

Spring 2017

## Malware Scores Based on Image Processing

Vikash Raja Samuel Selvin  
*San Jose State University*

Follow this and additional works at: [https://scholarworks.sjsu.edu/etd\\_projects](https://scholarworks.sjsu.edu/etd_projects)



Part of the [Information Security Commons](#)

---

### Recommended Citation

Selvin, Vikash Raja Samuel, "Malware Scores Based on Image Processing" (2017). *Master's Projects*. 546.  
DOI: <https://doi.org/10.31979/etd.347x-pf32>  
[https://scholarworks.sjsu.edu/etd\\_projects/546](https://scholarworks.sjsu.edu/etd_projects/546)

This Master's Project is brought to you for free and open access by the Master's Theses and Graduate Research at SJSU ScholarWorks. It has been accepted for inclusion in Master's Projects by an authorized administrator of SJSU ScholarWorks. For more information, please contact [scholarworks@sjsu.edu](mailto:scholarworks@sjsu.edu).

Malware Scores Based on Image Processing

A Project

Presented to

The Faculty of the Department of Computer Science

San José State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

by

Vikash Raja Samuel Selvin

May 2017

© 2017

Vikash Raja Samuel Selvin

ALL RIGHTS RESERVED

The Designated Project Committee Approves the Project Titled

Malware Scores Based on Image Processing

by

Vikash Raja Samuel Selvin

APPROVED FOR THE DEPARTMENT OF COMPUTER SCIENCE

SAN JOSÉ STATE UNIVERSITY

May 2017

Dr. Mark Stamp      Department of Computer Science

Dr. Robert Chun      Department of Computer Science

Dr. Thomas Austin      Department of Computer Science

## **ABSTRACT**

Malware Scores Based on Image Processing

by Vikash Raja Samuel Selvin

Malware analysis can be based on static or dynamic analysis. Static analysis includes signature-based detection and other forms of analysis rely only on features that can be extracted without code execution or emulation. In contrast, dynamic analysis depends on features extracted at runtime (or via emulation) such as API calls, patterns of memory access, and so on. Dynamic analysis can be more informative and is generally more robust, but static analysis is typically more efficient. In this research, we implement, test, and analyze malware scores based on image processing. Previous work has shown that useful malware scores can be obtained when binaries are treated as images. We test a wide variety of image processing techniques and machine learning techniques. Further, we develop a dataset that is designed to evade detection mechanisms that employ image analysis.

## ACKNOWLEDGMENTS

I am very thankful to my advisor Dr. Stamp who has been a guiding light, constantly supporting me throughout the Master's project and helping me focus on the right path to complete the project.

I would also like to thank my committee members Dr. Thomas Austin and Dr. Robert Chun for their valuable time and suggestions for this project.

I would also like to thank all my family and friends for always supporting me and believing in me.

Last but not least, I would like to thank Christ for all the blessings.

## TABLE OF CONTENTS

### CHAPTER

<b>1</b>	<b>Introduction</b> . . . . .	<b>1</b>
<b>2</b>	<b>Types of Malware</b> . . . . .	<b>3</b>
2.1	Types of Malware - Based on Action, Concealment Strategy . . .	3
2.2	Types of Malware - Based on Evasion Strategy . . . . .	4
2.3	Malware Detection Methods . . . . .	5
<b>3</b>	<b>Drawbacks of Traditional Malware Analysis Methods</b> . . . . .	<b>7</b>
3.1	Drawbacks of Static Analysis . . . . .	7
3.2	Drawbacks of Dynamic Analysis . . . . .	10
<b>4</b>	<b>Image Processing Techniques</b> . . . . .	<b>13</b>
4.1	Gist Features . . . . .	13
4.2	Evaluation of gist descriptors . . . . .	14
4.3	Feature Selection and Further Experiments . . . . .	16
<b>5</b>	<b>Experimenting with Deep Learning</b> . . . . .	<b>31</b>
5.1	Tensorflow for Deep Learning . . . . .	31
5.2	Experimenting with simple Softmax Regression . . . . .	32
5.3	Convolutional Neural Networks . . . . .	41
5.3.1	What is Convolution . . . . .	41
5.3.2	Architecture . . . . .	47
5.3.3	Putting it all together in the experiment . . . . .	49
5.3.4	Disadvantages of Neural Networks . . . . .	50

6 Conclusion and Future Work . . . . .	52
<b>LIST OF REFERENCES . . . . .</b>	<b>53</b>



## LIST OF TABLES

1	Experimental Setup for gist based scoring. . . . .	14
2	Maling Dataset. . . . .	15
3	Malicia Dataset. . . . .	17
4	Accuracy Results - 30/70 Train/Test Split. . . . .	18
5	Accuracy Results - 40/60 Train/Test Split. . . . .	19
6	Accuracy Results - 50/50 Train/Test Split. . . . .	19
7	Accuracy Results - 60/40 Train/Test Split. . . . .	20
8	Accuracy Results - 70/30 Train/Test Split. . . . .	20
9	Accuracy Results - 80/20 Train/Test Split. . . . .	21
10	Accuracy Results - 90/10 Train/Test Split. . . . .	21
11	Experimental Setup for simple neural networks. . . . .	37
12	No of families - 1. . . . .	37
13	No of families - 2. . . . .	39
14	No of families - 3. . . . .	39
15	No of families - 4. . . . .	40
16	No of families - 5. . . . .	40
17	Experimental Setup for deep learning. . . . .	49

## LIST OF FIGURES

1	Register reassignment techniques by using multiple registers to achieve same result. . . . .	9
2	Malware analysis using IDA Pro for extraction of function calls. .	11
3	All variants of the malware family Dialplatform.B. . . . .	14
4	Confusion Matrix with all features. . . . .	16
5	Malicia Dataset - Accuracy vs % Training Data. . . . .	22
6	Confusion Matrix with 60 features. . . . .	23
7	Confusion Matrix Results - Before Salting. . . . .	25
8	Confusion Matrix Results - After Salting. . . . .	26
9	Confusion Matrix Results - After Salting. . . . .	27
10	Accuracy Comparision - Before and After Salting. . . . .	27
11	Confusion Matrix - After Salting with Benign Exes. . . . .	28
12	Confusion Matrix - After Salting with Benign Exes and using SVC.	29
13	Confusion Matrix - After Interleaving with Benign Exes and using SVC. . . . .	30
14	Logistic Regression vs. Softmax Regression. . . . .	33
15	Softmax Regression Matrix Form. . . . .	34
16	Softmax Regression in Simplified Matrix Form. . . . .	34
17	Cross Entropy Example. . . . .	35
18	Gradient Descent. . . . .	36
19	Simple Neural Network Scores. . . . .	38
20	Convolutional Neural Network - Convolution Example. . . . .	41

21	Convolutional Neural Network - Convolution Step 1. . . . .	42
22	Convolutional Neural Network - Convolution Step 2. . . . .	42
23	Convolutional Neural Network - Convolution Step 3. . . . .	42
24	Convolutional Neural Network - Convolution Step 4. . . . .	42
25	Convolutional Neural Network - Convolution Step 5. . . . .	43
26	Convolutional Neural Network - Convolution Step 6. . . . .	43
27	Convolutional Neural Network - Convolution Step 7. . . . .	43
28	Convolutional Neural Network - Convolution Step 8. . . . .	43
29	Convolutional Neural Network - Convolution Step 9. . . . .	44
30	Convolutional Neural Network - Blurring Filter Example Image. .	45
31	Convolutional Neural Network - Filter Example. . . . .	46
32	Regular Neural Network. . . . .	47
33	Convolutional Neural Network. . . . .	47
34	Convolutional Neural Network - Max-pooling. . . . .	48
35	Real World Convolutional Neural Network. . . . .	48
36	Convolutional Neural Network Results. . . . .	50
37	Convolutional Neural Network Results. . . . .	50

# CHAPTER 1

## Introduction

When computers became ubiquitous so did malware. Any software written with the intention to damage or disable computers and computer systems is called as Malware. Malware writers started out writing code to show off their skills or for research purposes but that soon took a turn for the worse when computer systems started becoming the cornerstone of all sorts of businesses and organizations. Nowadays, most malware are targeted at profit through gaining access to confidential data or destruction of data or denial of services. Hence, it is important to develop efficient methods that could shield the computer systems from these malware.

Classifying malware is a very hard problem to tackle, the reason being innumerable variants being created on an everyday basis. Antivirus vendor Symantec came out with a report that said it had over 286 million variants of malware in its database [1]. Static and dynamic analysis are the most commonly employed techniques in analyzing malware. Signature based detection methods form a core part of static analysis [2]. Typically, in static analysis, the malware is not executed and features such as the machine level “opcodes”, the actual binary representation of the executable are used. Dynamic analysis typically refers to methods wherein the malware is executed under a controlled environment and various actions of malware such as the Application Program Interface (API) calls it makes, pattern of memory access etc are used as features. Drawbacks of the aforementioned approaches [3, 4, 5, 6] will be discussed in the next two chapters, from these discussions it will become apparent that there is a need for new robust mechanisms that are unaffected by the strategies that are currently being used to defeat traditional malware analysis methods.

In this research, we will be using techniques from the domain of image detection. The underlying idea is to convert malware binaries in to images and then calculate gist

descriptors from the images. A gist descriptor gives a low dimensional representation of the image which could be used to reliably recognize similar images. This technique would help us to identify variants of the same malware family as they tend to be visually similar even if code obfuscation techniques were used for generating the variations. This research also aims to determine the optimum number of features required to successfully cluster similar images and also discusses methods that aim to defeat the gist based detection technique. It also looks at the use of and effectiveness of neural networks for clustering similar malware images.

This paper is structured as follows. Chapter 2 provides details on types of malware. Chapter 3 provide details on drawbacks of static and dynamic analysis respectively. Chapter 4 discusses about using gist descriptors for image recognition and feature selection techniques for reducing the dimensionality of the data. Chapter 5 discusses about using deep learning techniques for malware image recognition and using the TensorFlow framework for it. Chapter 6 concludes the paper after discussing attempts at trying to defeat the gist based scores and future work.

## CHAPTER 2

### Types of Malware

Malware refers to any software that intends to damage or disable computers or computer systems. There are many types of malware including but not limited to Adware, Spyware, Ransomware, Virus, Worm, Trojan, Rootkit, Backdoors, Keyloggers, Rogue security software, Browser hijacker. In this chapter, we will look into different types of malware based on their action, concealment strategy and detection evasion techniques used.

#### 2.1 Types of Malware - Based on Action, Concealment Strategy

Adware displays ads on your computer and generates revenue based on that for the malware writer. This is mostly innocuous but might be irritating in that it might open multiple browser windows to display multiple ads at the same time [7].

Spyware is a kind of malware that tracks your online presence in order to send targeted ads to your computer. For example, say if you are looking for cars, then a car company might use this data to send you ads about their new cars [7].

Ransomware is a type of malware that encrypts data in infected system or alters its access privileges or blocks access to the computer until a sum of money is paid. This is currently used against individual computers most of the time [8].

Virus is a self replicating program that mostly attach themselves to other programs. They try to infect their targets and conceal themselves, they are parasitic by nature and reside in the boot sector, executables or data files. They are usually spread by sharing files and portable memory sticks between computers [9].

Worms are programs that replicate and multiply and repeat this process over and over again thereby effectively choking the system and its resources. They are usually spread through the network [10].

Trojans are programs that conceal or disguise themselves as other benign files

and they aim to discover bank credentials, take over computer resources, and in larger systems creates a denial-of-service attack. Even though they are usually disguised as some innocuous file like images or benign executables or videos if you look close enough you will find that the extension differs [11].

Rootkit is a type of malware that resides in the boot sector and as a result is amongst the first programs to start in the machine when it gets power. Since it is very hard to detect it is also hard to remove them. These type of malware are typically used to create a back door in to the system so that other, more advanced malware could exploit it for gain [12].

Backdoor works by bypassing authentication and gaining remote access to the infected system. It usually waits for commands from a command and control center. Harebot.M is an example of a backdoor malware [13].

Keyloggers are designed to conceal themselves and keep running in the background and record, transmit every key stroke being executed on the machine. Nowadays they are advanced enough to capture screens and transmit them securely over to a remote server [14].

Rogue security software impersonate an anti-virus product. They often turn off the real anti-virus products and trick you into believing that these are legitimate programs [15].

Browser hijacker is designed to route you to pages that would bring money to the malware writers. They work by redirecting the browser traffic to pages of the malware writer's choice [16].

## **2.2 Types of Malware - Based on Evasion Strategy**

In order to evade detection, malware writers have come up with various effective techniques, we will be seeing some of those techniques below.

Encrypted Virus work by encrypting the critical code, so that they do not show up on the signature scan. When the virus executes based on some predefined condition the decryption code kicks in and decrypts the critical part at runtime. However, the problem is that the decryption part would be fairly standard and that could be used for scanning the signatures [9].

Oligomorphic Virus is an improvement over the encrypted virus. They try to offset the weakness in the encrypted virus by having multiple versions of the decryption code. But this will not be that detrimental to the anti-virus product as it will simply add all versions of the decryption code to their databases [17].

Polymorphic virus is written by encapsulating the layer of encryption and the decryption code is programmed in such a way that it changes its form and hence evades detection. These can be detected by allowing them to run in an emulator and let the code decrypt itself and after that looking for the signature [18].

