voting system; the Mental Image Analysis, which analyses the mental images for each discriminator in order to find where are the differences between the classes; and the Decision Tree Analysis, which use a decision tree classifier.

The Simple Analysis employs a WiSARD to differentiate between the four classes of virus. It works similarly to the other classes, but this work shows that this technique by itself has lower accuracy when compared to the other classes. Because of this, another techniques were used in order to increase the accuracy. As different addressing can lead to different results, the idea of the Ensemble Analysis is to explore a range of different number of addressing bits. In this manner, the final class is the most chosen class in the output list of the several WiSARD. As the malware classification field presents fast development of new software every day, the Mental Image Analysis and the Decision Tree Analysis aim to provide a different approach to the system, as the input data can be quite volatile. As the mental images can show what patterns the WiSARD extracted from the presented data, the idea of the Mental Image Analysis is to extract features from the mental images generated by each discriminator and find the regions in the input data which represents the difference among the classes. Finally, the Decision Tree Analysis builds a classification tree in order to differentiate between the classes. The decision tree is built after a training session using the CART algorithm with the Gini impurity as a metric.

Chapter 5

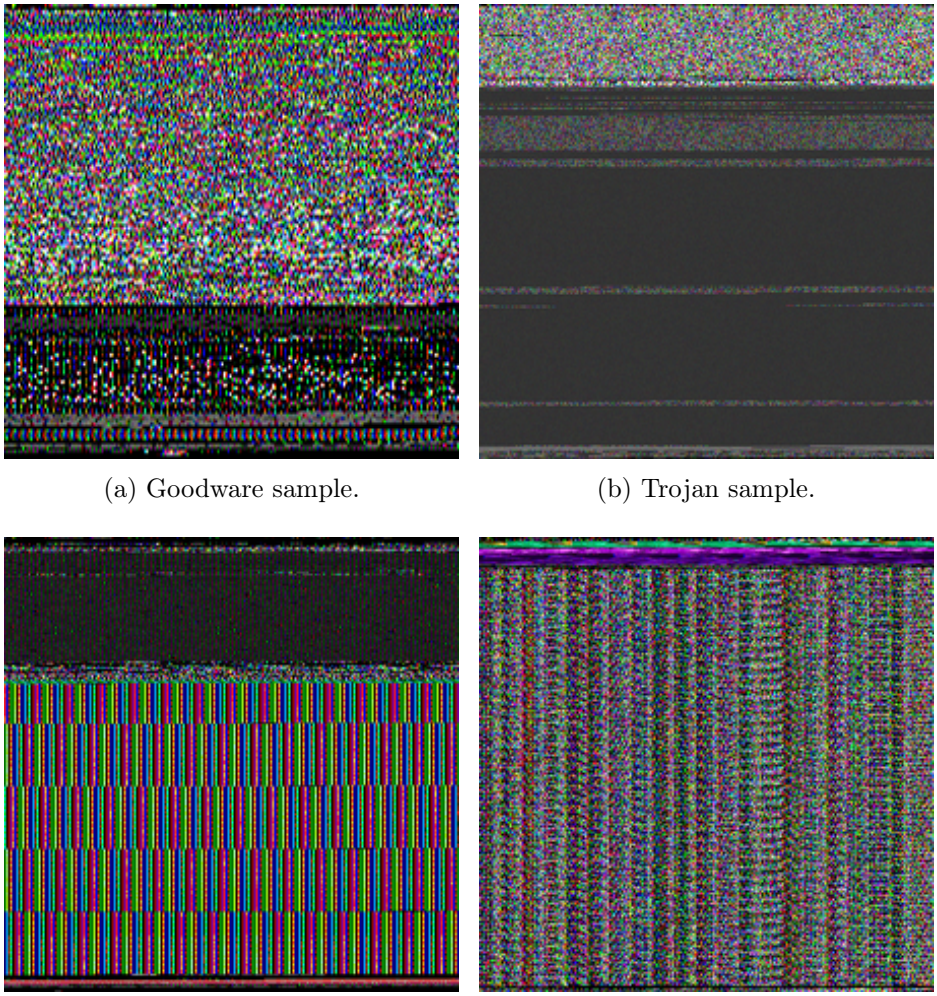
Results and Discussion

This chapter discusses all the experimental infrastructure to validate MalWiSARD. Section 5.1 explicits the database used in the experiments. Subsequently, Section 5.2 details the experimental setup of the experiments and Section 5.3 explores the experiments done and the results obtained. Finally, Section 5.4 discusses the results.

5.1 Database

MalWiSARD was validated the *MaleVis* database was used [44]. This is an open-set image dataset generated from the 25 malware classes that MalWiSARD aims to classify plus one legitimate software class. Figure 5.1 shows samples from the database. It contains 224×224 px^2 images with red, green and blue components (channels) for each pixel.

This database contains image representations of software, which already went through the conversion phase described in Section 4.2.1. Therefore, MalWiSARD will only apply the binarization, training and testing steps for this dataset.



(a) Goodware sample. (b) Trojan sample.

(c) Virus sample. (d) Worm sample.

Figure 5.1: Sample images from database.

This dataset was originally used in [1], which employed a DenseNet based convolutional neural network to classify the images, achieving 97.48% accuracy. Table 5.1 presents the content for each class in the dataset.

Table 5.1: Contents of the MaleVis dataset.

Goodness	Type	Class	Train/Validation Samples
Goodware	—	—	1832/1482
Malware	Adware	Adposhel	350/144
		Amonetize	350/147
		BrowseFox	350/143
		InstallCore	350/150
		MultiPlug	350/149
		Neoreklami	350/150
	Trojan	Agent-fyi	350/120
		Dinwod	350/149
		Elex	350/150
		HackKMS	350/149
		Injector	350/145
		Regrun	350/135
		Snarasite	350/150
		VBKrypt	350/146
		Vilsel	350/146
	Virus	Neshta	350/147
		Sality	350/149
		Expiro	350/150
		VBA	350/150
	Worm	Allaple	350/128
		Autorun	350/146
		Fasong	350/150
		Hlux	350/150
	Backdoor	Androm	350/150
		Stantinko	350/150

As can be seen in Figure 5.1, sample images for 4 classes, there are notable differences among the program image representation, which presents the core opportunity for classification through machine learning techniques.

5.2 Experimental Setup

This section presents the experimental setup used for the experiments. All experiments were conducted using a server equipped with a Intel(R) Core(TM) i7-6700 CPU @ 3.40 GHz and 4 cores with hyperthreading disabled, provided by the Com-

puter Architecture and Microelectronics Laboratory at the Federal University of Rio de Janeiro.

The experiments were conducted using the Python 3.9.6 programming language exploying the *wisardpkg* module, a library developed in order to facilitate the production of codes using WiSARD-based models [45].

Each experiment was repeated ten times and the metrics used to validate the model are shown in Table 5.2, where TP is the number of true positives; TN is the number of true negatives; FP is the number of false positives; and FN is the number of false negatives. Besides, a confusion matrix was constructed for each test.

Table 5.2: Metrics used to validate the model.

Metric	**Formula**
Precision	$\frac{TP}{TP+FP}$
Recall	$\frac{TP}{TP+FN}$
F1-score	Harmonic mean between Precision and Recall
Accuracy	$\frac{TP+TN}{TP+TN+FP+FN}$
Training time	Time to train a set of samples
Classification time	Time to classify a set of samples

5.3 Experiments and Discussion

This section explores and discusses the experiments done in order to validate MalWiSARD. Every experiment was repeated 10 times and, when the WiSARD is employed, the bleaching technique was activated.

