Master of Science in Computer Science
June 2021

Comparing state-of-the-art machine learning malware detection methods on Windows

Filip Ahlgren

Faculty of Computing, Blekinge Institute of Technology, 371 79 Karlskrona, Sweden

This thesis is submitted to the Faculty of Computing at Blekinge Institute of Technology in partial fulfilment of the requirements for the degree of Master of Science in Computer Science. The thesis is equivalent to 20 weeks of full time studies.

The authors declare that they are the sole authors of this thesis and that they have not used any sources other than those listed in the bibliography and identified as references. They further declare that they have not submitted this thesis at any other institution to obtain a degree.

Contact Information:
Author(s):
Filip Ahlgren
E-mail: fiaa16@student.bth.se

University advisor:
Emiliano Casalicchio, Ph.D
Department of Computer Science and Engineering Department

Faculty of Computing
Blekinge Institute of Technology
SE–371 79 Karlskrona, Sweden

Internet : www.bth.se
Phone : +46 455 38 50 00
Fax : +46 455 38 50 57

Abstract

Background. Malware has been a major issue for years and old signature scanning methods for detecting malware are outdated and can be bypassed by most advanced malware. With the help of machine learning, patterns of malware behavior and structure can be learned to detect the more advanced threats that are active today.
Objectives. In this thesis, research to find state-of-the-art machine learning methods to detect malware is proposed. A dataset collection method will be found in research to be used in an experiment. Three selected methods will be re-implemented for an experiment to compare which has the best performance. All three algorithms will be trained and tested on the same dataset.
Methods. A literature review with the snowballing technique was proposed to find the state-of-the-art detection methods. The malware was collected through the malware database VirusShare and the total number of samples was 14924. The algorithms were re-implemented, trained, tested, and compared by accuracy, true positive, true negative, false positive, and false negative.
Results. The results showed that the best performing research available are image detection, N-Gram combined with meta-data and Function Call Graphs. However, a new method was proposed called Running Window Entropy which does not have a lot of research about it and still can achieve decent accuracy. The selected methods for comparison were image detection, N-Gram, and Running Window Entropy where the results show they had an accuracy of 94.64%, 96.45%, and 93.71% respectively.
Conclusions. On this dataset, it showed that the N-Gram had the best performance of all three methods. The other two methods showed that, depending on the use case, either can be applicable.

Keywords: Malware, Machine Learning, Static Analysis

Acknowledgments

I would like to thank Dr. Emiliano Casalicchio for being the supervisor for this thesis and providing feedback on the work and writing process.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Malicious software or malware, is an application made with the intent of damaging a target. This can be through extracting sensitive information, holding files for a ransom or being used as a zombie in a Botnet. The range of the term "malware" is quite large, however, most of the applications has a lot in common. Every malicious application has a payload, which is the code that will perform the malicious action on the system. Even if two applications are from two different malware families, they can still contain the same payload.

Ever since the first malware was released back in 1988 called The Morris Worm, malware has been a large issue and the number of new malware samples collected every year is growing[1]. The initial response to this problem was to use signature scanning in order to detect if a program has previously been detected as malware. As times goes on, not only are more unseen malware created, but the malware evolves to bypass this simple solution through obfuscation and polymorphism[37]. As the malware nowadays can with ease bypass this outdated method, new solutions based on structure and behaviour has to be adopted. This is where machine learning can be useful. With an almost infinite supply of malware, features can be selected in order to extract important behaviour characteristics and train a classifier.

Various machine learning methods has been proposed over the years, most commonly using deep learning and supervised learning. Common deep learning methods usually use Convolution Neural Networks or Artificial Neural Networks. While these can be effective, they take a long time to train and requires large datasets to do so. An alternative is to use Supervised Learning with classifiers such as Support Vector Machine, Random Forest or Gradient Boosting. These algorithms are faster to train, and therefore a lot of different solutions can be tested to see which performs the best. There is no right or wrong when selecting a classifier[13], different methods will perform different depending on the approach and features.

This study proposed a comparison between three state-of-the-art machine learning methods for malware detection that is found through a literature review. As most papers does not include code or datasets used in the implementation, papers with a good description of all components are required. These methods will be re-implemented and compared against each other in an experiment to see which has the best performance in terms of accuracy, true positives, false positives, true negatives and false negatives. The dataset used to train all three classifiers are privately collected and includes a total of 14924 samples, where 7269 are malware from 496 different families and 7655 are benign.

1.1 Problem Statement

Using machine learning on malware in not easy and has been done for many years. And due to the complexity of malware there is constant race between security researchers and malware creators. Because of this, research in this field will always be relevant. Even if researchers come up with a new method that detects all malware, malware creators will eventually find a way to bypass the new method.

IBM Security posted a report[3] where they stated that the average cost of a security breach is $3.86 million and it takes on average 280 days to identify and contain the breach. This is not only big corporations that can easily afford this cost, but the most targeted industry is healthcare.

The work proposed in this thesis is necessary due to the overwhelming information regarding malware classification. Many methods are proposed and tested on their own private dataset, making it hard for other researchers to validate their results. Also, these methods are highly optimized for that dataset in order to achieve the highest possible accuracy. Meaning, it is a possibility that a well performing method will not be as effective in the real world. Therefore, three state-of-the-art malware detection methods are selected to be compared on the same dataset, to see if they can achieve good performance on a dataset they were not designed to be trained on.

1.2 Aim and Objectives

The aim of this study is to find state-of-the-art machine learning methods to detect malware on the Windows operating system. As Windows specific software are often in form of portable executables(PE) files, these will be collected for the dataset. These methods will be re-implemented and trained on a privately collected dataset in order to find which method has the best performance. To do this, the following objectives are followed:

- Research what state-of-the-art machine learning methods are available and what datasets and data collection methods are used in recent research.
- Re-implement and train the three best methods from the first objective.
- Collect a large sample of malware and benign Portable Executable files to test the malware detection algorithms performance.
- Compare the different algorithms to see which has the best performance.

In the following section the research questions will be presented. The first objective will answer research question one and two, while the rest of the objectives will answer research question three.

1.3 Research Questions

The research questions for this thesis are:

- What state-of-the-art machine learning methods and algorithms are used for malware detection on Windows operating system?
- What collection methods or datasets are available to test performance of the algorithms?
- Which detection methods from RQ1 has the best performance when detecting malware on Windows operating system?

1.3.1 Hypothesis

To successfully answer the third research question, it is necessary to state a null hypothesis in order to correctly determine if there is a difference in performance. Therefore, the following null hypothesis will state:

H_0: There is not a statistical significant difference in performance when making the comparison for RQ3.

1.4 Scope

Malware can be found on almost any device, such as PC, MAC, IoT, Android and more. To find any meaningful results, the scope of this study has to be narrowed down to only malware for the Windows operating system. Also, portable executable(PE) files will be the only file type looked at. The majority of these files are ".exe" and ".dll".

As for the algorithms, since a requirement was to include enough detail on how to implement the algorithm, which is discussed further in chapter 4, the study will only include static analysis methods as they tend to be more specific in details.

1.5 Overview

The rest of this thesis will be structured as follows. The Background chapter will discuss some of the basic concepts and terminology related to malware detection. This is followed by Related Work, where solutions made by other researchers are presented and what research gap is found. Thereafter, the Method chapter will discuss the approach taken when conducting this study. Then the results will be presented in the Results and Analysis and discussed in the Discussion chapter. Finally, a conclusion of the study where the content is summarized and what future work is possible.

Chapter 2

Background

2.1 Analysis Methods

In malware analysis, there are two different ways a malicious application can be analyzed. Static analysis or dynamic analysis, and each have their advantages and disadvantages. That is why it is common to use both to complement each other when analyzing malware. However, this study will focus on static analysis since this is the more time-efficient method, and the reasoning is described in more detail in chapter 4.

2.1.1 Static Analysis

Static analysis is when the content of the malware is examined without executing it. This is done by looking at the source code, often in a disassembler if it is a portable executable(PE). With this, important strings can be identified such as filenames, comments, print statements, where the harmful blocks of code can be found. The code can be read to see what files are created, API calls are made to the operating system, and if network sockets are created, as an example. During static analysis, it is important to know if the code is obfuscated in some way, which will be described more in detail in the following section. This is a way of hiding malicious content in a file by encrypting or packing the payload, and if static analysis is run on this file without unpacking it, the content will make little to no sense.

The advantages of using static analysis are with the help of tools, an overview of the malware can be formed and malicious code snippets can be found quickly. It is also very helpful to get an understanding of how the malware is trying to hide from detection. Examples of tools could be IDA pro, PEid, VirusTotal, and Ghidra.

The limitations of static analysis are that it can be difficult to get an overview of the malware. For example, if ransomware was run with dynamic analysis you would get direct feedback that it is in fact ransomware. Because with ransomware, all files on the computer will be encrypted and a ransom note will appear or even lock the desktop. However, doing this through static analysis would take longer since small pieces of information has to be gathered to finally get a bigger picture.

Using this in combination with machine learning can be extremely effective. Techniques such as string extraction, PE header information, and N-Grams can find patterns in malware other methods cannot. However, most malware are packed, encrypted, or are using some type of obfuscation. If this is not treated with caution, it can make the analysis process more difficult.

2.1.2 Dynamic Analysis

Dynamic analysis is when the application is analyzed when executed. To do this, tools are made to capture calls made to the operating system, network traffic sent, running processes, created files, and much more. This can give direct feedback on how malware is functioning and in what order events occur. Examples of tools could be Process Hacker, Wireshark, or Process monitor.

The advantages of running the application are that it is easier to understand the bigger picture. The tools will capture in real-time what files are created, what processes are spawned, and what is changed in the Windows registry. Also, packed malware is no concern due to it unpacking itself when executing. This makes it harder for the malware to hide the malicious actions.

The disadvantages are that it can be time-consuming and intensive to run a virtual machine for a malware sample. For example, some malware can sleep for a few seconds or minutes before running, making it necessary to wait long periods for the malware to run the payload.

2.2 Anti-malware Evasion Techniques

Over the years, malware has adapted to the various detection techniques developed. I. You et al.[37], made a summary of some popular methods that malware use to evade these techniques. Some of these will be discussed in this section since this has to be considered when creating a dataset of malware for static analysis.

Encryption is a common way of evading anti-malware scanners. When the application is run, the en encrypted copy of the malware is extracted from within the malware and later executed. Every time the malware is distributed to a target, the decryption key is changed, making it hard to automatically unpack the malicious code.

Polymorphic malware will change the content of the file every time it mutates. This type of mutation will only change the decryption routine used to extract the malicious code.

Metamorphic is an extension of the polymorphic malware, where not only is the decryption routine change but also the content of the malicious code. Meaning that if an anti-malware tool successfully decrypts the code, it still does not have the same signature as other versions of the malware.

2.2.1 Obfuscation

Obfuscation[37] of malware is the technique of hiding the content of the payload through encoding or reordering bytes. This is can be used in combination with the evasion methods mentioned previously.

Dead-Code Insertion means code that will never be executed is added to the final binary file. A common way of doing this is to insert the NOP instruction between bytes.

Instruction Substitution is a way of replacing instructions with alternatives to create a different signature while not modifying the outcome of the program. As an

example, making a jump condition that stores the compared variables in different registers, or even making the comparison with an external library.

Code Transposition is a method of moving code around. The code is divided into multiple blocks and thereafter the position of these block are moved around, creating a new signature while still having the execution path identical.

2.2.2 Packers

When computers first started to become popular and disk space was a big issue for most people, packers were introduced to reduce the size of applications. Later, when malware started infecting computers and anti-viruses started scanning for signatures, they adapted the packing[34] technique to change the signature of the file. This was one of the ways malware started to obfuscate itself. Packers are simple to use and almost anyone can make a packed executable, some popular options are UPX, ExeStealth, Morphine, and FSG. However, this is often defeated by static analysis. But for anti-malware software and machine learning algorithms, this can have a big impact on performance, since a packed ransomware file will not match the unpacked ransomware files.

2.2.3 Machine Learning Process

To make a working malware detection algorithm with machine learning, there are some terminology and basic steps that are required in most cases[15]. In this section, these concepts will briefly be described.

The first step is to create a dataset. A dataset is a collection of data on which the machine learning algorithm will train and be tested on. The dataset will be divided into two parts, training and testing, where training will be around 70% of the dataset and testing will be 30%. However, if a deep learning algorithm is used, taking a part of the training data for validation is very common. With the validation set, it is visible during training how the model performs on unseen data and the algorithm can make corrections based on this.