Metamorphic virus are the most advanced of all malware. They work by changing the entire code every time they are run. They usually have a mutation engine which serves up several obfuscation techniques to evade signature detection. Since this is the most advanced of all malware it is also the most difficult of malware's to write. This is a challenging problem for anti-virus companies and researchers alike [17].

### **2.3 Malware Detection Methods**

There are two traditional analysis methods used for malware detection, they are static and dynamic analysis.

The unique characteristic about static analysis is that they do not execute the malware file and extract features that include but are not limited to opcodes, binary representation of the program, function names, etc. This results in a faster turn around time and lesser risk. Antivirus programs then try signature matching, i.e.,



they compare the signature obtained from the malware file to a known database that they maintain. If a match is found, then the file is tagged as a malware. Algorithms like Aho-Corasick [5, 19] allow for efficient string matching which keeps the running time of this approach manageable. However, this approach could be defeated by using various code obfuscation techniques which will be discussed in the next chapter.

Dynamic Analysis of malware refers to techniques in which the malware is run in a sandboxed environment and various features such as windows API calls, stack contents, actual function parameters, etc are extracted from it which is then used for classifying it as a malware. Dynamic analysis is costlier than static analysis but generally yields better performance. Though it is harder to defeat it has been shown that it is possible to defeat it, we will discuss some of these techniques in the next chapter.

## CHAPTER 3

### Drawbacks of Traditional Malware Analysis Methods

Some of methods used by malware writers for defeating static and dynamic analysis techniques are explained herein. In general, static analysis has been shown to be weak against code obfuscation techniques and dynamic analysis has been shown to be expensive and it produces lots of false positives.

#### 3.1 Drawbacks of Static Analysis

As discussed in the previous section, static analysis refers to the technique of extracting features from a malware without running it. Machine level opcodes, the original binary program, function calls and API, variable names. From these features a signature, which is a just a string of bits, is extracted and matched against a known database. Efficient string comparison algorithms like Aho-Corasick [5, 19] ensure that this process runs in acceptable time frames. [3] discusses some of the techniques that could be used for defeating scores based on static analysis, these include using opaque constants, control flow obfuscation, data location obfuscation and data usage obfuscation.

1. Opaque constants --- Constant values are used in many places in the binary and are often involved in branching control flows, supplied as operands of arithmetic expressions, etc. Substituting expressions that produce values which are equal to the constant's value [3] would result in different bits being generated in the binary form of the malware. For example, instead of using constant value take an exclusive or with zero to get an equivalent value. The examples shown below explain how a code fragment could be written to manipulate opaque constants.

Suppose the value 40 is to be assigned to a variable, then this is one way to do it:

```
constant = 40;
```

The above value could be generated in a number of other ways, for example, given below is one other way to generate the same constant:

```
for (i=0; i<= 40; i++)  
    {  
        Pass;  
    }  
constant = i;
```

2. Control flow obfuscation --- A control flow graph is defined as a directed graph  $G = (V, E)$  in which the vertices represent basic blocks and an edge represents the flow of control between the two vertices [3]. Unconditional jump and calls could be replaced with a sequence of instructions that perform the same thing but is difficult to determine.
3. Data location obfuscation --- Data elements are located by specifying a constant address. A static analyzer could be misled by using opaque constants for specifying the location and hiding the data elements [3].
4. Data usage obfuscation --- Static analyzers are good at identifying the chain of operations that lead to a value being stored in a register, so with data usage obfuscation the idea is to spill values to an obfuscated memory location and then reload it later [3, 6] discusses techniques like encryption which could also be used for scrambling the binary form of the malware. Encrypting parts of the malware program makes sure that the binary form of it is just a meaningless random combination of zeros and ones. The decryption code then decrypts this

encrypted block when the malware is executed. The code fragment below shows an example of a malware decrypting itself based on some condition.

```
if (Hash(cmd) == Hash('DECRYPT'))
{
    X = decrypt(Hash(cmd), payload)
    execute X
}
```

[4] discusses techniques like dead-code insertion, register reassignment, subroutine reordering, code transposition, code integration.

5. Dead-code insertion --- This is a very simple mechanism in which the malware writer inserts some random instructions that have no effect on the flow of the program or no operation instructions [4].
6. Register reassignment --- In this technique the register being used is switched around for each generation of then malware [4]. Figure 1 shows an example of a code fragment that performs register reassignment using the well know EAX, EBX and EDX registers.



Figure 1: Register reassignment techniques by using multiple registers to achieve same result.

7. Subroutine reordering --- In this technique the order of the various subroutines is changed in some random order in each malware generation [5].
8. Code transposition --- In this technique the sequence of instructions in the original code are reordered without having an impact on the behavior of the malware [5].
9. Code integration --- In this technique the malware appends itself to the code of its target program, the target being a benign program [5].

The consensus is that malware writers have become wise to the use of signature based detection methods and have begun incorporating various code obfuscation techniques into their malware to evade the antivirus products. Hence, there is a need for a detection mechanism which is robust in the face of code obfuscation techniques.

### **3.2 Drawbacks of Dynamic Analysis**

Analyzing actions performed by a malware while it is running is called dynamic analysis. In dynamic analysis, the malware binary is run in an emulated operating system environment and its security related events are monitored. The windows API calls, native system calls that the program uses are monitored [20]. Figure 2 shows the usage of IDA Pro software for tracing function calls of a malware script.

Function call monitoring, function parameter analysis, information flow tracking, instruction trace, AutoStart Extensibility Points(ASEP) are some of the information being collected during dynamic analysis [20].

1. Function call monitoring --- Functions are used to abstract implementation details. For analyzing the behavior of a malware program it is very useful to see what happens in these functions. The process of intercepting functions is called hooking. Once the function is hooked it can be used to analyze the contents of

```

-----
; START OF FUNCTION CHUNK FOR sub_403A20
loc_42840A:
    push    0Bh                ; CODE XREF: sub_403A20+17↑j
    push    offset aNotrayicon ; size_t
    push    esi                ; wchar_t *
    mov     ebx, ecx
    call    __wcsnicmp
    add     esp, 0Ch
    test   eax, eax
    jnz    short loc_42842D
    mov     byte ptr [ebx], 1

loc_428423:
    mov     eax, 1
    jmp    loc_403A42
-----
loc_42842D:
    push    0Dh                ; CODE XREF: sub_403A20+249FE↑j
    push    offset aRequireadmin ; size_t
    push    esi                ; wchar_t *
    call    __wcsnicmp
    add     esp, 0Ch
    test   eax, eax
    jnz    short loc_42844F
    mov     byte ptr [ebx+1], 1
    mov     eax, 1
    jmp    loc_403A42
-----
loc_42844F:
    push    0Dh                ; CODE XREF: sub_403A20+24A1F↑j
    push    offset aNoautoit3execu ; "#NoAutoIt3Execute"
    push    esi                ; wchar_t *
    call    __wcsnicmp
    add     esp, 0Ch
    test   eax, eax
    jnz    short loc_428471
    mov     byte ptr [ebx+2], 1
    mov     eax, 1
    jmp    loc_403A42
-----
loc_428471:
    push    16h                ; CODE XREF: sub_403A20+24A41↑j
    push    offset a0nautoitstartr ; "#0nAutoItStartRegister"
    push    esi                ; wchar_t *
-----

```

Figure 2: Malware analysis using IDA Pro for extraction of function calls.

the stack, parameters and so on [20].

2. Function parameter analysis --- Dynamic function parameter analysis is used for getting the arguments that are actually passed to the function during runtime. Tracking these parameters would be useful in finding correlation of individual function calls that operate on the same object [20]. This provides a detailed insight into the program's behavior from an object centric view.

3. Information flow tracking --- The goal of information flow tracking is to track the flow of “interesting” information as the program executes. Taint sources and taint sinks form an important part of this technique. Taint sources are the ones from where tainted data is introduced into the program flow and taint sink is a specific component of the system that reacts in a certain way when simulated with the tainted source [20].
4. Instruction trace --- In this technique a trace of the low level machine instructions is traced as the program executes. This could reveal information that was potentially missed when looking at high level instructions.
5. AutoStart Extensibility Points(ASEP) --- ASEPs provide a mechanism for programs to be automatically invoked upon the operating system boot process [20]. Typically, malwares would persist by using these ASEPs.

Though dynamic analysis techniques offer a very good precision in recognizing malware it would not scale well, it takes a lot of time for extracting these features. Since it takes almost close to 3 minutes in some cases, this cannot be used for production cases. Also, it tend to produce a lot of false positives as it finds benign processes which might be making calls that might be out of the norm. False positives drastically reduce the confidence that users have on the anti-virus product which is why anti-virus products are calibrated to have high tolerance about processes they are not sure about classifying as malware.

## CHAPTER 4

### Image Processing Techniques

In order to overcome the shortcomings of both static and dynamic analysis [3, 4, 5, 6], it is important to find techniques that are robust in the face of code obfuscation and can provide detection rates that are of practical use without raising a lot of false positives. Image processing techniques come in handy to fill that void. The idea is that malware binaries are visualized as grey scale images, with the observation being that malware variants belonging to the same family produce similar images [21]. From these images, gist features are then extracted which are used to group together malware belonging to the same family [21].

#### 4.1 Gist Features

Gist is a low dimensional representation of the scene, which does not require any form of segmentation [22]. Gist descriptors capture the dominant spatial structures of a scene along perceptual dimensions of Naturalness, Openness, Roughness, Expansion, Ruggedness. They yield a low-dimensional descriptor (feature vector) of the entire image, which is called a Spatial Envelope.

Gist is used in scene recognition by considering global configurations without detailed object configuration [22, 23]. Gist computes the spectral information in an image through Discrete Fourier Transform. The spectral signals are then compressed by the Karhunen-Loeve Transform. The above mentioned perceptual dimensions were shown to be reliably estimated from these spectral signals.

The principal idea behind using gist descriptors is that images produced by different variations of the same malware family tend to be very similar and code obfuscation techniques do not transform the image sufficiently for it to not be grouped together when considering the gist features.

Gist has been proved to return very good results in web-scale image search [23].



And the results from using gist for classifying malware variants belonging to 25 different families also proved to be good returning almost 98 percent accuracy. Figure 3 shows different variants of the malware family Dialplatform.B

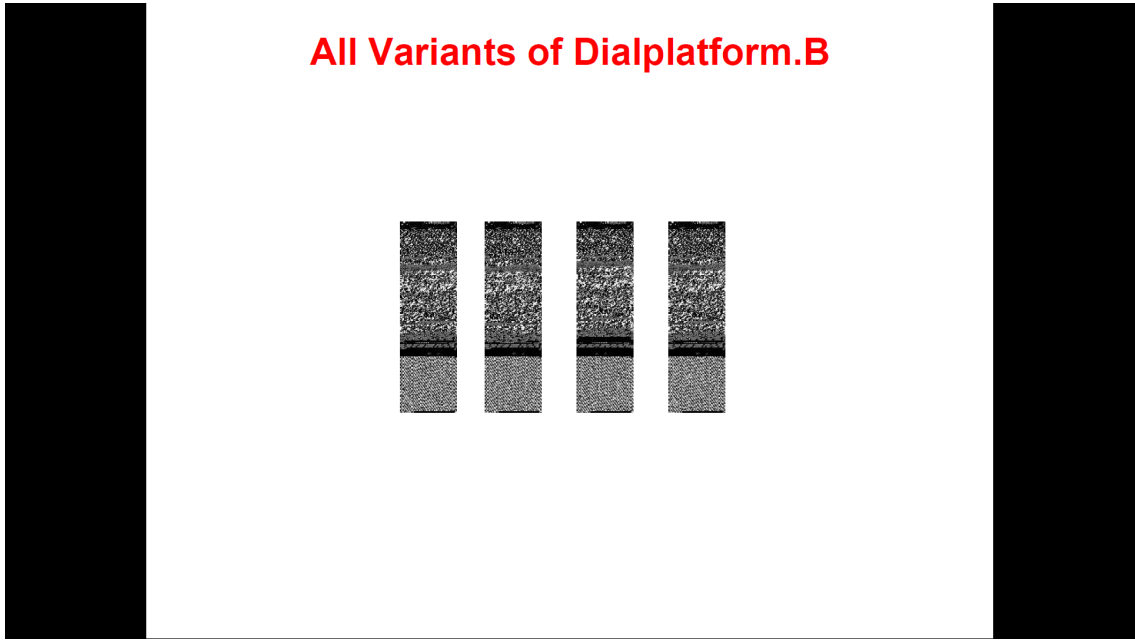


Figure 3: All variants of the malware family Dialplatform.B.

#### 4.2 Evaluation of gist descriptors

Table 1 lists the conditions under which this experiment was carried out. Python 2.7 was used on Ubuntu 14.04. Leargist, Numpy, sklearn were some of the important packages used, when it came to datasets both “Malicia” and “Malimg” were used in these experiments.

OS	Ubuntu 14.04
Python	2.7
Python Libraries	Numpy, Leargist, PIL, scipy, sklearn, Matplotlib
Datasets	Malicia, Malimg

Table 1: Experimental Setup for gist based scoring.

By using the gist descriptor as features and running a K-Means algorithm we

were able to achieve very good results with the Maling Dataset. The Maling dataset consists of 9000 malware files belonging to 25 malware families and their variants. Table 2 shows the various families in “Maling” dataset. All these malware binaries were converted to images and gist features were extracted from them.