5.3.1 Preprocessing

In order to compare the preprocessing techniques, a WiSARD to differentiate between goodware and malware was employed [46], which can be called a detection method. In this experiment, all classes of malware present in Table 5.1 were labelled as 'malware'. The number of addressing bits chosen was $n = 20$bits after a grid search from $n = 1$ to $n = 64$; the criteria was the highest accuracy. Figure 5.2 shows the time spent to binarize every image in the database using different methods, and Figure 5.3 illustrates the accuracy obtained by a binary classification WiSARD after every binarization method.

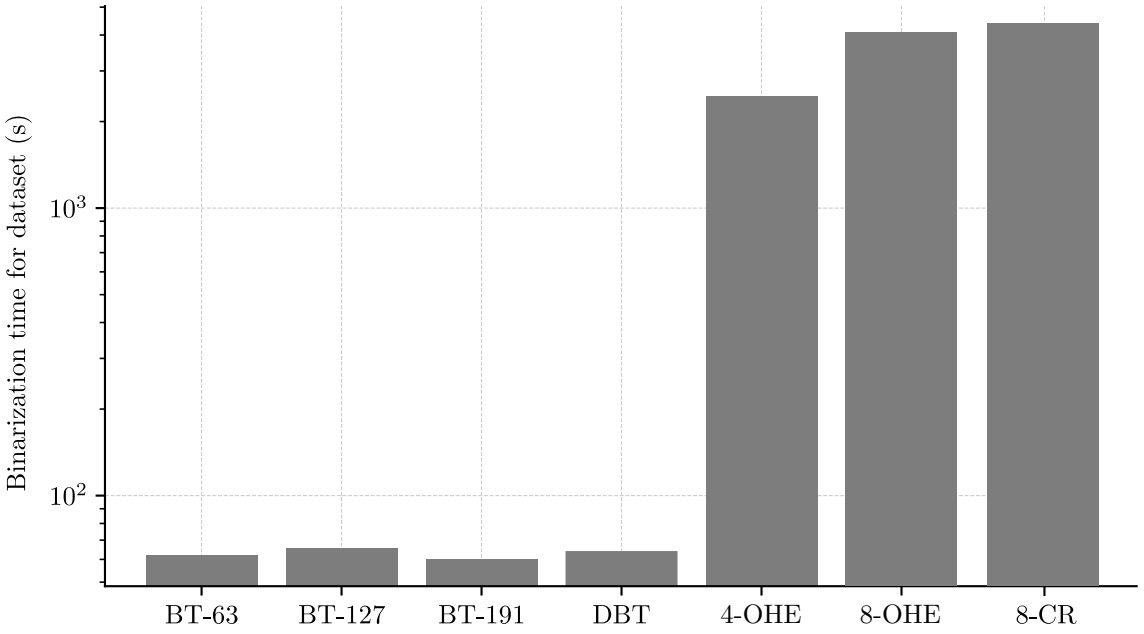


Figure 5.2: Binarization time for different techniques.

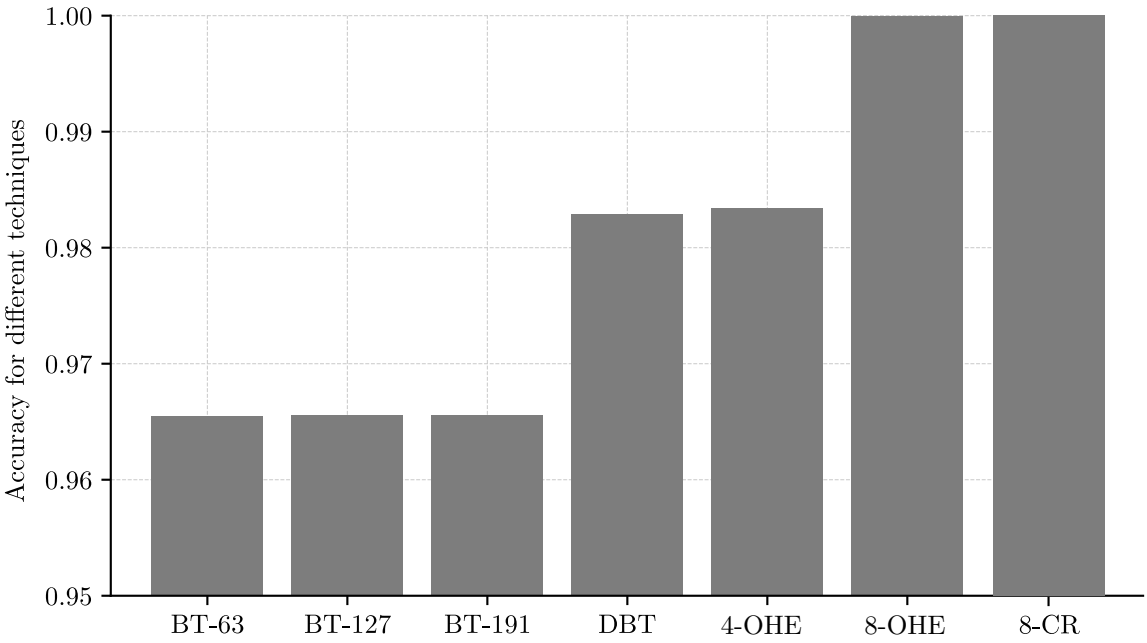


Figure 5.3: WiSARD accuracy for different binarization techniques.

The circular representation and the one-hot encoding binarization had similar results. Based on the result, the binarization preprocessing technique chosen for the next experiments is the 8-bit one-hot-encoding, because of the higher accuracy among the preprocessing techniques. It is important to note that the binarization time for the 8-bit one-hot encoding was also higher in more than an order of magnitude. Nevertheless, this process can be easily parallelized or computed via hardware.

5.3.2 MalWiSARD with Simple WiSARD System

Using an 8-bit one-hot encoding as the binarization technique, all 26 classes were submitted to the Simple WiSARD System. The results are shown in Table 5.3 and the confusion matrix is presented in Figure 5.4.

Table 5.3: Results for the Simple WiSARD System.

	Simple WiSARD System	
Metric	Mean	Std. Deviation
F1-Score	0.6938	0.0000
Precision	0.6483	0.0000
Recall	0.7748	1.1102×10^{-16}
Accuracy	0.7748	1.1102×10^{-16}
Training time (s)	159.441	4.457
Classification time (s)	1926.608	47.302

