Malware Classification using Image Representation

A thesis submitted in fulfilment of the requirements

for the degree of Master of Technology

by

Ajay Singh

Department of Computer Science and Engineering

INDIAN INSTITUTE OF TECHNOLOGY KANPUR

July 2017

Statement of Thesis Preparation

1. Thesis title: Malware Classification using Image Representation.
2. Degree for which the thesis is submitted: M.Tech
3. Thesis Guide was referred to for preparing the thesis.
4. Specifications regarding thesis format have been closely followed.
5. The contents of the thesis have been organized based on the guidelines.
6. The thesis has been prepared without resorting to plagiarism.
7. All sources used have been cited appropriately.
8. The thesis has not been submitted elsewhere for a degree.

Ajay Singh
(Signature of the student)

Name: AJAY SINGH

Roll No.: 15111005

Department/IDP: CSE

Certificate

It is certified that the work contained in this thesis entitled "Malware Classification using Image Representation" by "Ajay Singh" has been carried out under my supervision and that it has not been submitted elsewhere for a degree.

Shukla
10.7.17

Dr. Sandeep Shukla

Professor

Department of Computer Science and Engineering

Indian Institute of Technology Kanpur

July 2017

Abstract

Name of the student: **Ajay Singh** Roll No: **15111005**

Degree for which submitted: **M.Tech.** Department: **Computer Science and Engineering**

Thesis title: **Malware Classification using Image Representation**

Thesis supervisor: **Dr. Sandeep Shukla**

Month and year of thesis submission: **July 2017**

In the recent years, there has been a rapid rise in the number of files submitted to anti-virus companies for analysis, so it has become very difficult to analyse functionality of each file manually. Malware developers have been highly successful in evading the signature based detection techniques. Most of the prevailing static analysis techniques involve a tool to parse the file. The entire analysis process becomes dependent to the efficacy of the tool, if the tool crashes the process is hampered. Most of the dynamic analysis techniques involve the binary file to be run in a sand-boxed environment to examine its behaviour. This can be easily thwarted by hiding the malicious activities of the file if it is being run inside a virtual environment. In this thesis, we have explored a new technique to represent malware as images. We then used existing neural network techniques, for classifying images, to train a classifier for classifying new malware files into their respective classes. By converting the file into an image representation we have made our analysis process independent of any tool also the process becomes less time consuming. With our model we have been able to get an accuracy of 98.21% in classifying malware samples.

Acknowledgements

I would extend my sincerest gratitude to Dr. Sandeep Shukla for allowing me to work on this topic and also for guiding me through all the ups and downs. I would also like to thank my co-researchers Pranjul Ahuja, Vineet Purswani, Abhay Kumar and Sourabh Kumar for their help and co-operation whenever I faced any difficulty. I am grateful to my family and friends for being there for me all the time. Finally I would also like to thank all the countless tech-samaritans over the Internet community who were at my rescue all the time.

Contents

List of Figures

List of Tables

Dedicated to my Parents.

Chapter 1

Introduction

Internet has become an integral part of our daily life. Almost 57% of the world's population as of 2017 is connected over the Internet [19]. We use it for banking, communicating, entertainment, shopping, and various other commercial and non-commercial activities. Albeit making our life convenient, the Internet has exposed us to the risk of getting attacked. Illegitimate users use malware programs to commit financial frauds or steal private/sensitive information from legitimate users. The number of reported attacks are increasing every year. Pandalabs has reported to identify 227,000 Malware samples every day in 2016 [39]. What started as a hobby of tech-enthusiast and researchers, has now evolved into an international community of highly trained programmers motivated by easy profits [38]. It has now grown to be a multi-million dollar industry [22] where hacking tools are sold and bought, just like the legal software industry. Even technical support and customer services are available to allure pillagers.

Proliferation of malware at an ever increasing rate poses a serious threat in the post-internet world. Malware detection and classification has become one of the most crucial problems in the field of cyber security. With the ever increasing risk of attack, the onus lies on the security researchers for devising new techniques for detecting malware and developing new security mechanisms against them.

1.1 Malware

Malware is an abbreviation to Malicious Software. Malwares are specifically written programs written to perform some malicious activities such as getting access to victim's machine, setting up a key logger, setting up a back door for a bot or stealing valuable private information. Malware can be defined as "Any software that does something that causes harm to a user, a computer, or a network" [51]. A malware can be in form of a script, an executable or any other software program. Broadly, malwares can be classified as Worms, Viruses, Trojans, Ransomwares, Adwares, Spywares, Bots, PUPs [1], Rootkits, Scarewares, and other malicious programs. Almost every attack we hear about, such as the Mirai malware attack [60] and WannnaCry Ransomware attack [62], involves some kind of malicious software. With the rising digital footprint the complexity of these attacks is also increasing.

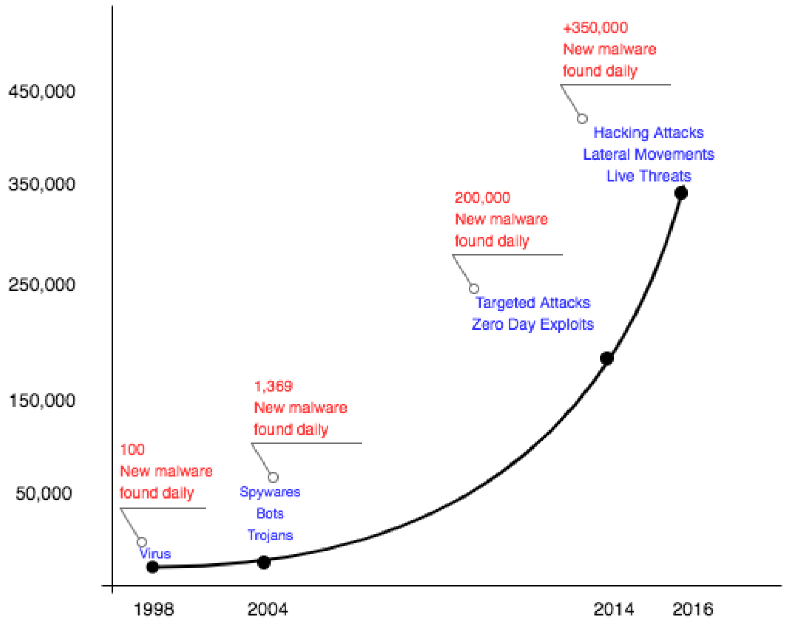


FIGURE 1.1: Increasing malware complexity[40].

1.1.1 Types of Malware

This section provides an overview about the most popular malware types. These types are neither comprehensive nor mutually exclusive. New malware types are discovered almost everyday and different types of malware may share exploitation techniques.

Worms: Worms are network viruses, that have the ability of spread over the network by replicating themselves[53]. They do not alter the user or system files, but they

[1]PUP stands for Potentially Unwanted Programs

reside in the main memory and keep on replicating and spreading themselves. With all these unwanted activities they end up using all the available resources thus making the system and the network unresponsive [46]. Though not directly harmful, they are generally used to inject payloads, which can be other malicious programs like trojan horses, viruses, backdoors, etc. Some famous worm attacks are Morris worm, loveletter worm, and Code Red.

Viruses: A virus is a software program that modifies other programs and attaches itself to their code. Just like the real viruses, it can not run on its own, it performs malicious activities only when the infected program is executed [52]. A virus propagates itself by infecting files on the host machine as well as shared files over the network. In the process of infecting user files they generally end up corrupting them, thus user may suffer from data loss. Some of the most famous computer viruses are Creeper virus, and Melissa virus.

Trojans: Trojans, as the name suggest, are malicious programs that deludes the user to install them by pretending to be a useful or benign software. On execution it starts performing its malicious activities in the background. They are generally propagated through some social engineering techniques such as e-mail attachment, fake pop-ups, or spam links. Once installed on the victim's computer they may download other malwares, create backdoors or infect system files. Some of the famous Trojans are Zeus, Dark comet, and Shedun Android malware.

Ransomwares: Recently ransomware attacks have increased drastically like the recent TorrentLocker, the WannaCry, and the Petya attacks [61]. They either encrypt all the user files or restricts the user from accessing her files. By restraining the users, the attacker forces them to pay in order to get access to their files. Ransomware attack are normally carried out using a Trojan, which tricks the user to deliver the payload, ransomware, into their system.

Adwares: Adware shows specific advertisements to the infected host that help the attacker to generate revenue. They are generally installed along with a PUP. They may not harm the victim directly but there might be some vulnerability in the adware which may be exploited by some attacker.

Spywares: Spywares, as the name suggest, are generally used to spy on users. They are malicious programs that aim at stealing private information from the infected host without getting noticed and then send this information to the attacker. They are used to track the activities of the user, steal passwords, bank account details, etc.

Bots: A Bot is a software that allows the attacker to remotely access the hosts machine. A collection of Bots controlled by a single server is called a Botnet. A Botnet can be used to launch DDoS attack, run spam campaigns, etc.

Rootkits: Rootkits are used to hide themselves or other malicious programs from the victim [51]. Their detection is difficult because they generally run with root privileges, which enables it to subvert system logs and intrusion detection softwares.

Scarewares: Scarewares tries to trick users into buying something that was unnecessary by frightening them. They use social engineering techniques, such as showing false warning messages, false security breach alerts, to create false sense of threat.

Backdoors: Backdoors allow access to a system through a remote terminal. It is used both for malicious and non-malicious purposes. A popular Backdoor tool is RAT, Remote Access Terminal.

Spyware, Adware, PUPs are sometimes classified as Grayware. Grayware are unwanted applications that don't directly harm the infected host but can consume system resources worsening performance and may contain some vulnerability that can serve as an entry point for an attacker.

1.1.2 Malware Nomenclature

Although all anti-virus companies follow a rigorous naming conventions for malwares but there are no standard conventions for naming malwares. So it is quite possible that different anti-virus engines may assign different names to the same malware file. For the purpose of this thesis we will discuss the naming convention followed by Microsoft anti-virus. Microsoft anti-virus uses the malware naming scheme proposed by Computer Antivirus Researchers Organisation (CARO).

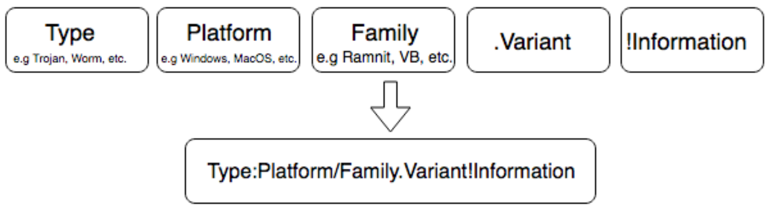


Figure 1.2: Naming scheme used by Microsoft [32].

For example Trojan:JS/BlacoleRef.W, Worm:Win32/Allaple.A, etc. As per the naming convention depicted in figure-1.2 from the name Trojan:JS/BlacoleRef.W we can easily figure out that, Trojan is the type, JS is the platform, BlacoleRef is the family of malware, and W is the variant.

Type field in the naming scheme specifies the type of the malware, that is the major activity performed by the malware. The major malware types have been discussed in the previous section such as Trojan, Backdoor, Virus, etc.

Platform field signifies the platform on which the malware is designed to execute. Platform may include operating system such as Windows, MacOS, Android, etc. It may also include programming language like HTML, JS, etc.

Family of a malware is determined based on structural similarity between different malwares. Malware belonging to the same family have code similarities, which help in creating generic detection and removal techniques. It is the most important field in the naming convention.

.Variant is used to distinguish between different version of the same family.

!Information field is used to specify some extra information about the malware such as whether it is corrupted, compressed, packed, etc. It may also include any other information which the analyst considers as an important characteristic of that malware. For example lnk may be added to specify that the malware is a shortcut file.

A malware name must include the family name and the variant name other fields may be dropped. Only if their is only one variant of a particular family is known, variant field is allowed to be dropped. At places in this thesis family has been replaced by class, thus malware class and malware family are essentially the same.

1.2 Malware Analysis

Malware analysis is the process of dissecting a binary file to understand its working and then devising methods to identify it and other similar files [51].