The second step is to extract features. Features are what values the machine learning algorithm will train and predict with. For example, if a classifier should predict malware, one of the features could be the names of the imported functions visible in the portable execution header.

The third step is to train a model or a classifier with the extracted features. With each classifier, parameters related to how the classifiers should train on the data are referred to as hyperparameters. These parameters are values such as learning rate and the maximum number of nodes.

The fourth step is to test the classifier. In the first step, the dataset was divided into two or three different parts, where 30% of all samples were put into the testing dataset. These samples have never been seen by the classifier and which means the accuracy can be tested by predicting, and then controlling if the prediction was correct by looking at the correct label.

2.3 Supervised Classifiers

Supervised classifiers[13] are one of the simpler ways of using a classifier. The input is features extracted from a large collection of data and labels assigned to each feature row. When these are trained, it is then possible to predict what label unseen features have. In this section, supervised classifiers used in the final experiment will be presented and discussed.

Decision Tree
A decision tree(DT) creates a simple tree-like structure where the nodes in the tree are true or false if statements and the leaves are the final predictions. When a new sample is presented, the features will go through the if statements and finally find the end node which is the prediction.

Random forest
Random forest(RF) is an ensemble machine learning algorithm, which is a classification method based on multiple machine learning algorithms to improve the ability to make predictions. This algorithm uses multiple Decision Trees to fix the issues DT usually has due to overfitting the dataset.

Gradient Boosting
Gradient boosting(GB) is much like Random forest since both are created to improve the weaker machine learning algorithm Decision Tree. However, instead of using multiple decision trees, it uses regression to estimate relations between dependent and independent variables.

Support Vector Machine
Support Vector Machine(SVM) places the samples on an x-dimensional axis. If the samples have two dimensions, a 2D grid will be created, and if the features of a sample can represent the labels in a good way, each class should be grouped together. Then a line will divide the samples and depending on which side of the line a new sample is placed, the prediction will change.

K-Nearest Neighbors
K-Nearest Neighbors(KNN) is much like SVM, where samples are placed on an x-dimensional axis. However, instead of drawing a line between the sample groups, the distance of K adjacent samples will determine the prediction.

Naive Bayes
Naive Bayes(NB) calculates the probability that a given feature belongs to a label. For example, if a malware sample has the "stdio" library imported in the PE Header, the algorithm calculates how probable it is for a malware sample to have that library imported. With the combination of multiple probability calculations, a prediction can be made.

Nearest Centroid
Nearest Centroid(NC) will put the mean of all classes on a grid, which is called the Centroid. When a sample has to be predicted, the distance between centroids are calculated and the closest will be the final prediction.

2.4 Neural Networks

A neural network[30] is an alternative way of predicting classes compared to supervised learning. This network is inspired by the neurons in the human brain which gives it a possibility to learn more complex patterns in data. The network is divided into three different components, input layer, hidden layer, and output layer, as seen in figure 2.1. The input layer is composed of nodes, and each node will take one value as input. This means that if a feature row has 256 values, there will be 256 input nodes in the layer. The hidden layer is composed of multiple columns of nodes, and this will do the most work in the calculation. Each node in a column is connected to all the nodes in the next column, and with each connection, there is a weight assigned to it. All the weights are then summarized and multiplied by the input, then a bias is added. The calculation can be seen as following, where b is bias, x is input, and w is weight:

$$b + \sum_{i=1}^{n} x_i w_i$$

If the value is below a set threshold, the node will not send the output value to the next nodes. If it is above, the output will be sent to the nodes in the next column and the process will repeat. The output layer will then receive the final values and make a prediction based on what output node has the highest value. During the training phase, depending on how wrong the prediction is, the weights and baizes will be changed for the nodes in the hidden layer.

2.5 Dataset

A dataset is a large collection of data collected to train a classifier and make predictions from it. For malware detection, datasets are often collected from malware databases such as VirusShare[9], VX heaven[11] and Malshare[6]. However, malware found on these sites can be outdated and when malware is changing all the time, it is important to train classifiers on newer samples. Another alternative is to capture live malware with a honeypot. This is an intentionally vulnerable server that is made to discover breaches in a system, or in this case to capture the malware planted on the system. This can however be a very time-consuming task, and it requires constant attention.

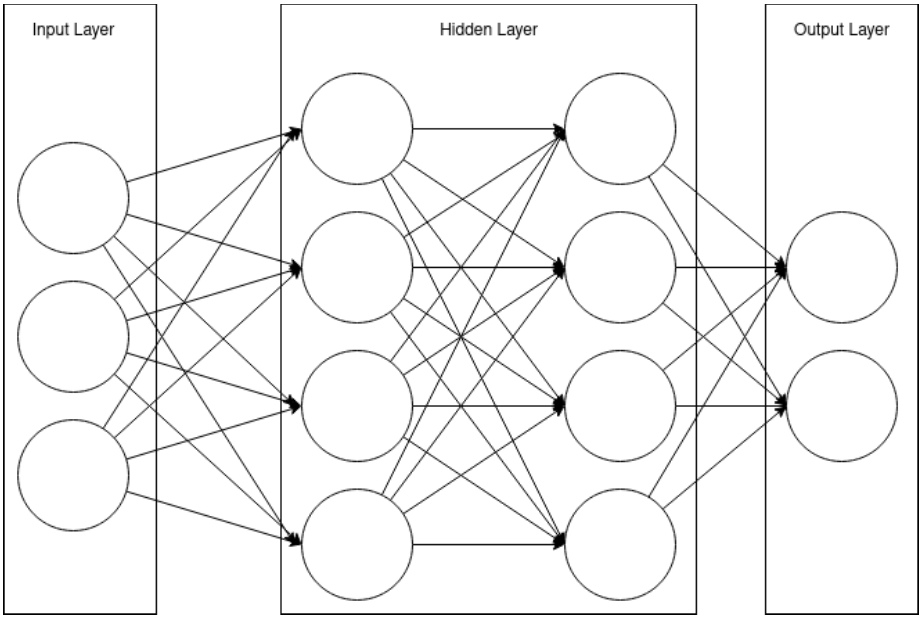

Figure 2.1: Neural Network structure

2.5.1 Balancing

When collecting samples for a dataset, it is generally advisable to collect the same amount of samples for each prediction class to avoid overfitting. If a class has 100 samples versus another class with 1 sample, it is more likely that the classifier will choose the larger class since it has more data from it. Also, in this case, where malware is collected, it would be beneficial to have the same amount of samples for each malware family even though only two labels are used, malware and benign.

2.5.2 Public and Private Datasets

When selecting a dataset, an important consideration is to capture the data from scratch or using a publicly available dataset.

A public dataset is when researchers, companies, or online challenges, release a dataset they want others to use to help the company or complete the challenge. This comes incredibly handy when researchers want to use these datasets for comparisons against other proposed methods. Then a standard dataset can be used and the best performing method is very apparent. However, while this is good for comparison, it can also create bias on that dataset. When researchers come up with new methods, and they have one specific dataset in mind, they can train their models around it, making it less useful in the real world. Also, many of these datasets are outdated and are rarely updated. Meaning that a classifier that performs well on one public dataset may not perform that well in the real world since most malware now has changed their structure. Some datasets also have pre-extracted features where string, byte sequences, and other data are already in a feature row. If this method would be used in the real world, these features have to be extracted in the same way, also packed and encrypted malware as described above may introduce problems if not handled correctly.

A private dataset is when the author of the paper uses the different malware collection methods described above. This will make sure the data is not as outdated as the publicly available ones. As described above, collection methods that include malware from databases such as VirusShare[9] can be outdated. However, to compare the Microsoft malware challenge[29] with the data collected from VirusShare, the Microsoft dataset is from 2015 and VirusShare's samples are from 2020 and back.

A private dataset can in some cases have a better balance of malware families, depending on how the data is collected. While this is not always the case, public datasets can have a very big difference in class sizes. The public dataset Malimg[25], which is commonly used for image detection, has a large variation in class size. The malware family "Allaple.A" has 2319 samples while "Skintrim.N"only has 59 samples.

2.6 Selected methods

In this paper, three malware detection methods are selected through a literature review. This section will describe these methods in more detail to help to understand the final results.

2.6.1 Image Detection

The paper by D. Vasan et al.[31], was selected as an example for the image detection method during the literature review. The main idea is to convert malware binary files into images. The process of this is kept very simple since both instructions and pixel values are one byte. After the conversion, the images can look like the samples in figure 2.2. As seen in the figure, not all images are the same aspect ratio, this can easily be solved by converting the images into 224x224 pixels which are required for the algorithm to work.

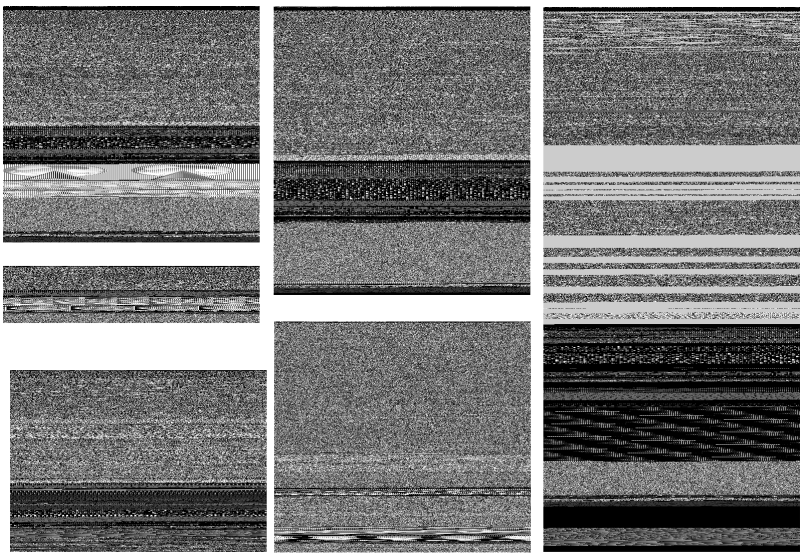

Figure 2.2: Malware samples converted into images

Figure 2.3, is an overview of the entire process of detecting malware with the proposed method. Starting with the VGG16 model, which is a Convolutional Neural Network(CNN) model previously trained on thousands of images from Imagenet, and it can go in two directions. First is towards the Softmax Classifier, this is the last layer in the CNN and is the standard way of a VGG16 model to predict the input layers. The second direction is to the PCA(Principal component analysis) feature reduction. Here, the model is reduced by making the FC2 layer (The second Dense layer in figure 2.4) output a 4096-dimensional vector, which in return are turned into a 410-dimensional vector with the PCA. Meaning that only the most important features are selected from the feature row to reduce the size for performance reasons. This feature row is sent to the top "One-vs-All Multi-class SVM", which is a Support Vector Machine(SVM) variant.

The other model is called ResNet50 and as seen in figure 2.3, this also has two directions. Again, to the Softmax classifier, which is the last layer in the ResNet50 model and the default way of making predictions with the ResNet50 model. The other direction is where a 2048-dimensional vector is extracted from the avg_pool layer(The third to last layer as seen in figure 2.5) and then reduced with PCA to get a 205-dimensional vector, which is then sent to the "One-vs-All Multi-class SVM" in the bottom.

The middle "One-vs-All Multi-class SVM" will get both the 410-dimensional vector from the VGG16 model and the 205-dimensional vector from the ResNet50 model. This will result in a concatenated 615-dimensional vector which the classifier will use as input.

With the two Softmax classifiers and the three Support Vector Machines, there is a total of five predictions being made. To combine these into one prediction, the values have to be fused with an algorithm. There are a few methods of doing this, where weights can be assigned to each classifier in case they are more accurate than others. However, the authors did not specify exactly what method was used in the paper, and that is why the most simple but yet effective method was used. By calculating the average of all values, it should give a good prediction involving all classifiers.