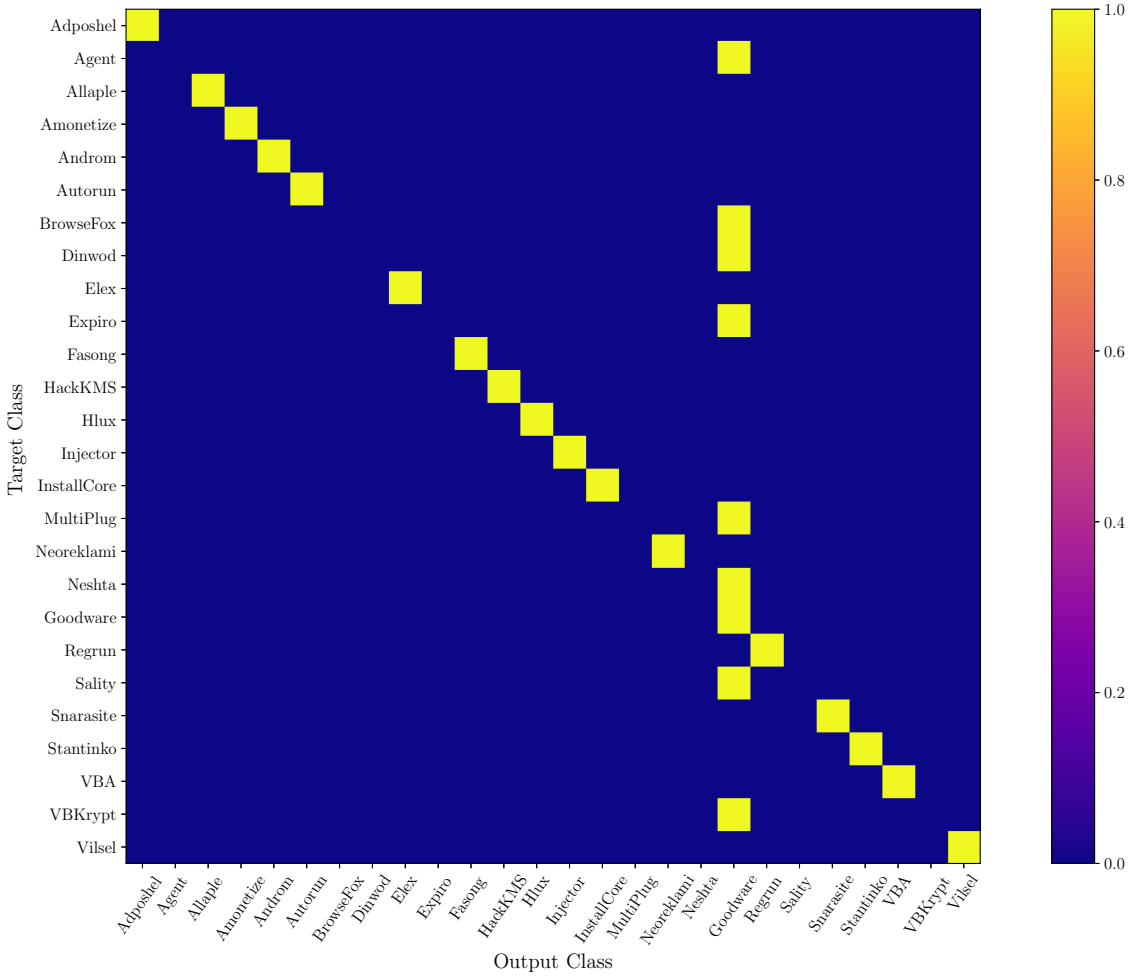


Figure 5.4: Confusion matrix for the Simple WiSARD System.

It can be seen that the WiSARD is not able to distinguish among some classes of programs, classifying them as Goodware. All experiments presented the same result, but the classes varied. Besides, these other classes are classified 100% erroneously and confused with other classes.

5.3.3 MalWiSARD with Tree WiSARD System

The Tree WiSARD System makes use of a flow of decisions based on the output of the networks. The experiment can be divided in three steps: the goodness test, which determines if the program is a goodware or a malware; the type test, which classifies the malware within its type; and the class test, which further classifies the class of malware within each type.

Goodness test

The results for the goodness test are shown in Table 5.4 and the confusion matrix is presented in Figure 5.5.

Table 5.4: Results for the Tree WiSARD System - Goodness Test.

	Tree WiSARD System - Goodness	
Metric	Mean	Std. Deviation
F1-Score	1.00	0.00
Precision	1.00	0.00
Recall	1.00	0.00
Accuracy	1.00	0.00
Training time (s)	212.1903	0.5173
Classification time (s)	230.6231	0.1149

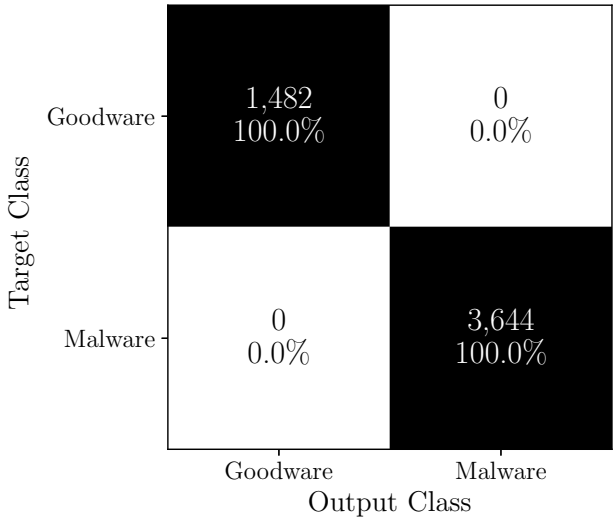


Figure 5.5: Confusion matrix for the Tree WiSARD System - Goodness Test.

This experiment shows that the WiSARD is able to distinguish between a goodware and a malware with 100% accuracy, which can be used as a detection system already.

Type test

The results for the type test are shown in Table 5.5 and the confusion matrix is presented in Figure 5.6.

Table 5.5: Results for the Tree WiSARD System - Type Test.

	Tree WiSARD System - Type	
Metric	Mean	Std. Deviation
F1-Score	0.9678	1.1102×10^{-16}
Precision	0.9727	2.2204×10^{-16}
Recall	0.9670	1.1102×10^{-16}
Accuracy	0.9670	1.1102×10^{-16}
Training time (s)	166.0064	3.8004
Classification time (s)	365.0621	3.7838

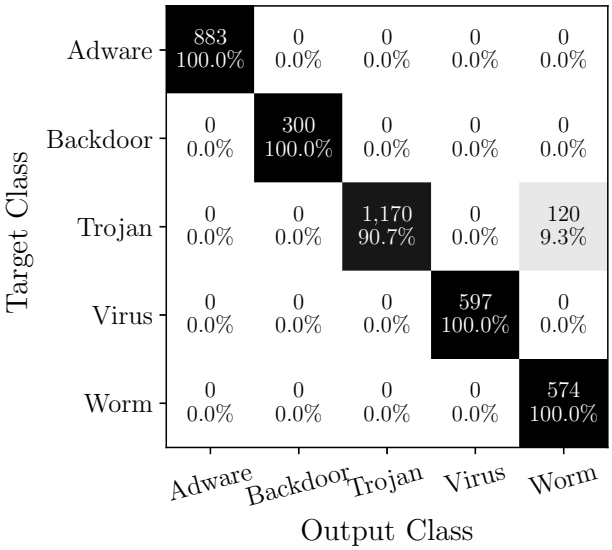


Figure 5.6: Confusion matrix for the Tree WiSARD System - Type Test.

It can be concluded, by the type test, that some Worms are confused with Trojans. As Worms and Trojan can be used to access file systems and can use the network, it is possible that some code reuse between these two types of malware were capture by the WiSARD, leading to confusion.

Class test

The results for each class test can be seen in Table 5.6. Furthermore, Figures 5.7, 5.8, 5.9 and 5.10 show the confusion matrices for each type of malware.

Table 5.6: Results for the Tree WiSARD System - Class Test.