Malware analysis aims at figuring out the actions performed by the malicious software and then developing methods to neutralise those actions and prevent further infections. It is not only the responsibility of anti-virus companies, security administrators of various organisations need to do malware analysis. System administrators need to carry out malware analysis to figure out the extent of damage to the system. Network administrators need to carry out analysis to figure out the extent to which the network has been damaged and then try to fix it. Academic researcher analyse malware samples to understand how they behave and study the techniques used by malware developers, to improve security of existing infrastructure.

Malware analysis is being used both for detecting malware and classification of malware. Malware detection is the process of labelling a binary executable as benign or malign. So basically, it is the first step of malware analysis. Only after we know a file is a malware we proceed for further analysis. Whereas classification is the process of figuring out the class to which a given malware belong. After we know a file is a malware, we need to figure out its family. Malware analysis can be performed in two ways static and dynamic. Static analysis refers to the technique that aims at analysing a sample without executing it, while in dynamic analysis we execute a sample to determine its behaviour.

1.2.1 Static Malware Analysis

Static Malware analysis involves inspecting the binary entirely without executing it. This method can be applied to various types of representations of an executable. If the actual code is present then it becomes easy to perform static analysis. Even if the code is not available, we can examine the raw hex code or first disassemble the binary file and then examine the assembly code. Static analysis may involve examining printable strings in the program, examining the header of the file, disassembling the program, finding byte sequences, etc.

1.2.1.1 Common Static Analysis Techniques:

1. We can use any available anti-virus to check if the file has already been labelled as malicious. This can be done by taking fingerprint of the file. *MD5* or *SHA-256* hash can serve as a fingerprint to distinguish a file from other files [51].

2. We can get all the printable characters from the binary using *strings.exe*. This may include some status information or error messages with which we can conclude about the functionality of the binary [51].

3. We can also examine the meta data present in the file header. Meta data may include information such as magic number, compilation time, system library imports, icons

used, etc. Magic number can help us identify the type of file, type of libraries imported may help us to predict the behaviour of the malware [51].

4. We can figure out if the file is packed or not by using packer identification tool for example the *PEiD*. *PEiD* manages a database of signatures of popular packers and if the file has been packed using one of them then it can report us. If the file is packed by a standard packer, such as *UPX Packer* then we must first unpack the packed binary before analysing it any further [51].
5. We can perform an in-depth analysis of the functionality of the malware by first disassembling the machine code to assembly code and then examine the opcodes. *IDA Pro* or a similar tool can be used to generate the assembly code and then *OllyDbg* or a similar tool can be used for debugging the assembly code [51].
6. Trying decompilation to get High Language Code from Machine Code. It does not recover the original code as there is information loss during compilation [51] but we can get some idea about the binary from the recovered code.

1.2.1.2 Advantages and Limitations

Major advantage of static analysis is that it poses no risk of spreading the infection while malware is being examined. Static analysis is comprehensive as it tends to cover all the possible execution paths. Only disadvantage with it is that, it can be very time consuming and requires an expert to perform the analysis.

Static Analysis can get thwarted if the binary is encrypted, compressed, obfuscated or packed. Some of the popular techniques to evade Static Analysis are:

Polymorphism generally means ‘change the appearance of‘. In this technique the malware developer changes the binary file in such a manner that it can evade detection techniques which use pattern matching. Packers or crypters are used for making polymorphic copies of binaries. These software first pack the binary file and then attach an unpacking routine at the top of the packed code. This new binary can’t be detected using pattern matching because its signature will be different. When the binary will be executed the packing routine will first unpack the packed binary in memory only and then pass the execution sequence to the start of the unpacked code. Thus even though file has been changed but it will continue performing the same function. The easiest way to detect polymorphic virus is to look for signatures in memory because the original malware is decrypted and kept in memory.

Metamorphism in malware mean self modifying binary executables. The malware modifies its own binary code whenever it propagates or is run. It is different from polymorphism, as polymorphic malware can't rewrite their own code. The resulting binary will perform the same activities as the original malware but its binary representation will be completely different. Simple techniques for metamorphism include adding varying lengths of NOP instruction, changing control flow using jumps, permuting the registers used, etc.

1.2.2 Dynamic Malware Analysis

Dynamic analysis involves executing a binary and observing its behaviour to deduce whether it is a malware or not. The malware is executed in a sand-boxed environment, or VM, so that the effects of the malware can be contained. Virtual Machines give us the advantage of taking snapshots of the state prior to execution of the malware and once the analysis is over we can easily revert back to the saved state.

1.2.2.1 Common Dynamic Analysis Techniques:

System call monitoring System calls help a program to interact with the operating system. A function is generally associated to a particular task, for example sort function is used for sorting. So monitoring the system calls used by an executable can give us better insights about its functionality.

Instruction trace Another way of tracking the activities of a malware is through instruction trace. It records the sequence in which the instructions were executed by the malware.

Tracking malware activities Actions performed by a process under execution on other processes and files can be monitored using tools such as *Process Monitor*. We can keep a track of the processes forked and the list of files modified by the malware. Another tool *Process Explorer* help us to track registry changes, file activities and new child process created during execution.

Registry tracking Malwares programs attempts to change system registry to alter system behaviour. So to track any changes to the registry we must take its snapshot before executing the binary file and then compare the initial snapshot with the registry after execution is over.

Tracking Network activity We can simulate a fake network to monitor network activity. Certain malware show malicious activity when they are connected to a network. So tools such as *INetSim* can be used to simulate the Internet, because giving Internet access to the sandbox can be risky.

Capturing File Modifications Goat files are specially created to track file modifications done by a malware. They generally contain just the NOP instruction, so if Malware tries to infect it then changes to the file can be easily identified [53].

Debugging Using a debugger we can monitor the changes made at every step of the execution. Also a debugger can help us to identify alternate execution paths, which may remain unexplored on simple execution.

1.2.2.2 Advantages and Limitations

Since, during dynamic analysis malware is being executed, this will help us to counter the techniques defeating static analysis such as packing and obfuscation.

Major limitations of Dynamic Analysis is that we are usually able to monitor a single path of execution, so it suffers from incomplete code coverage. Also, if a malware is able to detect the sandbox then it may alter it behaviour by hiding the malicious activities, thus evading the analysis. Another limitation is that, if there is any bug in the sandbox environment then malware can infect the host computer or other computers on the network.

1.2.3 Need for Automated Malware Analysis

Traditional malware identification tool, anti-virus software, use signature matching to identify malware. Analyst at security companies analyse a sample received by them and develop unique signatures related to that sample. The anti-virus database is then updated with these signatures, which then compares the signature of every file on the machine with its database. If the signature of a particular file matches with any of the signatures in the anti-virus database then that file is labelled as malware. Due to their high matching speed and high accuracy in finding known threats signature based techniques are effective, but these techniques fail to cope with code obfuscation and fails to effectively identify newly arrived threats. More than 800,000 files are submitted everyday to VirusTotal [58] for analysis. Since the volume of samples obtained by Antivirus companies for analysing is very large, so doing the analysis manually is not possible. So the process of analysis needs to be automated, which will help in reducing the number of files which requires manual analysis.

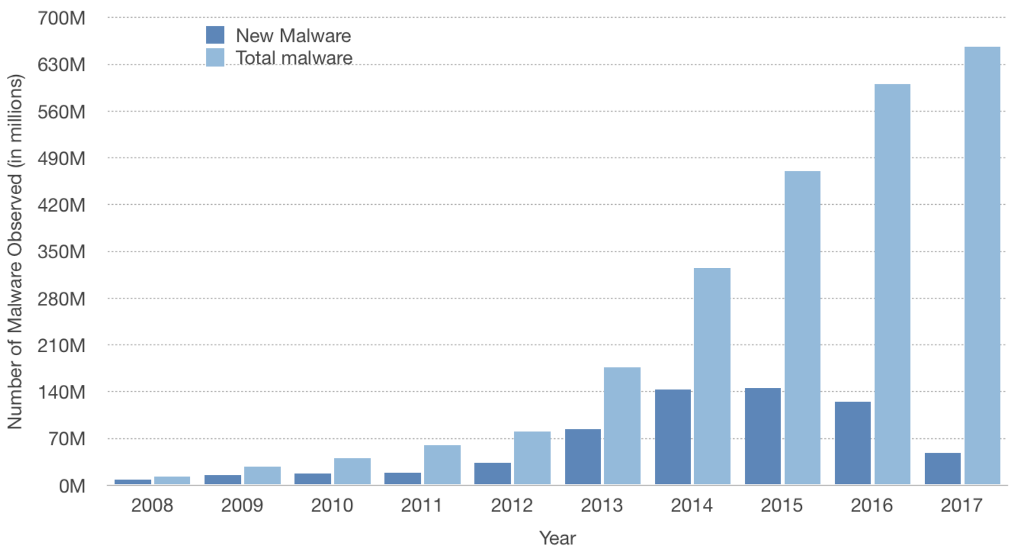


FIGURE 1.3: New and total amount of Malware observed yearly. [4]

Cohen has shown that detecting whether a given program is a malware is an undecidable problem [11]. Automated detection and analysis techniques are limited by this theoretical result and might fail in some cases. So for such cases manual intervention is required to understand new attacks and analyses evasion techniques. The new techniques learned from such manual analysis is then incorporated into the automated software improving its efficacy.

1.3 Our Objective

As it is said "A picture is worth a thousand words", in this thesis we will examine whether it holds for malwares also. Visualisations have always proven to be beneficial in getting a comprehensive view of any system or data. We are intuitively more capable of making more sense out of images than any other representation. So this thesis explores how we can generate a visual representation for representing a binary file and examine whether there are any patterns visible in those visual representations. Then it leverages those representations for classifying malware samples into their respective classes/families.

Classification of malware is helpful for the analyst as it helps them to get a better insight into the functioning of the malware. Malware samples which have similar code structure are grouped into one class. This is very helpful for analysts, because just by knowing the class/family of the malware they can have an idea about how to devise sanitation and detection techniques for that malware. Also by knowing the family to which a malware belong we have a general idea about its behaviour. This helps in sharing of knowledge between malware analysts.

1.4 Contribution of this Thesis

This thesis proposes a new method for visually representing malwares. Using this new representation technique we have been successful in effectively classifying malware to their respective families/classes. Also our technique can be used for real time classification because the pre-processing time is almost negligible and the image can be directly given as input to the classifier. With the new representation we have been able to achieve an improvement on results shown in previous work. Also we have successfully applied state of the art neural network technique for malware analysis, which was found missing in the work we surveyed.

1.5 Organisation

Chapter 2 gives a general overview of some of the popular and common malware analysis and their advantages and disadvantages. Chapter 3 discusses about neural networks and the model used for classification in this work. Chapter 4 includes the experiments performed and results in the present work. Chapter 5 discusses future possibilities in this field.

Chapter 2

Previous Work

Malware are being used to attack critical infrastructures, for espionage against a nation, for stealing private information or for conducting financial frauds. All the attacks use network as a medium. Almost all the malware detection system in industry use either signature based approach or anomaly based approach. A signature is a unique sequence of bytes that is present in the malicious binary and in the files that have been damaged by that malware [48]. Signature based methods use the unique signatures developed by the anti-virus companies using the known malware to capture the threat. This approach is fast and has high accuracy, but it fails in detecting previously unseen malware. So generally, after a new malware has infected numerous systems an analyst may be able to generate its signature. Also the signature database has to be prepared manually which is a time consuming process [27]. In anomaly based approach the anti-virus companies forms a database of actions that are considered safe. If a process breaks any of these pre defined rules, it is labelled as malicious [55]. Although with anomaly based method we are able to identify new unseen malware samples but the false alarm rate is very high.

Another method used is heuristic [6] based, in this approach analysts use machine learning techniques to train a malware classifier. Static, dynamic, visual feature representations, or a combination are used for training a classifier on a dataset composed of both malign and benign binaries. Various machine learning techniques such as Support Vector Machines, Random Forests, Decision Trees, Naive Bayes, K-means Clustering and Gradient Boost and Ada-boost have been suggested for classifying and detecting malware samples into either their respective classes or to filter out the malware that require further exhaustive analysis by an analyst. A few of these techniques, used in the literature, are discussed in this section. Rest of the chapter discusses past work on static and dynamic based approaches in the first two sections 2.1 and 2.2 and visualisation based techniques in section 2.3.