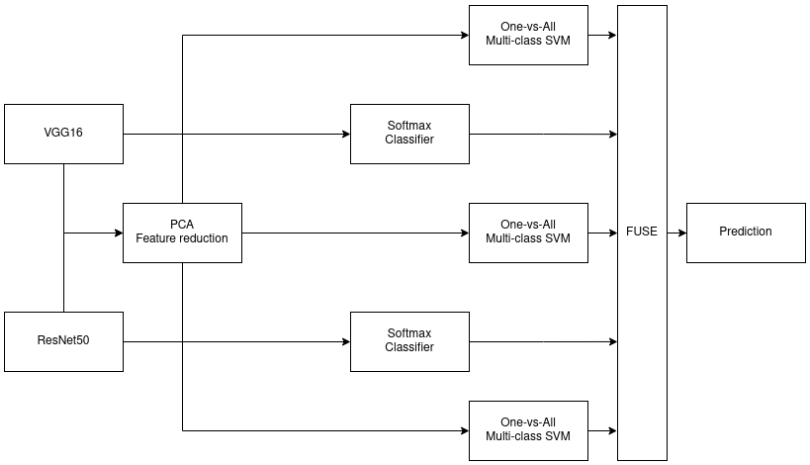

Figure 2.3: Overview of image detection

VGG16

The first model is called VGG16 and the idea is to use transfer learning to save time on training and the structure of the model can be found in figure 2.4. When the model first was created, it was a part of the yearly ImageNet competition, where a total of 1000 different objects has to be detected in a wide range of images. To make this work with malware, transfer learning is used, To do this, the first nodes are kept static while the last few nodes are trained again on the malware images. This is possible since the first layers will often work the same during image detection, regardless of what object or pattern is being detected.

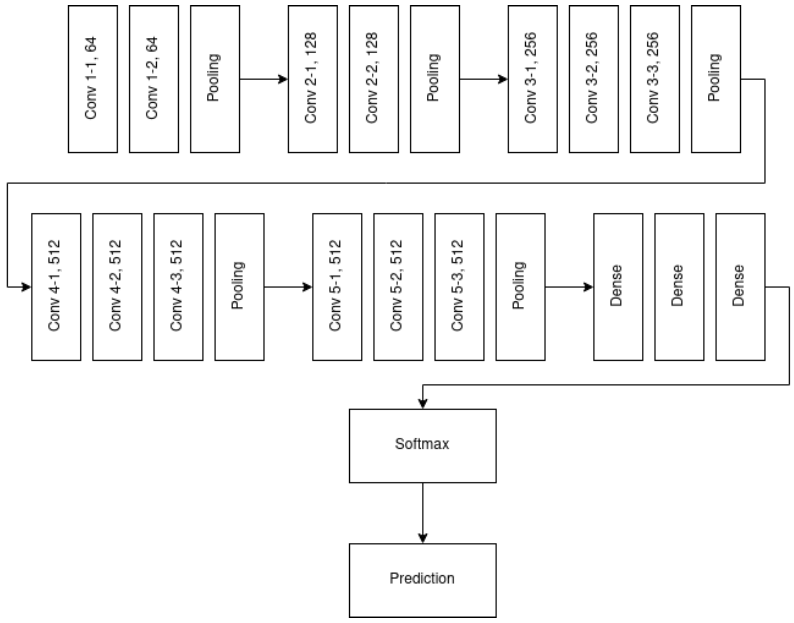

Figure 2.4: Overview of the VGG16 Model

ResNet50

The ResNet50 model as can be seen in figure 2.5, is much like the VGG16 model because it is also trained on object detection for images prior to this work. Compared to the VGG16 model which is incredibly slow, this will take less time to train and make predictions with, however, it is not as accurate.

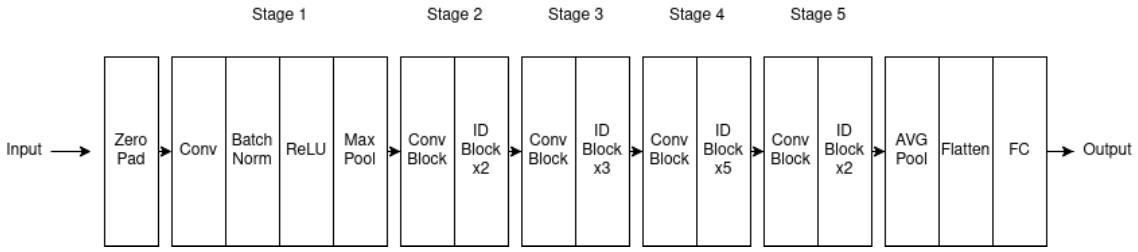

Figure 2.5: Overview of the ResNet50 Model

2.6.2 N-Gram

The paper by L. Yang et al.[35] proposed a method that extended the Malconv method[27]. The idea is to split the two different methods into two layers as seen in figure 2.6. The first layer will be the standard Malconv method, which is the faster of these two, however, it is not as accurate. The Malconv model will output a number between 1 and 0 of how sure it is the sample file is malware. Zero meaning it is benign and 1 meaning it is malware. Since the Malconv model is not as accurate compared to the other method, the output number has to be below the threshold of 0.000005 or above 0.99 to make the classification. Otherwise, the next layer will have to extract more features to make the final prediction.

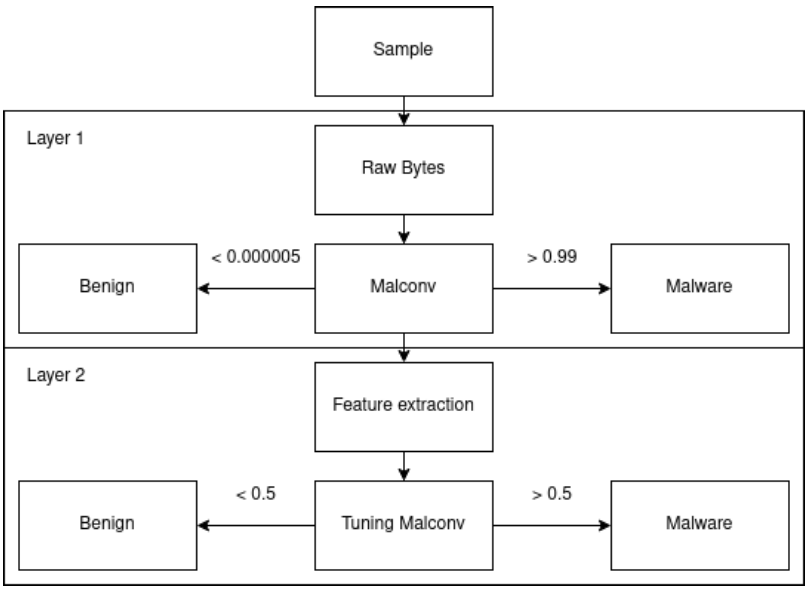

Figure 2.6: Overview of TuningMalconv

The TuningMalconv model starts with extracting 4 different features from the binary file.

- **Fixed Byte Embedding**, is the same method Malconv[27] used as detection in the first layer.
- **N-Grams Of Code Bytes**, is a common method of finding context in text from the field Natural Language Processing(NLP). In this case, the 512 most common byte sequences were used as features after the bytes were converted into 2-grams.
- **PE Imports and Section Names** were extracted and converted into a numerical vector using a HashingVecotizer from sklearn. In addition to this, the number of section names was appended to the end of the vector resulting in a total vector with 513 dimensions, where section names and imports are 256 each.
- Null-terminated **String Patterns** with 5 being the minimum amount of characters are extracted from the raw bytes using regex. The strings are then converted into a 256-dimensional numerical vector using hash trick.

When all features are extracted, a classifier will make a prediction on how certain it is the sample file is malware or not. Since this is the last layer, if the prediction is below or equal to 0.5 the file is benign, and above 0.5 the file is malware. In the proposed study there were a few different classifiers tested and they found that the best was Gradient Boosting.

2.6.3 Running Window Entropy

The third and final model was proposed by K. Jones[21], and the idea is to classify malware based on Window Entropy. As the implementation was available on Github[5], the need to worry about implementation errors was smaller. An overview of the method can be found in figure 2.7.

The process starts with gathering malware from Virus Total[10], which can be done either by keywords or random samples. However, since a dataset is already collected, this part can be skipped.

The Running Window Entropy(RWE) features will then be extracted. The script allows any value for Window size and number of datapoints. Depending on the chosen value, the features row will have different lengths.

From the collected dataset, the VirusTotal API has to be used to classify which malware family the malware file belongs to. The response from the API will then be processed and through a combination of multiple anti-malware scanners, a suitable family will be selected.

To make sure all malware samples are scanned with the VirusTotal API, a hash list is created and any missed sample will be excluded from the final dataset.

The classifiers can then be trained with a wide number of classifiers, which is incredibly useful to test which has the best performance on the created dataset. The authors found that the Random Forest classifier performed the best for their dataset.

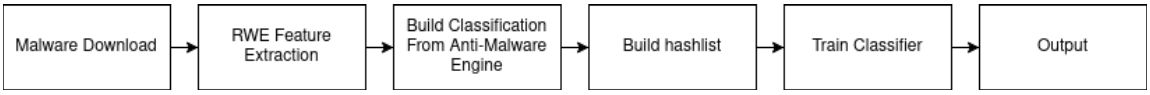

Figure 2.7: Overview of Running Window Entropy Method

Entropy Definition

Entropy measures how much useful information can be found in data[22]. If a low value is presented, the data is predictable and is seen as less useful. If a high value is presented, the data is seen as unpredictable and therefore is seen as more useful. For example, if an entire section of a binary file has a section filled with NOP instructions or Zero bytes, the entropy value will be less. This is why entropy can be useful to see if a binary file is encrypted or packed[24]. If some data is in "plain-text", the entropy value will not be too high. But if the data is encrypted, there will be less structure and more random noise which will lead to a higher entropy value.

Chapter 3

Related Work

In malware analysis, there are two major analysis methods which are static and dynamic analysis. This chapter will explain in detail the research that has been done in the two fields and how it relates to this thesis. Also, methods such as image analysis, N-gram, and running window entropy will be looked at since they are used in this study and fall under the static analysis category. Lastly, publicly available datasets will be discussed to see which is the most used by other researchers.

3.1 Static Analysis

Static analysis is a type of method commonly used when analyzing files to verify if it is malicious or not. It often involves looking through the available code, used imports, and function calls. This can be transported into machine learning algorithms with various techniques that will be discussed further in this section.

M. Hassen et al.[18] proposed a method improving the Function Call Graph(FCG) method. A Function Call Graph is a way of representing relations between functions, and by mapping out how each function in the malware relates to each other, a vector can be created with features to train a machine learning algorithm. Since functions are unlabeled, it is hard to compare two graphs. Their solution to this problem is to give them cluster-ids with the help of Locality Sensitive Hashing. They used the dataset Microsoft Malware Classification Challenge[29] and were able to achieve an accuracy of 97.9% when classifying different malware types.

Y. Zhang et al.[38] recognized that most malware solutions can only detect malware families that are included in the training data. The proposed solution to this issue is to use a soft relevance value based on multiple trained models. There are six different subsets of features and each feature subset corresponds to one model. These features include information such as file sizes, dll and function call names, n-grams and so on. When the models are trained, they will try to predict which malware family from the dataset they belong to. From the results of the trained models, the soft relevance value is used to find if the malware belongs to one of the original malware families or not. With this approach, the authors achieved an accuracy of 99.8% on the Microsoft challenge dataset[29].

3.1.1 Image Analysis

A popular method of static analysis is image analysis. The idea is to take the structure of a binary file and convert it into a greyscale image which can be used to

find patterns with object detection techniques from other fields. D. Vasan et al.[31] used two popular image detection models called VGG16 and ResNet50 with transfer learning in order to achieve 99% accuracy on the Malimg[25] dataset. This means they took the algorithms that have been trained on detecting objects in images, then by only re-training the last nodes in the neural network, they can use it to detect patterns in the malware files converted to images.

R. Vinayakumar et al.[32] proposed a method where Convolutional Neural Networks were used, which is a neural network that has been getting a lot of attention over the last couple of years since it is very efficient at object detection. In this study, two different datasets were used in order to make it more robust while it also shows that there can be a difference in accuracy between datasets. The first dataset was the public Malimg[25] dataset which had an accuracy of 93.3% while the second dataset which was privately collected had an accuracy of 96.3%.

S. Ni et al.[26] used a different approach when extracting images. They use SimHash encoding, and it is a way of turning similar strings into similar hashes, which is then used on the binary data before turning it into a greyscaled image. Then a standard Convolutional Neural Network was trained to make the classification. The authors were able to achieve an accuracy of 98.8% on a privately collected dataset.

3.1.2 N-Gram

N-gram is a commonly used method to capture the structure of a sentence in Natural Language Processing. Over the years, this method has become a popular method for analyzing bytecode to detect malware, since binary applications are also written in a structural language. L. Yang et al.[35] proposed an extension of the Malconv algorithm[27], where two detection methods were divided into two layers to improve performance and accuracy. In the first layer, an implementation of the Malconv algorithm was used since it has fast detection time, however, it is not as reliable as the second layer. The second layer will only be used if the first layer is not very sure that the given file is malware or not. The second layer is a combination of n-gram and metadata analysis, and is slower due to the time to extract all features, however, it is more accurate when detecting malware. With this method, the authors achieved an accuracy of 98% on a large privately collected dataset.