	Adware		Trojan		Worm		Backdoor	
Metric	Mean	Std. Dev.	Mean	Std. Dev.	Mean	Std. Dev.	Mean	Std. Dev.
F1-Score	1.00	0.00	1.00	0.00	1.00	0.00	1.00	0.00
Precision	1.00	0.00	1.00	0.00	1.00	0.00	1.00	0.00
Recall	1.00	0.00	1.00	0.00	1.00	0.00	1.00	0.00
Accuracy	1.00	0.00	1.00	0.00	1.00	0.00	1.00	0.00
Training time (s)	37.7001	1.0146	40.8444	1.8907	21.0938	0.4681	11.0040	0.0552
Classification time (s)	116.6996	3.1755	162.5134	3.9988	35.5540	0.3304	9.6630	0.0145

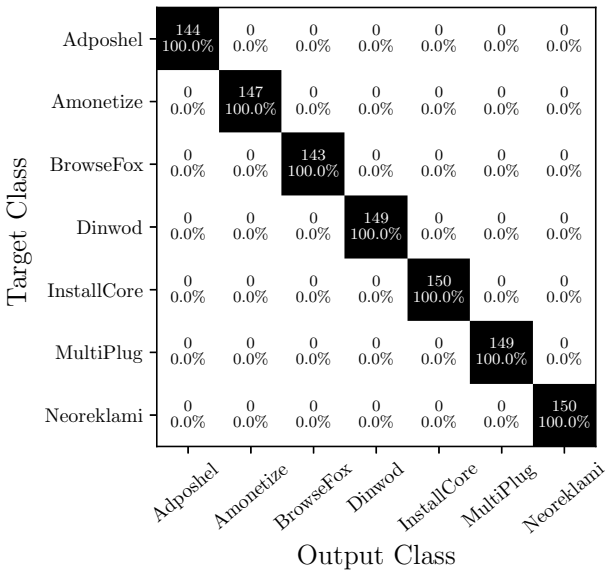


Figure 5.7: Confusion matrix for the Tree WiSARD System - Class Test for Adware.

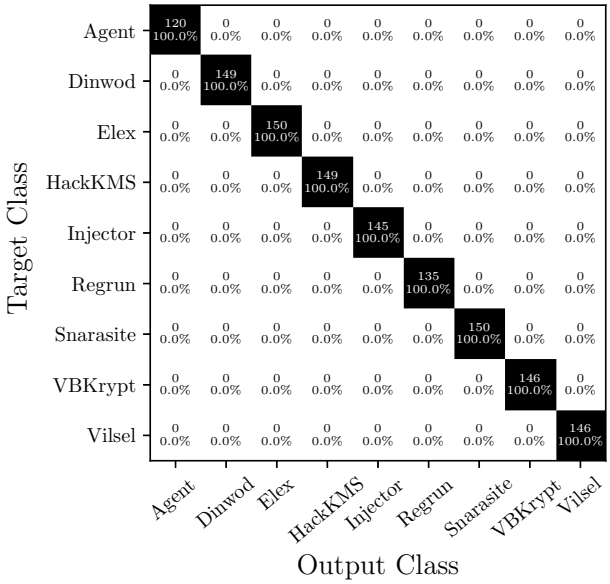


Figure 5.8: Confusion matrix for the Tree WiSARD System - Class Test for Trojan.

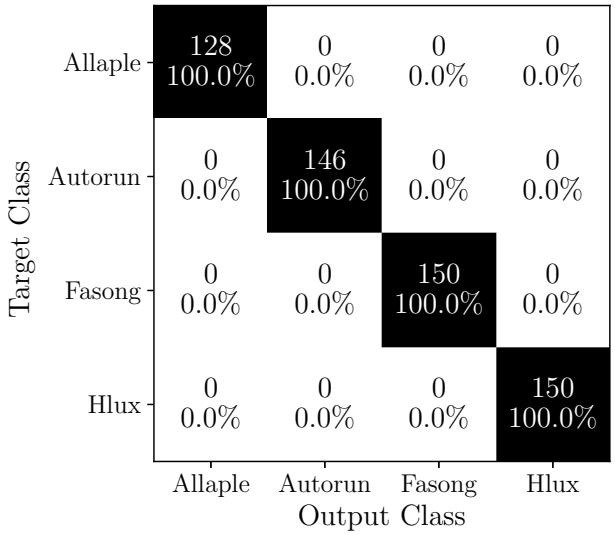


Figure 5.9: Confusion matrix for the Tree WiSARD System - Class Test for Worm.

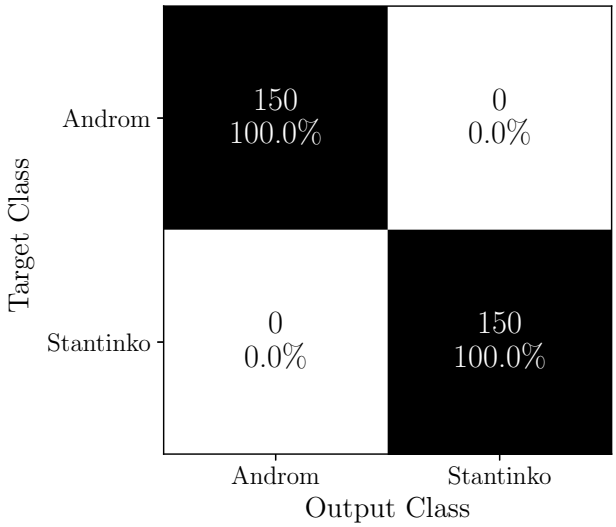


Figure 5.10: Confusion matrix for the Tree WiSARD System - Class Test for Backdoor.

For the types adware, trojan, worm and backdoor, the WiSARD is able to fully distinguish among them. The virus type is analyzed separately in Section 5.3.4.

5.3.4 Virus Analysis Techniques

As explained in Section 4.4, the virus classification will be presented separately in order to further explore the difficulties in classifying this type of malware.

Simple

Table 5.7 shows the results for the classification among virus using the WiSARD, and the corresponding confusion matrix is presented in Figure 5.11.

Table 5.7: Results for the Simple Analysis for Virus.

	Simple Analysis	
Metric	Mean	Std. Deviation
F1-Score	0.4213	5.5511×10^{-17}
Precision	0.3794	5.5511×10^{-17}
Recall	0.5041	0.000
Accuracy	0.5041	0.000
Training time (s)	24.3200	0.1851
Classification time (s)	34.8086	0.1787

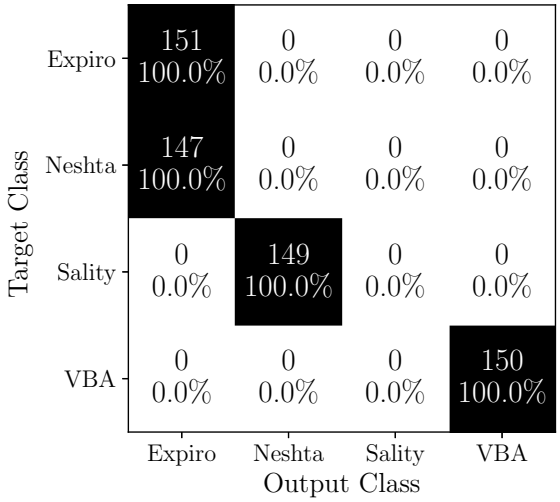


Figure 5.11: Confusion matrix for the Simple Analysis for Virus.

It can be seen that there is one class, VBA, which was fully distinguish. Therefore, the WiSARD in the simple analysis is able to classify VBA correctly. However, the system presented worst results in the other three classes, and two of them were 100% confused with another. This is due to the high code reuse and obfuscating effort employed in the creation of new virus, which is one of the most popular types of malware.