2.1 Static Analysis Approaches

Static analysis includes extraction of static features from the binary file using binary analysis tools. Matthew *et al.* [49] used details such as the list of DLLs functions called by the executable, the count of unique system calls used within each DLL, list of DLLs used by the executable, printable characters or strings encoded in the binary file and byte sequences using hex-dump as features for classification. They used Multinomial Naive Bayes algorithm to classify a malware dataset of 3265 malicious and 1001 benign samples and reported an accuracy of 97.11%. They were one of the first to try performing malware analysis using data mining techniques.

Kolter *et al.* [26] examined the classification accuracy of different machine learning techniques, such as Naive-Bayes, Support Vector Machine, Decision Tree and their boosted versions, for malware classification into different malware families using the features proposed by Schultz *et al.* [49] and showed that boosted decision trees gave the best accuracy for classification. Zhang [66] proposed an automated static analysis method for identifying metamorphic malware programs, which generally evade conventional signature-based detection methods. They computed the degree of similarity between disassemblies of two executables using the list of library function calls made by the two binaries and showed good accuracy in identifying metamorphic variants.

Supervised learning algorithms for malware detection require a significantly large amount of labelled binaries for both malign and benign classes. But in reality obtaining such a labelled dataset would be very difficult, so Santos *et al.* [48] proposed a semi-supervised learning technique to detect previously unseen malware samples. A semi-supervised algorithm is useful when the amount of labelled data available is small. It tries to learn a classifier using the available labelled data and tries to predicts the labels for the unlabelled data over a number of iterations and in each iteration some of the unlabelled data whose label predictions is above a confidence threshold is shifted into labelled data category. Byte n-gram distribution approach has been used to represent the executables. They have used LGGC(Learning with Local and Global Consistency) [67], a semi-supervised algorithm, in their work and reported an accuracy of 86% for malware detection. Although the accuracy of their model is less as compared to other models reported earlier in the literature, but they have been able to reduce the number of labelled samples required for learning while maintaining a considerable prediction results.

Siddiqui *et al.* [50] presented a novel approach of using variable length instruction sequence to classify worms from benign binaries using machine learning. They used random forest and decision trees to classify a data set consisting of 1444 worms and 1330 benign files and reported classification accuracy of 96%. Kang *et al.* [8] applied machine learning on n-opcode sequences with SVM classifier and got 98% accuracy for malware Detection. Opcodes are machine code mnemonics and techniques such as cosine similarity and critical instruction sequences have been used in their detection process. Sung *et al.* [1] explored the use of API call sequence for malware detection. They showed that all versions of the same malware share a common core signature, which can be captured through core API call sequences.

Moser *et al.* [34] proposed a scheme based obfuscation technique to explore the drawbacks of static analysis approaches. Their experiments showed that static analysis alone is not enough for effective malware analysis. Since static analysis can be easily evaded if the malware is obfuscated or packed, so we required some robust behavioural features for our analysis. This brings us to various Dynamic Analysis Approaches.

2.2 Dynamic Analysis Approaches

Common to all dynamic analysis based approaches is, execution of binary sample in a controlled environment for extracting behavioural features inside a virtual machine. Zolkipli *et al.* proposed a comprehensive approach to perform behaviour based malware analysis and classify malware into new groups using artificial intelligence techniques. They used honeypot [41] and intrusion detection system, like HoneyClients and Amun, to collect malware samples for analysis. Behaviour report for each of the collected sample was generated using virtual machine platforms like CWSandbox [63] and Anubis [25] and each report was manually analysed. Then using AI techniques the malware samples were grouped into Worms and Trojans. The only limitation was they didn't automate the report analysis, so considering the huge volume of malware generated in present times, human analysis of reports is not feasible.

Christodorescu *et al.* [9] automated the time consuming manual analysis process. They proposed a technique to mine the malicious behaviour shown by a malware that is not shown by a benign program. For this they compared execution behaviour of a malware with the execution behaviour of a set of benign samples. Since their approach gave a concise account of the malicious behaviour, so it can give great insight to security researchers for understanding a malware.

Rieck *et al.* [45] suggested a new method for automated identification of new classes of malware with similar type of behaviour (clustering) and classifying previously unseen malware to these discovered classes (classification) using machine learning. Using both, clustering and classification, an incremental approach is used to process the behaviour of large number of malware binaries. The incremental approach reduced the run-time overhead of current analysis techniques significantly. They recorded state change features such as opening a file, locking a mutex, network activity, infecting running processes or setting a registry key and thus embed the malware behaviour into a high-dimensional vector space. They used more than 10,000 malware samples, belonging to 14 different families, in their experiment. These malware samples were collected using honeypots and spam-traps. They reported an accuracy of 88% on family classification using simple SVM based classifier. A limitation of their work is that they only considered one execution path of a binary in their analysis which was stopped if the binary exits itself from the run.

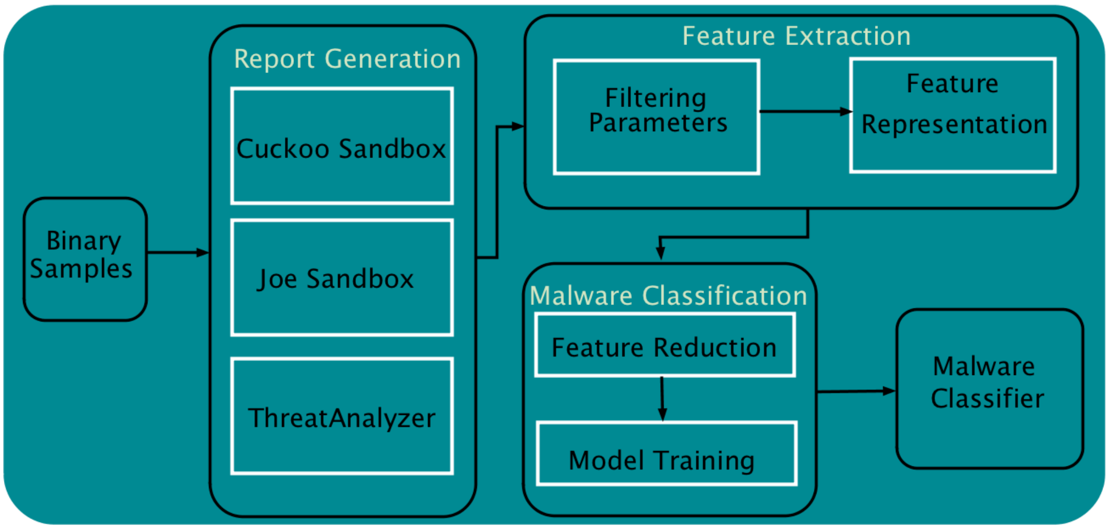
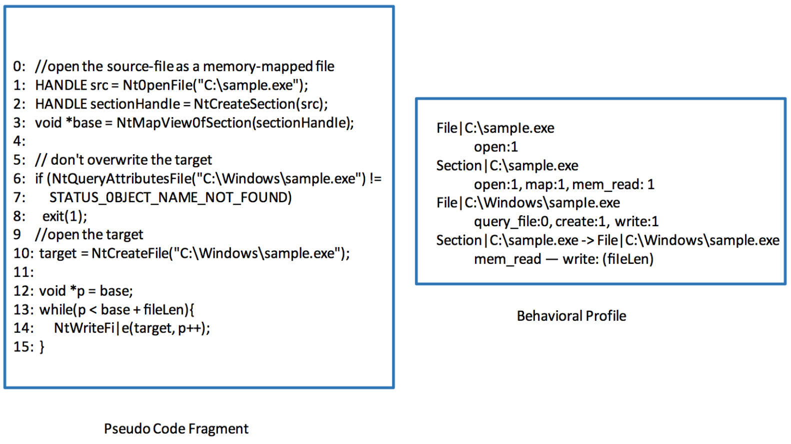


FIGURE 2.1: System Flow for Dynamic Analysis.

Youngjoon *et al.* [23] developed a novel technique by applying DNA sequence alignment algorithms (MSA and LCS) to the runtime API sequence generated by malwares and extracted some common API sequence patterns found in different types of malwares. DNA sequence alignment algorithm assisted them in capturing patterns in API call sequences that were otherwise difficult to capture. Making use of both extracted API call sequence patterns and critical API call sequences in their model helped them achieve high accuracy in detecting previously unknown malware. Since API call sequence can be extracted from almost all systems, so their method can be used to detect malware ubiquitously. They showed certain functions or API call sequence patterns existed among malware even in different categories which can be used to detect new unknown malwares. A limitation of their work is that DNA sequence alignment algorithms are highly resource and time consuming algorithms which slows down the detection process.

Anderson *et al.* [3] presented a novel approach, by modelling Markov chain graphs from the instruction traces generated by executing a binary on Ether malware analysis framework, for detection of malware. The vertices in the graphs (Markov chains) are the instructions and the transition probabilities are estimated from the data contained in the trace report. Concoction of graph kernels capture similarities between instruction trace graphs on both global and local levels and thus construct a similarity or kernel matrix between the instruction trace graphs. Local similarity between the graphs is measured using a Gaussian kernel while global similarity is measured through a spectral kernel. These similarity matrices are given as input to support vector machine algorithm for performing classification. They used a dataset comprising 615 benign samples and 1,615 malign samples in their experiment and reported classification accuracy of 96.41%. This approach albeit showing good results is limited by the very high computation time, which makes it unfit for real world application.



```
//open the source-file as a memory-mapped file
HANDLE src = NtOpenFile("C:\sample.exe");
HANDLE sectionHandle = NtCreateSection(src);
void *base = NtMapViewOfSection(sectionHandle);

// don't overwrite the target
if (NtQueryAttributesFile("C:\Windows\sample.exe") !=
   STATUS_OBJECT_NAME_NOT_FOUND)
  exit(1);
//open the target
target = NtCreateFile("C:\Windows\sample.exe");

void *p = base;
while(p < base + fileLen){
   NtWriteFi|e(target, p++);
}
```

Pseudo Code Fragment

```
File|C:\sample.exe
      open:1
Section|C:\sample.exe
      open:1, map:1, mem_read: 1
File|C:\Windows\sample.exe
      query_file:0, create:1, write:1
Section|C:\sample.exe -> File|C:\Windows\sample.exe
      mem_read — write: (fileLen)
```

Behavioral Profile

Figure 2.2: Example of Behaviour Profile

Bayer *et al.* [5] proposed a system that can be easily scaled to automatically clusters large number of malware samples into groups/classes based on their behaviour effectively. They extended Anubis [25] system to include taint tracking and additional network analysis and generated automated trace report for all malware samples using this extended system. The behavioural profiles, depicted in Figure-2.2, are created by abstracting system calls, their dependencies, and the network activities to a generalised representation consisting of operating system objects and operations that were performed on these objects for each trace report from previous step. These behavioural reports are able to characterise the activities of a program in more abstract terms. The profiles serve as input to the clustering algorithm, Locality Sensitive Hashing (LSH), [20] it is a sub-linear method for approximating nearest neighbour problem. To exhibit the scalability of the proposed method, the author clustered a dataset consisting of 75,000 malicious files in less than 3 hours.

Nari *et al.* [35] presented a framework for automated classification of malware samples into their classes using their network activities. Network trace generated on executing a binary is captured in form of pcap file, a pcap file is a network activity dump file which is used by applications such as WireShark, which is given as input to the model. From the pcap files communication information such as IP address, port number and protocol used are extracted, which altogether denotes the network flows. A behaviour graph is constructed to abstractly represent the network behaviour of a malware and the dependencies between network flows. Features like graph size, root out-degree, average out-degree, maximum out-degree, number of specific nodes are extracted from the behaviour graphs, which are used to classify malware. They used a dataset of 9610 malware samples and labelled them using 11 anti-viruses. Malware samples were scanned using all the 11 anti-viruses and the label which appeared above a fixed threshold was assigned to a sample. Classification algorithm provided in WEKA java library were used to classify malicious files [13] and it was shown that J48 decision tree gave an accuracy of 95.23% which surpassed other classifying algorithms.