M. Chowdhury et al.[16] used the n-gram approach on PE(Portable Executable) files. They used 5 grams and information from the PE header as features, then PCA (Principal Components Analysis) was used for feature reduction to only focus on the impactful values. To classify the malware an artificial neural network was used and it achieved an accuracy of 97.7% on a privately collected dataset.

3.1.3 Running Window Entropy

K. Jones et al.[21] proposed an algorithm that was based on Running Window Entropy(RWE). Window Entropy is a way of measuring how much useful information can be found in a set of data. A high number can indicate that the data is random or unpredictable, which means that the data is useful, while a low number can indicate that the data is predictable and has little randomness to it. With a private dataset, the authors achieved an accuracy of 95% when classifying malware types.

Entropy is not often used in malware classification and therefore not many papers are published. The more common use case for Entropy is to detect packed or encrypted malware. This was done by R. Lyda et al.[22], and with Entropy analysis they were able to quickly detect if a malware sample was packed or encrypted. It showed that plain text and plain executable files had an entropy of 4-5, while packed or encrypted files had an entropy of 6.8-7.

3.2 Dynamic Analysis

This paper mostly focuses on static analysis, however, dynamic analysis can also be used to generate features for machine learning classifiers. Dynamic analysis will gather information about what the malware does when executed. Common features to extract is API calls made, network packets sent and any event the executable might spawn.

In the past, there have been issues with applying spectral based graph neural networks to directed graphs. Z. Zhang et al.[40] proposed a solution to this problem named Spectral-based Directed Graph Network(SDGNet). With this approach, the authors claim to have a better than the state-of-the-art performance with an accuracy of 97.3% on a privately collected dataset.

M. Rhode et al.[28] saw the issue that dynamic analysis can take up to 5 minutes to get enough information for a good prediction. They proposed a solution that would take snapshots of the running malware every second, and within 5 seconds of execution, a prediction can be made. Since this solution does not have much time to capture API calls, which is a common feature in dynamic analysis, machine activity is therefore collected instead. The authors are classifying the data with a recurrent neural network and can achieve an accuracy of 96% on a privately collected dataset.

R. Agrawal et al.[12] proposed a method looking at emulation event sequence with a maximum of 200 events to train a neural network. They state that the events that cause the malicious action is either grouped or scattered across the event sequence, and therefore it is important to both have the events present as well as keeping the correlation between events. The trained models in the paper are Long Short Term Memory and Convolutional Neural Network, which are commonly used models for malware detection. The authors achieved an accuracy of 95.6% with a privately collected dataset.

3.3 Datasets

Datasets for machine learning can either be privately collected or publicly available for anyone. This section will describe what popular datasets are available and are commonly used in research.

In 2015 Microsoft announced a malware classification challenge called “The Microsoft Classification Challange”[29] with a dataset of 0.5 terabyte data. This dataset became widely popular amongst researchers and is a commonly used dataset to compare malware classification methods.

Ember[14] is a large publicly available dataset with static features extracted from

PE files. The dataset includes 1.1 million features from binary files and can easily be used when testing and comparing new malware detection techniques.

Malimg[25] is a image dataset converted from malicious binary files. This collection of grey-scaled images has 9,458 samples from 25 different malware families and is commonly used by researchers when comparing image detection on malware samples.

3.4 Comparison

To conclude the literature review, 20 papers were selected out of all reviewed papers based on their performance and replicability. From these papers, a few interesting methods were found with high performance and are presented in table 3.1. The chosen papers are only static analysis since it is the analysis technique this study focuses on.

As seen in figure 3.1, the best performing algorithms seem to be image detection and Function Call Graphs since most of them have accuracy above 99%. While both the N-Gram methods and the RWE methods have papers with less accuracy. While this is very much dependent on the datasets collected, the presented accuracies show a small variation between all proposed methods. To find which method has the best performance, the results from the experiment need to be considered.

Method	Authors	Title	Accuracy
Image	R. Vinayakumar et al.[32]	Robust Intelligent Malware Detection Using Deep Learning	98.8%
Image	D. Vasan et al.[31]	Image-Based malware classification using ensemble of CNN architectures (IMCEC)	99.50%
Image	S. Ni et al.[26]	Malware identification using visualization images and deep learning	99.26%
N-Gram	L. Yang et al.[35]	TuningMalconv: Malware Detection With Not Just Raw Bytes	99.03%
N-Gram	M. Chowdhury et al.[16]	Protecting data from malware threats using machine learning technique	97.7%
FCG	G. Y. Zhang et al.[38]	The Classification and Detection of Malware Using Soft Relevance Evaluation	99.8%
FCG	M. Hassen et al.[18]	Scalable Function Call Graph-based Malware Classification	99.33%
RWE	K. Jones et al.[21]	Malgazer: An Automated Malware Classifier With Running Window Entropy And Machine Learning	95%

Table 3.1: Table of literature review examples

3.5 Research Gap

Throughout the literature review, it was apparent that a lot of research has been done in this field. With so many different methods proposed, it is hard to know which has the better performance. Even when researchers reference related works and compare the results, it is impossible to know if there is a difference in performance due to many uncontrolled variables. The biggest factor among these differences must be the dataset, as it can change the results of a method drastically.

Taking a step back and exploring the state-of-the-art methods, this thesis can help uncover the differences of the proposed methods and whether or not there actually is a significant difference between them.

Chapter 4

Method

The proposed methods for the study were a literature review and an experiment. The literature review was proposed to find necessary information about the field, and the chosen technique was Snowballing[33]. The review had multiple potential detection methods, whereas three were selected for the final experiment. Also, throughout the process of reviewing papers, the most common public datasets and malware collection methods were considered. This was then followed by the experiment, where the three detection methods were compared on the same dataset to see which had the better performance.

With the literature review, research question 1 and 2 can be answered, while research question 3 can be answered with the experiment. The reasoning behind choosing a literature review for finding information about the subject is because there are not many options to choose from. An alternative could be interviewing knowledgeable researchers. However, this could create a bias towards a certain field or paper and would not give a broad understanding of the entire field. As for choosing the Snowballing method, the reasoning is that the different possible methods that are available in this field were not known prior to the review. This means that if only a keyword search was done, some methods could have been left out because of the lack of knowledge in the domain. With the help of Snowballing, other researchers would refer to other methods in the related works section to give a much broader search.

The experiment method was chosen due to the need for control during testing. For a successful comparison between the three algorithms, dependent and independent variables have to be examined one by one in a controlled environment. Only then the most optimal result and the reasons can be found. If an alternative method was chosen, such as a case study, a different approach from the beginning is required. This could be finding an implemented machine learning algorithm from a company and have that examined. While this is possible, this does not necessarily answer the questions proposed in this study.

The rest of this chapter will go into details about how the work was set up. First, the literature review is discussed on how it was performed, then how the dataset was collected. Thereafter, the method used to implement all three algorithms is presented, and finally how the experiment are set up.

4.1 Literature Review

The goal of the literature review was to answer the first two research questions. The required information for this was to find state-of-the-art machine learning methods to detect malware and if any public available malware datasets are commonly used. For this review, the Snowballing[33] method was used to iterate research papers, as it allows for finding a better variety of research papers without knowing the keyword to search for as explained earlier in the chapter. The database used for the initial keyword search was Google Scholar[2] since it will give a wide variety of publication databases. Google scholar has been getting a lot of critique due to the overwhelming amount of papers and publications that are poorly written and should not be considered in an academic paper. However, this is considered during the review and only well-written papers from reputable sources are used with a very strict search query. Some of the included databases were ACM Digital Library, IEEE Explorer, Science Direct, and Springer Link, as an example.

Since the related work for this papers is the results from the literature review, the results are presented chapter 3.

4.1.1 Criteria

For a paper to be considered for the final experiment some criteria must be met. The following list of items were the rules for selecting a paper all the papers:

- **Date:** No papers older than 2017 should be included.
- **Database:** The paper should come from a database that is trustworthy and peer-reviewed before accepted.
- **Title:** Should be relevant and give a good idea that the content is about malware detection with machine learning.
- **Abstract:** Should provide information about if the paper proposed an algorithm that could be used for this study.
- **Results:** Will provide information about if the paper has good enough results to be included in this study.
- **Full-text:** Shows if the paper as a whole is valid based on used methods and replicability.

4.1.2 Review Process

The initial step for starting the Snowballing method was to use one keyword search in the Google Scholar database. The following query was used to get 281 results:

"machine learning" malware classification detection windows -survey -IoT -android -"OS X" -OSX -validating -Linux PE

From the first selection of only inspecting data and title, 59 papers were gathered. After this, further inspection of the papers was done with all the other criteria, to get the number of papers down to 32. The initial thought was to do forward and backward iterations on all papers until no more interesting papers are found. However, due to the time constraints and the large volume of papers, the max number of iterations done was 2. The main idea of this review was not to do a large review but to get an understanding of what work is done in the field and which methods have the best performance. From the total 116 papers reviewed, 20 of the most promising papers were selected for a final evaluation. Then the decision of only focusing on static analysis was made since most of the papers are written in a way that is more feasible to re-implement. Out of the 20 papers, three of the top-performing papers were selected with replicability in mind. The selected papers can not be the top-performing ones due to a lack of detail in those papers.

During the review of all papers, the choice of the dataset were also considered. A large amount of research has been done with Ember[14], the Microsoft challenge dataset[29] and Malimg[25], which is further described in the Related Work chapter.

4.2 Dataset

When selecting a dataset to use, publicly available datasets were considered. However, none of the commonly used datasets made it possible to use all three selected algorithms on it. The decision was then made to gather all malware samples from VirusShare[9]. With this method, the size of the dataset can be controlled, while also giving it a wide range of malware. While some datasets can give a small range of malware families to classify, this work requires malware that the classifiers have not seen before. This is because the classification made is only for malware or benign files and to have a small range of malware families will impact the final results negatively. It is required that the dataset has some real-world elements to it, and this can be done by introducing malware that is not commonly seen by these classifiers.

4.2.1 Collection

VirusShare provides large malware collections that can be downloaded and the required samples can be extracted. The used collections for this study can be seen in table 4.1. Since the collections do not only have the PE files wanted for this study, the required samples had to be sorted out with a custom script. The malware collections are from the year 2020 which means the sample selection is relatively new. The only possibility to get more up-to-date malware is to create a honeypot and gather the malware by hand. However, that is a very time-consuming task, which means VirusShare is a good option to get up-to-date malware samples from.

4.2.2 Sorting

To extract the Portable Executable files from the malware collection, a custom-made script in Python3 was used. This script used the PeFile[8] module to identify which files were Portable Executables and they were then extracted with the zip module.

Name	Size	Date
VirusShare_00389.zip	71.98 GB	2020-09-26
VirusShare_00386.zip	79.19 GB	2020-09-04
VirusShare_00385.zip	86.68 GB	2020-09-02
VirusShare_00378.zip	73.15 GB	2020-04-21

Table 4.1: Table of VirusShare collections

All Zip files were extracted at the same time and when a total of 10081 samples were collected, the collection was stopped. Meaning that the entirety of all collections are not used, only parts of them.

To ensure the samples are in fact malware, Metadefender[7] API was used to scan the PE files. After removing all files that were not recognized as malware, only 7284 samples remained.

To collect benign samples, a fresh installation of Windows 10 was created. Then with a custom script, 7737 Portable Executable files were extracted from the "C:\" drive. This means that the total collected malware samples were 15021. However, due to limitations in the Running Window Entropy algorithm, 97 samples had to be removed. This was because files over 30MB could crash to application due to overflow in Random Access Memory(RAM), and files under 1KB would get stuck in an infinite loop. This means the final dataset consisted of 14924 samples, where 7284 were malware and 7737 were benign.

The collection events can be seen in table 4.2.

Event	Malware samples	Benign samples	Total Samples
Malware extraction	10081	0	10081
Malware filtering	7284	0	7284
Benign extraction	7284	7737	15021
Algorithm limitations/ Final dataset	7269	7655	14924

Table 4.2: Table of number of dataset samples

4.2.3 Final Dataset

The final dataset had 7269 malware samples and 7655 benign samples, which is a total of 14924. Out of these 14924 samples, there were 496 different malware families. However, the number of malware families may not be too accurate since it is a guess made by the MetaDefender[7] API, but it should be within that range. The reason to keep so many malware families was so the classifiers would have some real-world training data, where not only families seen before are classified. Since not all malware families can be in a single dataset, unseen samples will be discovered in the real world and therefore even malware families with low amounts are kept. Also, since only two outcomes are possible, the important part is to keep those two balanced to avoid overfitting. While it would probably be good to avoid having families with few samples, it should be fine in this case, since the classifications are not made on

family types. In addition to this, there are 682 packed executables in the dataset. As mentioned, the reason for this is to have a realistic dataset where some malware can be packed. It is very common for malware to be packed[34], whether it is a known packer or not. Meaning that these algorithms have to be able to detect malware that are packed since not all packers can be detected. If only unpacked executables were included, unknown packers could bypass the detection. This is why 682 packed executables are included in the final dataset.