<b>FAMILY NAME</b>	<b>NO OF FILES</b>
Adialer.C	122
Agent.FYI	116
Allapple.A	2949
Allapple.L	1591
Alueron.gen!J	198
Autorun.K	106
C2LOP.gen!g	200
C2LOP.P	146
Dialplatform.B	177
Dontovo.A	162
Fakerean	381
Instantaccess	431
Lolyda.AA1	213
Lolyda.AA2	184
Lolyda.AA3	123
Lolyda.AT	159
Malex.gen!J	136
Obfuscator.AD	142
Rbot!gen	158
Skintrim.N	80
Swizzor.gen!E	128
Swizzor.gen!I	132
VB.AT	408
Wintrim.BX	97
Yuner.A	800
Total	9339

Table 2: Maling Dataset.

The gist features were then clustered together using K-Means clustering algorithm. The average classification accuracy was 97%. Figure 4 shows the confusion matrix for this dataset.

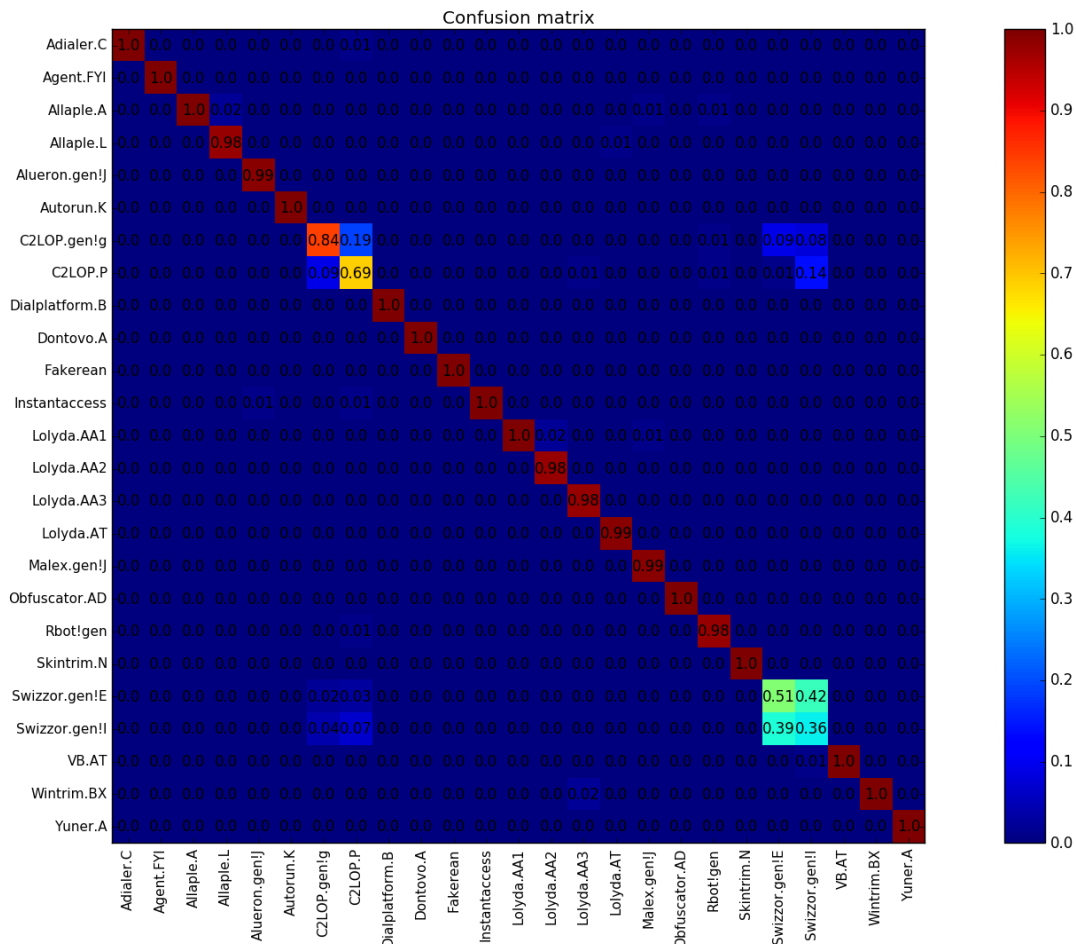


Figure 4: Confusion Matrix with all features.

### 4.3 Feature Selection and Further Experiments

In order to ascertain that the gist based scores would work on all general malware images and not just the “Maling” dataset, it was tested with the more challenging “Malicia” dataset. “Malicia” dataset contains 11363 samples which are primarily made up of ZBot, WinWebSec, ZeroAccess malware families. Table 3 below gives the details about the various malware families in the “Malicia” dataset and the corresponding number of files in each family.

<b>FAMILY NAME</b>	<b>NO OF FILES</b>
cleaman	32
CLUSTER:184.82.148.49	1
CLUSTER:217.20.115.99	1
CLUSTER:46.105.131.121	20
CLUSTER:85.93.17.123	45
CLUSTER:91.234.32.10	5
CLUSTER:astaror	24
CLUSTER:azonpowzanadinoar.com	1
CLUSTER:bundlemonkey.com	1
CLUSTER:chapterleomemorykombo.eu	9
CLUSTER:clarkclark	1
CLUSTER:dzony3777.su	1
CLUSTER:eikons.com	1
CLUSTER:foreign	6
CLUSTER:in7cy	1
CLUSTER:justontime-12.com	1
CLUSTER:landtechdata.com	1
CLUSTER:m9swachu.be	1
CLUSTER:mergenew	4
CLUSTER:newavr	29
CLUSTER:online-police.com	2
CLUSTER:paydayloatsstc.ru	1
CLUSTER:positivtkn.in.ua	14
CLUSTER:price.dtdns.net	1
CLUSTER:up2x.com	1
CLUSTER:whatismyip.com	1
crindex	74
cutwail	2
fakeav-rena	3
fakeav-webprotection	3
harebot	53
NULL	1646
ramnit	5
smarthdd	68
spyeeye-ep	7
ufasoft-bitcoin	3
winwebsec	5820
zbot	2186
zeroaccess	1306
Total	11363

Table 3: Malicia Dataset.

Experiments were carried out by taking a subset of the Malicia dataset and Benign exe's (Windows executables) so as to prevent the bias which arises out of having a very large number of files belonging to one class. From the table above, it could be seen that the families of zbot, winwebsec, zeroaccess could very easily create a bias and hence this approach is taken. The experiment was repeated for various splits of training and test data ranging from 30%,70% split to 90%,10%. The results from these experiments are discussed below.

Table 4 shows that the accuracy of classification when using 30% data for training and 70% data for testing is around 84%.

	FAMILY				
	BENIGN	W.WEBSEC	ZBOT	Z.ACCESS	OTHERS
TRAINING	69	51	13	14	21
TESTING	182	93	40	28	49
% OF TRAIN	30%	NO OF NEIGHBOURS			3
% TEST	70%				
ACCURACY	84.43%				

Table 4: Accuracy Results - 30/70 Train/Test Split.

Table 5 shows that the accuracy of classification when using 40% data for training and 60% data for testing improves to around 87%.

	FAMILY				
	BENIGN	W.WEBSEC	ZBOT	Z.ACCESS	OTHERS
TRAINING	100	57	22	22	23
TESTING	151	87	31	20	47
% - TRAIN	40%	NO OF NEIGHBOURS			3
% - TEST	60%				
ACCURACY	87.79%				

Table 5: Accuracy Results - 40/60 Train/Test Split.

Table 6 shows that the accuracy of classification when using 50% data for training and 50% data for testing improves to around 88%.

	FAMILY				
	BENIGN	W.WEBSEC	ZBOT	Z.ACCESS	OTHERS
TRAINING	131	67	26	21	35
TESTING	120	77	27	21	35
% - TRAIN	50%	NO OF NEIGHBOURS			3
% - TEST	50%				
ACCURACY	88.21%				

Table 6: Accuracy Results - 50/50 Train/Test Split.

Table 7 shows that the accuracy of classification when using 60% data for training and 40% data for testing the accuracy drops to around 85%.

	FAMILY				
	BENIGN	W.WEBSEC	ZBOT	Z.ACCESS	OTHERS
TRAINING	142	94	29	25	46
TESTING	109	50	24	24	17
% - TRAIN	60%	NO OF NEIGHBOURS			3
% - TEST	40%				
ACCURACY	85.26%				

Table 7: Accuracy Results - 60/40 Train/Test Split.

Table 8 shows that the accuracy of classification when using 70% data for training and 30% data for testing the accuracy improves to around 89%.

	FAMILY				
	BENIGN	W.WEBSEC	ZBOT	Z.ACCESS	OTHERS
TRAINING	176	97	35	32	52
TESTING	75	47	18	10	18
% - TRAIN	70%	NO OF NEIGHBOURS			3
% - TEST	30%				
ACCURACY	89.88%				

Table 8: Accuracy Results - 70/30 Train/Test Split.

Table 9 shows that the accuracy of classification when using 80% data for training and 20% data for testing the accuracy improves to around 92%.

	FAMILY				
	BENIGN	W.WEBSEC	ZBOT	Z.ACCESS	OTHERS
TRAINING	196	188	41	38	55
TESTING	55	26	12	4	15
% - TRAIN	80%	NO OF NEIGHBOURS			3
% - TEST	20%				
ACCURACY	92.85%				

Table 9: Accuracy Results - 80/20 Train/Test Split.

Table 10 shows that the accuracy of classification when using 90% data for training and 10% data for testing the accuracy stays at around 92%.

	FAMILY				
	BENIGN	W.WEBSEC	ZBOT	Z.ACCESS	OTHERS
TRAINING	227	134	46	38	59
TESTING	24	10	7	4	11
% - TRAIN	80%	NO OF NEIGHBOURS			3
% - TEST	20%				
ACCURACY	92.85%				

Table 10: Accuracy Results - 90/10 Train/Test Split.

Figure 5 shows the variation in accuracy with different splits for training and test data.



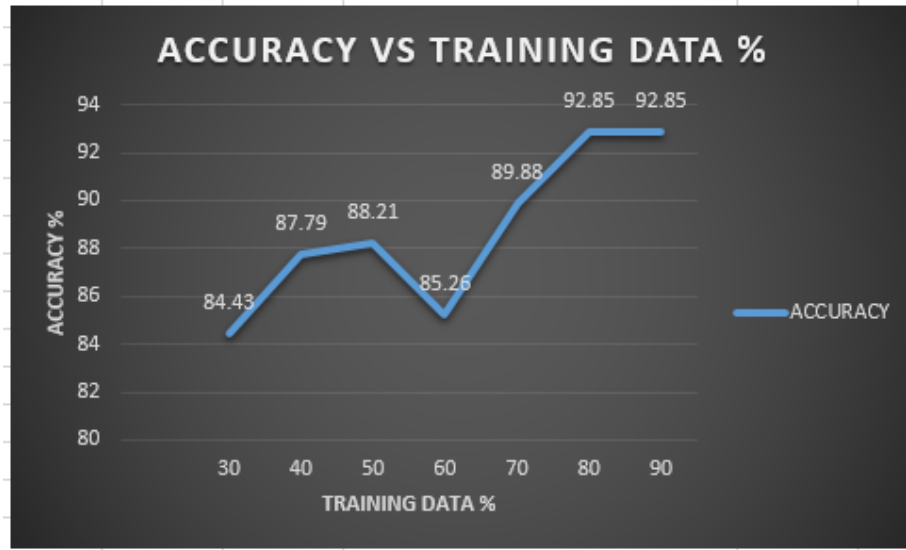


Figure 5: Malicia Dataset - Accuracy vs % Training Data.

Feature selection was carried out next. The aim of performing feature selection was to ascertain what would be the optimum set of features that need to be used for achieving high accuracy. SVM recursive feature elimination was the choice to see when we get the best RoC for the Malicia dataset, it was found that 320 was the optimal number of features.

Further, experiments were carried out with “univariate feature selection” from the python scikit package. Univariate statistics focuses on identifying features that are able to describe the data best. It can be seen as a preprocessing step to an estimator. “SelectKBest” retains only the  $k$  top features using metrics like false positive rate, false discovery rate, or family wise error for each feature. It was found that even if only the top 60 features are chosen, we still get very good results.

Figure 6 shows the confusion matrix for Maling dataset with just 60 of the original number of features.