Ensemble

In order to surpass the problem of virus classification, and ensemble system was employed. The ensemble consists of six different WiSARDs, each one with different number of addressing bits, varying in the set $\{2, 4, 8, 16, 32, 64\}$. The idea is to use a voting classifier consisting of WiSARDs, that is, among the six output presented, the most chosen is elected as the classified class. If there is a tie, a random one is chosen. This method was applied to the three classes which were confused in the simple analysis, as the VBA class was perfectly classified. Table 5.8 shows the

results for the classification among virus using the WiSARD, and the corresponding confusion matrix is presented in Figure 5.12.

Table 5.8: Results for the Ensemble Analysis for Virus.

	Ensemble Analysis	
Metric	Mean	Std. Deviation
F1-Score	1.00	0.00
Precision	1.00	0.00
Recall	1.00	0.00
Accuracy	1.00	0.00
Training time (s)	136.5568	3.0898
Classification time (s)	185.1278	2.3675

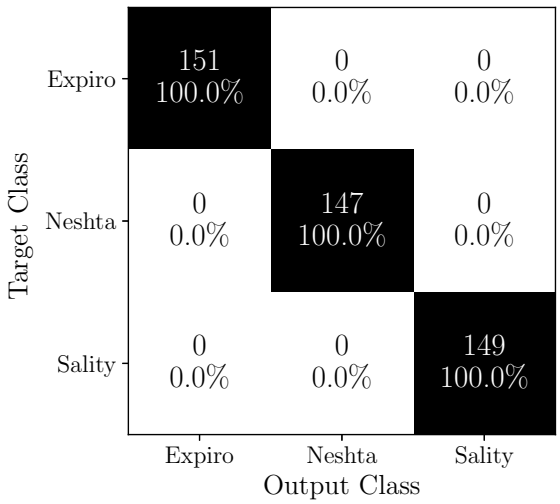
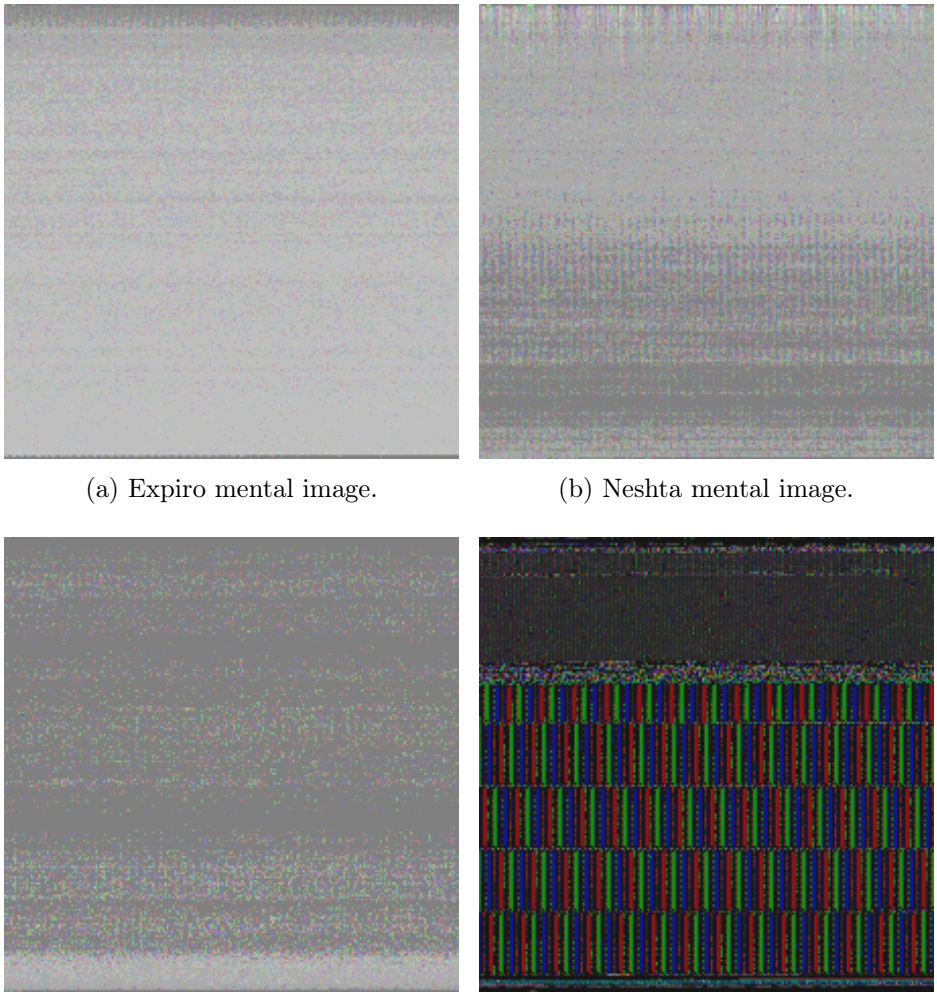


Figure 5.12: Confusion matrix for the Ensemble Analysis for Virus.

With different numbers of addressing bits, the ensemble analysis was able to capture a wider range of features hidden in the training data, leading to 100% accuracy.

Mental Image

As can be seen in the Simple Analysis, the WiSARD was not able to differentiate between three classes. In order to understand what patterns were learned by the system, Figure 5.13 shows the mental images produced the the Simple Analysis.



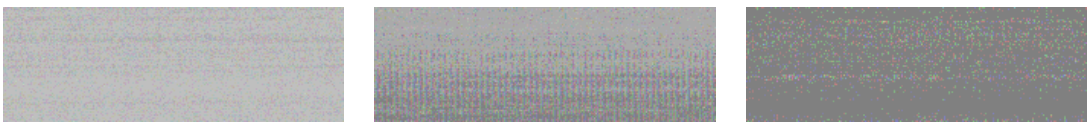
(a) Expiro mental image. (b) Neshta mental image.

(c) Sality mental image. (d) VBA mental image.

Figure 5.13: Mental images for the Virus Type.

It can be seen that Figure 5.13d, representing the VBA class, is quite different from the other three. This can explain why the WiSARD was able to completely classify this class.

The mental image analysis presented here tries to find patterns in the mental images and, instead of using the whole image, using only the regions in which a difference can be seen. In this analysis, the horizontal middle third, presented in Figure 5.14, was chosen in order to represent the images.



(a) Expiro cut mental image. (b) Neshta cut mental image. (c) Sality cut mental image.

Figure 5.14: Cut mental images for the Virus Type.

In this manner, Table 5.9 shows the results for the classification among virus

using the WiSARD trained with the cut images, and the corresponding confusion matrix is presented in Figure 5.15.

Table 5.9: Results for the Mental Image Analysis for Virus.

	Mental Image Analysis	
Metric	Mean	Std. Deviation
F1-Score	0.5516	1.1102×10^{-16}
Precision	0.4966	1.1102×10^{-16}
Recall	0.6629	0.000
Accuracy	0.6629	0.000
Training time (s)	7.9621	0.0376
Classification time (s)	8.2631	0.0292

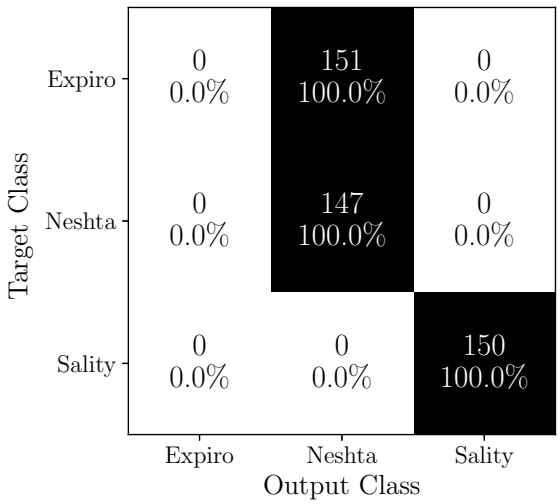


Figure 5.15: Confusion matrix for the Mental Image Analysis for Virus.