From the above discussion, it is evident that either static analysis or dynamic analysis alone is not enough to accurately and efficiently classify malware samples. When used alone these methods can be easily evaded using the various analysis evasion techniques such as code obfuscation or various execution stalling techniques. Also with dynamic analysis alone we are not able to explore all the execution paths of an executable. So another field of study forked out in the area of malware analysis, hybrid analysis, which used a combination of static and dynamic features simultaneously to improve detection and classification accuracy for malwares.

2.3 Hybrid Approaches

Santos *et al.* [47] proposed a novel approach of using both static and dynamic features collectively to train a malware classifier, named OPEM. They combined frequency of occurrence of operational codes, static feature, with execution trace of an executable retrieved by monitoring operations performed, system calls and exceptions raised, dynamic feature, to train their model and showed that hybrid approach performs better than both approaches when run separately. They validated their approach using two different datasets along with a multitude of machine algorithms, Decision Tree, KNN, Bayesian network and SVM.

Islam *et al.* [21] also proposed a hybrid model for classifying the binaries into benign and malicious files, integrating both dynamic based and static based features. They used static features such as printable sting information and frequency of function length and dynamic features such as API parameters and API function names. For testing their model they used 2939 malign samples and 541 benign samples. They used integrated meta classifiers such as SVM, IB1, DT and RF for classifying malware samples and reported an accuracy of 97.055% . Their results were an improvement on the previous results.

Hybrid approaches are very promising as they provided significant improvement over just static and just dynamic approaches.

2.4 Visualisation Approaches

Several Hex viewer tools were already available to visualise and edit a binary file, but they just display the file in hexadecimal and ASCII formats and fails to convey any structural information to the analyst. First we will discuss how various researchers tried to visualise binaries.

Helfman [16] applied the dotplot data visualisation technique to software programs and showed that visualisation can be helpful for identifying software design patterns. Dotplot is a technique of visualising patters in string matchers and gives us a visual overview of the structure of an enormous system. Such visual patterns are useful for design of software systems through *Successive Abstraction* - a design pattern that helps to eliminate redundancy. A sequence is broken down into tokens and the tokens are plotted around the two axes with a dot where the tokens match and blank where they don't match.

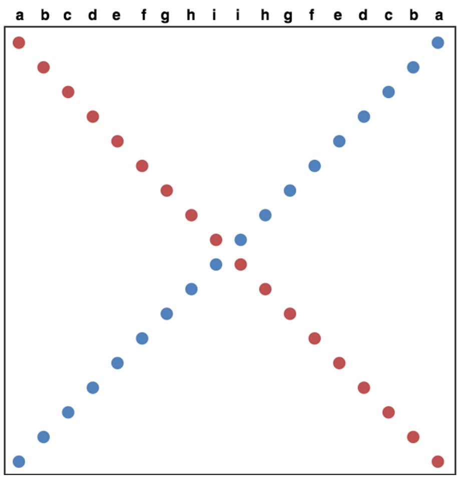


FIGURE 2.3: Dot plot of a palindrome. [16]

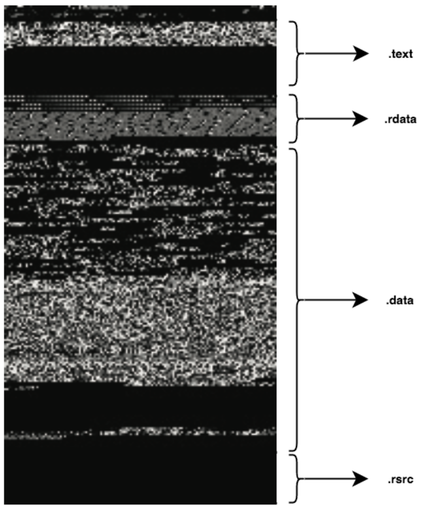


FIGURE 2.4: Various Sections of Binary File. [12]

Binary files are assorted collection of primitive types such audio, images, video, text and code. Conti *et al.* [12] proposed an automated binary mapping technique using Byte plot visualisation, that is a technique to map or classify regions within a file. A Byte plot is generated by considering each byte in a file as a pixel value, 0 as white and FF as black. Using Byte plot they automated the problem of finding the start and stop offsets of each distinct region within a binary and also predict the its primitive data type. Primitive type is defined as a homogeneous section of a binary which has related binary structure like random number sequences, similar data groups. The Byte plot visualization helped in finding distinctive patterns, even transformations such as encoding, encryption are applied. Their work was limited only to identify primitive patterns in a binary file. From Figure-2.4, we can observe that different sections of a binary evince different textures.

Yoo *et al.* [64] tried to detect and visualise viruses embedded inside an executable using self organising maps (SOM) [24], without using any signature information. SOM is a type of Artificial Neural Networks (ANN) used for visualising multi-dimensional data. SOM combines vector quantisation and vector projection and creates a topologically ordered mapping of the data. They showed that just like every person has a unique DNA, every virus family has a virus mask like DNA which is visible when the binary is visualised using SOM. They used these unique features identified using SOM to predict the virus family. They showed that the variant of each virus also can be covered with each virus mask, which is produced by SOM.

Nataraj *et al.* [37] were the first to explore the use of byte plot visualisation for automatic malware classification. They converted all the malware sample to greyscale byte plot representations and extracted texture based features from the malware image. They used an abstract representation technique, GIST, for computing texture features from images. Their dataset consists of 9,458 malware samples belonging to 25 different classes, collected from Anubis [25] system. They used the global image based features to train a K-Nearest Neighbour model, with Euclidean distance as distance measure, to classify malware samples into their respective classes and an accuracy of 97.18% was obtained. The results obtained are comparable to those shown by dynamic or hybrid analysis and showed that image processing based malware classifying techniques can classify malware more quickly than existing dynamic approaches. Inspite of that, their approach has a huge computational overhead of calculating texture based feature.

Han *et al.* [14] proposed a new way of visualising malware using op-code sequences to detect and classify malware samples. They used image matrices to visually represent malware which assisted in detecting features of malware and also in finding similarities between different samples swiftly. First the binary file is disassembled using IDA Pro or OllyDgb and the op-code sequence is divided into blocks. Then they used two hash functions and for each block of op-code sequence, computed a coordinate and the RGB value using the two hash functions. Then they plotted all the RGB values to their corresponding coordinates in a matrix of dimension 8 by 8 to get an image matrix. They used "selective area matching" for calculating similarities between image matrices and evaluated their model on malware samples from 10 different families. Their results showed that image matrices of malware from the same family had higher similarity score than with malware from different family, image matrices can effectively classify malware families.

Makandar *et al.* [30] converted malware into a 2-dimension greyscale image and classified the samples using texture based features. They extracted texture based global features using Gabor wavelet transform and GIST and used Mahenhur dataset [30] for experiments, which is comprised of 3131 binaries samples from 24 unique malware families. They used The Artificial Neural Networks (ANN) for classifying malware and reported an accuracy of 96.35%.

Liu *et al.* [28] proposed an incremental approach to automatically assign malware to their respective families and detect new malware. They used a combination of feature greyscale byte plot, Op-code n-gram, and import functions. The decision making module uses these features to classify malware samples to their respective families and to identity new unknown malware. They used Shared Nearest Neighbour (SNN) as the clustering algorithm to detect new malware families. Their model is evaluated on dataset consisting of 21,740 malware samples from 9 different families and reported classification accuracy of 98.9%, and detection accuracy of 86.7%.

Chapter 3

Neural Network Background

3.1 Basics

This chapter is intended to give basic concepts behind the techniques used in this thesis. This theory will help is the subsequent understanding of concepts used in later chapters where the theory is used.

The basic task of the project is classification of malware to their respective families. This task can be performed by a human who has thorough understanding of malware. As the number of malware can be theoretically infinite, there is need of a machine to perform the task. So we need a digital classifier to perform the same task. Classifiers can be of many types. A classifier can be made using heuristics generated by a human by studying different types of malwares. For example if some particular types of headers are found in a file, we may know with high probability that it is a malware. But again there may be many different ways in which malwares can be made and its a hard task to create heuristics for all the different types. Also we may need to examine the malware first to update our heuristics, and by then many machines might already been infected. So there is a need for classifiers that learn the patterns and key features in malware automatically and generalize well. To solve this issue we have used artificial neural networks in our approach.

3.1.1 Artificial Neural Networks (ANNs)

ANNs are somewhat motivated by how the actual brain works along with statistics and applied maths. Human brain contains a large amount of interlinked brain cells called neurons. Information is transferred from one neuron to other in the form of electric signals. Brain receives many sensory inputs and the neuron web manipulates it to the concious part of brain. Each neuron can be modelled as a non-linear function f(x), which takes input signals from all the neurons it is connected to, in the form of electric signals. These signals get manipulated in terms of strength while travelling from one neuron to other depending upon the chemical composition of the connection. It then outputs a signal y if the collective input signal is greater then a certain threshold, which is then taken as input by other neurons, as shown in fig. 3.1. This goes on in a very complex connected structure of neurons resulting in a very complex function. The input of a neuron X can be written as $w_1x_1 + w_2x_2 + ...w_nx_n$, where w_is are the manipulating factors. The neuron then has a continuous activation function, $\phi(X)$. As we experience new things and learn the connections get modified, new connections are formed. This is how the learning is believed to happen.

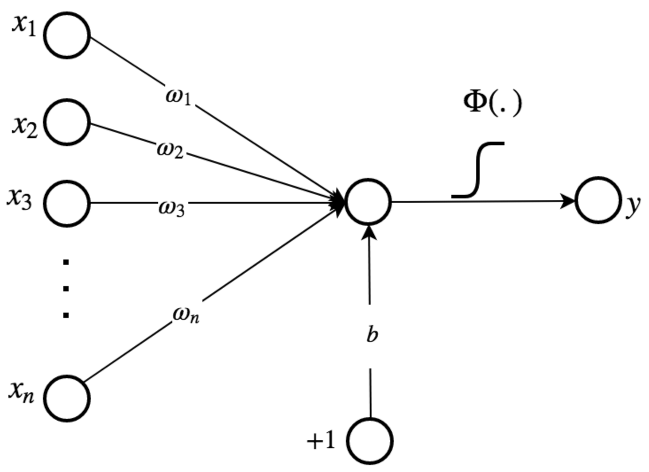


FIGURE 3.1: Graphical representation of how a neuron works in simple terms. In this y is the output signal, ϕ is the activation function, x_i are the inputs from other connected neurons, w_i are the respective weights. B is to incorporate threshold. [17]

Similar is the structure of ANNs. An ANN comprises of connected units called artificial neurons. These are organised in layers. Data travel from the input to the output layer, traversing through many layers. Each connection between two neurons has a weight w_i , typically a real number between 0 and 1 governing how the input is modified before being transmitted to next neuron. In addition to that, there might be threshold or a limiting (activation) function at each node. These layers can be fully connected (each neuron is connected to all neurons in the next layer), or partially connected(chosen randomly). So each neuron can be represented by a function

$$y = \phi(\sum_{i=1}^{n} w_i x_i + b)$$

where y is the output, ϕ is the activation function, n is the number of neurons feeding into a neuron, w_i is the value of weight of the i^{th} connection, x_i is the output of the neurons feeding into the neuron and b is the threshold. One may think of b as a neuron with constant output of value -1. The value of b varies from one neuron to other, allowing different activation thresholds for different neurons. Fig 3.2 shows a full neural network. There are broadly three types of layers, input layer, hidden layer and output layer. Input layer deals with the raw information provided as input to the network. However the hidden layers get some other representation of the layer determined by the weights and activation functions from previous layers. This is important because hidden layers are therefore free to learn their own representation of input and thus simplify the classification problem for the subsequent layers. The whole network can be thought of as a set of layers where each layer learns a different representation of the input(removing irrelevant information and only keeping useful information for the task). These artificial neurons are also called Perceptron and when there are multiple layers of neurons, the whole neural network is called Multilayer Perceptrons (MLPs). Neural networks in which outputs from each layer are fed as inputs to next layer only are also called Feed Forward Networks.