4.3 Implementation

This section will discuss the method of implementing the three algorithms and the deviations from the original implementation. The details of the algorithms have already been discussed in chapter 2, so only key parts of the implementations will be brought up in this section.

4.3.1 Image Detection

The image detection algorithm was implemented in Python 3.8.5 with help of the Keras 2.4.3 module. The Keras module has the ability to instantly create the VGG16 and ResNet50 models and also download the pre-trained dataset directly.

The implementation of the image detection algorithm was straightforward since the models used are pre-created where not much can change. However, there can be some variation in the hyperparameters used and the number of layers re-trained. As discussed in chapter 2, transfer learning was used in order to train the last layers in the model. However, in the original paper, the authors did not mention exactly how many layers are needed to be re-trained to get the same results. In this case, the VGG16 model was trained with 19 static layers out of 23, while the ResNet50 model was trained with 175 layers static out of 181. As for the hyperparameters, they are tweaked in the experiment to see which yields the best performance.

The limitations of this method are mostly the time it takes to train both models. The final model took over 8 hours to train and 1 hour to make predictions. While some more parameters could have been tested, it came down to time limitations for this algorithm.

4.3.2 N-Gram

The N-Gram approach had two layers. The first layer was the Malconv[35] algorithm which was implemented with the help of a community submission from GitHub by "EndGameInc"[4]. The second layer was implemented with the help of the SciKit-Learn module version 0.24.1, while the code was written in Python 3.8.5.

For this approach, an alteration had to be made to the method to make it work. As the authors stated in the original paper and mentioned in chapter 2, the PE imports and Section Names used the HashingVecorizer from sklearn to create a vector of the list of names. However, with a lack of knowledge in the domain and the lack of explanation of how to deal with this class, hash tricking was used instead. The idea

behind both methods is to create a vector of integers from the strings, so making this switch should not impact the performance too much.

4.3.3 Running Window Entropy

The proposed method for Running Window Entropy submitted the code along with the paper, meaning the source code was available. Like the other methods, this was implemented with python version 3.8.5, and with the use of the Keras module for classification. In order to fit the algorithm into predicting malware or benign PE files instead of malware families, slight modifications had to be done to the code. Also, to make the code work with the collected dataset, some small structural changes were done. However, these alterations did not change anything in the algorithms used for feature extraction or the methods used for predictions.

As discussed in the previous section, the limitation of this algorithm is that it is very RAM intensive, and therefore larger files are hard to extract features from. During the test, 1024 datapoints and 1024 windows were used and when the file size was over 30MB, the application crashed due to an overflow in RAM. The workstation had 16GB RAM with 2GB in swap space, meaning that even not too large files are quite intensive to extract features from.

The selection of this model may be questionable since not a lot of research has been done on this method for malware detection. However, with the lack of research this method still has some impressive results, so including it in the study will still be of value. Since this work was published in 2020, it still has a possibility to become a more stable solution and therefore it is compared to the more common methods found today.

4.4 Experiment

This section will go through how the experiment is setup and what evaluation metrics are used to compare the performance of the algorithms.

4.4.1 Workstation

The experiment was run on Ubuntu 20.04 with Python 3.8.5 installed. The CPU was an Intel Core i7-2700K CPU @ 3.50GHz with 16GB RAM. Since all methods use static analysis, none of the malware samples are executed, which means there is no requirement to set up an environment in a specific way.

4.4.2 Independent Variables

The independent variables of the experiment are:

- **Machine learning methods**, the methods proposed in the paper selected from the literature review.
- **Dataset**, which will be used to train and test the methods.

- **Malware samples**, the malware that the algorithms will try to detect and which is gathered from another source.

4.4.3 Dependent Variables

The dependent variables of the experiment are:

- **Accuracy**, how many samples that were correctly classified.
- **False positive**, how many samples that were classified as malware but was not.
- **False negative**, how many samples that were classified as benign software but was not.
- **True positive**, how many samples that were classified as malware and was.
- **True negative**, how many samples that were classified as benign software and was.

4.4.4 Training

During the training of the data, it is important to find which classifier and hyperparameters are the most optimal for the dataset. The following classifiers will be used for the two methods:

- Gradient Boosting (GB)
- Random Forest (RF)
- Decision Tree (DT)
- Support Vector Machine (SVM)
- K-Nearest Neighbours (KNN)
- Naive Bayes (NB)
- Nearest Centroid (NC)

Depending on what classifier has the best performance out of the gate, that will be chosen to perform hyperparameters tweaks to maximize performance. This is not done with all classifiers since most of them have very different parameters and if a poorly performing classifier is tweaked, it will still be performing poorly. So the hyperparameters tweaked will be presented in chapter 5.

K-Fold Cross-Validation

On the N-Gram and the RWE method, k-fold cross-validation is used to reduce bias from the classifiers. According to R. Kohavi[23], the recommended k value is 10, however, this will create a rather small testing set. It is also mentioned that with a higher number of k such as between 10 and 20, the bias will increase and the variation will decrease[23]. With lower numbers such as 5, the variance will increase. But this is mostly for instances where there are multiple classes being classified. In this study, the k will therefore be reduced to 5 to keep the testing set as large as possible, while still having a low variance due to only two classes being predicted.

The image detection algorithm cannot change the classifiers used, so during training, only the hyperparameters are tested. Neither can the k-fold cross-validation technique be used on this algorithm due to the long training time. The dataset split for this method will then be 50% training, 20% validation, and 30% testing.

4.4.5 Validating

It can be quite challenging to validate that all three algorithms are working correctly. While multiple iterations of reading the original papers and checking that the implementation is handled correctly, without the original dataset, it is almost impossible to be 100% sure. The only available dataset was the image detection method, and the results will be presented in chapter 5. However, with the other two methods, only conclusions from the results from the new dataset can be drawn if it is correctly implemented or not.

4.4.6 Evaluation Metrics

The dependent and independent variables must be looked at to evaluate the algorithms, as the dependent variables will change the outcome based on the independent variables. To measure the difference between the different outcomes, a variety of metrics are used. M. Hossin et al.[19] made a summary of evaluation metrics commonly used in machine learning. This was taken into consideration when selecting the following metrics:

- **Accuracy**, how many samples were correctly classified.
- **False positive (FP)**, how many samples were classified as malware but were not.
- **False negative (FN)**, how many samples were classified as benign software but was not.
- **True positive (TP)**, how many samples were classified as malware and was.
- **True negative (TN)**, how many samples were classified as benign software and was.
- **Recall**, how many positive samples were identified correctly.
- **Precision**, how many of the malware samples were identified correctly.

- **F-Measure**, uses the harmonic mean between recall and precision.
- **Receiver Operating Characteristic(ROC)**, a figure describing the relation between true positive rate and false positive rate.

4.4.7 Significant Difference

To answer the third research question and reject or accept the hypothesis, it is important to know if there is a statistical significant difference between the three methods. Therefore, the Friedman test will be used to measure the difference. If the P-value is less or equal to 0.05 there is a significant difference between the results. If the P-value is above 0.05, there is not enough data suggesting that there is a difference between the three methods.

Chapter 5

Results and Analysis

This chapter will show and present the results from the proposed experiment.

5.1 Validation and Tweaking

This section will contain information about how the algorithms perform compared to the original papers. Then different parameters are tweaked to see which yields the most optimal results.

5.1.1 Validation

To validate the implementation of the algorithms, they are compared to the original papers. However, only the image detection method has access to the original dataset, and it was trained with the best-performing hyperparameters found in the experiment. For the other two methods, the best accuracy results from this experiment are compared against the best accuracy from the original paper. As the Running Window Entropy method had the source code available and only trivial modifications were made, it can be safe to say that the method is implemented correctly.

As seen in table 5.1, the original image detection method has an accuracy of 99.5%, while the implemented version has 99.15%. This should be an acceptable accuracy since small tweaks could be done to increase the accuracy to match the original paper.

For the other two methods, since the dataset are not available, the results from the experiment were used to validate the implementation. For the N-Gram method, the accuracy from the original paper was 99.03% while the results from the experiment were 96.45%, which is a difference of 2.58%. Since these results are from two different datasets, it can be some differences in how the dataset is collected. Or it can be a difference in hyperparameters for the classifiers. However, the results are close to the original paper and should not be dismissed, even if a minor implementation error occurred. The RWE method achieved an accuracy of 95% from the original paper and 93.71% from the final results, leading to a difference of 1.29%. Since the implementation of the RWE method had available source code and is most likely implemented correctly, there is still a difference between the original paper and the final results. Allowing the N-Gram method to have some difference in accuracy compared to the original paper. Even if an implementation error occurred, it should have an accuracy that represents the N-Gram method.

Method	Authors	Accuracy	Val Accuracy
Image	D. Vasan et al.[31]	99.50%	99.15%
N-Gram	L. Yang et al.[35]	99.03%	96.45%
RWE	K. Jones et al.[21]	95%	93.71%

Table 5.1: Table of accuracies from the original papers compared to experiment results

5.1.2 Classifier Selection

This section describes the results of selecting the most optimal classifiers for the methods. It is based on the accuracy metric since it is required for the classifiers to have high accuracy. Because image detection uses a deep learning technique, it is not possible to change it, and will not be included in this section. All the selected classifiers are run with the default parameters and only the top-performing algorithms will be tweaked in the next section. The selected classifiers can be read more about in chapter 4 or 2.

N-Gram

As seen in table 5.2, Gradient Boosting(GB), Random Forest(RF) and Decision Trees(DT) are the top performing classifiers with an accuracy of 94.94%, 94.54% and 93.62% respectively. Since they are close in accuracy, all three have to be selected for the tweaking to find the best performance.

Classifier	Accuracy	Error Rate
GB	**94.94%**	5.06%
KNN	49.26%	50.74%
DT	93.62%	6.38%
SVM	77.91%	22.09%
NB	81.96%	18.04%
RF	94.54%	5.46%
NC	71.1%	28.9%

Table 5.2: Table of N-Gram classifier results

Running Window Entropy

For Running Window Entropy, it is required to select the size of the Window as well as the number of datapoints. The original paper found that 1024 for both values had the best performance on their dataset. This can vary depending on the dataset, however, this is used as a starting point for the classifier test. As seen in table 5.3, only Random Forest and Gradient Boosting are over 90% accuracy, which means they have the best potential for higher performance.

Classifier	Accuracy	Error Rate	Windows	Datapoints
GB	91.86%	8.14%	1024	1024
KNN	87.13%	12.87%	1024	1024
DT	88.85%	11.15%	1024	1024
SVM	88.98%	11.01%	1024	1024
NB	77.18%	22.81%	1024	1024
RF	**93.11%**	6.89%	1024	1024
NC	75.60%	24.40%	1024	1024

Table 5.3: Table of RWE classifier results

5.1.3 Hyperparameter Tweaking

In this section, the hyperparameters are tweaked to boost the performance of the classifiers. Again, the metric used to test the hyperparameters are accuracy.

Image detection

Since the image detection method had a long training time, the number of epochs was set to 20 to run more tests. The author mentioned in the original paper that the accuracy will not increase much after 10 epochs, which means that this is sufficient to test parameters.

For this method, the accuracy is from the training set and not the test set. Because neural networks present the accuracy of each epoch throughout the training process and an estimation of how well-performing the models are can be based on those values. It is known that this is a biased approach, but it should not matter when only selecting the hyperparameters.

The image detection method has two trained models, VGG16 and ResNet50, as described in chapter 2. Table 5.4 shows that each test was run on both models. There are a few hyperparameters that can be changed for these models. However, the most important ones are Steps per epoch(Steps), Learning rate(Learn Rate), and Momentum. By increasing the steps per epoch, the amount of training done is increased, and therefore a higher accuracy is achieved. As the learning rate is decreased, the accuracy will increase since smaller learning increments are done, leading to a smaller chance of overfitting. The momentum is to increase the speed of learning, while also helping the algorithm from getting stuck in a local minimum. Therefore, decreasing this value will decrease accuracy.