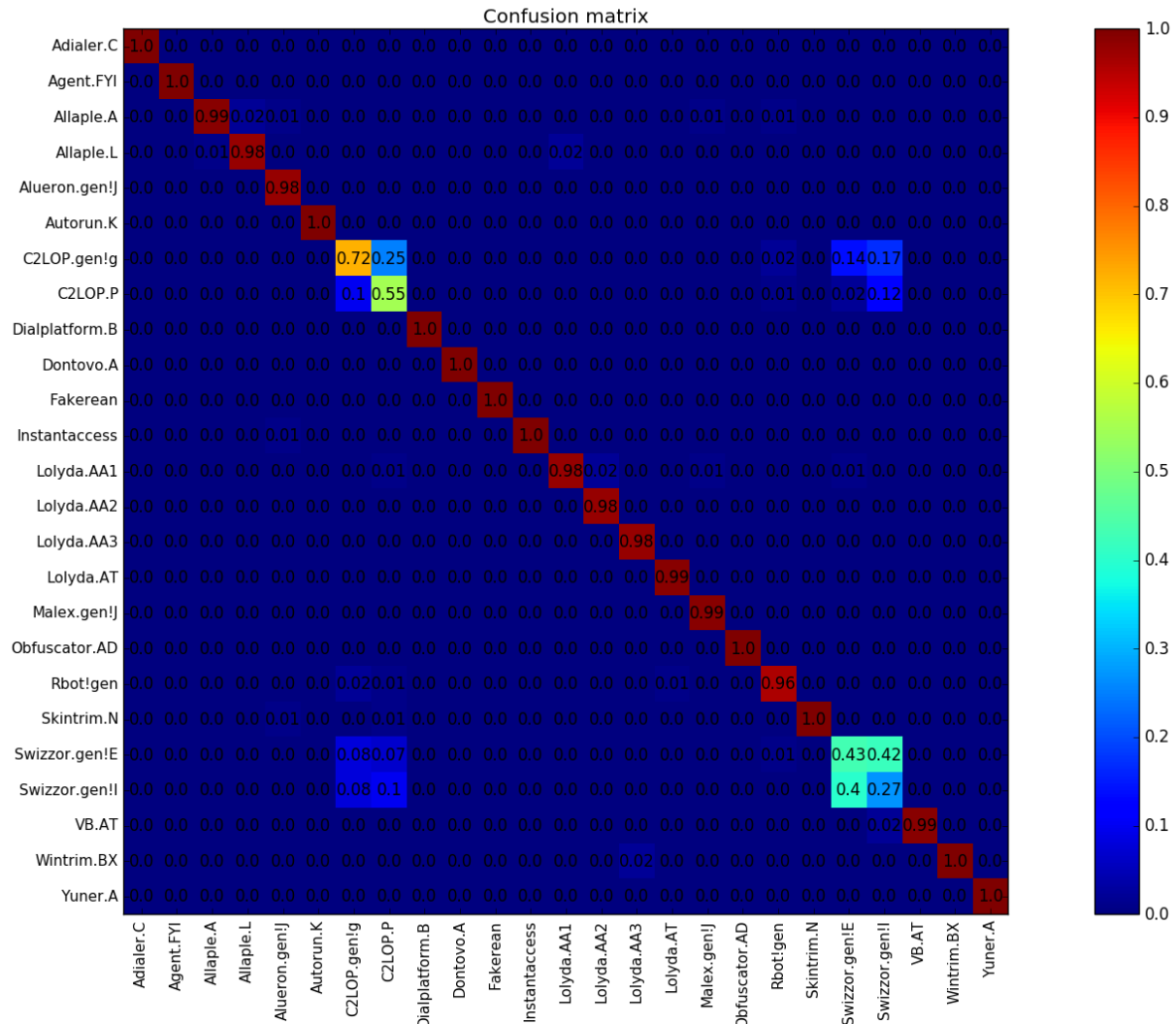


Figure 6: Confusion Matrix with 60 features.

Also, on the Malicia dataset we achieve 98.5% and 99.25% accuracy with 80% and 90% training data respectively. Hence it is concluded that combining univariate feature selection with first 60 gist descriptors gives us a very good scoring algorithm which is also very efficient.

It follows that the next step would be to find ways to defeat the score obtained using these gist features. The first idea that was tried was to salt the malware images

with another image and this leads to several questions like:

1. What happens when only one family's images are salted with an external image?
2. What happens when closely related families are salted with similar external images?
3. What happens when all families are salted with the same external images?

Experiments were conducted to answer the above questions and the findings are reported below.

When only one family's images are salted with an external image it should follow that the family's classification score should improve. This is because due to the addition of the salted image, the images belonging to this family would now be more unique and hence they will be fewer mis-classifications.

To ascertain this an experiment was conducted wherein only the images belonging to "Allaple.L" family were salted. This resulted in the score going up from 98% to 100%. Figures 7, 8 show the scores before and after the salting of the images respectively.

The next question is what would happen when closely related families are salted with similar external images? It should follow that the scores should go down for both the families. Our experiments also confirmed this. For this run we chose "Allaple.L" and "Allaple.A" families and salted both these families with images from an external image dataset. The result is that the scores came down to 85% and 93% respectively. Figure 9 shows the same.

Figure 10 shows the comparison of the accuracy scores before and after the salting of images.

Next, the malware images were salted with benign exe files to see if that was able

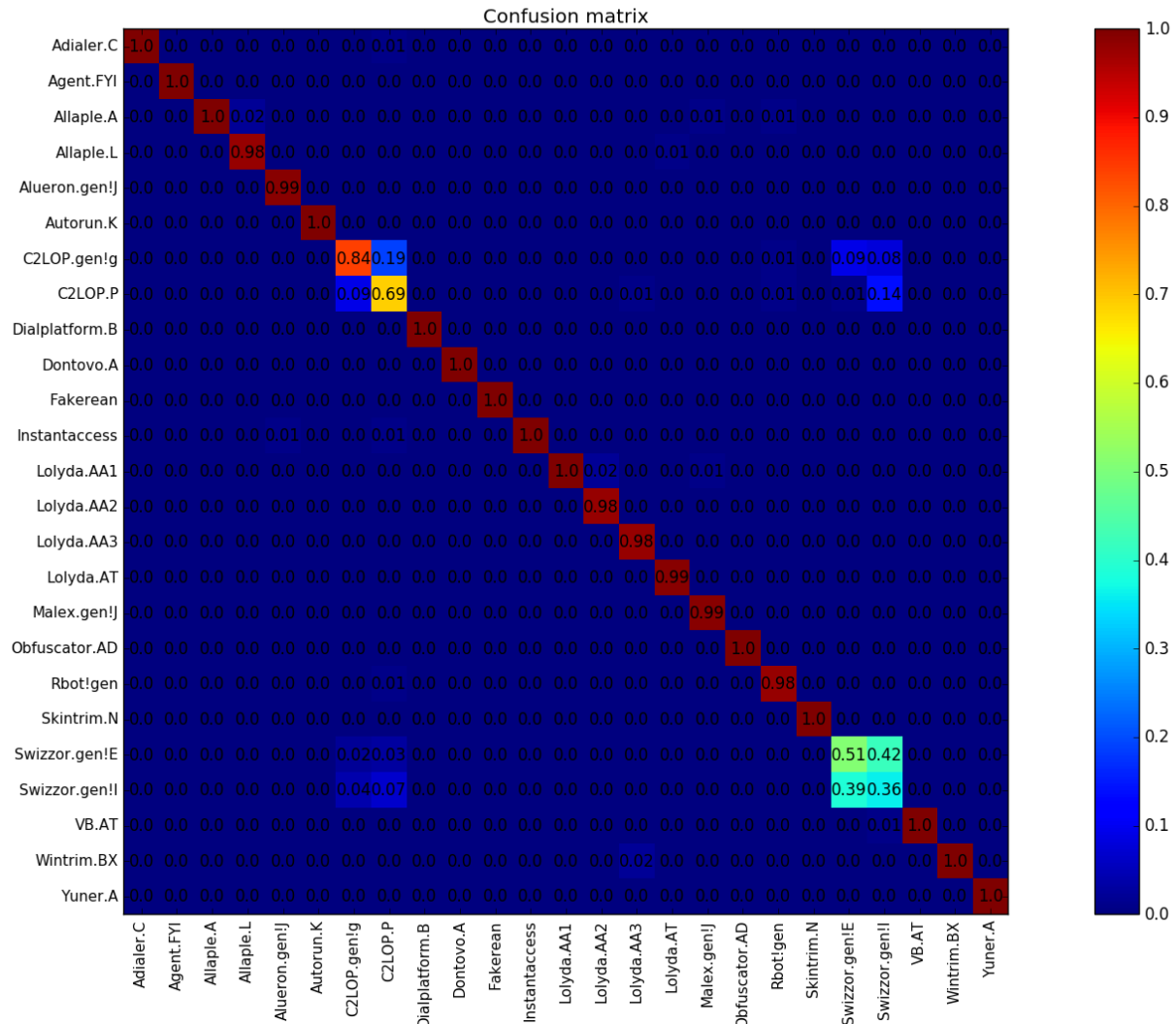


Figure 7: Confusion Matrix Results - Before Salting.

to defeat the scores. The confusion matrix is shown in Figure 11. Though it performed better than salting by random images it still did not break the scores completely.

A Support Vector Classifier (SVC) was tried next instead of the usual K-Nearest Neighbours algorithm. The confusion matrix is shown in Figure 12. It could be seen from the figure that the SVC did not fare well.

The next approach towards breaking the scores was to see what effect interleaving

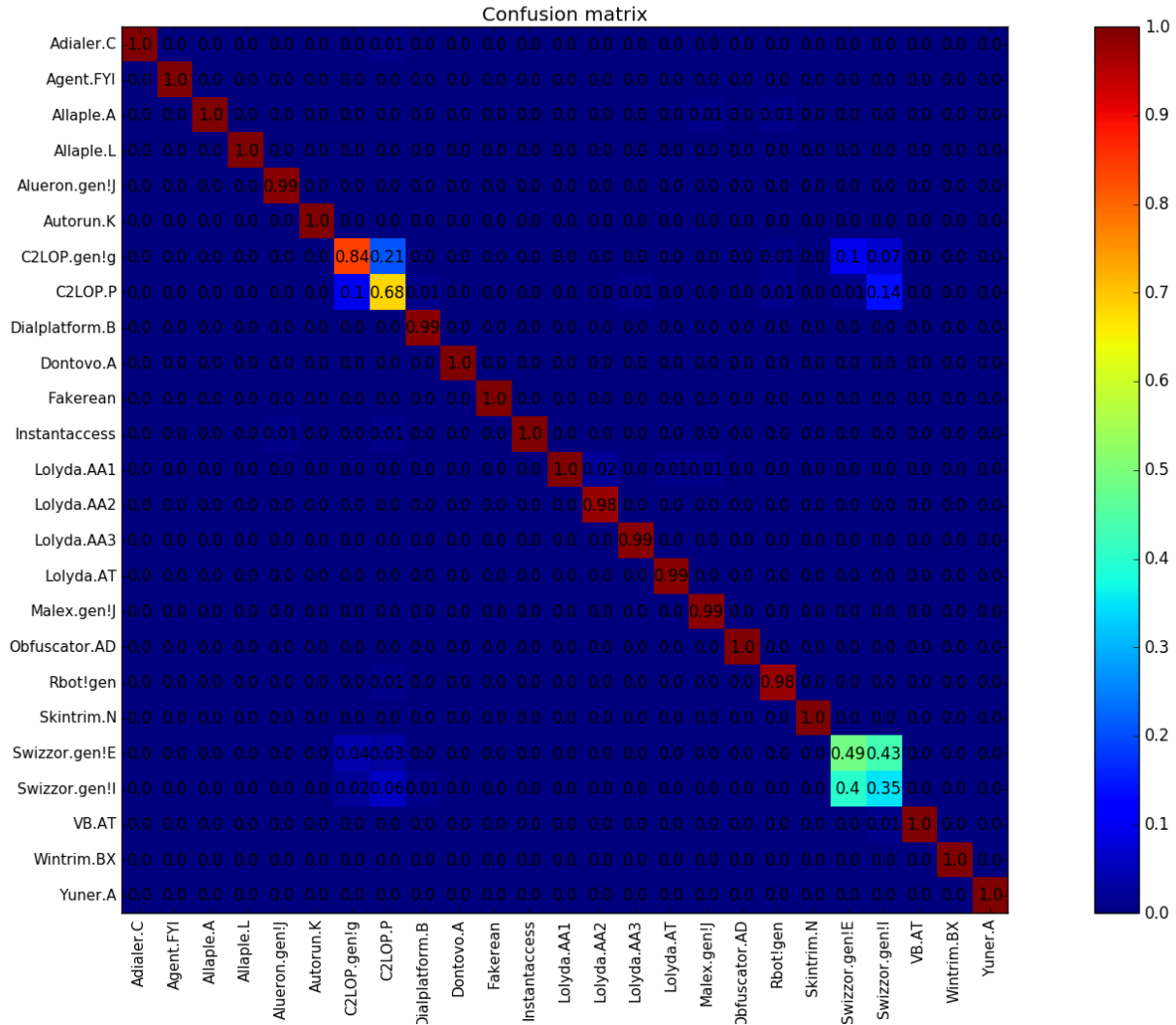


Figure 8: Confusion Matrix Results - After Salting.

images has on the gist features and the corresponding scores. All the methods mentioned above stack the images vertically hence it was interesting to look at what interleaving does to the scores. Accordingly, we interleaved images belonging to “Allaple.A”, “Allaple.L”, “Fakerean” families using benign executables. The results are surprisingly not that bad, gist feature scores are still good enough. Figure 13 shows the confusion matrix for this approach.