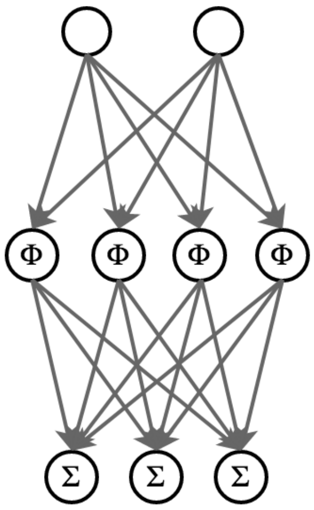


FIGURE 3.2: A graphical representation of ANN with only one hidden layer. [17].

3.1.2 Learning Process

As mentioned earlier, we want our classifier to learn features automatically. ANNs does this by adjusting their weights such that its output gets closer to the desired output for each example we provide. There are two types of learning process one is supervised, another is unsupervised. In supervised learning we know the actual output corresponding to each input and we use that information during training. On the other hand, in unsupervised learning we do not know the actual outputs, we just want to cluster similar inputs/separate out different inputs, or learn some distribution from which inputs are picked. Supervised learning can be used when we have labelled data as was our case. In training phase we feed in examples from dataset one by one and modify the weights every time based upon the error made by the model. This can be thought of as method of learning through mistakes. This process is repeated until difference between the network outputs and desired outputs become very small if at all its possible. There are many ways of doing this. One popular way to start is to randomly initialise weights using some distribution. An input is fed to the network and output, y, is obtained. Next error is calculated describing how far is y from the actual desired output y'. Our goal is to minimise this error as fas as we can. How this error is calculated is very important as it defines our optimisation problem. One of the techniques used is calculating mean squared error. Based on this error the weights are then updated using technique called back propagation. Back propagation is based upon the fact that the optimum change in the weights (resulting in maximum reduction in error) would be when the error decrease maximum by taking a small step delta. From linear algebra, we know that the error will decrease the fastest in the direction where the negative slope of error function is maximum, considering fixed size step. Also we do not want to miss the minimum, thus we have to take small steps in this direction until we reach the minimum. The training happens in batches. One batch includes all the examples in data set or a fixed set of example randomly chosen from dataset. We keep feeding batches until we reach sufficiently low error or the error stop showing further change.

3.1.3 ANNs in image recognition

Image recognition is one of the most important applications, where neural networks have been successfully implemented and have showed really great results. ANNs are already being used as popular practice in industries, health care, risk management ets. As previously discussed, the structure of ANNs include input layer, one or more hidden layers, and a output layer. Many tasks in image recognition can become very complicated and thus cannot be learned using single hidden layer. If we think of each layer as a function f_i on the input, we can learn more and more complex functions by using multiple layers $....f_4(f_3(f_2(f_1)))$. Fig 3.3 shows an example of how different layers can contribute to overall task in case of image recognition. Thus increasing the number of layers lets us learn more complicated representations, or in other words learn harder tasks. The number of layers in neural network represents its depth. Thus models using many hidden layers are called deep models and this learning is termed as deep learning. Using deep learning in image recognition we now get much more accuracy then any previous methods. In the next section we discuss deep nets in details.

3.2 Deep Networks and deep learning

Ivakhnenko and Lapa in 1965, were the first to propose a working supervised learning algorithm for feed-forward, deep, multilayer perceptrons. Since then a lot of research was carried out over the years on various models but the first major change was observed in 2006. As the computers were now capable of carrying out huge amount of computations. For the first time it was showed how a many layered feed-forward network can be pre-trained and used for various tasks once trained. Since then it has become the most popular technique in many practical domains. To understand the capabilities of neural networks, first let us go back a little and see how deep learning is different from other classification techniques and how it overcomes their limitation.

The simplest classifiers are the linear models. Linear models can be applied to problems where the classification task can be solved by applying some linear transformation to input. One popular example is Linear Regression. In linear regression we have a scalar dependent variable y (output) and one or more explanatory variables denoted by X(input). Using many data points we try to predict a model β such that

$$y_i = X_i^T \beta + \epsilon_i i = 1, 2, 3.....n$$

where ϵ_i denotes a random variable, modelling the noise involved in collecting data. Linear regression is used in many statistical models. For example, modelling of constants in some polynomial relationship in physics using data points generated from experiments. As the name suggests, the issue with linear regression is it can only solve linear dependencies. What if the relationship is of the form *tanh*, or some high degree polynomial, or mixture of these two. To model such relationships we need to incorporate non-linear function in our model. This lead to development of many non-linear machine learning classification techniques. The most widely used technique for image recognition and classification was SVMs (Support Vector Machines). In SVM, we first transform input using some non-linear function, $\phi(x)$, popularly known as the kernel trick, and then apply linear model to the transformed input. The best choice for function ϕ can be anything and totally depends upon the dataset. If we pick a very generic ϕ (infinite dimensional ϕ for example), we can always fit any training set, but this then leads to poor generalization. Thus we have to manually engineer ϕ, and that is what people used before the advent of deep learning. In deep learning we learn ϕ. This is great for two reason. We let the model to learn any type of non-linear function. We don't need to worry about coming up with a complicated ϕ. Next we see a typical feed-forward deep learning model.

3.2.1 Deep Learning Model

In deep learning output y can be modelled as $f(x;\theta,w)$, where θ is a parameter to learn ϕ and w are the weights that map ϕ to y. Thus,

$$y = \phi(x;\theta)w$$

We use optimisation algorithms to find a good representation of y. Backpropagation is one such algorithm which is used to solve the optimization problem. Like many other models there are many parameters that we need to decide before training the neural network. These are design choices and are termed as hyperparameters. We need to choose the cost function, the form of output units, the activation function, the depth of network, width of each hidden layer, the connectivity of layers, etc. Common choice for activation function is softmax function $p_j = \frac{exp(x_j)}{\sigma_k exp(x_k)}$. Softmax function maps k-dimensional real-valued vector X to k-dimensional vector X' in range [0,1] and sum of all values in vector is 1. For the cost function popular choice is cross entropy defined as $C = -\sigma_j d_j log(p_j)$. Where d_j represents the desired probability of the output and p_j is the probability output from the neural net. All these parameters depend upon the problem, dataset, computational capability etc. Because of large number of hidden layers deep learning techniques suffer from over-fitting. To avoid over-fitting various techniques like weight-decay, sparsity are applied during training.

DNNs work good with image recognition and classification but suffers from the curse of dimensionality and thus can only be used for low dimension images. For example an image as small as 32*32 pixels would require 3072 weights at the input. Increase in size, results in huge number of weights and thus renders it computationally hard. Also because of their flat structure DNNs are unable to capture the correlation in neighbouring pixels and thus fails to take full advantage of the 2D structure of images. These limitations were overcome by Convolutional Neural Network(CNNs).

3.3 Convolutional Neural Network (CNNs)

CNNs were inspired from the biological structures from cat's visual cortex. It is comprised of complex arrangement of cells as studied by Hubel and Weisel in ther work [18], called a receptive field. These cells behave as filters and exploit local spatial correlation in the images. Two types of cells have been identified. One type mainly extracts edges like features and other are complex filters. CNNs are designed to exploit features in similar way.

3.3.1 Structure

The structure of a CNN contains a few sets of convolution layers followed by pooling and in the end fully connected layers for classification. Fig 3.6 shows a typical LeNet for image classification. The first step is convolution. A number of filters, initially randomised and later learnt during training, are applied to the image through convolution. This step extracts important features from the image. For example one filter can extract all the edges, other can extract contrast. Outputs from each filter are stacked one after the other to form a 3D net. Non-linear function is applied to introduce non-linearity and limit the output values. After that spatial pooling is applied to reduce the dimensionality and retain only the important features from the image. Pooling also makes the classier more invariant to small transformations and noises. And in the end we have a scale invariant representation of our image, which is very important. There are various types of pooling used such as sum, average, max, etc. Max pooling is the most used technique. This process can be repeated multiple times as shown in the figure-3.3. In the end we have fully connected Multi-layer Perceptron, MLP, with soft-max function. The MLP does the task of classifying the set of rich features. All the filters and weights of MLP are learnt using back-propagation only. As same filter is applied over the whole image the classifier becomes invariant to the position of the object.

CNNs are best suited for classification using image recognition tasks. But as the number of classes increases the task becomes harder. The net size increases leading to various issues like vanishing gradient problem. Front layers get harder and harder to train. To tackle this Microsoft introduced ResNets in 2015. The network used by Microsoft had 152 layers and performed best in ILSVRC 2015 image classification competition [15].

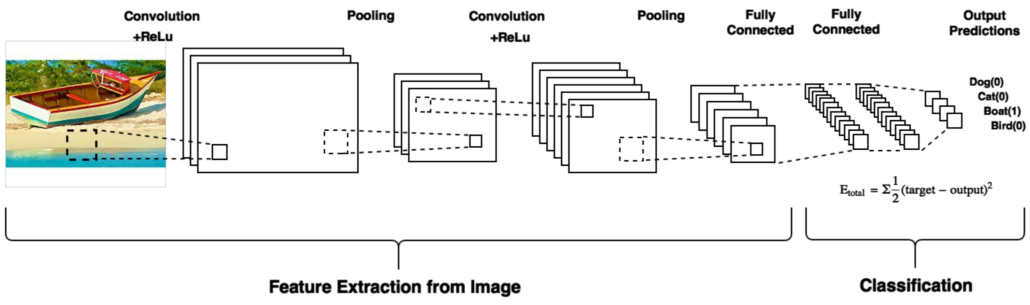


FIGURE 3.3: An Example of convolution network with two convolutional layers and two fully connected layer, Performing classification between dog, cat, boat and bird [10].

3.3.2 Training

Training CNNs is computationally more efficient than training a DNN and involves same steps. So same techniques and code be used with small modifications. The process starts with initialising all weights and parameters to random values. Next input is fed in and output gets calculated as a vector of probabilities corresponding to each class. Next the errors is calculated as some of probability differences. Next back propagation is done for updating weights and parameters. Weights are updated in proportion to their contribution to error. The difference lies when filters need to be updated. Note that the only difference between CONV layer and FC layer is that neurons are only connected to a region in the previous layer and same connections are used for all the regions. So an average can be taken to compute the update for all the inputs sharing common weights.

3.4 Deep Residual Networks (ResNets)

Weizmann Institute of Science has recently published a mathematical proof that establishes the utility of having deeper network as compared to wider. In the future, deep nets are going to be deeper. As standard fully connected MLPs get deeper, problems such as exploding/vanishing of gradients [7] start appearing, that disturbs the convergence. Degradation problem also starts to appear with increasing depths, accuracy gets saturated and then degrades rapidly. In residual networks, this is tackled by feeding inputs of a layer to other layers ahead. Which help the stack of layers in between to learn a particular mapping. So instead of just hoping that deep nets will divide stack of layers and learn desired mappings better on their own as we increase the depth, we explicitly make a stack of layers learn the mapping. Fig. 3.7 shows a unit of such network. As shown, input is shorted to later layers. These stack of layers generally contains convolutional networks.

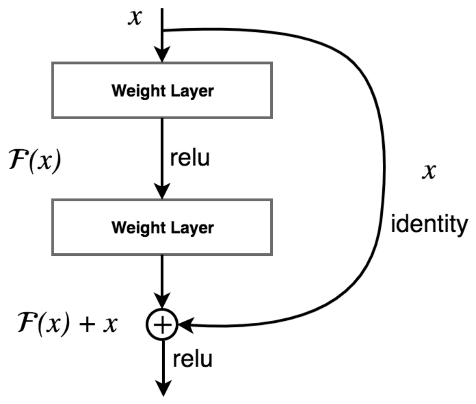


FIGURE 3.4: A building block of ResNet. The no of MLP layers might differ. Many such blocks are stacked with varying MLP layers. [15].