It can be seen that the accuracy, for instance, from the Simple Analysis, rose from 50.41% to 66.29%, testifying that the DRASiW can provide insight for the different patterns which represent each class.

Decision Tree

As the domain of malware classification can be deeply volatile, in order to be resilient to trend changes, a decision tree classifier was also employed to be compared with the WiSARD. Thus, Table 5.10 shows the results for the classification using the decision tree analysis, and the corresponding confusion matrix is presented in Figure 5.16.

Table 5.10: Results for the Decision Tree Analysis for Virus.

	Decision Tree Analysis	
Metric	Mean	Std. Deviation
F1-Score	0.8327	0.0124
Precision	0.8338	0.0125
Recall	0.8326	0.0123
Accuracy	0.8326	0.0123
Training time (s)	12.1495	0.4715
Classification time (s)	0.0564	0.0002

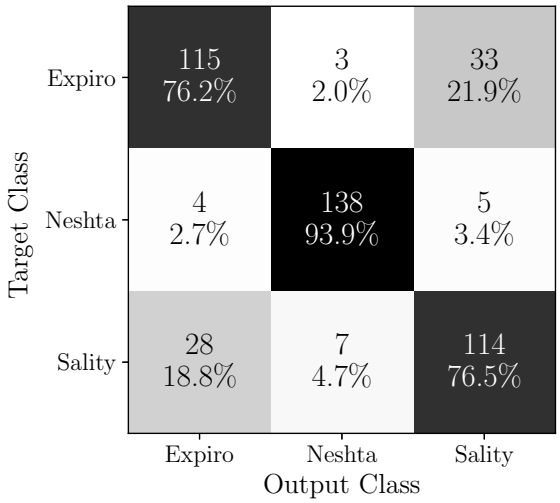


Figure 5.16: Confusion matrix for the Decision Tree Analysis for Virus.

The decision tree analysis gives higher accuracy when compared to the simple analysis and the mental image analysis. Moreover, the training time is comparable to the other methods, however the classification time can be almost 150 times faster. Besides the lowest accuracy when compared to the ensemble analysis, depending on the application, the decision tree can be used to provide faster response times.

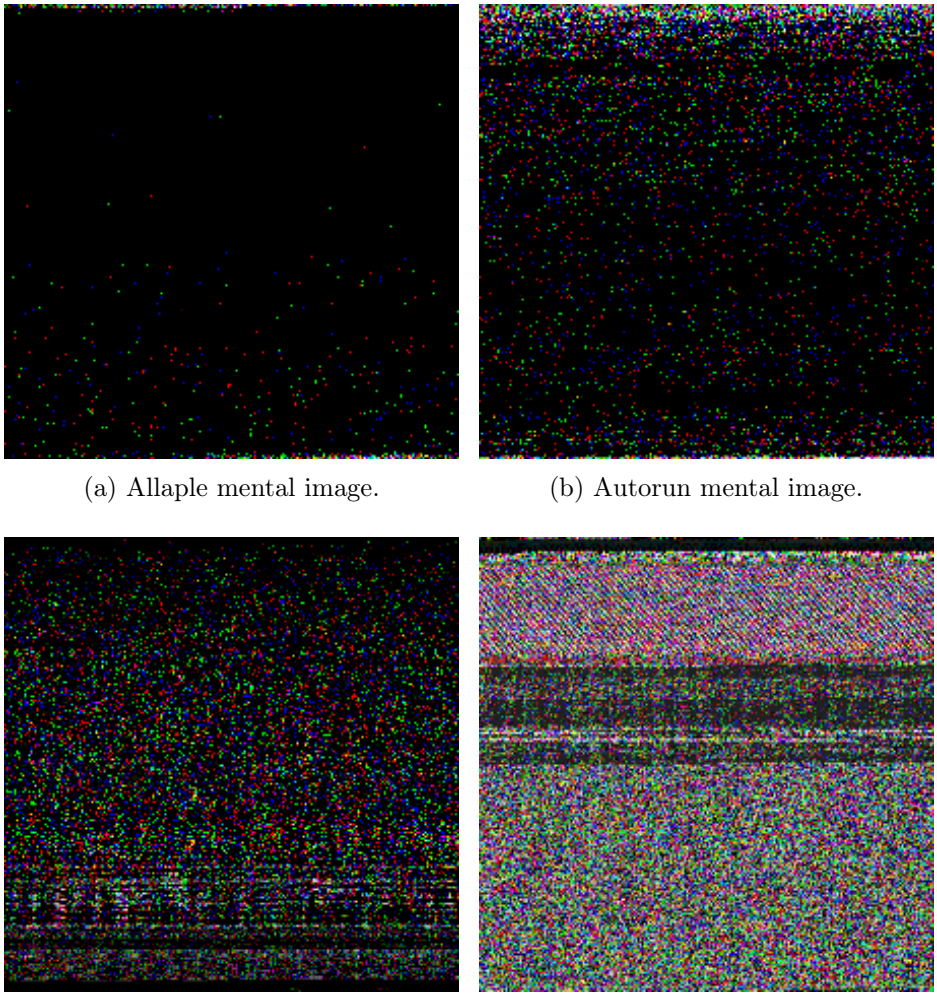
5.4 Discussion

The overall accuracy of the MalWiSARD using the Tree WiSARD System and the Ensemble Analysis for the virus type was 98.5692%, which is higher than the state-of-the-art method for the MaleVis dataset presented in [1], which achieved an accuracy of 97.48%. Besides, the overall average time to train was 10.90 minutes, which is almost 13 times faster than the fastest model presented in [1]. Table 5.11 summarizes the results obtained by each variation of MalWiSARD in comparison with the state-of-the-art DenseNet.

Table 5.11: Comparison among variants.

Method	Accuracy
MalWiSARD: simple	77.48%
MalWiSARD: tree	91.89%
MalWiSARD: tree - mental images	93.73%
MalWiSARD: tree - decision tree	95.71%
DenseNet (Bozkir et. al. 2019)	97.48%
MalWiSARD: tree - ensemble	**98.56%**

Besides, Figure 5.17 presents, as an example, the mental images learned for the Worm Type. As can be seen, the four classes of Worm present quite different learning patterns. The mental images can be used to explain how the WiSARDs presented in the system were able to learn the differences between the training data.



(a) Allaple mental image.

(b) Autorun mental image.

(c) Fasong mental image.

(d) Hlux mental image.

Figure 5.17: Mental images for the Worm Type.

Chapter 6

Conclusion

This work presented MalWiSARD, a weightless neural network system used to classify among 26 classes of software, which includes malware. The highest accuracy among all variants of MalWiSARD was achieved by the Tree WiSARD System and the Ensemble Analysis for the virus type, with 98.5692% accuracy, which is higher than the state-of-the-art method for the MaleVis dataset presented in [1], which achieved an accuracy of 97.48%. Besides, the overall average time to train was 10.90 minutes, which is almost 13 times faster than the fastest model presented in [1].

The WiSARD implementation, besides being faster to train and having higher accuracy rates, is simpler, as the system is tuned by the number of addressing bits of each WiSARD, the bleaching option and the preprocessing technique, in constrast with a Densely Connected Convolutional Networks, which need millions of parameters to be tuned.