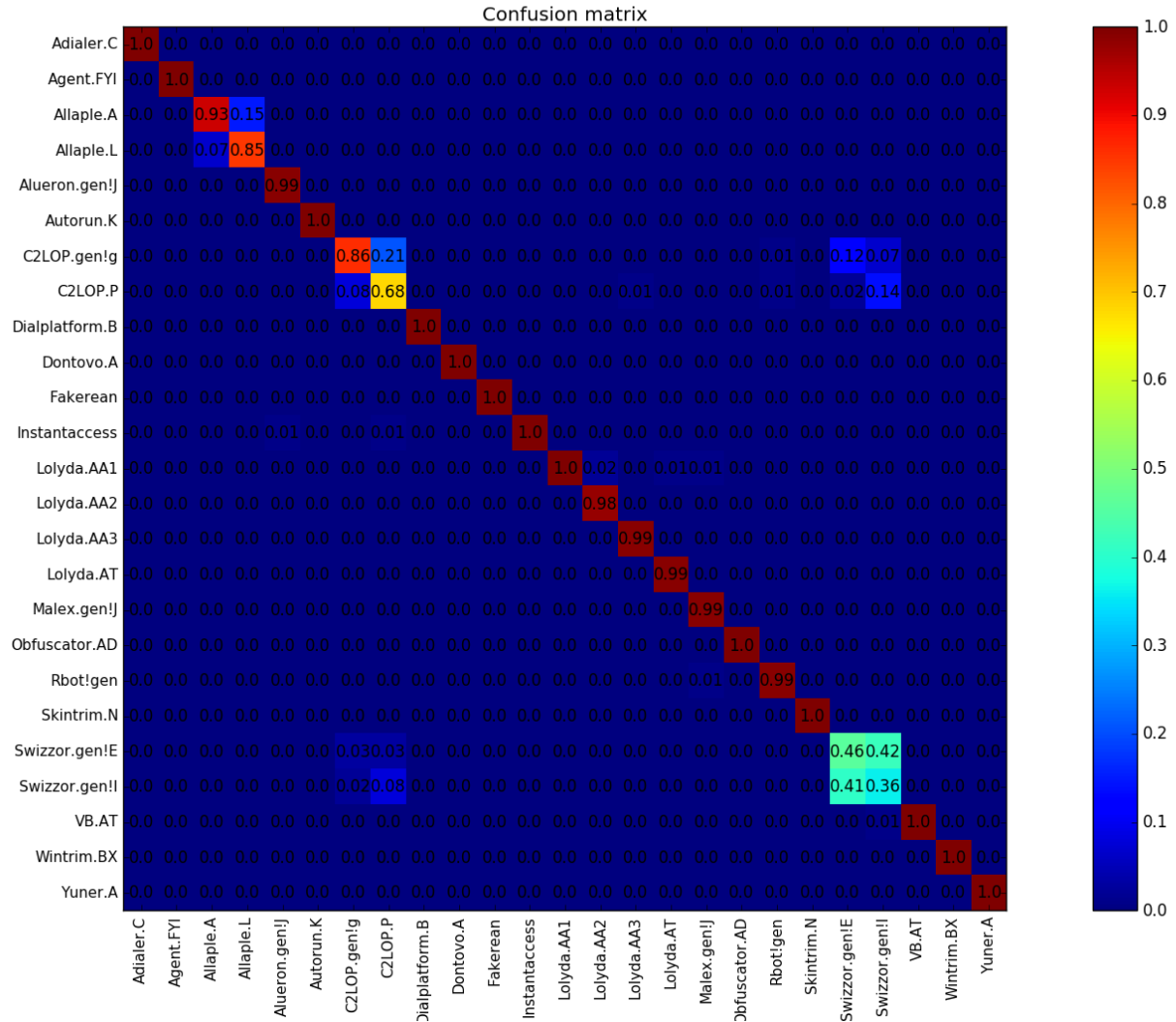


Figure 9: Confusion Matrix Results - After Salting.

FAMILY NAME	ACCURACY BEFORE SALTING	ACCURACY AFTER SALTING
Allaple.A	1	0.93
Allaple.L	0.98	0.85

Figure 10: Accuracy Comparison - Before and After Salting.

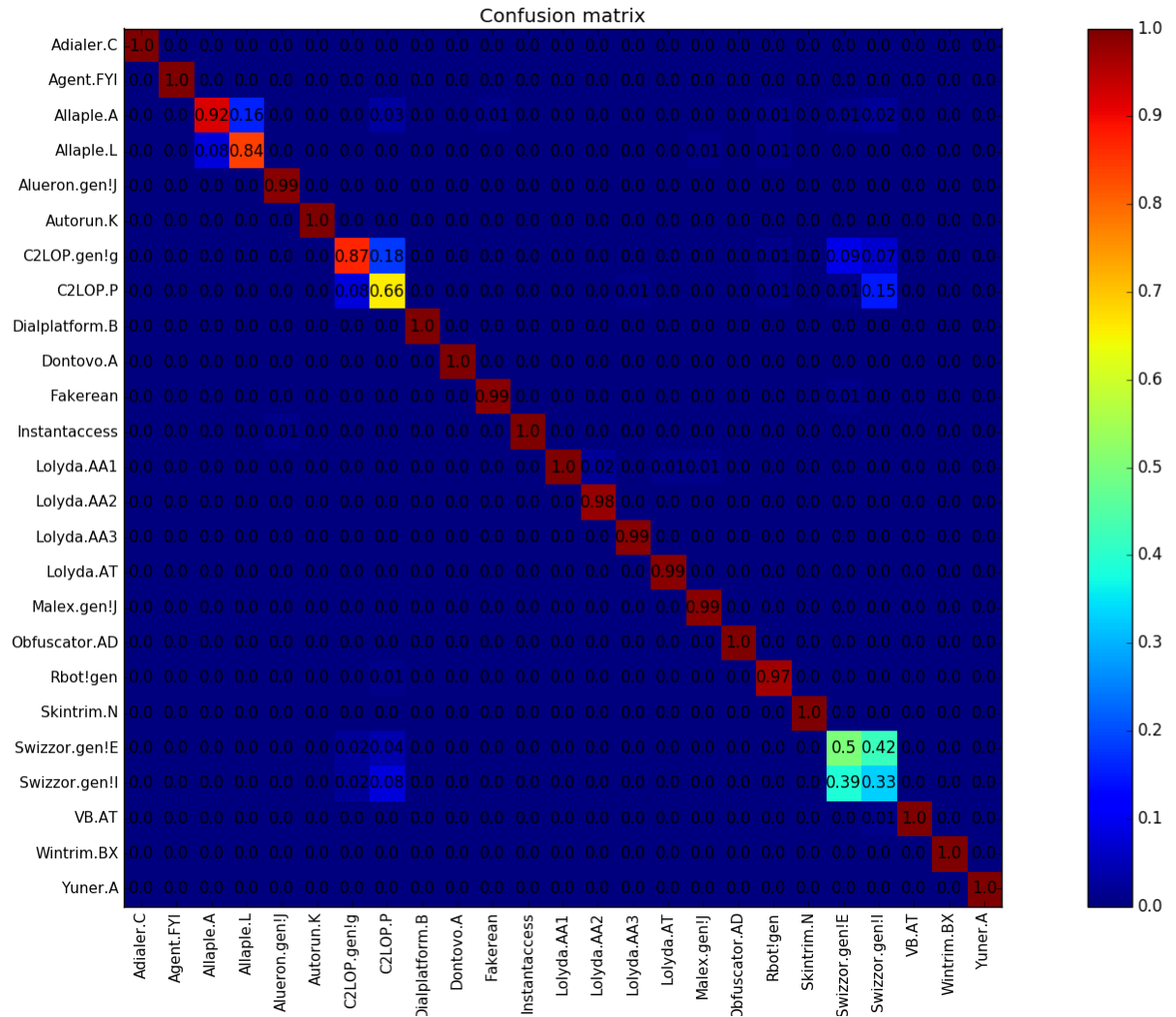


Figure 11: Confusion Matrix - After Salting with Benign Exes.

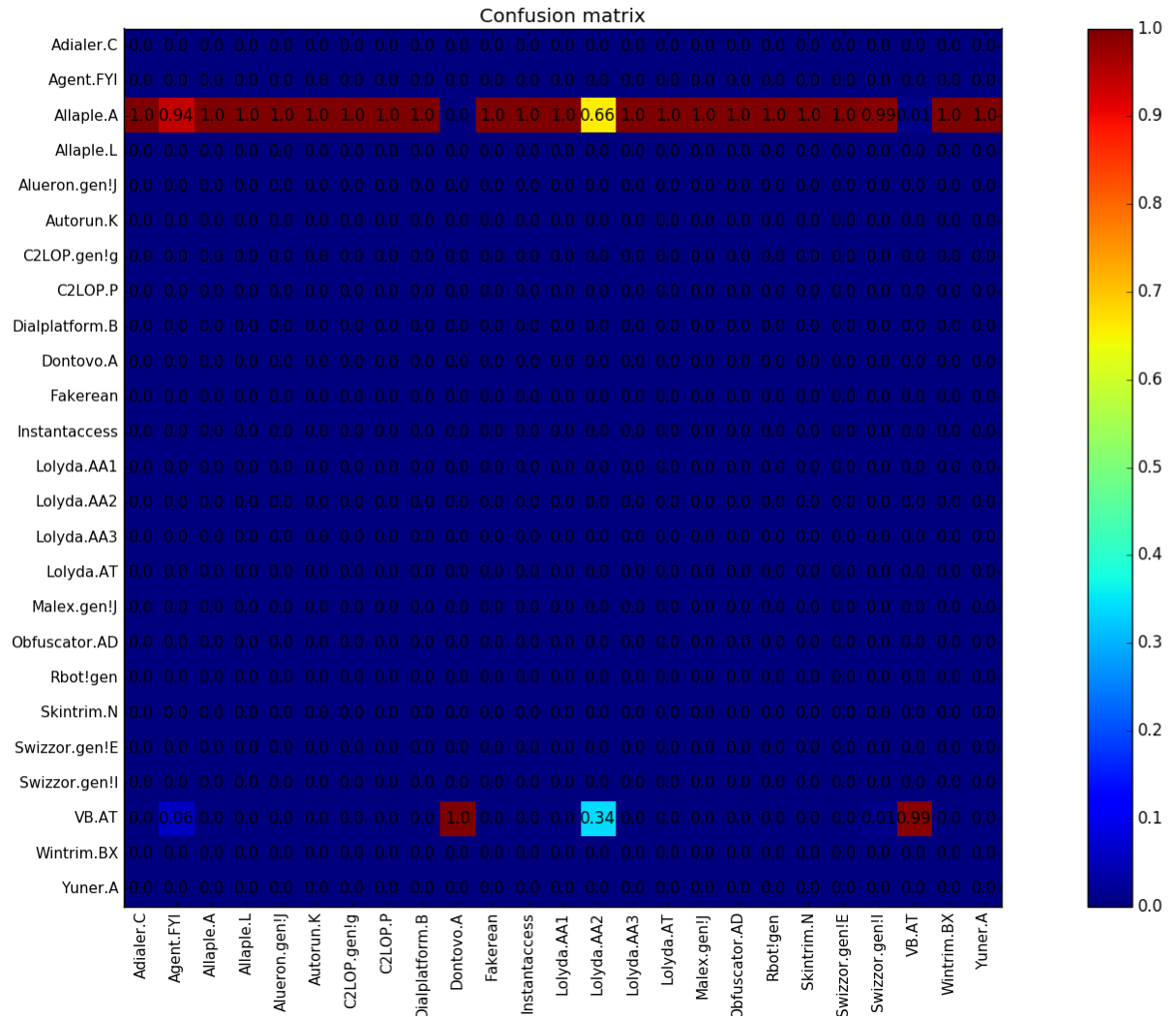


Figure 12: Confusion Matrix - After Salting with Benign Exes and using SVC.





## CHAPTER 5

### Experimenting with Deep Learning

Deep learning using layers of neural networks is the goto technique of computer scientists nowadays when dealing with challenges in computer vision and image processing [24]. Neural networks have been shown to be successful with various categories of image processing problems including but not limited to image recognition, image segmentation et all. Google’s Inceptionv3 and Facebook Artificial Intelligence Research(FAIR) teams came up with pretty good success rates when dealing with the problems in the domain of image classification and object segmentation respectively [25, 26]. Hence, with all these advancements in technology related to deep learning, it was a natural next step to see if these techniques could be leveraged to gain an improvement over the accuracy achieved with gist features.

#### 5.1 Tensorflow for Deep Learning

Neural networks were out of vogue till the recent past owing to the humongous amount of data needed to train them. One of the reasons why neural networks have gained traction in the recent years is the advances made in the fields of hardware and software required to process the large amount of data associated with them [27]. With the recent advancement in GPU technology and software like TensorFlow it is no longer impractical to train neural networks [27, 28].

TensorFlow™ is an open source software library by Google for numerical computation using data flow graphs. Tensorflow is the successor of the earlier closed source “DistBelief” from Google which was used for training and deploying neural networks for pattern recognition [28].

The unit of data in TensorFlow is a set of primitive values shaped into a n-dimensional array. TensorFlow programs can be thought of as comprising the below sections:

1. Building the Computational Graph
2. Running the Computational Graph

A computational graph is a series of TensorFlow operations arranged into a graph of nodes. A node may or may not have a tensor as its input but it usually produces a tensor as output. Once the computational graph is created you could evaluate / run it by creating a session, which encapsulates the control and state of the TensorFlow runtime, and running the graph within it [28].

## 5.2 Experimenting with simple Softmax Regression

Softmax Regression can be thought of as multi-class logistic regression. Logistic regression is a machine learning algorithm that could be used for binary classification problems where the dependent variable is dichotomous (binary). Softmax regression could be thought of as a generalization of logistic regression that could handle multiple classes. The sigmoid function from logistic regression is replaced by the softmax function. There are a set of weights and biases, that along with the input images are used as the input for the softmax function. After processing, it outputs the probability for the image belonging to each input class. Figure 14 [29] would help in understanding the difference between the logistic regression and softmax regression machine learning algorithms [29].