A stack of layers takes a mapping $H(x)$ in the neural net, given x is input to the first layer. Using the hypothesis that multiple non-linear layers can asymptotically approximate complicated functions (this hypothesis is still an open question) [33], one can say same stack of layers can take up form of $H(x) - x$ (based on the assumption of compatible dimension). Thus the layers approximate new function, $F(x) = H(x) - x$ the input to the next stack of layers remain the same $F(x) + x$ i.e. $H(x)$. This passing of input to layers ahead is purely based upon the counter-intuition to the degradation problem. As we see both function can be learnt by the network, but difference may lie in the ease of learning. The structure of one stack of layers may vary. ResNets have been able to reduce the degradation problem and thus producing networks as deep as 150 layers.

Chapter 4

Methodology and Experimental Results

A large number of malware samples are created using polymorphic and metamorphic techniques, so different malware will possess some structural similarity, share some attributes and behaviour. Despite most of the new malware being very similar to the known malware samples signature-based anti-virus programs still fail to detect them because they do not take into account structural and behavioural properties for detection. Image based detection and classification proves to be effective, because it leverages the structural similarity between the known and new malware samples. Moreover, visual analytics helps analysts to recognise patterns in malwares' code and behaviour, thus helping them to come up with better results. We used a novel way of representing a binary file as a coloured image matrix to analyse and classify malwares. Our approach require no code extraction or decompilation or execution. We converted the malware samples into images and used machine learning algorithm to classify them into their respective classes.

Basic outline:

- Data Collection and Preprocessing
- Malware to Image Conversion
- Training a neural network model
- Testing the model

Later in this chapter we will discuss about the malware dataset in section 4.1 and the representation used in section 4.2 and finally experiment and results in section 4.3.

4.1 Dataset

4.1.1 Collection

We collected more than sixty thousand malware samples from various malware repositories such as Malshare[31], VirusShare[56], VirusTotal[57]. These portals collect malware using Honeypots and also users around the world submit files over them for analysis and sharing malware samples. Then we removed the duplicates from the malware collection by comparing MD5 hash of each file. To assert that all of those samples were valid malware samples, we analysed them using VirusTotal and selected only those which were labelled as malicious by more than 50% of the anti-virus engines in the VirusTotal report. After this we were left with more than forty thousand valid malware samples.

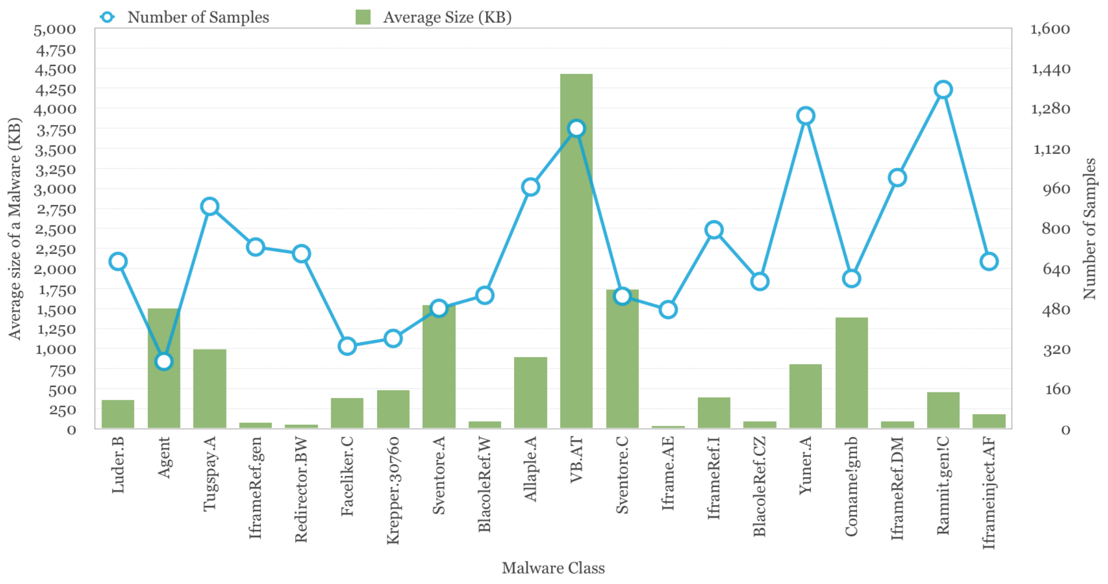


Figure 4.1: Dataset Statistics

4.1.2 Labelling

Since we are using supervised learning for classification, therefore we require labelled samples. So for labelling these samples we used the label provided by Microsoft anti-virus engine in the VirusTotal report. For this we had to provide the MD5 hash of the binary

Table 4.1: The Dataset

Malware Type	Malware Family	Number of Samples
TrojanDropper	Sventore.C	1,503
	Sventore.A	1,654
Exploit	IframeRef.gen	2,268
	IframeRef.DM	3,134
Worm	Yuner.A	3,906
	Allaple.A	3,017
	VB.AT	3,748
Trojan	Redirector.BW	2,185
	Iframe.AE	1,487
	BlacoleRef.W	1,665
	IframeRef.I	2,483
	BlacoleRef.CZ	1,838
	Iframeinject.AF	2,088
	Comame!gmb	1,875
Virus	Krepper.30760	1,127
	Luder.B	2,088
	Ramnit.gen!C	4,233
Backdoor	Agent	840
TrojanClicker	Faceliker.C	1,031
TrojanDownloader	Tugspay.A	2,775
Total		44,945

file to VirusTotal. If that file had already been analysed by their engine then it would return a report otherwise we had to upload the file to get the report. There were a few samples in our dataset that Microsoft anti-virus engine failed to classify, we decided to drop such samples to maintain consistency of labels.

4.1.3 Binary File to Image

Generally, all binary files can be considered as a sequence of ones and zeros. So first we converted each binary file into a string of ones and zeros. Then we divide the content of the string into units of 8 bits each, that is, 8 characters for every unit. Now considering each unit as a Byte take their upper and lower nibbles as indices of a 2-dimensional colour map that stores RGB values corresponding to that Byte. Repeating this for every unit we will get a sequence of RGB values (pixel values) corresponding to each Byte in the binary file. Now we converted this sequence of pixel values into a 2-dimensional matrix, thus getting an image representation for a binary file. For this work we have fixed the width of the matrix to 384 Bytes or units. So the height of image is variable and depends on the size of the binary file.

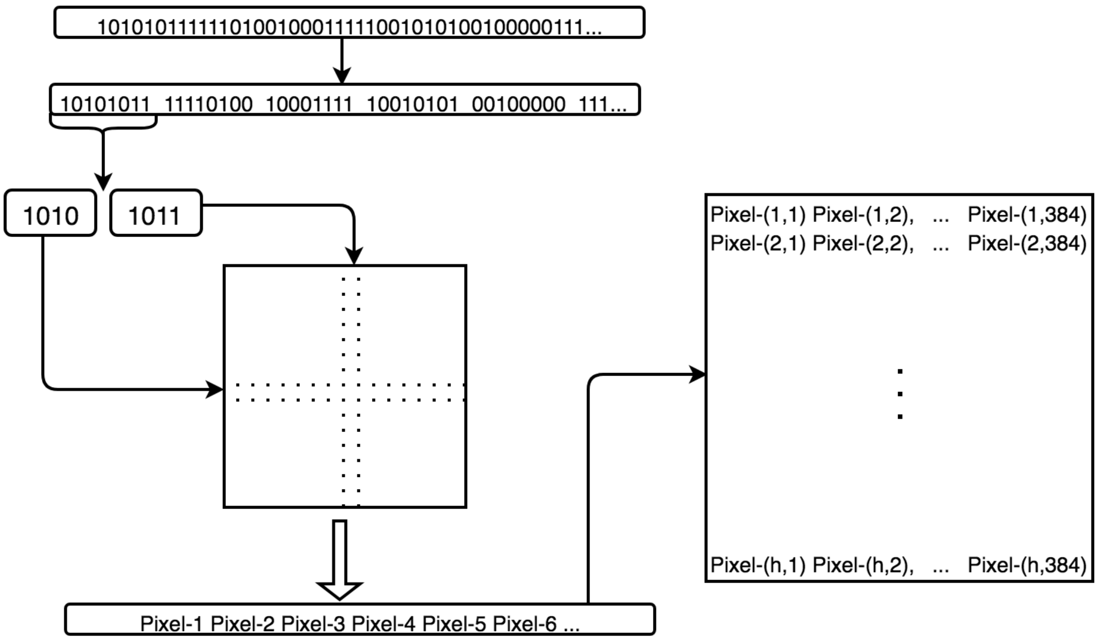
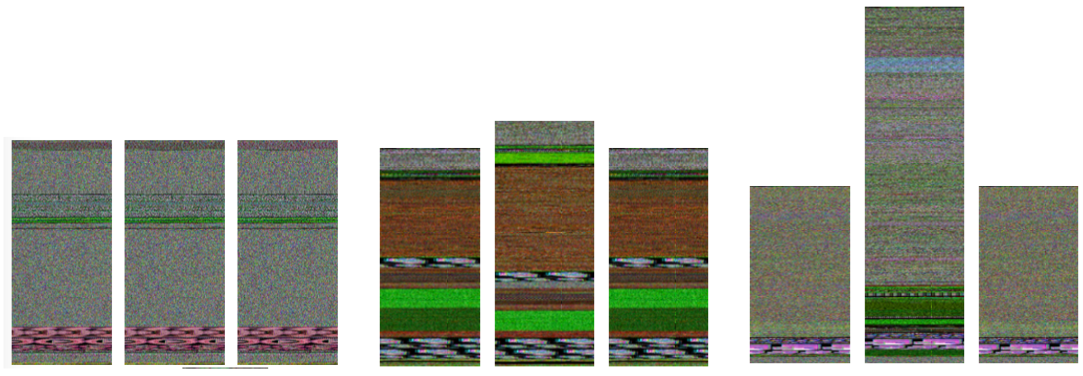

10101011111110100100011111001010100100000111...
10101011 11110100 10001111 10010101 00100000 111...
1010
1011
Pixel-1 Pixel-2 Pixel-3 Pixel-4 Pixel-5 Pixel-6 ...
Pixel-(1,1) Pixel-(1,2), ... Pixel-(1,384)
Pixel-(2,1) Pixel-(2,2), ... Pixel-(2,384)
Pixel-(h,1) Pixel-(h,2), ... Pixel-(h,384)

FIGURE 4.2: Conversion of binary file to coloured image.



(a) Allaple.A (b) Tugspay.A (c) Yuner.A

FIGURE 4.3: Samples Images of Malware belonging to different Families.

Figure-4.3 shows image representations of malware from three different families. From the images we can easily discern that there is some textural similarity among the malware of the same class and also some differences among malware from different classes. This is due to the recycling of old malware code, as most of the new malware created reuse the preexisting code and to evade detection by signature matching they use techniques such as obfuscation, packing or encryption. Further we will see how we could leverage these similarities to classify malware samples to their respective families.

4.2 Model

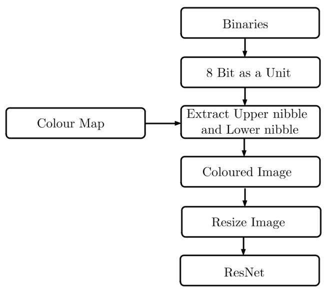


Figure 4.4: Flow of Algorithm.

Neural networks have been very successful in finding meaning or recognising patterns from a set of images. We used an 18-Layer Residual Neural Network for training a classifier to group malware into their classes [44]. Residual learning means that every layer is responsible for fine tuning the outputs from its previous layer.

The images are resized to a common size of 32 rows and 32 columns. The dataset comprising of samples from 20 classes is divided and 30106 samples are used for training and 14829 samples are used for validation. The training set samples are fed to an eighteen layered residual network. The ResNet comprises of a layer of convolution, followed by a layer of max polling, followed by sixteen layers of convolution and a layer of average pooling.

We initially used CNN for classification, it gave about 95.24% test accuracy. The results were comparable with those in some of the previous work but on using ResNet we got test accuracy of 98.21%, which is a significant improvement both on CNN and previous work.