As future work, the challenges of hardware implementation of WiSARD will be studied. With this, systems like MalWiSARD will be able to be employed in real time systems for rapid software analysis in big data flows in a network, for instance.

Another path of research is to extract patterns from the mental images obtained by the system, which definitely can be analyzed to bring explanation of what was learned by the neural network. The difference between the mental images clearly shows the WiSARD ability to learn and differentiate between patterns, but an algorithm to extract and identify these patterns would present a great advance in the explainable machine learning field.

References

[1] BOZKIR, A. S., CANKAYA, A. O., AYDOS, M. "Utilization and Comparision of Convolutional Neural Networks in Malware Recognition". In: *27th Signal Processing and Communications Applications Conference (SIU)*, p. 1–4, Apr 2019. doi: 10.1109/SIU.2019.8806511.

[2] CHRISTODORESCU, M., JHA, S., SESHIA, S., et al. "Semantics-aware malware detection". In: *2005 IEEE Symposium on Security and Privacy (S P'05)*, pp. 32–46, 2005. doi: 10.1109/SP.2005.20.

[3] MCGRAW, G., MORRISETT, G. "Attacking Malicious Code: A Report to the Infosec Research Council", *IEEE Software*, v. 17, n. 5, pp. 33–41, 2000. doi: 10.1109/52.877857.

[4] VINOD P., V. LAXMI, M. G. "Survey on Malware Detection Methods". In: *3rd Hackers' Workshop on Computer and Internet Security*, pp. 74–79, India, mar. 2009.

[5] SAEED, I., SELAMAT, A., ABUAGOUB, A. "A Survey on Malware and Malware Detection Systems", *International Journal of Computer Applications*, v. 67, pp. 25–31, 04 2013. doi: 10.5120/11480-7108.

[6] IDIKA, N., MATHUR, A. "A survey of malware detection techniques", *Purdue University*, 03 2007.

[7] LANDWEHR, C. E., BULL, A. R., MCDERMOTT, J. P., et al. "A Taxonomy of Computer Program Security Flaws", *ACM Comput. Surv.*, v. 26, n. 3, pp. 211–254, set. 1994. ISSN: 0360-0300. doi: 10.1145/185403.185412. Disponível em: <`https://doi.org/10.1145/185403.185412`>.

[8] GORDON, L., LOEB, M., LUCYSHYN, W., et al. "CSI/FBI Computer Crime and Security Survey", *Computer Security Institute*, v. 22, 01 2000.

[9] YE, Y., LI, T., JIANG, Q., et al. "Intelligent File Scoring System for Malware Detection from the Gray List". In: *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD

'09, p. 1385–1394, New York, NY, USA, 2009. Association for Computing Machinery. ISBN: 9781605584959. doi: 10.1145/1557019.1557167. Disponível em: <https://doi.org/10.1145/1557019.1557167>.

[10] YE, Y., LI, T., ADJEROH, D., et al. "A Survey on Malware Detection Using Data Mining Techniques", *ACM Comput. Surv.*, v. 50, n. 3, jun. 2017. ISSN: 0360-0300. doi: 10.1145/3073559. Disponível em: <https://doi.org/10.1145/3073559>.

[11] YE, Y., LI, T., ZHU, S., et al. "Combining file content and file relations for cloud based malware detection". In: *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 222–230, 2011.

[12] MOSKOVITCH, R., FEHER, C., ELOVICI, Y. "A Chronological Evaluation of Unknown Malcode Detection". pp. 112–117, 04 2009. ISBN: 978-3-642-01392-8. doi: 10.1007/978-3-642-01393-5_12.

[13] EGELE, M., SCHOLTE, T., KIRDA, E., et al. "A Survey on Automated Dynamic Malware-Analysis Techniques and Tools", *ACM Comput. Surv.*, v. 44, n. 2, mar. 2008. ISSN: 0360-0300. doi: 10.1145/2089125.2089126. Disponível em: <https://doi.org/10.1145/2089125.2089126>.

[14] ELLIS, D., AIKEN, J., ATTWOOD, K., et al. "A behavioral approach to worm detection". pp. 43–53, 01 2004. doi: 10.1145/1029618.1029625.

[15] ILGUN, K., KEMMERER, R., PORRAS, P. "State transition analysis: a rule-based intrusion detection approach", *IEEE Transactions on Software Engineering*, v. 21, n. 3, pp. 181–199, 1995. doi: 10.1109/32.372146.

[16] CHRISTODORESCU, M., JHA, S. "Static Analysis of Executables to Detect Malicious Patterns". In: *Proceedings of the 12th Conference on USENIX Security Symposium - Volume 12*, SSYM'03, p. 12, USA, 2003. USENIX Association.

[17] KUMAR, S., SPAFFORD, E. "A generic virus scanner for C++". In: *[1992] Proceedings Eighth Annual Computer Security Application Conference*, pp. 210–219, 1992. doi: 10.1109/CSAC.1992.228218.

[18] SULAIMAN, A., RAMAMOORTHY, K., MUKKAMALA, S., et al. "Malware examiner using disassembled code (MEDiC)". In: *Proceedings from the Sixth Annual IEEE SMC Information Assurance Workshop*, pp. 428–429, 2005. doi: 10.1109/IAW.2005.1495985.

[19] TREADWELL, S., ZHOU, M. "A heuristic approach for detection of obfuscated malware". In: *2009 IEEE International Conference on Intelligence and Security Informatics*, pp. 291–299, 2009. doi: 10.1109/ISI.2009.5137328.

[20] ELOVICI, Y., SHABTAI, A., MOSKOVITCH, R., et al. "Applying Machine Learning Techniques for Detection of Malicious Code in Network Traffic". v. 4667, pp. 44–50, 09 2007. ISBN: 978-3-540-74564-8. doi: 10.1007/978-3-540-74565-5_5.

[21] WANG, T.-Y., HORNG, S.-J., SU, M.-Y., et al. "A Surveillance Spyware Detection System Based on Data Mining Methods". In: *2006 IEEE International Conference on Evolutionary Computation*, pp. 3236–3241, 2006. doi: 10.1109/CEC.2006.1688720.

[22] ANDERSON, B., QUIST, D., NEIL, J., et al. "Graph-based malware detection using dynamic analysis", *Journal in Computer Virology*, v. 7, pp. 247–258, 11 2011. doi: 10.1007/s11416-011-0152-x.

[23] NATARAJ, L. *A signal processing approach to malware analysis*. University of California, Santa Barbara, 2015.

[24] NATARAJ, L., KIRAT, D., MANJUNATH, B., et al. "Sarvam: Search and retrieval of malware". In: *Proceedings of the Annual Computer Security Conference (ACSAC) Worshop on Next Generation Malware Attacks and Defense (NGMAD)*, 2013.

[25] GARCIA, F. C. C., MUGA II, F. P. "Random Forest for Malware Classification", *arXiv:1609.07770 [cs]*, Sep 2016. Disponível em: <http://arxiv.org/abs/1609.07770>. arXiv: 1609.07770.

[26] FARROKHMANESH, M., HAMZEH, A. "A novel method for malware detection using audio signal processing techniques". In: *2016 Artificial Intelligence and Robotics (IRANOPEN)*, p. 85–91, Apr 2016. doi: 10.1109/RIOS.2016.7529495.

[27] GARCÍA-TEODORO, P., DÍAZ-VERDEJO, J., MACIÁ-FERNÁNDEZ, G., et al. "Anomaly-based network intrusion detection: Techniques, systems and challenges", *Computers & Security*, v. 28, n. 1, pp. 18–28, 2009. ISSN: 0167-4048. doi: https://doi.org/10.1016/j.cose.2008.08.003. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0167404808000692>.