Softmax regression is especially useful for the task in our hand as we want to effectively discover to which malware family a given image belongs to from amongst the various families. Softmax regression outputs the probability of the given image belonging to each class in the input. It consists of two steps: Adding up evidence that our input belongs to a certain class and then we convert that evidence into probabilities. Expressed in mathematical notation, it is defined as

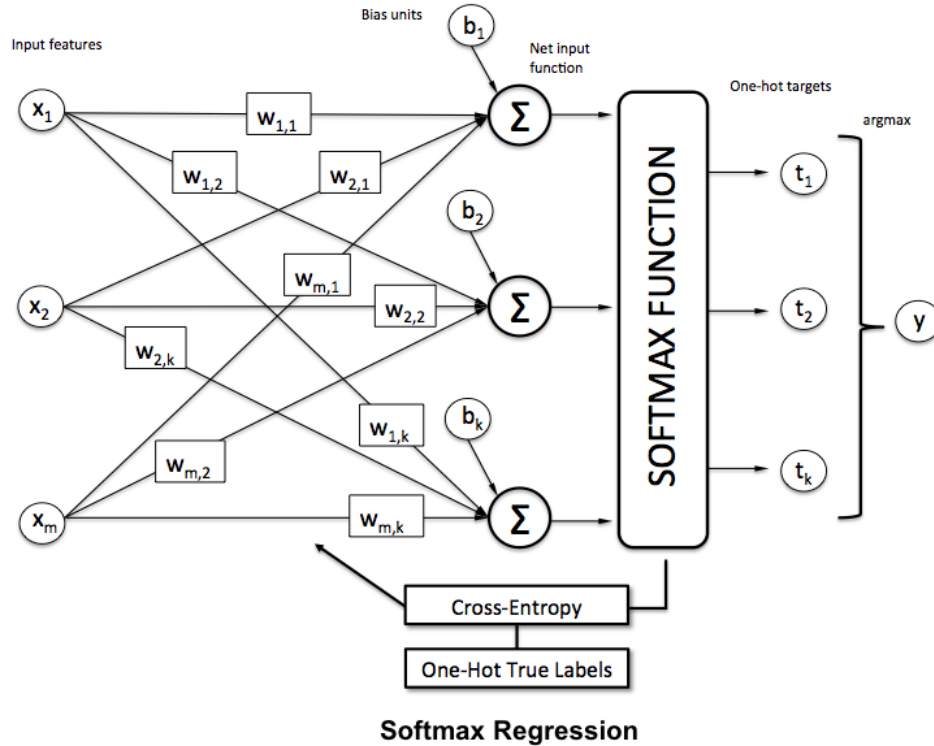
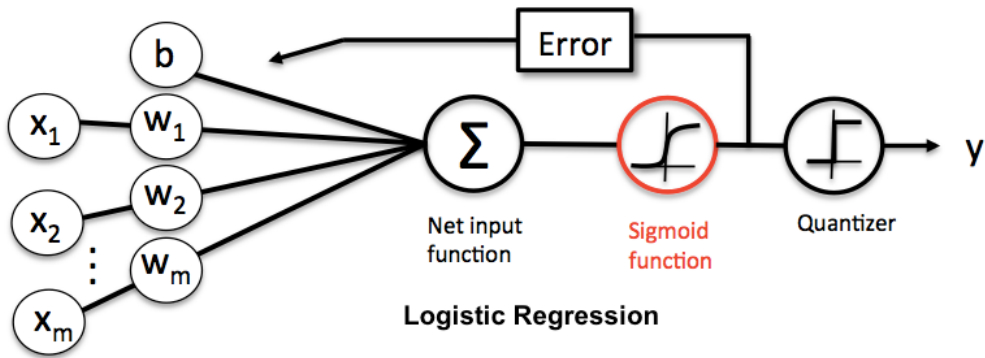


Figure 14: Logistic Regression vs. Softmax Regression.

$$P(y = j | z^{(i)}) = \phi_{(softmax)}(z^{(i)}) = \frac{e^{z_k^{(i)}}}{\sum_{j=0}^k e^{z_k^{(i)}}} \quad (1)$$

$$z = w_0x_0 + w_1x_1 + \dots + w_mx_m = \sum_{l=0}^m w_lx_l = w^{(T)}x. \quad (2)$$

This can be expressed in matrix form it is as shown in Figure 15 [28].

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \text{softmax} \left( \begin{bmatrix} W_{1,1}x_1 + W_{1,2}x_2 + W_{1,3}x_3 + b_1 \\ W_{2,1}x_1 + W_{2,2}x_2 + W_{2,3}x_3 + b_2 \\ W_{3,1}x_1 + W_{3,2}x_2 + W_{3,3}x_3 + b_3 \end{bmatrix} \right)$$

Figure 15: Softmax Regression Matrix Form.

This could be vectorized as shown in Figure 16 [28].

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \text{softmax} \left( \begin{bmatrix} W_{1,1} & W_{1,2} & W_{1,3} \\ W_{2,1} & W_{2,2} & W_{2,3} \\ W_{3,1} & W_{3,2} & W_{3,3} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} \right)$$

Figure 16: Softmax Regression in Simplified Matrix Form.

It would be clear from the above expression that we would represent the variables,  $W, x$  and  $b$  as tensors. These would then be used in the construction of the computational graph. This graph would then be run by creating a session. This way all the heavy computations have been offloaded from Python while still retaining the easy to code and read attributes of Python.

In order to train the model, we would need a cost function and we would need to minimize the cost. The lower the cost the better the model. For this experiment we would be using “cross-entropy” to determine the loss of the model which is as defined as

$$H_{y'}(y) = - \sum_i y'_i \log(y_i) \quad (3)$$

It could be thought of as a function wherein the closer you are to the output the smaller the cross-entropy and is a better measure than mean squared error when it comes to the case of neural networks which output probabilities. To illustrate this further consider an example wherein you are trying to predict the probability of 3 images as belonging to either of 3 malware families (Family A, B, C ).

NEURAL NETWORK 1						
PREDICTED PROBABILITIES			TARGETS			CORRECT PREDICTION?
Family A	Family B	Family C	Family A	Family B	Family C	
0.8	0.1	0.1	1	0	0	YES
0.1	0.8	0.1	0	1	0	YES
0.3	0.4	0.3	0	0	1	NO
NEURAL NETWORK 2						
PREDICTED PROBABILITIES			TARGETS			CORRECT PREDICTION?
Family A	Family B	Family C	Family A	Family B	Family C	
0.4	0.3	0.3	1	0	0	YES
0.3	0.4	0.3	0	1	0	YES
0.3	0.4	0.3	0	0	1	NO

Figure 17: Cross Entropy Example.

From the above table it could be seen that both the neural networks have predicted the same outcomes and hence have the same classification error but Neural Network 1 has more confidence in the predictions that were right (Predicted Probability of 0.8 vs 0.4 for Neural Network 2) and hence is the better model. Also, you will see that mean squared error (MSE) places much emphasis on incorrect predictions and hence using cross-entropy is much more preferable [30].

For this experiment we will be using the gradient descent algorithm to optimize or minimize the cross-entropy function. What this means is that gradient descent

algorithm will find the best values for  $W$  and  $B$  by iteratively trying out different values and in each iteration it picks a better value for these parameters. Think of it as a bowl, wherein the goal is to reach the bottom of the bowl, this is shown in Figure 18 [31]. Gradient descent is usually used when it is impossible or very hard to compute the parameters analytically, say by using linear algebra etc.

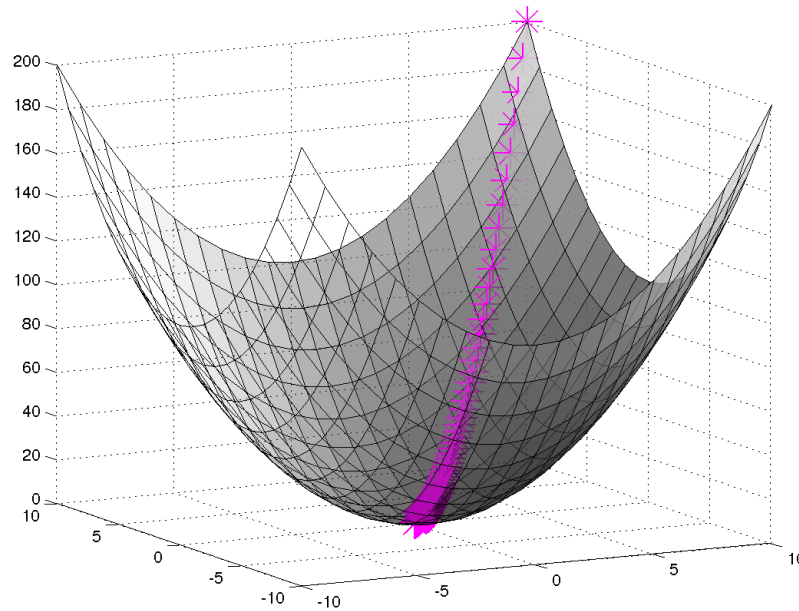


Figure 18: Gradient Descent.

The learning rate is the parameter that defines how big of a step we take downhill, if we take a very big step we run the risk of missing the minimum point and if we take a step that is too small it would take longer for the algorithm to converge. 0.5 is usually a safe value to start with, but one should experiment with various values for this parameter so as to arrive at a value that converges fast as well as finds the minimum point. Finally, a session is created and the computational graph created early is run using it.

Table 11 lists the conditions under which this experiment was carried out. For

this experiment Python 2.7 was used on Ubuntu 14.04. TensorFlow, Numpy were some of the important packages used, when it came to datasets “Maling” was used in these experiments.

OS	Ubuntu 14.04
Python	2.7
Python Libraries	TensorFlow, Numpy, PIL
Datasets	Maling

Table 11: Experimental Setup for simple neural networks.

The experiment was performed with 1,2,3,4 and 5 families. The cost function was “cross-entropy”, the optimization function was “gradient descent” with the “learning rate” set to 0.5 and the activation function was “softmax”. It was observed that as we increased the number of families, the accuracy of the model kept decreasing as shown in Figure 19.

Table 12 shows that the accuracy was 100% when there was only one malware family in the test run.

NO. OF FAMILIES	OPTIMIZER	LOSS FUNCTION	ACCURACY RESULT
1	Gradient Descent Optimizer	Cross Entropy	100
FAMILIES IN THE RUN	TRAINING FILES	TESTING FILES	TOTAL FILES
Adialer.C	100	22	122
Total	100	22	122

Table 12: No of families - 1.

Table 13 shows that the accuracy did not drop when two families were considered in the test run.



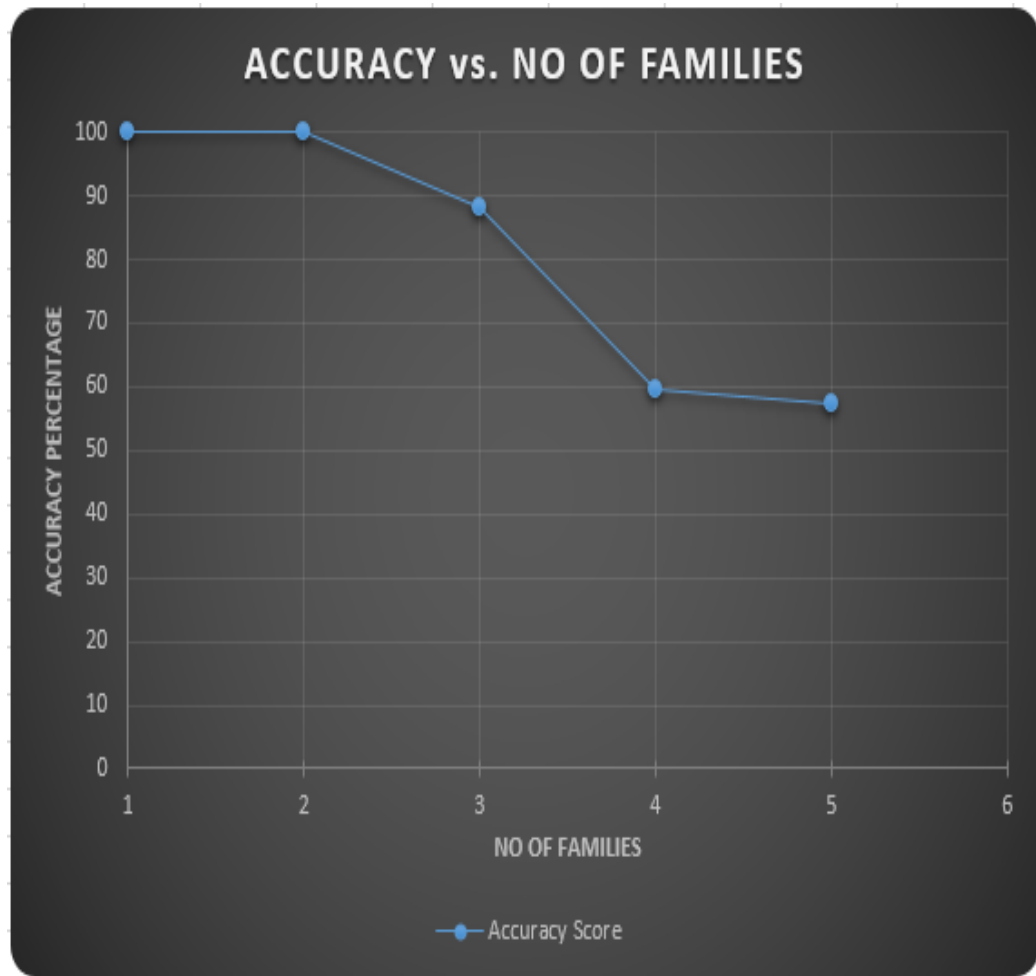


Figure 19: Simple Neural Network Scores.