Table 4.2: Experimental Results

Classification Model	Malware Dataset			Accuracy
	Samples	Train Set	Test set	
CNN	44935	30106	14829	95.24%
ResNet	44935	30106	14829	98.21%

We also compared our model with that proposed by Natraj et al. [37] who used byte plot representation and classified malware using GIST [54] features extracted from the images. GIST are a set of global features which tends to capture textural similarity between images. He used a K-Nearest Neighbour model to classify a malware dataset of 9339 samples belonging to 25 classes. He got classification accuracy of 97.18% using his approach for the following dataset table-4.3.

For implementing the proposed system we used python programming language and a plethora of different python libraries. The entire code for dataset preparation and labelling using VirusTotal API service is implemented in python. PIL library of python was used for image generation and resizing. For learning models we used keras with tensorflow as background. Scikit-learn was used for cross validation, performance evaluation and matplotlib along with seaborn was used for plotting.

Malware Type	Malware Family	Number of Samples
Worm	Allaple.L	1591
	Allaple.A	2949
	Yuner.A	800
	VB.A T	408
PWS	Lolyda.AA1	213
	Lolyda.AA2	184
	Lolyda.AA3	123
	Lolyda.AT	159
Trojan	C2Lop.P	146
	C2Lop.gen!g	200
	Skintrim.N	80
	Alueron.gen!J	198
	Malex.gen!J	136
TDownloader	Swizzot.gen!I	132
	Swizzor.gen!E	128
	Wintrim.BX	97
	Dontovo.A	162
	Obfuscator.AD	142
Backdoor	Agent.FYI	116
	Rbot!gen	158
Dialer	Adialer.C	122
	Dialplatform.B	177
	Instantaccess	431
Worm:AutoIT	Autorun.K	106
Rogue	Fakerean	381
Total		9339

Table 4.3: Malimg dataset from Natraj et al. [36]

4.3 Experiment and Results

4.3.1 Experimental results for CNN

Initially we tried to classify the malware images using Convolutional Neural Networks. We used a 4 layer CNN with 2 convolutional layers and 2 dense layers. This gave us an accuracy of 95.24%. The table-4.4 lists the precision and recall score for each class.

Classes	Precision	Recall	F1-score	#
0	0.981268	0.965957	0.973553	705
1	0.99308	0.97619	0.984563	294
2	0.997847	0.997847	0.997847	929
3	0.78125	0.790139	0.785669	791
4	1.0	0.957627	0.978355	708
5	0.650273	0.760383	0.701031	313
6	0.973333	0.99455	0.983827	367
7	0.994071	0.994071	0.994071	506
8	0.990403	0.923077	0.955556	559
9	0.998944	0.98954	0.99422	956
10	0.98677	0.996072	0.9914	1273
11	0.998106	1.0	0.999052	527
12	1.0	0.993927	0.996954	494
13	0.990172	0.98533	0.987745	818
14	0.99308	0.97619	0.984563	588
15	1.0	1.0	1.0	1324
16	0.983498	0.996656	0.990033	598
17	0.879959	0.859841	0.869784	1006
18	0.995652	0.980029	0.987779	1402
19	0.72517	0.794337	0.758179	671
Average	0.95482	0.952458	0.95338	14829

TABLE 4.4: Results for CNN model.

Although the results are good but they fall short in accuracy percentage when compared with those in some previous work. Since deeper network give better results but CNN suffers from vanishing gradient problem, so we moved to ResNet, residual networks. ResNet have been able to handle the vanishing gradient problem and give better results than CNN on image classification.

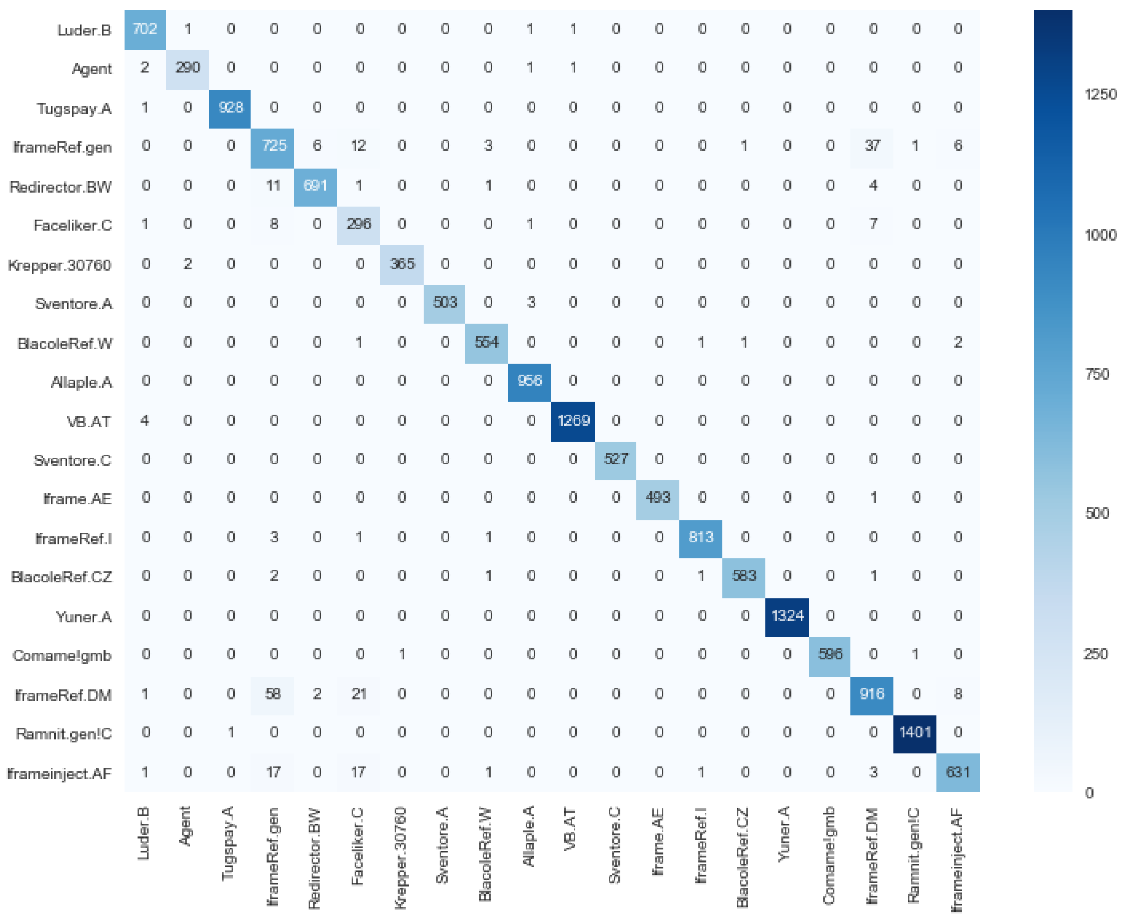
4.3.2 Experimental results for ResNet

We used an eighteen layered ResNet for malware image classification consisting of 17 convolutional layers and 1 max pool layer. This gave us an accuracy of 98.206%. The table-4.5 lists the precision and recall score for each class.

Classes	Precision	Recall	F1-score	#
0	0.985955	0.995745	0.990826	705
1	0.989761	0.986395	0.988075	294
2	0.998924	0.998924	0.998924	929
3	0.879854	0.916561	0.897833	791
4	0.988555	0.975989	0.982232	708
5	0.848138	0.945687	0.89426	313
6	0.997268	0.99455	0.995907	367
7	1.0	0.994071	0.997027	506
8	0.987522	0.991055	0.989286	559
9	0.993763	1.0	0.996872	956
10	0.998426	0.996858	0.997642	1273
11	1.0	1.0	1.0	527
12	1.0	0.997976	0.998987	494
13	0.996324	0.993888	0.995104	818
14	0.996581	0.991497	0.994032	588
15	1.0	1.0	1.0	1324
16	1.0	0.996656	0.998325	598
17	0.945304	0.910537	0.927595	1006
18	0.998574	0.999287	0.99893	1402
19	0.97527	0.940387	0.957511	671
Average	0.982523	0.982062	0.982176	14829

Table 4.5: Results for Residual Network model.

As expected residual network gave an improvement over the CNN results. Also the results are an improvement on previous results achieved in classification through image representation. Figure-4.5 shows the confusion matrix corresponding to the above experiment. Confusion matrix help us to better understand the results, each row of the matrix represents an actual class and each column of the matrix represents a predicted class, and the count in each element signifies the number of images predicted.



	Luder.B	Agent	Tugspay.A	IframeRef.gen	Redirector.BW	Faceliker.C	Krepper.30760	Sventore.A	BlacoleRef.W	Allaple.A	VB.AT	Sventore.C	Iframe.AE	IframeRef.I	BlacoleRef.CZ	Yuner.A	Comame!gmb	IframeRef.DM	Ramnit.gen!C	Iframeinject.AF
Luder.B	702	1	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0
Agent	2	290	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0
Tugspay.A	1	0	928	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
IframeRef.gen	0	0	0	725	6	12	0	0	3	0	0	0	0	0	1	0	0	37	1	6
Redirector.BW	0	0	0	11	691	1	0	0	1	0	0	0	0	0	0	0	0	4	0	0
Faceliker.C	1	0	0	8	0	296	0	0	0	1	0	0	0	0	0	0	0	7	0	0
Krepper.30760	0	2	0	0	0	0	365	0	0	0	0	0	0	0	0	0	0	0	0	0
Sventore.A	0	0	0	0	0	0	0	503	0	3	0	0	0	0	0	0	0	0	0	0
BlacoleRef.W	0	0	0	0	0	1	0	0	554	0	0	0	0	1	1	0	0	0	0	2
Allaple.A	0	0	0	0	0	0	0	0	0	956	0	0	0	0	0	0	0	0	0	0
VB.AT	4	0	0	0	0	0	0	0	0	0	1269	0	0	0	0	0	0	0	0	0
Sventore.C	0	0	0	0	0	0	0	0	0	0	0	527	0	0	0	0	0	0	0	0
Iframe.AE	0	0	0	0	0	0	0	0	0	0	0	0	493	0	0	0	0	1	0	0
IframeRef.I	0	0	0	3	0	1	0	0	1	0	0	0	0	813	0	0	0	0	0	0
BlacoleRef.CZ	0	0	0	2	0	0	0	0	1	0	0	0	0	1	583	0	0	1	0	0
Yuner.A	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1324	0	0	0	0
Comame!gmb	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	596	0	1	0
IframeRef.DM	1	0	0	58	2	21	0	0	0	0	0	0	0	0	0	0	0	916	0	8
Ramnit.gen!C	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1401	0
Iframeinject.AF	1	0	0	17	0	17	0	0	1	0	0	0	0	1	0	0	0	3	0	631

FIGURE 4.5: Residual Network Confusion Matrix

4.3.3 Comparison with Previous Work

To validate that our model gives at par results with the past work, we compared it with the model presented by Natraj *et al.* [37]. The authors have made their dataset [36] available for verifying their results. For comparing our technique we applied our model to their dataset by converting it to the new colour map representation. Since Natraj et al. have only shared their byte plot images and not the actual malware binaries, so we first had to convert the images to binaries and then binaries to colour map representation. With the model proposed in this thesis we got classification accuracy of 96.08% on their dataset. This is comparable to the accuracy with the model proposed by Natraj et al. who got an accuracy of 97.18%.

4.4 Conclusion

Most of the malware classification methods require executing the malware to capture its behaviour or use disassembly techniques to predict its behaviour. Both execution of a malware and disassembling are time consuming and have high computation requirements. With the help of Image representations we are able to achieve comparable results and significant improvements in performance. Also the dataset used for analysis is larger than previous works done in classification using visual representation.

Behavioural or static based analysis are platform dependent, therefore we need to have different classifiers for different platforms. Whereas image based approach presented here is independent of platform as it classifies files based on the degree of similarity between binary of the same kind and dissimilarities with binaries of other kind. Also it is more secure as compared to dynamic based approaches as binaries are converted into image format and are never executed. The model proposed in this thesis gives an improvement on the previous works done and also paves the path for applying state of the art neural network techniques in malware analysis.