[28] CHAUGULE, A., XU, Z., ZHU, S. "A Specification Based Intrusion Detection Framework for Mobile Phones". In: Lopez, J., Tsudik, G. (Eds.), *Applied Cryptography and Network Security*, pp. 19–37, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[29] BLOUNT, J. J., TAURITZ, D. R., MULDER, S. A. "Adaptive Rule-Based Malware Detection Employing Learning Classifier Systems: A Proof of Concept". In: *2011 IEEE 35th Annual Computer Software and Applications Conference Workshops*, pp. 110–115, 2011. doi: 10.1109/COMPSACW.2011.28.

[30] AHMED, M., PAL, R., HOSSAIN, M. M., et al. "NIDS: A Network Based Approach to Intrusion Detection and Prevention". In: *2009 International Association of Computer Science and Information Technology - Spring Conference*, pp. 141–144, 2009. doi: 10.1109/IACSIT-SC.2009.96.

[31] BLEDSOE, W. W., BROWNING, I. "Pattern recognition and reading by machine". In: *IRE-AIEE-ACM Computer Conference*, pp. 225–232, Boston, Massachusetts, dec 1959.

[32] GRIECO, B. P., LIMA, P. M., DE GREGORIO, M., et al. "Extracting fuzzy rules from "mental" images generated by modified WiSARD perceptrons". In: *17th European Symposium on Artificial Neural Networks*, pp. 313–318, Belgium, apr 2009.

[33] ALEKSANDER, I., THOMAS, W., BOWDEN, P. "WISARD· a radical step forward in image recognition", *Sensor review*, v. 4, n. 3, pp. 120–124, mar 1984.

[34] DE SOUZA, C. R. *Redes Neurais sem-peso aplicadas na categorização de subtipos do HIV-1.* M.Sc. dissertation, PESC/COPPE - Universidade Federal do Rio de Janeiro, Rio de Janeiro, RJ, Brazil, 2011.

[35] DE GREGORIO, M., C.M, S., C.L, F., et al. "Uma Implementação em Software do Classificador WISARD". 12 1998.

[36] BURATTINI, E., CORAGGIO, P., DE GREGORIO, M., et al. "Agent WiSARD in a 3D World." pp. 272–280, 01 2005.

[37] KAMINSKI, B., JAKUBCZYK, M., SZUFEL, P. "A framework for sensitivity analysis of decision trees", *Central European Journal of Operations Research*, v. 26, 03 2018. doi: 10.1007/s10100-017-0479-6.

[38] CATLETT, J. *Megainduction: Machine Learning on Very Large Databases*. Tese de D.Sc., Basser Department of Computer Science, University of Sydney, Sydney, Australia, 1991.

[39] NARAYANAN, R., HONBO, D., MEMIK, G., et al. "An FPGA Implementation of Decision Tree Classification". In: *2007 Design, Automation Test in Europe Conference Exhibition*, pp. 1–6, 2007. doi: 10.1109/DATE.2007.364589.

[40] LEO BREIMAN, JEROME H. FRIEDMAN, R. A. O. C. J. S. "Classification and regression trees". 1 ed., Monterey, Brooks/Cole Publishing, 1984.

[41] LABER, E., MURTINHO, L. "Minimization of Gini Impurity: NP-completeness and Approximation Algorithm via Connections with the k-means Problem", *Electronic Notes in Theoretical Computer Science*, v. 346, pp. 567–576, 2019.

[42] MADHUKAR, B., NARENDRA, R. "Lanczos resampling for the digital processing of remotely sensed images". In: *Proceedings of International Conference on VLSI, Communication, Advanced Devices, Signals & Systems and Networking (VCASAN-2013)*, pp. 403–411. Springer, 2013.

[43] JAIN, A. K. *Fundamentals of digital image processing*. Prentice-Hall, Inc., 1989.

[44] "MaleVis: A Dataset for Vision Based Malware Recognition". Disponível em: <`https://web.cs.hacettepe.edu.tr/~selman/malevis/`>.

[45] FILHO, A. S. L., GUARISA, G. P., FILHO, L. A. D. L., et al. "wisardpkg – A library for WiSARD-based models". 2020.

[46] L. C. S. RAMOS, L. A. D. LUSQUINO FILHO, F. M. G. F. P. M. V. L. "Detecção estática e dinâmica de malwares usando redes neurais sem peso". In: *Brazilian Symposium on Information and Computational Systems Security (SBSeg-2020)*. SBC, 2020.

Appendix A

Papers Published

Conference Papers

L. C. S. RAMOS, L. A. D. LUSQUINO FILHO, F. M. G. FRANÇA, P. M. V. LIMA; Detecção estática e dinâmica de malwares usando redes neurais sem peso, *Brazilian Symposium on Information and Computational Systems Security*, 2020.

Detecção estática e dinâmica de *malwares* usando redes neurais sem peso

Luiz C. S. Ramos[1], **Leopoldo A. D. Lusquino Filho**[1], **Felipe M. G. França**[1], **Priscila M. V. Lima**[1]

[1]Programa de Engenharia de Sistemas e Computação (PESC/COPPE)
Universidade Federal do Rio de Janeiro (UFRJ)
Rio de Janeiro – RJ – Brasil

{sampaio,lusquino,felipe,priscilamvl}@cos.ufrj.br

Resumo. *A preocupação com a segurança e a integridade dos dados em sistemas de computação, incluindo áreas importantes como Internet das Coisas e Indústria 4.0, estão crescendo dramaticamente. Dessa forma, a existência de um malware pode ameaçar o bom funcionamento de sistemas inteiros, trazendo consequências irreversíveis. Este trabalho visa utilizar redes neurais sem peso para detecção estática e dinâmica de malwares. Na utilização de técnicas estáticas baseadas na imagem 2D do arquivo binário e de técnicas dinâmicas baseadas em API Calls, a rede WiSARD mostrou resultados próximos de técnicas do estado da arte utilizando redes neurais com peso, porém com tempos de treinamento e classificação uma ordem de grandeza menor.*

Abstract. *Concerns with security and integrity of data in computer systems, including important areas such as Internet of Things and Industry 4.0, are dramatically increasing. Therefore, the existence of malware can threaten the smooth functioning of whole systems, bringing irreversible consequences. This work aims to use weightless neural networks for static and dynamic detection of malwares. Using static techniques based on the 2D image of binary files and dynamic techniques based on API Calls, the WiSARD network showed results close to state-of-the-art techniques using convolutional neural networks, but shorter training and classification times one lower order of magnitude. The method presented shows the WiSARD as a faster alternative in detecting malware.*

1. Introdução

Malwares, ou *softwares* maliciosos, são programas que buscam perturbar as operações de um sistema computadorizado por meio de acesso não autorizado para conseguir informações sensíveis, ameaçando usuários [Aycock 2006]. Esses programas podem comprometer a integridade das informações nesse sistema, de forma que a detecção de *malwares*, ou seja, técnicas para identificar se um programa é malicioso ou não, é alvo de pesquisadores, já que novos *malwares* constantemente são lançados e as técnicas devem acompanhar tais mudanças [Bazrafshan et al. 2013].

Com a criação constante de novos programas maliciosos, além dos métodos tradicionais de detecção de *malwares*, como *signature-based methods* e *behavior-based methods*, métodos baseados em heurística, utilizando técnicas de aprendizado de máquinas, como Naïve Bayes [Schultz et al. 2001] começaram a ser explorados