N-Gram

From the last section, Gradient Boosting(GB), Random Forst(RF), and Decision Tree(Dt) were selected based on their performance. In chapter 2, these classifiers were discussed and it was mentioned that they are all based on the Decision Tree classifier. Meaning that some of the hyperparameters are the same, while others are not included. As seen in table 5.5, there are some values specified as "N/A". This indicates that the parameter does not exist for that specific classifier. However, the value None means that there are no limitations on the presented value.

Model	Steps	Learn Rate	Momentum	Accuracy
VGG16	10	0.0005	0.9	90.94%
VGG16	10	5e-06	0.5	94.06%
VGG16	10	5e-06	0.9	95.00%
VGG16	25	5e-06	0.9	95.63%
VGG16	50	5e-06	0.9	**97.31%**
ResNet50	10	0.0005	0.9	92.50%
ResNet50	10	5e-06	0.5	90.94%
ResNet50	10	5e-06	0.9	92.81%
ResNet50	25	5e-06	0.9	90.87%
ResNet50	50	5e-06	0.9	**94.69%**

Table 5.4: Table of image detection tweaking of hyperparameters results

In the table, by increasing the number of estimators there are an increase in accuracy. However, this results in a large increase in time while not giving much benefit in accuracy. The best value for this parameter is 150, since it is faster and there is only a small difference in accuracy compared to 200.

For the learning rate, a low value is good, but if it is too low, it will not have enough time to learn all patterns. The optimal value is around 0.75.

Depending on the classifier, a different max depth is required. For the Gradient Boost classifier, the time is increased a lot when the max depth is over 5. This means that 5 has the best accuracy while still keeping training time down.

As seen in the table, the Gradient Booster has the best performance of all three classifiers. The best values are 200, 0.75 and 7, however, this takes a long time to train and 150, 0.75, 5 almost has as good accuracy while taking less train time.

Running Window Entropy

The Running Window Entropy method does not only have hyperparameters to tweak but also sizes of Windows and Datapoints. In table 5.6, it can be seen how the different Window and Datapoint sizes affect accuracy. When increasing Window sizes and Datapoints, the accuracy also increases. At the largest number of 2048, the accuracy is 94.16%. The reason for not getting higher values than 2048 is because the sizes will get too large for the smaller PE files and the feature extraction will get stuck in an infinite loop.

When tweaking the hyperparameters, the set Window and Datapoints were both set to 2048 which was the best performing sizes in the previous test. As can be seen in table 5.7 for the Random Forest classifier, there are two different hyperparameters tested. These two are the most effective parameters that can be changed. The initial values for the classifier are 100 number of estimators and no max depth, which are optimal values for this dataset. By increasing or decreasing any of the values will only overfit the tree, leading to less accuracy.

Classifier	Accuracy	N-Estimators	Learn Rate	Max Depth
GB	94.29%	50	1	3
GB	94.94%	100	1	3
GB	95.12%	150	1	3
GB	95.64%	150	0.75	3
GB	94.04%	150	1.25	3
GB	95.78%	150	0.5	3
GB	94.4%	150	0.75	2
GB	95.6%	150	0.75	4
GB	96.34%	150	0.75	5
GB	95.88%	150	0.5	5
GB	96.47%	150	0.75	7
GB	**96.54%**	200	0.75	7
RF	94.34%	50	N/A	None
RF	94.45%	100	N/A	None
RF	94.76%	150	N/A	None
DT	94.09%	N/A	N/A	25
DT	93.98%	N/A	N/A	50
DT	93.48%	N/A	N/A	100

Table 5.5: Table of N-Gram tweaking of hyperparameters results

Classifier	Accuracy	Windows	Datapoints
RF	91.37%	256	256
RF	92.04%	512	512
RF	93.11%	1024	1024
RF	**94.16%**	2048	2048
GB	90.48%	256	256
GB	91.26%	1024	1024
GB	91.84%	2048	2048
Any	N/A	4096	4096

Table 5.6: Table of RWE tweaking of Windows and Datapoints results

Classifier	Accuracy	N-Estimators	Max Depth
RF	93.36%	50	None
RF	**93.56%**	100	None
RF	93.47%	150	None
RF	93.40%	200	None
RF	93.45%	100	25
RF	93.52%	100	100

Table 5.7: Table of RWE tweaking of hyperparameters results

5.2 Comparison

This section will present and compare the best results from the experiment. These results are performed using 5-fold cross-validation which is run 3 times, which can be seen in table 5.8. Figure 5.1, 5.2 and 5.3 is the average value from all three runs from table 5.8.

For this test, the algorithms had the hyperparameters resulting in the highest accuracy from the previous tests, except for the N-Gram method. It uses 150 as N-Estimators, 0.75 as learning rate, and 5 as max depth.

As table 5.8, or figure 5.1 and 5.2 shows, the N-Gram method has an accuracy of 96.45 and is the highest of all methods. This includes having the best false positives and false negatives as well. Even if one or the other is more important, the N-Gram method will always give the best results. This method is also the most stable, as seen in the standard deviation column(stdev). A low standard deviation means that when splitting the dataset into different parts, the accuracy will stay relatively the same between multiple runs. Interestingly, while the image detection method has fewer false positives, the Running Window Entropy solution has fewer false negatives. Depending on the more important metric , there can be a use case for both.

Table 5.9 or figure 5.3, shows that the Running Window Entropy method has less recall than the other methods and higher precision. This is expected since the Running Window Entropy method had higher False Positive compared to the False Negatives than the other methods.

Method	Run	Accuracy	TP	TN	FP	FN
Image	1	94.61%	94.73%	94.51%	5.48%	5.27%
Image	2	94.64%	94.73%	94.55%	5.44%	5.27%
Image	3	94.64%	94.73%	94.55%	5.44%	5.27%
N-Gram	1	**96.45%**	96.23%	95.53%	3.7%	3.33%
N-Gram	2	96.29%	96.27%	95.17%	4.06%	3.29%
N-Gram	3	96.30%	96.23%	95.23%	4.00%	3.33%
RWE	1	93.68%	91.54%	95.65%	7.97%	4.57%
RWE	2	93.50%	91.42%	95.41%	8.09%	4.81%
RWE	3	93.71%	91.73%	95.53%	7.79%	4.69%

Table 5.8: Table of final results

Method	Run	stdev	Recall	Precision	F-Measure
Image	1	N/A	94.25%	94.73%	94.49%
Image	2	N/A	94.29%	94.73%	94.51%
Image	3	N/A	94.29%	94.73%	94.51%
N-Gram	1	0.16%	96.11%	96.66%	96.38%
N-Gram	2	0.35%	95.75%	96.7%	96.22%
N-Gram	3	0.23%	95.81%	96.66%	96.23%
RWE	1	0.38%	91.6%	95.25%	93.39%
RWE	2	0.59%	91.48%	95.0%	93.21%
RWE	3	0.50%	91.8%	95.14%	93.44%

Table 5.9: Table of final results

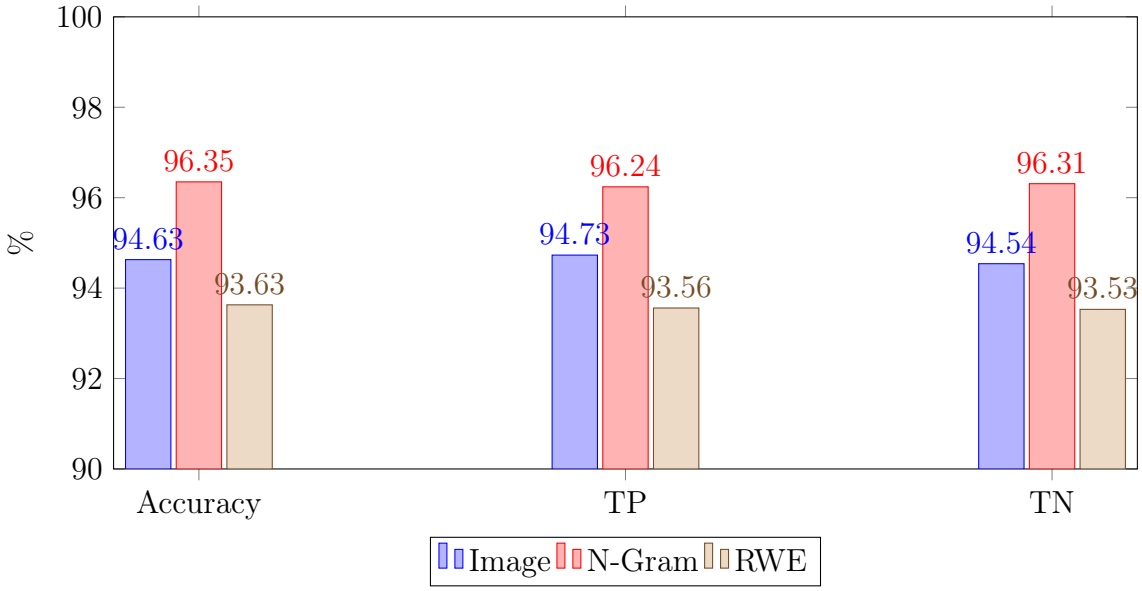

Figure 5.1: Comparing final results of accuracy, True Positive and True Negative

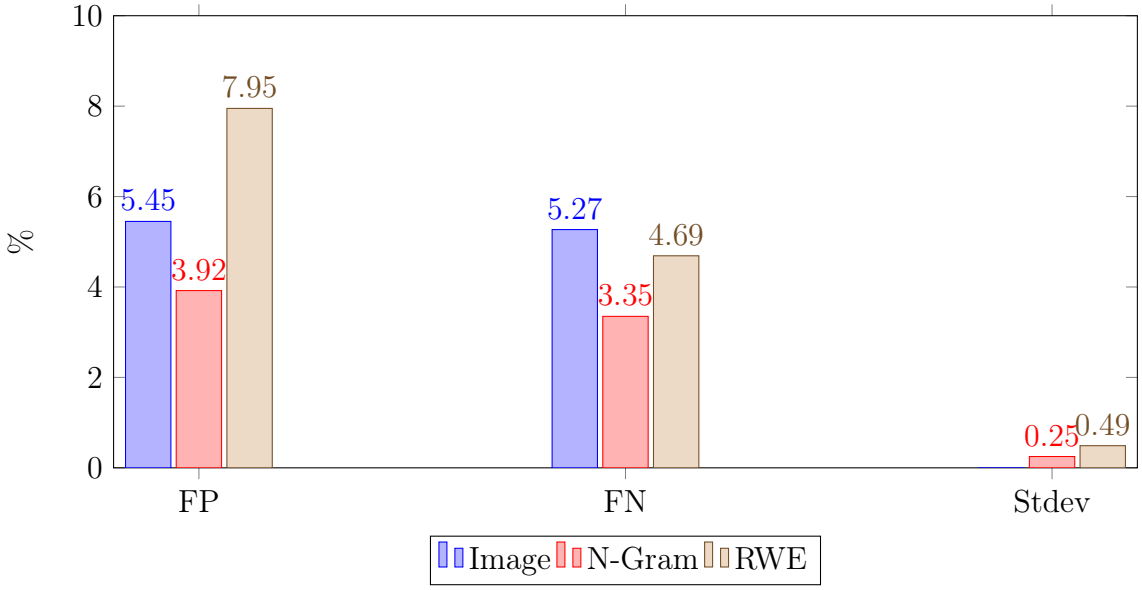

Figure 5.2: Comparing final results of False Positive, False Negative and Standard Deviation

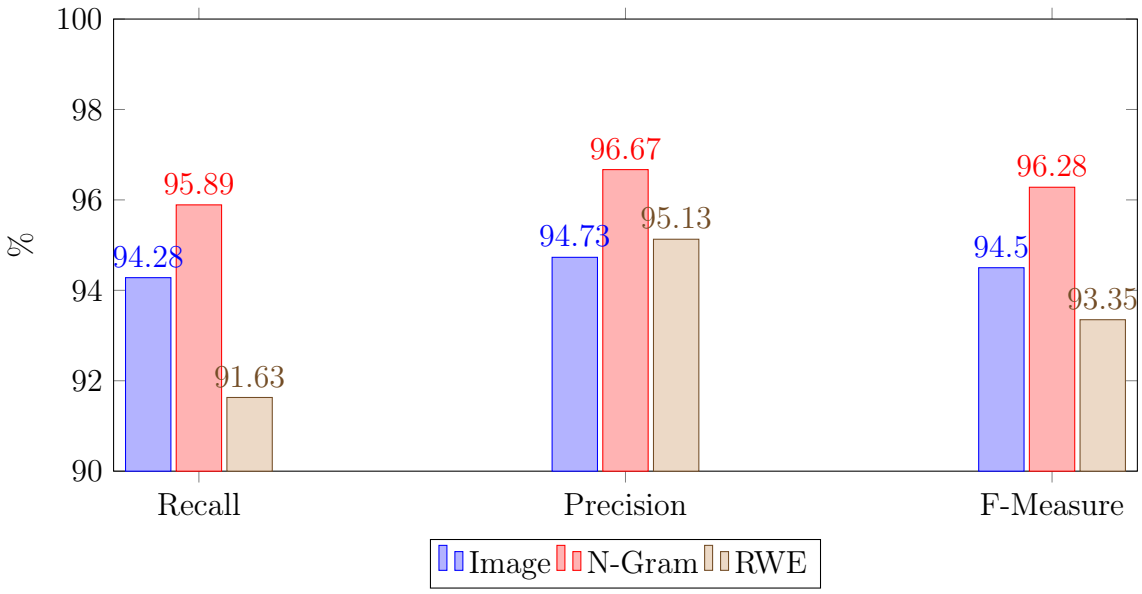

Figure 5.3: Comparing final results of Recall, Precision and F-Measure

Figure 5.4, support the previous statements that N-Gram has the better performance, then image detection and Running Window Entropy is slightly behind. The figure shows that the true positive rate for the N-Gram method will always be the closest method to 1 and therefore it shows that N-Gram has the best performance when measuring true positive rate and false positive rate.