Chapter 5

Future Work

In this work we have fixed the image width to 384 Bytes. Since there can be Malware samples which could be very close to 384, so for those an effective representation could not be generated. So, one possible future work is to try coming up with representations which are immune to the size of the malware binary. Such representation can prove beneficial, as they would be able to analyse malware samples of all sizes.

Also quality of dataset label can be improved. Since we have used anti-virus engines to label our dataset and every anti-virus engine follows a different convention for labelling malware. So it may happen that the same malware may be labelled differently by different anti-virus engines. Also those anti-virus engines could mislabel some of the samples in the dataset as they use signature to label malwares. So another possible future work could be devising better techniques for labelling the dataset.

Bibliography

[1] A. Sung, J. Xu, P. C. and Mukkamala, S. (2004). Static Analyzer of Vicious Executables (SAVE). In *Proceedings of the 20th Annual Computer Security Applications.*

[2] Alazab, M., Venkataraman, S., and Watters, P. (2010). Towards Understanding Malware Behaviour by the Extraction of API calls. In *2nd CTC 2010 Ballarat (VIC), Australia.*

[3] Anderson, B.and Quist, D., Neil, J., Storlie, C., and Lane, T. (2011). Graph Based Malware Detection Using Dynamic Analysis. In *Journal in Computer Virology.*

[4] AV-test (2017). Malware Statistics. `http://www.av-test.org/en/statistics/malware/`.

[5] Bayer, U., Comparetti, P., Hlauschek, C., and Kruegel, C. (2009). Scalable, Behavior-Based Malware Clustering. In *Proceedings of the 16th Annual Network and Distributed System Security Symposium.*

[6] Bazrafshan, Z., Hashemi, H., Fard, S. M. H, and Hamzeh, A. (2013). A survey on heuristic malware detection techniques. In *Information and Knowledge Technology.*

[7] Bengio, Y., Simard, P., and Frasconi., P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks.*

[8] BooJoong Kang, Suleiman Y. Yerima, K. M. S. S. (2016). N-opcode Analysis for Android Malware Classification and Categorization.

[9] Christodorescu, M., Jha, S., and Kruegel, C. (2007). Mining specifications of malicious behavior. In *Proceedings of the 6th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESECFSE).*

[10] Clarify (2014). Convolutional Neural Networks. `https://www.clarifai.com/technology`.

[11] Cohen, F. (1987). Computer Viruses: Theory and Experiments. `http://web.eecs.umich.edu/~aprakash/eecs588/handouts/cohen-viruses.html`.

[12] Conti, G., Bratus, S., Sangster, B., Ragsdale, R., Supan, M., Lichtenberg, A., Perez-Alemany, R., and Shubina, A. (2010). Automated Mapping of Large Binary Objects Using Primitive Fragment Type Classification. In *The proceedings of The Digital Forensic Research Conference DFRWS*.

[13] Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., and Witten, I. (2009). The WEKA Data Mining Software. In *ACM SIGKDD Explorations Newsletter*.

[14] Han, K. S., Lim, J. H., and Im, E. G. (2013). Malware analysis method using visualization of binary files. In *Proceedings of Research in Adaptive and Convergent Systems ACM*.

[15] He, Kaiming; Zhang, X. R. S. S. J. (2015). Deep Residual Learning for Image Recognition. *eprint arXiv:1512.03385, ARXIV*.

[16] Helfman, J. (1995). Dotplot patterns: A literal look at pattern languages. *TAPOS*, 2:31–41.

[17] Honkela, A. (2001). Nonlinear switching state-space models. `URLhttp://www.hiit.fi/u/ahonkela/dippa/dippa.html`.

[18] Hubel, D. and Wiesel, T. (1968). Receptive fields and functional architecture of monkey striate cortex. *Journal of Physiology (London)*.

[19] ICT (2016). ICT: Facts and Figures. `http://www.itu.int/en/ITU-D/Statistics/Documents/facts/ICTFactsFigures2016.pdf`.

[20] Indyk, P. and Motwani, R. (1998). Approximate Nearest Neighbor: Towards Removing the Curse of Dimensionality. In *Proceedings of 30th Annual ACM Symposium on Theory of Computing, Dallas*.

[21] Islam, R., Tian, R., Batten, L. M., and Versteeg, S. (2013). Classification of malware based on integrated static and dynamic features. In *Journal of Network and Computer Applications*.

[22] Kaspersky (2014). Cybercrime, Inc.: How profitable is the business? `https://blog.kaspersky.com/cybercrime-inc-how-profitable-is-the-business/15034/`.

[23] Ki, Y., Kim, E., and Kim, H. K. (2015). A Novel Approach to Detect Malware Based on API Call Sequence Analysis. *International Journal of Distributed Sensor Networks*.

[24] Kohonen, T. (1995). Self-Organizing Maps. *Springer.*

[25] Kolbitsch, C. (2011). Anubis. `https://seclab.cs.ucsb.edu/academic/projects/projects/anubis/`.

[26] Kolter, J. and Maloof, M. (2004). Learning to detect malicious executables in the wild. In *In Proc. of the 10th ACM Int. Conf. on Knowledge Discovery and Data Mining.*

[27] Kong D, Y. G. (2013). Discriminant malware distance learning on structural information for automated malware classification. In *Proceedings of the 19th ACM SIGKDD international conference on knowledge discovery and data mining. ACM.*

[28] LIU, L., Bao-sheng WANG, YU, B., and Qiu-xi ZHONG (2016). Automatic Malware Classification and New Malware Detection using Machine Learning. In *Frontiers of Information Technology and Electronic Engineering.*

[29] M. Schultz, M. Eskin, E. Z. and Stolfo, F. (2001). Data Mining Methods for Detection of New Malicious Executables. In *In Proc. of the 22nd IEEE Symposium on Security and Privacy.*

[30] Makandar, A. and Patrot, A. (2015). Malware Analysis and Classification using Artificial Neural Network. In *Trends in Automation Communications and Computing Technology.*

[31] Malshare (2012). Malware Repository. `http://malshare.com/`.

[32] Microsoft (2017). Naming malware. `https://www.microsoft.com/en-us/security/portal/mmpc/shared/malwarenaming.aspx`.

[33] Montufa, G., Pascanu, R., Cho, K., and Bengio., Y. (2014). Learning long-term dependencies with gradient descent is difficult on the number of linear regions of deep neural networks. *NIPS.*

[34] Moser, A., Kruegel, C., and Kirda, E. (2007). Limits of Static Analysis for Malware Detection. In *IEEE Computer Society.*

[35] Nari, S. and Ghorbani, A. (2013). Automated Malware Classification Based on Network Behaviour. In *Proceedings of International Conference on Computing, Networking and Communications (ICNC), San Diego.*

[36] Nataraj, L., Karthikeyan, S., Jacob, G., and Manjunath, B. (2011a). Malimg Dataset. `http://old.vision.ece.ucsb.edu/spam/malimg.shtml`.

[37] Nataraj, L., Karthikeyan, S., Jacob, G., and Manjunath, B. (2011b). Malware Images: Visualization and Automatic Classification. In *Proceedings of International Symposium on Visualization for Cyber Security.*

[38] Ollmann, G. (2008). The Evolution of commercial malware development kits and colour-by-numbers custom malware. In *Computer Fraud and Security.*

[39] Pandalabs (2016). Quaterly Report. `http://www.pandasecurity.com/mediacenter/src/uploads/2016/05/Pandalabs-2016-T1-EN-LR.pdf`.

[40] Pandalabs (2017). Quaterly Report. `http://www.pandasecurity.com/mediacenter/src/uploads/2017/05/Pandalabs-2017-T1-EN.pdf`.

[41] Peter, E. and Schiller, T. (2008). A practical guide to honeypots. `http://www.cs.wustl.edu/~jain/cse571-09/ftp/honey.pdf`.

[42] R. Tian, L. M. B. and Versteeg, S. C. (2008). Function length as a tool for malware classification. In *In Proc. of the 3rd Int. Conf. on Malicious and Unwanted Software.*

[43] Radu, S. P., Hansen, S. S., Larsen, Thor. M. T., Stevanovic, M., Pedersen, J. M., and Czech, A. (2015). Analysis of malware behavior: Type classification using machine learning. In *CyberSA*.

[44] Raghakot (2015). ResNet. `https://github.com/raghakot/keras-resnet`.

[45] Rieck, K., Holz, T., Willems, C., Dussel, P., and Laskov, P. (2008). Learning and classification of Malware behaviour. In *Detection of Intrusions and Malware, and Vulnerability Assessment.*

[46] S. Staniford, V. P. and Weaver, N. (2002). How to own the internet in your spare time. In *Proceedings of the 11th USENIX Security Symposium.*

[47] Santos, I., Devesa, J., Brezo, F., Nieves, J., and Bringas, P. G. (2012). OPEM: A Static-Dynamic Approach for Machine-Learning-Based Malware Detection.

[48] Santos, I., Nieves, J., and Bringas, P. (2011). Semi-Supervised Learning for Unknown Malware Detection. In *Symposium on Distributed Computing and Artificial Intelligence Advances in Intelligent and Soft Computing.*

[49] Schultz, M. G., Eskin, E., and Zadok, F. (2001). Data Mining Methods for Detection of New Malicious Executables. In *In Proc. of the 22nd IEEE Symposium on Security and Privacy.*

[50] Siddiqui, M. and Wang, M. C. (2009). Detecting Internet Worms Using Data Mining Techniques. In *Journal of Systemics, Cybernetics and Informatics.*

[51] SIKORSKI M., H. A. (2012). *Practical Malware Analysis: The Hands-On Guide to Dissecting Malicious Software.*

[52] Spafford, E. H. (1989). The Internet worm incident. In *Proceedings of the 2nd European Software Engineering Conference. 446?468.*

[53] Szor, P. (2005). *The Art of Computer Virus Research and Defense.*

[54] Torralba, A., Murphy, K., Freeman, W., and Rubin, M. (2003). Classification of malware based on integrated static and dynamic features. In *In Proceedings of ICCV.*

[55] Vinod, P., Jaipur, R., Laxmi, V., and Gaur, M. (2009). Survey on malware detection methods. In *Proceedings of the 3rd Hacker's Workshop on Computer and Internet Security (IITKHACK?09).*

[56] VirusShare (2011). Malware Repository. `https://virusshare.com/`.

[57] VirusTotal (2004). Online Malware Report Generator. `https://www.virustotal.com/`.

[58] VirusTotal (2017). Daily Statistics. `https://www.virustotal.com/en/statistics/`.

[59] Wagener, G., State, R., and Dulaunoy, A. (2008). Malware behaviour analysis. Journal in Computer Virology. In *Proceedings of the 5th International Conference on Malicious and Unwanted Software 2010.*

[60] Wikipedia (2016). Mirai Malware. `https://en.wikipedia.org/wiki/Mirai_(malware)`.

[61] Wikipedia (2017a). Ransomware. `https://en.wikipedia.org/wiki/Ransomware`.

[62] Wikipedia (2017b). WannaCry Ransomware. `https://en.wikipedia.org/wiki/WannaCry_ransomware_attack`.

[63] Willems, C., Holz, T., and Freiling, F. (2007). Toward Automated Dynamic Malware Analysis Using Cwsandbox. In *IEEE Security and Privacy.*

[64] Yoo, I. S. (2004). Visualizing windows executable virus using self-organizing maps. In *Proceedings of ACM workshop on Visualization and data mining for computer security.*

[65] Zeiler, Matthew D; Fergus, R. (2013). Visualizing and Understanding Convolutional Networks. *eprint arXiv:1311.2901S.*

[66] Zhang, Q. and Reeves, D. (2007). Metaware: Identifying Metamorphic Malware. In *Proceedings of the 23rd Annual Computer Security Applications Conference.*

[67] Zhou, D., Bousquet, O., Lal, T. N., Weston, J., and Scholkopf, B. (2003). Learning with local and global consistency. In *Advances in Neural Information Processing Systems 16: Proceedings of the 2003.*