However, when the 3rd family was introduced, we see that the accuracy dropped to 88% in Table 14.

With the addition of one more family the accuracy went down drastically to 59% as shown in Table 15.

Adding the 5th family resulted in the accuracy dropping to 57% as shown in Table 16.

NO. OF FAMILIES	OPTIMIZER	LOSS FUNCTION	ACCURACY RESULT
2	Gradient Descent Optimizer	Cross Entropy	100
FAMILIES IN THE RUN	TRAINING FILES	TESTING FILES	TOTAL FILES
Adialer.C	100	22	122
Agent.FYI	98	18	116
Total	198	40	238

Table 13: No of families - 2.

NO. OF FAMILIES	OPTIMIZER	LOSS FUNCTION	ACCURACY RESULT
3	Gradient Descent Optimizer	Cross Entropy	88.02
FAMILIES IN THE RUN	TRAINING FILES	TESTING FILES	TOTAL FILES
Adialer.C	100	22	122
Agent.FYI	98	18	116
Allapple.A	2655	294	2949
Total	2853	334	3187

Table 14: No of families - 3.

NO. OF FAMILIES	OPTIMIZER	LOSS FUNCTION	ACCURACY RESULT
4	Gradient Descent Optimizer	Cross Entropy	59.63
FAMILIES IN THE RUN	TRAINING FILES	TESTING FILES	TOTAL FILES
Adialer.C	100	22	122
Agent.FYI	98	18	116
Allaple.A	2655	294	2949
Allaple.L	1432	159	1591
Total	4285	493	4778

Table 15: No of families - 4.

NO. OF FAMILIES	OPTIMIZER	LOSS FUNCTION	ACCURACY RESULT
5	Gradient Descent Optimizer	Cross Entropy	57.42
FAMILIES IN THE RUN	TRAINING FILES	TESTING FILES	TOTAL FILES
Adialer.C	100	22	122
Agent.FYI	98	18	116
Allaple.A	2655	294	2949
Allaple.L	1432	159	1591
Allueron.gen!J	179	19	198
Total	4464	512	4976

Table 16: No of families - 5.

The results are as expected, the simple neural network had a 92% accuracy in classifying the MNIST dataset (Handwritten Digits), so it is not that surprising that it has not done well with the “Maling” malware images dataset.

### 5.3 Convolutional Neural Networks

Convolutional neural networks (CNN) can be thought of as a specialized case of the regular neural networks in that they also have a set of learnable weights and biases but they make the explicit assumption that they will be used for problems involving image recognition. This assumption helps to encode certain properties in the architecture, make the forward function easier to implement and vastly reduce the number of parameters in the network [32].

#### 5.3.1 What is Convolution

A convolutional neural network gets its name from the “convolution” operator. This is the function that extracts features from the input image space. Given below is a small example of how the convolution works over an image. Consider a 5 x 5 matrix which represents the image and a 3 x 3 matrix called the “filter” like the one in Figure 20 [33].

IMAGE DATA					FILTER		
1	1	1	0	0			
0	1	1	1	0	1	0	1
0	0	1	1	1	0	1	0
0	0	1	1	0	1	0	1
0	1	1	0	0			

Figure 20: Convolutional Neural Network - Convolution Example.

You move the “filter” over the image matrix, multiply each cell and add them up to come up with the resultant 3 x 3 matrix. This series of steps is illustrated in the below sequence of images.

Each step involves taking the sub matrix of size 3 x 3, starting at 0,0. The dot product of this matrix with the filter is then calculated. The sum of all dot products make up the value of the first cell in the resultant matrix.

Figure 21 [33] shows the convolution step for the 3 x 3 matrix starting at index 0,0.

1x1	1x0	1x1		1+0+1		4		
0x0	1x1	1x0		0+1+0				
0x1	0x0	1x1		0+0+1				

Figure 21: Convolutional Neural Network - Convolution Step 1.

Figure 22 [33] shows the convolution step for the 3 x 3 matrix starting at index 0,1.

1x1	1x0	0x1		1+0+0		4	3	
1x0	1x1	1x0		0+1+0				
0x1	1x0	1x1		0+0+1				

Figure 22: Convolutional Neural Network - Convolution Step 2.

Figure 23 [33] the convolution step for the 3 x 3 matrix starting at index 0,2.

1x1	0x0	0x1		1+0+0		4	3	4
1x0	1x1	0x0		0+1+0				
1x1	1x0	1x1		1+0+1				

Figure 23: Convolutional Neural Network - Convolution Step 3.

Figure 24 [33] the convolution step for the 3 x 3 matrix starting at index 1,0.

0x1	1x0	1x1		0+0+1		4	3	4
0x0	0x1	1x0		0+0+0		2		
0x1	0x0	1x1		0+0+1				

Figure 24: Convolutional Neural Network - Convolution Step 4.

Figure 25 [33] shows the convolution step for the 3 x 3 matrix starting at index 1,1.

1x1	1x0	1x1		1+0+1		4	3	4
0x0	1x1	1x0		0+1+0		2	4	
0x1	1x0	1x1		0+0+1				

Figure 25: Convolutional Neural Network - Convolution Step 5.

Figure 26 [33] shows the convolution step for the 3 x 3 matrix starting at index 1,2.

1x1	1x0	0x1		1+0+0		4	3	4
1x0	1x1	1x0		0+1+0		2	4	3
1x1	1x0	0x1		1+0+0				

Figure 26: Convolutional Neural Network - Convolution Step 6.

Figure 27 [33] shows the convolution step for the 3 x 3 matrix starting at index 2,0.

0x1	0x0	1x1		0+0+1		4	3	4
0x0	0x1	1x0		0+0+0		2	4	3
0x1	1x0	1x1		0+0+1				

Figure 27: Convolutional Neural Network - Convolution Step 7.

Figure 28 [33] shows the convolution step for the 3 x 3 matrix starting at index 2,1.

0x1	1x0	1x1		0+0+1		4	3	4
0x0	1x1	1x0		0+1+0		2	4	3
1x1	1x0	0x1		1+0+0		2	3	

Figure 28: Convolutional Neural Network - Convolution Step 8.

Figure 29 [33] shows the convolution step for the 3 x 3 matrix starting at index 2,2.

1x1	1x0	1x1		1+0+1		4	3	4
1x0	1x1	0x0		0+1+0		2	4	3
1x1	0x0	0x1		1+0+0		2	3	4

Figure 29: Convolutional Neural Network - Convolution Step 9.

As shown above, the “filter” is slid over the input image pixel by pixel, which is sometimes referred to as “stride”, and for every position the element wise product is computed and is summed up across the row. The resultant 3 x 3 matrix is called “Feature Map”, these act as feature detectors for the input image.

One could understand from the above explanation that using different “filter” will result in the CNN learning different features from the input image. Different filters produce different outcomes on the image. Operations such as edge detection, sharpening of the image, blurring of the image etc could be done with the help of filters. Figure 30 [33] shows an example of a blurring filter.

A CNN learns the values for these “filters” over the period of its training, but parameters such as number of filters, filter size, architecture of the network need to be specified before the training process is started. It is intuitive that there is a tradeoff between the number of filters and the training time. The more the number of filters the more features the CNN learns but the longer the training time will be. Figure 31 [33] summarizes what different “filters” will produce.

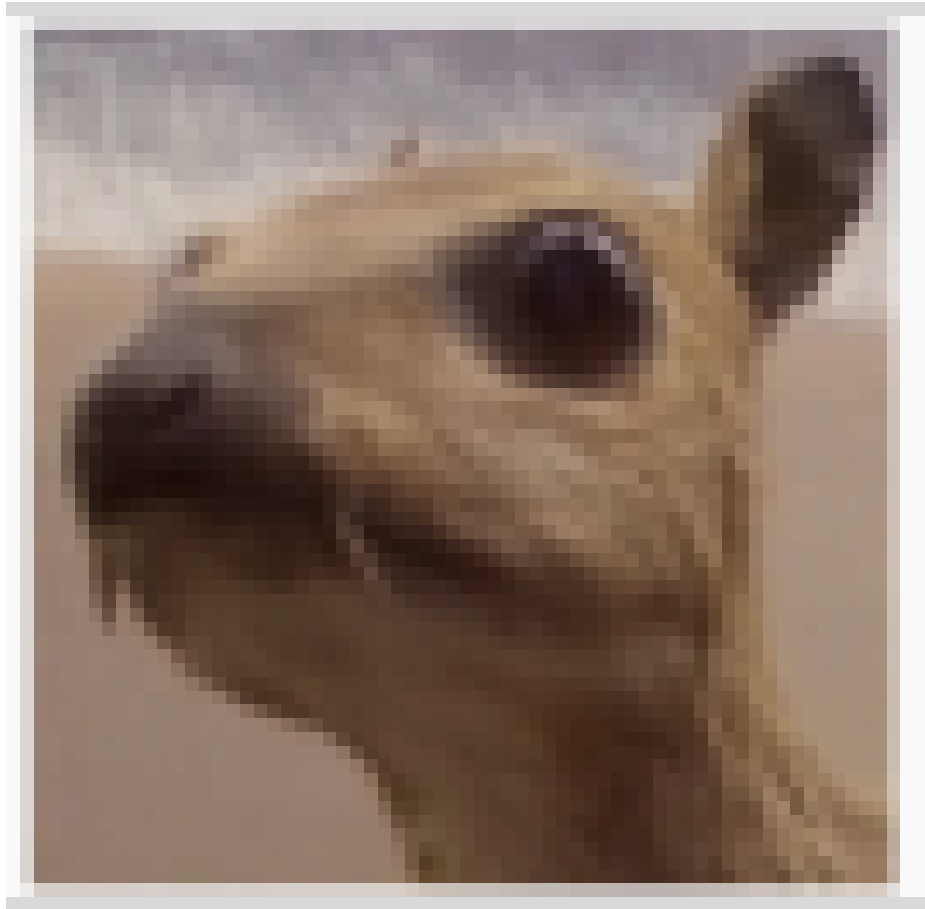


Figure 30: Convolutional Neural Network - Blurring Filter Example Image.










Operation	Filter	Convolved Image
<b>Identity</b>	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
<b>Edge detection</b>	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
<b>Sharpen</b>	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
<b>Box blur</b> (normalized)	$\frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
<b>Gaussian blur</b> (approximation)	$\frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	

Figure 31: Convolutional Neural Network - Filter Example.

### 5.3.2 Architecture

In a CNN, since we have made the design consideration that we will be working with images only we are allowed the freedom to make some changes to how the neurons are stacked in comparison to a regular Neural Networks. A CNN has neurons stacked in three dimensions of width, height and depth. Instead of being connected to all neurons from the previous layers, a neuron will only be connected to a small subset of the neurons in the previous layers. Figures 32, 33 [32] illustrates the differences between a regular neural network and a convolutional neural network.

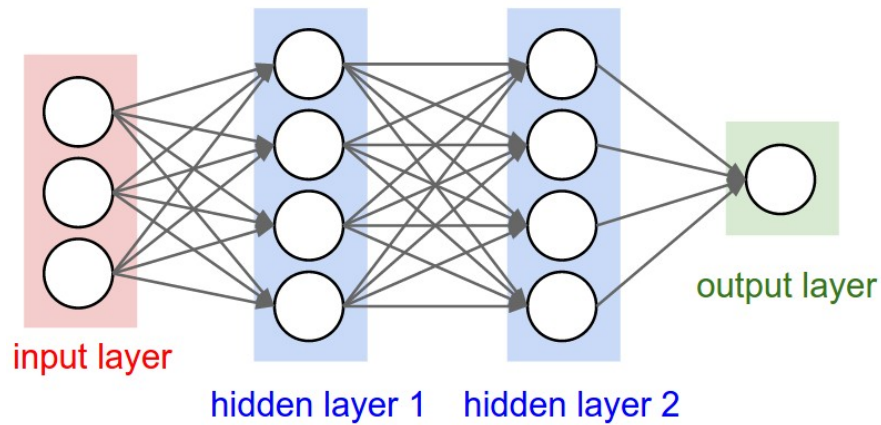


Figure 32: Regular Neural Network.

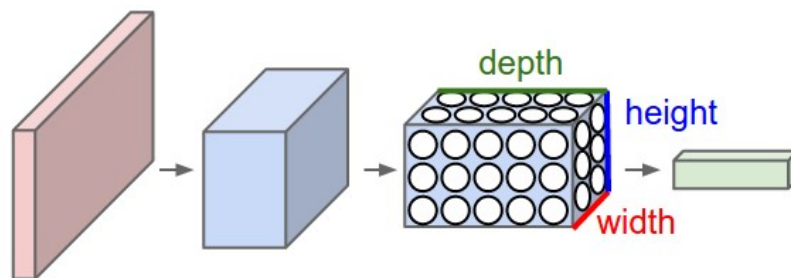


Figure 33: Convolutional Neural Network.

The CNN is made up of a sequence of layers, and each layer transforms the

activation it gets as input to an output through some function which could be differentiated (The Cost function). There are three main types of layers, they are convolutional layer, pooling layer and fully-connected layer. The operations of the convolutional layer were explained in the previous section. The pool layer performs downsampling operation along the height and width dimensions. Downsampling is just a fancy way of saying that you look at each “feature map” and choose the most interesting pixel. The most interesting pixel is just the pixel with the maximum value, hence the name “max-pooling”, Figure 34 [33] gives a visual representation of the process.