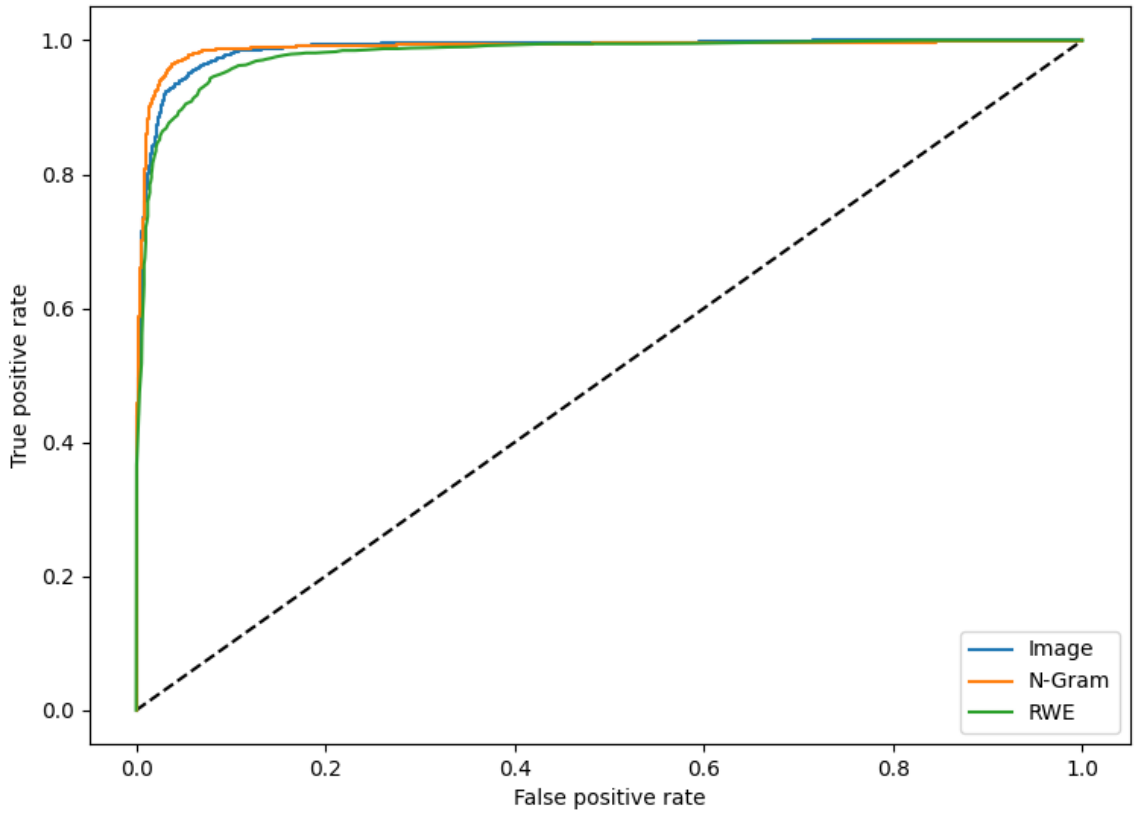

Figure 5.4: ROC Curve results

Without a statistical significance test, there is no certainty that the presented results show that there is a difference in performance between the proposed methods. In table 5.10, the Friedman test is run on the results to find if there is a statistically significant difference between the three methods. The results show a P-value of 0.049 and according to the Friedman test, if the P-value is less than 0.05, there is a statistically significant difference. This proves that there is a difference between the proposed methods. However, the P-value is not far from the limit which indicates that even if there is a difference, it is not large.

Statistics	P-Value
6.000	0.049

Table 5.10: Table of results of Friedman statistical significant difference test

Chapter 6

Discussion

The discussion of this paper are divided into three sections. The first two will discuss the research questions proposed for this study and whether or not the results are valid. The last section will discuss the validity of the paper.

6.1 Literature Review

The literature review was used to answer the first two research questions. The main idea was to get a good understanding and of the field, to be able to select three malware detection methods and select or create a dataset.

6.1.1 Approach Limitations

A concern to the literature review approach can be that the number of iterations had to be limited to keep the proposed schedule. This can lead to papers not being recognized which are well written with high performance, or even methods that were not explored in the first iterations. It could have been a better solution to select a more narrow starting set to get more iterations. While this is the case, a lot of papers were considered and there was a wide range of different methods proposed. Meaning that the suggested papers should cover the popular methods and therefore the results would not change too much if further research was conducted.

There can also be a benefit to start with a large starting sample. As the snowballing technique has critique against it saying some important parts of the field might be missed. With the initial keyword search, there will not be much bias towards any method since there is not much known about the field yet. This will then give a large range of possible methods, and each referring to at least one new method. However, this could lead to more different methods, but the better performing methods are not found since there is no deep dive into that specific method.

6.1.2 Results

RQ1

The results showed that for static analysis, the most common methods are image detection, N-Gram, and Function Call Graphs which answer the first research question. While all of these have good performance in general, there can certainly be better methods available. When selecting the top 20 papers, the most dominating method

found was image detection. While this can be a sign of a popular and well-performing approach, it can also be because there was a bias towards this method in the initial samples. Meaning, a lot more well-performing algorithms are found compared to other methods that only had a few well-performing algorithms. However, this can only be validated by multiple studies with larger samples collected than this. Even if the results were biased towards these three methods, the results could still be valid since these are still popular methods in the field, and a performance review of them from a third party are always needed.

Though results of different methods are compared to see which method has the better performance, it mostly depends on what dataset is used. If there is a difference of one percent on two datasets, it is impossible to know if it is because of the dataset or if the algorithm is better. This is why this study is required. By taking different implementations and comparing them on the same dataset will remove this variable and make it easier to compare.

RQ2

The dataset collection method used in this paper were commonly used in papers throughout the research, and while all methods have drawbacks, most are valid enough to have in academic work. Other methods such as public datasets and honeypot collection are possible, they were too limiting to be used in this study. With this, The Microsoft Classification Challenge[29], Ember[14] and MalImg[25] are the most common datasets. Collection from malware databases such as VirusShare[9] and MalShare[6] are also used a lot, which answers the second research question.

6.2 Experiment

The experiment will answer the third research question by taking the three selected algorithms and compare them in an experiment.

6.2.1 Classifiers and Hyperparameters

The hyperparameters chosen for the experiment were based on the documentation pages from sklearn. As there may be more tweaks that could be done to the classifiers, the selected ones should be sufficient. Even with more tweaking, it would be hard to increase the accuracy.

As described in chapter 2 and 4, the image detection method had two models called VGG16 and ResNet50 with a different amount of layers static to avoid re-training them. The number of static layers was arbitrarily chosen from different articles about the two models. By modifying these values, it could potentially increase the accuracy even more by making the model better trained for malware detection. But due to the time limitations of this study and the long training time of the algorithms, it was not possible.

To save time during training for image detection, fewer epochs were used to find the best hyperparameters. Fewer epochs should be fine since most parameters had reached most of their potential by then. As seen in the results, the parameter testing

has better accuracy than the final results. This is due to this neural network outputs accuracy from the training set during training, to better visualize the progress of each epoch. This metric was then used to select the best hyperparameters. This should still be valid since most parameters are to increase or decrease training time which in return means an increase or decrease in accuracy.

The Running Window Entropy method had the best accuracy with 2048 windows and datapoints. The limitation of this algorithm is it will not be able to make more windows and larger datapoints. As this increases, fewer malware samples can be included in the dataset since the samples will be too small for feature extraction. This was tested with 4096 in windows and datapoints and it could not extract the values from the all files in the dataset.

6.2.2 Results

Detection and Classification

The results show less accuracy than the original papers even though two of then are classifying malware or not malware instead of multiple malware families. This may be a concern since the initial thought may be that it is easier to classify two classes instead of 25 or more. However, this may not be the case since research do not have as high accuracy for malware detection. From a study proposed by A. Yewale et al.[36], and could only achieve an accuracy of 96.67%. A survey done by H. Merabet et al.[17] presented multiple studies and it showed that none were 99% or above. The most accurate algorithm had 98.9% and the median was 94%. A more in depth study has to be done to see if this is actually the case, but it is much more common to see classification of malware families have 99% and above accuracy. This may be due to malware detection does not know how the a certain malware family is structured or what imports are usually used, while in family classification, most malware samples will look alike.

As the image detection and Running Window Entropy methods were originally created as malware family classification algorithms, and it may be a concern that they were implemented as malware detection algorithms. For example, the image detection method relies on the structure of a malware to classify it, it may be hard for it to find patterns in malware it had not seen before. As this was one of the more promising algorithms during the literature review, it may be the case that it is better suited for malware family classification. However, none of the algorithms had a bad performance if it is compared to the survey done by H. Merabet et al.[17]. This means that even if the algorithms did worse than expected, the results are still usable for malware detection.

Implementation validation

Since the implementation of the algorithms was not done by the author, except for the RWE algorithm, it can be difficult to know if the algorithms were implemented correctly. As the results show in section 5.1.1, there is a slight decrease in accuracy. However, it is very hard to judge since there are changing variables. For some of the algorithms, there is a change from classification to detection, a new dataset that

probably has more variety of data, and hyperparameter tweaks to fit the new dataset. With this in mind, it is impossible to validate the implementation to 100% without having access to the original dataset, and even with it, inconsistencies can still occur. But the accuracy is still at a level where all algorithms are performing well, and from there, a conclusion can be drawn that even if there are small inconsistencies in implementation, the algorithms are good enough to represent the research being done in the field. Meaning that, even if the algorithms are not exactly as the original author presented, it will still evaluate the method used overall.

Since the image detection method had a long training time, it was not feasible to perform K-fold cross-validation. This means that the results are less generalizable because there are less data collected about the method. However, this should not dismiss the results entirely, only that the results will not be 94% every time. Also, as the other algorithms only have a standard deviation of 0.59% or less, it shows that even with different parts of the dataset is used as training and testing, it should be stable. Meaning that it should take an implementation error to make the method drop a few percent in accuracy.

As mentioned in chapter 4, hash tricking was used instead of the HashingVecotizer from sklearn suggested in the original report. As this could affect the final results, both methods have the same idea of converting strings into a vector. This could result in the final feature row looking a bit different than the original, but the patterns should be recognized by the classifiers.

RQ3 and Hypothesis

Before answering the research question, the answer of the hypothesis has to be discussed to see if there even is a difference in performance between the three methods. The results presented in chapter 5, showed that the P-value from the Friedman test was 0.049 which is below 0.05. Therefore, there is enough variation in the data to say that there is a significant difference between the methods. Therefore, the null hypothesis can be rejected.

To answer the third research question about which method has the best performance, the results from chapter 5 have to be considered. The method with the most accuracy was N-Gram, which also had the best TP, TN, FP, and FN. This means that from the results on this dataset, the N-Gram method had the best performance in all metrics. For the other two methods, depending on what metric is more important, there might be a difference in which performed the best. As the image detection algorithm had better accuracy and fewer false positives, this method is better if it is important to not alert with benign files. However, the RWE approach had less accuracy and fewer false negatives. Meaning that less malware will be classified as benign files, which can be more important than accuracy for certain scenarios. Since there is not a significant difference between the two, it is hard to say that any of them would perform better in most cases.