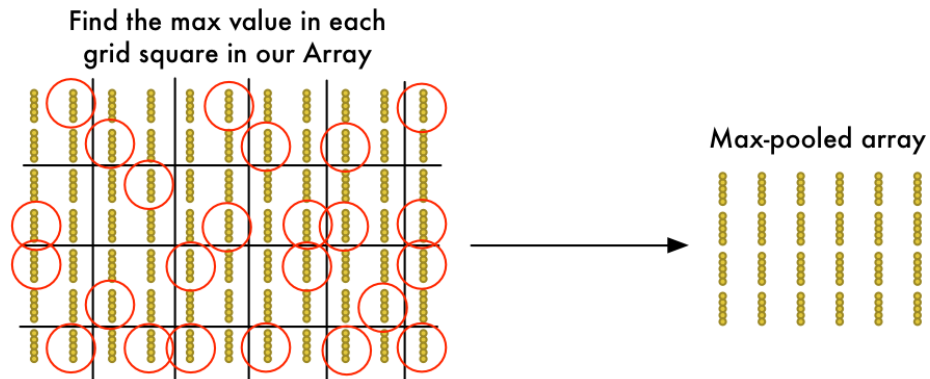


Figure 34: Convolutional Neural Network - Max-pooling.

Once the max-pooled array is formed the only remaining step left is to pass it to a “Fully connected network” that can make a prediction as to the class to which the input image belongs to. An example of a real world CNN is given in Figure 35 [33].

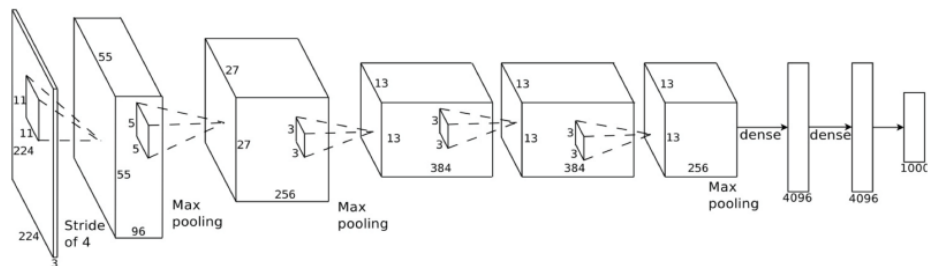


Figure 35: Real World Convolutional Neural Network.

### 5.3.3 Putting it all together in the experiment

Table 17 lists the conditions under which this experiment was carried out. For this experiment Python 2.7 was used on Ubuntu 14.04. TFLearn, a layer of abstraction on top of TensorFlow, Numpy were some of the important packages used, when it came to datasets “Malicia” was used in these experiments.

OS	Ubuntu 14.04
Python	2.7
Python Libraries	TensorFlow, TFLearn, Numpy, PIL
Datasets	Maling

Table 17: Experimental Setup for deep learning.

As explained in the previous sections a convolutional neural network consists of the below steps:

1. Breaking the input image into overlapping tiles
2. Feeding each tile to a smaller neural network
3. Saving the result from each tile into a new array
4. Downsampling the array to reduce the size of the array
5. Making a prediction

There is no limit as to the number of convolutions, max-pooling or fully connected layers involved. The principal idea is to break down a given imaged into smaller portions and filter it down to a single final result. The more the number of convolutions the more complex the features the CNN learns.

A convolutional neural network was trained on 2 families of the “Mallmg” dataset. It had 3 convolution layers, 2 max-pool layers and 2 fully connected layers. The

results from the experiment showing the runtime and the accuracy scores are shown in Figure 36 and Figure 37.

Family 1		Allapple.A		
Family 2		Allapple.L		
<b>CNN Accuracy Stats</b>				
		No of Training Images	No of Testing Images	Method Used
Family 1		2655	294	CNN
Family 2		1432	159	
<b>Accuracy Score</b>		55%		
<b>CNN Runtime Stats</b>				
		Start Time	End Time	Total Time Taken
		2:24 PM	11:00 PM	8:36 AM

Figure 36: Convolutional Neural Network Results.

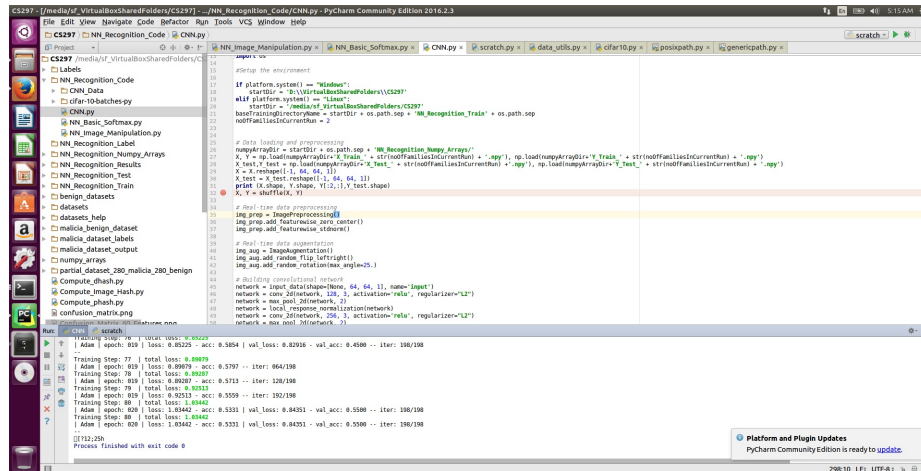


Figure 37: Convolutional Neural Network Results.

### 5.3.4 Disadvantages of Neural Networks

1. Neural networks work like a black box, wherein it is not possible to ascertain with certainty what features are being learned. In essence this makes it difficult

to fix misclassification's.

2. Arriving at the right convolutional network usually requires a lot of experimentation. There is no one size fits all and each image classification problem requires fine tuning the parameters to arrive at the right model as evident from the results.
3. The time taken to run these on a CPU only machine is prohibitive. The same experiment could probably be run in around an hour using the latest GPUs though.
4. Neural networks work like a black box, wherein it is not possible to ascertain with certainty what features are being learned.

## CHAPTER 6

### Conclusion and Future Work

The gist features based scores along with univariate feature selection proved to be both efficient and better than some of the other more complex neural network based scores from our tests. However, when some of the malware images are salted with other images, the accuracy suffered.

Some of the future work that would be interesting is to answer questions like what would happen if all images from across families are salted. It would be interesting to see if that results in gist features picking up important information from those images used for salting and if that would result in good clustering or if the salted images completely confuse the gist based scoring and hence beat the system.

Arriving at the right number of convolution layers, filters and max pooling layers is usually done by trial and error. It would also be a good experiment to run various convolutional neural networks to identify the correct set of parameters that should be used for this dataset and to see if this could be extended do well on malware belonging to other datasets. Another interesting question to answer when salting images is to see if the images that are used for salting purposes would become interesting features when learning features with convolutional neural networks.

It would be interesting to use principal component analysis on these salted images, as it would most probably separate the image used for salting and the actual malware image as they are very different in structure. Hence, this could act as the first layer of filtering before either running tests using gist features or convolutional neural networks.

## LIST OF REFERENCES

- [1] “Venture beat.” [Online]. Available: <http://venturebeat.com/2011/04/04/symantec-identified-286m-malware-threats-in-2010/>
- [2] S. Ranveer and S. Hiray, “Comparative analysis of feature extraction methods of malware detection,” *International Journal of Computer Applications*, vol. 120, no. 5, pp. 1--7, 2015.
- [3] A. Moser, C. Kruegel, and E. Kirda, “Limits of static analysis for malware detection,” *Twenty-Third Annual Computer Security Applications Conference (ACSAC 2007)*, 2007.
- [4] I. You and K. Yim, “Malware obfuscation techniques: A brief survey,” in *2010 International Conference on Broadband, Wireless Computing, Communication and Applications*, Nov 2010, pp. 297--300.
- [5] U. Bayer, A. Moser, C. Kruegel, and E. Kirda, “Dynamic analysis of malicious code,” *Journal in Computer Virology*, vol. 2, no. 1, pp. 67--77, 2006. [Online]. Available: <http://dx.doi.org/10.1007/s11416-006-0012-2>
- [6] M. I. Sharif, A. Lanzi, J. T. Giffin, and W. Lee, “Impeding malware analysis using conditional code obfuscation.” in *NDSS*, 2008.
- [7] “Spyware, adware, systemware and cookies,” *Network Security*, vol. 2002, no. 9, pp. 4 -- 5, 2002. [Online]. Available: <http://www.sciencedirect.com/science/article/pii/S1353485802090062>
- [8] A. Gazet, “Comparative analysis of various ransomware virii,” *Journal in Computer Virology*, vol. 6, no. 1, pp. 77--90, 2010. [Online]. Available: <http://dx.doi.org/10.1007/s11416-008-0092-2>
- [9] C. Nachenberg, “Computer virus-coevolution,” *Communications of the ACM*, vol. 50, no. 1, pp. 46--51, 1997.
- [10] A. Aziz, “Computer worm defense system and method,” Aug. 23 2011, uS Patent 8,006,305.
- [11] Z. M. X. Q. L. Chunming, “Analysis of trojan horse and its detection [j],” *Computer Engineering and Applications*, vol. 28, p. 053, 2003.
- [12] M. B. Schmidt, A. C. Johnston, and K. P. Arnett, “An empirical investigation of rootkit awareness,” *RESEARCH YEARBOOK*, 2006.



- [13] P. K. Singh and A. Lakhotia, "Analysis and detection of computer viruses and worms: An annotated bibliography," *SIGPLAN Notices*, vol. 37, no. 2, pp. 29--35, 2002.
- [14] T. Osama, M. ELzubair, O. Altom, and A. Eldewahi, "Keylogger prevention technique," 2016.
- [15] D. Samosseiko, "The partnerka - what is it, and why should you care," in *Proc. of Virus Bulletin Conference*, 2009.
- [16] A. Stamminger, C. Kruegel, G. Vigna, and E. Kirda, "Automated spyware collection and analysis," in *International Conference on Information Security*. Springer, 2009, pp. 202--217.
- [17] P. Ferrie and P. Ször, "Hunting for metamorphic," *Virus, págs*, pp. 123--143, 2001.
- [18] Z. Wang and H. Wang, "Study on anti-virus engine based on heuristic search of polymorphic virus behavior," *Research and Exploration in Laboratory*, vol. 25, no. 9, pp. 1089--1091, 2006.
- [19] T. H. Lee, "Generalized aho-corasick algorithm for signature based anti-virus applications," in *2007 16th International Conference on Computer Communications and Networks*, Aug 2007, pp. 792--797.
- [20] M. Egele, T. Scholte, E. Kirda, and C. Kruegel, "A survey on automated dynamic malware-analysis techniques and tools," *ACM Comput. Surv.*, vol. 44, no. 2, pp. 6:1--6:42, Mar. 2008. [Online]. Available: <http://doi.acm.org/10.1145/2089125.2089126>
- [21] L. Nataraj, S. Karthikeyan, G. Jacob, and B. S. Manjunath, "Malware images: Visualization and automatic classification," in *Proceedings of the 8th International Symposium on Visualization for Cyber Security*, ser. VizSec '11. New York, NY, USA: ACM, 2011, pp. 4:1--4:7. [Online]. Available: <http://doi.acm.org/10.1145/2016904.2016908>
- [22] A. Oliva and A. Torralba, "Modeling the shape of the scene: A holistic representation of the spatial envelope," *Int. J. Comput. Vision*, vol. 42, no. 3, pp. 145--175, May 2001. [Online]. Available: <http://dx.doi.org/10.1023/A:1011139631724>
- [23] M. Douze, H. Jégou, H. Sandhawalia, L. Amsaleg, and C. Schmid, "Evaluation of gist descriptors for web-scale image search," in *ACM International Conference on Image and Video Retrieval*. ACM, 2009, p. 19.

- [24] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” in *Advances in neural information processing systems*, 2012, pp. 1097--1105.
- [25] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, “Rethinking the inception architecture for computer vision,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 2818--2826.
- [26] P. O. Pinheiro, R. Collobert, and P. Dollar, “Learning to segment object candidates,” in *Advances in Neural Information Processing Systems*, 2015, pp. 1990--1998.
- [27] K.-S. Oh and K. Jung, “{GPU} implementation of neural networks,” *Pattern Recognition*, vol. 37, no. 6, pp. 1311 -- 1314, 2004. [Online]. Available: <http://www.sciencedirect.com/science/article/pii/S0031320304000524>
- [28] “Getting started with tensorflow.” [Online]. Available: [https://www.tensorflow.org/get\\_started/get\\_started](https://www.tensorflow.org/get_started/get_started)
- [29] “What is softmax regression and how is it related to logistic regression?” [Online]. Available: <http://www.kdnuggets.com/2016/07/softmax-regression-related-logistic-regression.html>
- [30] “Why you should use cross-entropy error instead of classification error or mean squared error for neural network classifier training.” [Online]. Available: <https://jamesmccaffrey.wordpress.com/2013/11/05/why-you-should-use-cross-entropy-error-instead-of-classification-error-or-mean-squared-error-for-neural-network-classifier-training/>
- [31] “Gradient descent visualization.” [Online]. Available: <https://www.mathworks.com/matlabcentral/fileexchange/35389-gradient-descent-visualization?requestedDomain=www.mathworks.com>
- [32] “Convolutional neural networks for visual recognition.” [Online]. Available: <http://cs231n.github.io/convolutional-networks/>
- [33] “An intuitive explanation of convolutional neural networks.” [Online]. Available: <https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/>