To relate to the literature review, the results do not correlate with the accuracy presented. As mentioned in chapter 3, the most promising algorithms were Function Call Graphs and image detection. Since the Function Call Graph was not included in this study, image detection should be the best performing method. The N-Gram method should not have much less accuracy than image detection. Then the Running

Window Entropy solution should have the least accuracy out of the three. The results presented in the experiment show that this is not the case, and the most performing method is N-Gram, while the image detection and RWE have slightly less accuracy.

6.2.3 Dataset

With a dataset with 7269 malware samples, 7655 benign samples, and 14924 in total, it is possible to speculate if these are enough samples to perform static analysis on. As the N-Gram and Running Window Entropy papers had more samples to train and test on, this could change the outcome of the results. At the same time, the image detection only had a sample size of 9458 and other researchers had presented papers with fewer samples than proposed in this study[20, 39]. Meaning that sample size can affect the results of a study, but successful research has been done with the same amount of gathered samples as the dataset proposed in this study.

The range of different malware families may affect the performance of the algorithms since they might not learn enough of the patterns from only one sample of a malware type. While this is true, the reason for including this range of malware families is because new families will be discovered in the real world. Therefore, a more accurate representation of the algorithms are created. In production, it may not be viable to keep this dataset as training data. However, these obtained results will not be too far away from reality.

In this paper, it has been mentioned that it is important to keep the dataset realistic to keep the accuracy as close to reality as possible. It could be argued that this would create an unbalanced training set for the classifiers and therefore the results would not be accurate. However, as the suggested method of classification is by malware or benign, it is important to keep the dataset with odd variations of malware since this is what is active in the real world. If the proposed method of classification was on malware families, then it would be a must to keep all the included malware families balanced.

6.3 Threats to Validity

Internal Validity

Some factors can not be controlled, such as the implementation of the methods. While the author has previous experience with machine learning and using machine learning to detect ransomware, the implementation of the methods were a new experience. Meaning, due to lack of domain knowledge the methods can have few variations in implementation compared to the original papers. However, the threat to validity is reduced by comparing the implemented methods and their results to the original papers. Another threat to validity is the misclassification of malware and benign files in the dataset. This was reduced by using an external source of multiple anti-malware software to ensure all samples are classified correctly. The hyperparameters can cause a threat to validity since if not all are explored, there is a chance that a well-performing algorithm is overlooked. This was reduced by experimenting with as many parameters as possible to see how the outcome changes.

External Validity

To be able to make this study applicable to other fields or use cases, the dataset was created to mimic real-world malware samples. The algorithms will then represent the actual accuracy they would have in production and therefore the results in this study are more useful. The algorithms were also run multiple times with different parts of the dataset as training and testing to ensure that the algorithms do not perform any differently based on the order of a dataset.

6.4 Contributions

As the field is well researched and new methods are proposed all the time, it is also required to take a step back and analyze the solutions that are already available. A common way of doing this is to do a literature review and compare results. However, in this case, it is hard to compare all available methods due to many uncontrollable variables when only looking at the results of the papers. If the experiments include different datasets, malware samples, and sizes, the results can vary a lot. This is why a comparison has to be done in a single experiment to control as many variables as possible. As in this study, three different malware detection methods are compared against each other on the same privately collected dataset to keep the results as realistic and accurate as possible.

This is only a comparison between three methods out of the hundreds available. That is why this is only the beginning of the possible research is done, and more comparisons have to be done.

Chapter 7

Conclusions and Future Work

7.1 Conclusion

Malware has been a major issue for many years, and old and outdated malware detection techniques are slowly being replaced. With the rise of machine learning, a lot of research has been done to find patterns in malware with deep learning and other classification methods. In this study, a literature review has been proposed to find which are the best performing research available. This review is followed by an experiment where the top three methods are re-implemented and compared to see which are the best. A dataset is collected with 14924 benign and malware samples to train and test the three methods.

The literature review used the snowballing technique to find as many papers as possible in the field. It was found that image detection, N-Gram combined with meta-data, and Function Call Graphs were the most popular and high-performing methods. An additional method called Running Window Entropy was discovered that had little research about it but could provide decent accuracy. For many of the researched algorithms, publicly available datasets were often used. It consisted of The Microsoft Classification Challenge, Ember, and MalImg datasets. Others collected their data from various malware databases such as VirusShare or MalShare.

The image detection, N-Gram, and Running Window Entropy methods were chosen for the experiment. The dataset was collected from VirusShare and then scanned for malware from external anti-malware software from Metadefender. The collected dataset included 7269 malware samples, 7655 benign samples which is a total of 14924 samples. The results showed that N-Gram had an accuracy of 96.45%, image detection had an accuracy of 94.64%, and Running Window Entropy had an accuracy of 93.71%.

7.2 Future Work

Since this research is limited by the implementation of the algorithms, it would be interesting to see how implementations by the authors would affect the results. Therefore, a research topic for the future could be investigating papers with available source code. This eliminates the need to validate the implementation and the results will be more clear.

Since one algorithm from a method cannot represent the entire field, it could be possible to investigate how much the performance of the algorithms varies. As an example, if the N-Gram method was selected, how would different approaches

to this affect the final results. Which parameters and features would yield the best performance.

The dataset has a big impact on whether or not an algorithm has good performance. It would be interesting to see how much difference there is between different kinds of datasets. Depending on if it is public, collected from malware databases, or collected by hand. Also, whether or not the malware is packed, encrypted, or uses any form of obfuscation.

Bibliography

[1] "Av-test, malware statistics and trends report," https://www.av-test.org/en/statistics/malware/, accessed: 2021-04-20.

[2] "Google scholar," https://scholar.google.com/, accessed: 2021-04-26.

[3] "Ibm security, cost of a data breachreport 2020," https://www.ibm.com/security/digital-assets/cost-data-breach-report/, accessed: 2021-04-22.

[4] "Malconv, community implemented by endgameinc," https://github.com/endgameinc/malware_evasion_competition, accessed: 2021-04-27.

[5] "Malgazer, source code," https://github.com/keithjjones/malgazer, accessed: 2021-04-23.

[6] "Malshare, malware database," https://malshare.com/, accessed: 2021-04-22.

[7] "Metadefender, malware scanning cloud service," https://metadefender.opswat.com, accessed: 2021-04-23.

[8] "Pefile, python3 module to handle pe files," https://pypi.org/project/pefile/, accessed: 2021-04-26.

[9] "Virusshare, malware database," https://virusshare.com/, accessed: 2021-04-22.

[10] "Virustotal, malware scanning cloud service," https://www.virustotal.com, accessed: 2021-04-23.

[11] "Vx heaven, malware database," https://www.vx-underground.org/, accessed: 2021-04-22.

[12] R. Agrawal, J. W. Stokes, M. Marinescu, and K. Selvaraj, "Robust neural malware detection models for emulation sequence learning," in *MILCOM 2018 - 2018 IEEE Military Communications Conference (MILCOM)*, 2018, pp. 1–8.

[13] D. Amancio, C. Comin, D. Casanova, G. Travieso, O. Bruno, F. Rodrigues, and L. da F. Costa, "A systematic comparison of supervised classifiers," *PloS one*, vol. 9, p. e94137, 04 2014.

[14] H. S. Anderson and P. Roth, "Ember: An open dataset for training static pe malware machine learning models," 2018.

[15] T. Ayodele, *Types of Machine Learning Algorithms*, 02 2010.

[16] M. Chowdhury, A. Rahman, and R. Islam, "Protecting data from malware threats using machine learning technique," in *2017 12th IEEE Conference on Industrial Electronics and Applications (ICIEA)*, 2017, pp. 1691–1694.

[17] H. El Merabet and A. Hajraoui, "A survey of malware detection techniques based on machine learning," *International Journal of Advanced Computer Science and*

Applications, vol. 10, 01 2019.

[18] M. Hassen and P. K. Chan, "Scalable function call graph-based malware classification," in *Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy*, ser. CODASPY '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 239–248. [Online]. Available: https://doi-org.miman.bib.bth.se/10.1145/3029806.3029824

[19] M. Hossin and S. M.N, "A review on evaluation metrics for data classification evaluations," *International Journal of Data Mining Knowledge Management Process*, vol. 5, pp. 01–11, 03 2015.

[20] H. Jiang, T. Turki, and J. T. L. Wang, "Dlgraph: Malware detection using deep learning and graph embedding," in *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*, 2018, pp. 1029–1033.

[21] K. J. Jones and Y. Wang, "Malgazer: An automated malware classifier with running window entropy and machine learning," in *2020 Sixth International Conference on Mobile And Secure Services (MobiSecServ)*, 2020, pp. 1–6.

[22] K. J. Jones and Y. Wang, "An optimized running window entropy algorithm," in *2018 National Cyber Summit (NCS)*, 2018, pp. 72–77.

[23] R. Kohavi, "A study of cross-validation and bootstrap for accuracy estimation and model selection," vol. 14, 03 2001.

[24] R. Lyda and J. Hamrock, "Using entropy analysis to find encrypted and packed malware," *IEEE Security Privacy*, vol. 5, no. 2, pp. 40–45, 2007.

[25] L. Nataraj, S. Karthikeyan, G. Jacob, and B. S. Manjunath, "Malware images: Visualization and automatic classification," in *Proceedings of the 8th International Symposium on Visualization for Cyber Security*, ser. VizSec '11. New York, NY, USA: Association for Computing Machinery, 2011. [Online]. Available: https://doi.org/10.1145/2016904.2016908

[26] S. Ni, Q. Qian, and R. Zhang, "Malware identification using visualization images and deep learning," *Computers Security*, vol. 77, pp. 871–885, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167404818303481

[27] E. Raff, J. Barker, J. Sylvester, R. Brandon, B. Catanzaro, and C. Nicholas, "Malware detection by eating a whole exe," 2017.

[28] M. Rhode, P. Burnap, and K. Jones, "Early-stage malware prediction using recurrent neural networks," *Computers Security*, vol. 77, pp. 578–594, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167404818305546

[29] R. Ronen, M. Radu, C. Feuerstein, E. Yom-Tov, and M. Ahmadi, "Microsoft malware classification challenge," 2018.

[30] D. F. Specht, "A general regression neural network," *Trans. Neur. Netw.*, vol. 2, no. 6, p. 568–576, Nov. 1991. [Online]. Available: https://doi-org.miman.bib.bth.se/10.1109/72.97934

[31] D. Vasan, M. Alazab, S. Wassan, B. Safaei, and Q. Zheng, "Image-based malware classification using ensemble of cnn architectures (imcec),"

Computers Security, vol. 92, p. 101748, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S016740482030033X

[32] R. Vinayakumar, M. Alazab, K. P. Soman, P. Poornachandran, and S. Venkatraman, "Robust intelligent malware detection using deep learning," *IEEE Access*, vol. 7, pp. 46 717–46 738, 2019.

[33] C. Wohlin, "Guidelines for snowballing in systematic literature studies and a replication in software engineering," *presented at the International Conference on Evaluation and Assessment in Software Engineering (EASE)*, 2014.

[34] W. Yan, Z. Zhang, and N. Ansari, "Revealing packed malware," *IEEE Security Privacy*, vol. 6, no. 5, pp. 65–69, 2008.

[35] L. Yang and J. Liu, "Tuningmalconv: Malware detection with not just raw bytes," *IEEE Access*, vol. 8, pp. 140 915–140 922, 2020.

[36] A. Yewale and M. Singh, "Malware detection based on opcode frequency," in *2016 International Conference on Advanced Communication Control and Computing Technologies (ICACCCT)*, 2016, pp. 646–649.

[37] I. You and K. Yim, "Malware obfuscation techniques: A brief survey," in *2010 International Conference on Broadband, Wireless Computing, Communication and Applications*, 2010, pp. 297–300.

[38] Y. Zhang, Z. Liu, and Y. Jiang, "The classification and detection of malware using soft relevance evaluation," *IEEE Transactions on Reliability*, pp. 1–12, 2020.

[39] Y. Zhang, X. Chang, Y. Lin, J. Mišić, and V. B. Mišić, "Exploring function call graph vectorization and file statistical features in malicious pe file classification," *IEEE Access*, vol. 8, pp. 44 652–44 660, 2020.

[40] Z. Zhang, Y. Li, H. Dong, H. Gao, Y. Jin, and W. Wang, "Spectral-based directed graph network for malware detection," *IEEE Transactions on Network Science and Engineering*, pp. 1–1, 2020.

Faculty of Computing, Blekinge Institute of Technology, 371 79 Karlskrona, Sweden