2020/2021

MINI PROJECT

Submitted in fulfillment of the requirements for the

MINI PROJECT GRADE FROM THE LEBANESE UNIVERSITY FACULTY OF ENGINEERING – BRANCH III

Major: Telecommunication Engineering

Prepared By:

Hussein Kazem

Comparison of Convolutional Neural Network architectures for Malware Classification on MalIMG and MaleVIS Datasets

Supervised by:

Dr. Ahmad Fadlallah

Defended on 23 July 2021 in front of the jury:

Dr. Ahmad Fadlallah	**President**

ACKNOWLEDGEMENTS

Immeasurable appreciation and deepest gratitude for the help and support are extended to the following persons who, in one way or another, have contributed in making this study possible.

Dr. **Hassan Shreim**, the faculty's director, and Prof. **Youssef Harkouss** the Electrical Engineering department Chief, for their well administration and exerting efforts to keep our faculty in its leadership.

Prof. **Ahmad Fadlallah**, our supervisor, for his constant encouragement and inspiration to conduct efficient work on this project and obtain promising results. We are extremely grateful and indebted for his valuable guidance and orientation.

All the doctors of the faculty of engineering that we had the opportunity to be their students, for their continuous support across the years.

ABSTRACT

The number of malicious files detected every year are counted by millions. One of the main reasons for these high volumes of different files is the fact that, in order to evade detection, malware authors add mutation. This means that malicious files belonging to the same family, with the same malicious behavior, are constantly modified or obfuscated using several techniques, in such a way that they look like different files. In order to be effective in analyzing and classifying such large amounts of files, we need to be able to categorize them into groups and identify their respective families on the basis of their behavior. In this paper, malicious software is visualized as images since its ability to capture minor changes while retaining the global structure helps to detect variations.

Multiple Different approaches and neural network approaches have been attempted and documented over the years, in this report we attempt to implement and compare the results of different architectures over 2 different malware images datasets: MalIMG and MaleVIS.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

GENERAL INTRODUCTION

One of the major challenges in the realm of security threats is malicious software which is also referred as malware. The main focus of malware is, to gather the personal information without the attention of users and to disturb the computer operations which makes problems for users. There are many kinds of malware i.e. Virus, Worm, Trojan-horse, Rootkit, Backdoor, Spyware, Adware etc. Annual reports from antivirus companies show that thousands of new malware are created every single day. These new malware become more sophisticated that they could no longer be detected by the traditional detection techniques such as signature-based detection, heuristic detection or behavior-based detection. Signature-based detection searches for specified bytes sequences into an object so that it can identify exceptionally a particular type of a malware. Its drawback is that it cannot detect zero-day or new malware since these malware signatures are not supposed to be listed into the signature database.

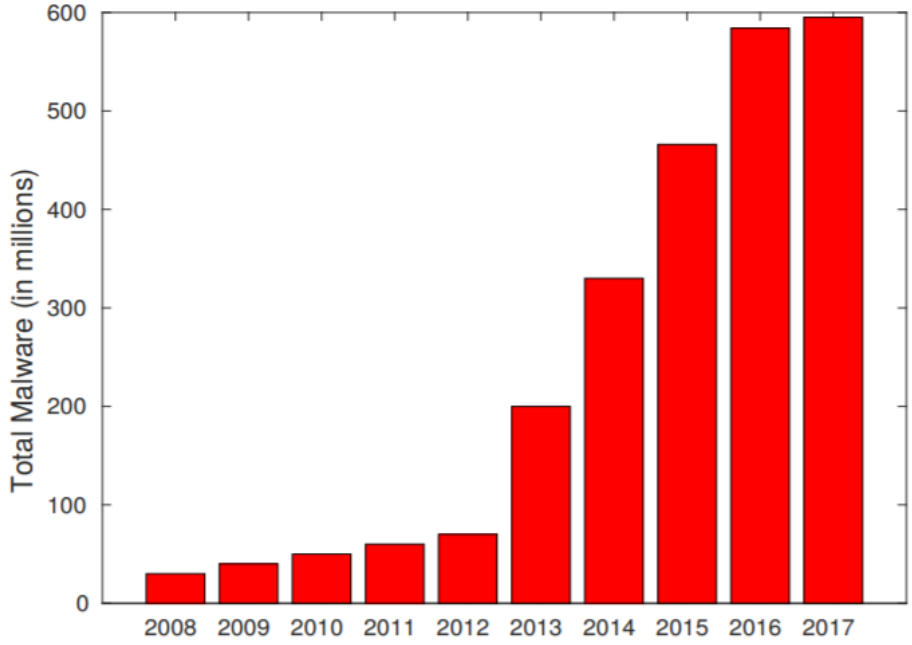

Fig. 0.1. Last 10 years malware statistics

Previous research on malware classification suggests that malware samples typically fall into a family that shares common behaviors, i.e. most new malware are variants of existing ones.

We use Convolutional Neural Networks (CNN), a deep learning architecture (DL), to tackle this problem. Recently, DL has produced state-of-the-art performance for various tasks in many fields such as natural language processing, computer vision, speech recognition, and bioinformatics. However, the capabilities for applying CNNs has not been well explored in many other fields. One field that may benefit significantly by advances in DL is cyber security. With the recent success of DL (especially CNNs) in various classification problems. We are going to be

testing different CNN architectures on 2 Malware Images datasets and comparing the resulting performance of each model on each dataset.

CHAPTER 1: MALWARES

1.1 Introduction

Malware is the collective name for a number of malicious software variants, including viruses, ransomware and spyware. Shorthand for malicious software, malware typically consists of code developed by cyber-attackers, designed to cause extensive damage to data and systems or to gain unauthorized access to a network. Malware is typically delivered in the form of a link or file over email and requires the user to click on the link or open the file to execute the malware.

Malware has actually been a threat to individuals and organizations since the early 1970s when the Creeper virus first appeared. Since then, the world has been under attack from hundreds of thousands of different malware variants, all with the intent of causing the most disruption and damage as possible.

Malware delivers its payload in a number of different ways. From demanding a ransom to stealing sensitive personal data, cybercriminals are becoming more and more sophisticated in their methods.

1.2 Types of Malwares:

The following is a list of some of the more common malware types and definitions.

1.2.1 Virus

Possibly the most common type of malware, viruses attach their malicious code to clean code and wait for an unsuspecting user or an automated process to execute them. Like a biological virus, they can spread quickly and widely, causing damage to the core functionality of systems, corrupting files and locking users out of their computers. They are usually contained within an executable file.

1.2.2 Worms

Worms get their name from the way they infect systems. Starting from one infected machine, they weave their way through the network, connecting to consecutive machines in order to continue the spread of infection. This type of malware can infect entire networks of devices very quickly.

1.2.3 Spyware

Spyware, as its name suggests, is designed to spy on what a user is doing. Hiding in the background on a computer, this type of malware will collect information without the user knowing, such as credit card details, passwords and other sensitive information.

1.2.4 Trojans

Just like Greek soldiers hid in a giant horse to deliver their attack, this type of malware hides within or disguises itself as legitimate software. Acting discretely, it will breach security by creating backdoors that give other malware variants easy access.

1.2.5 Ransomware

Also known as scareware, ransomware comes with a heavy price. Able to lockdown networks and lock out users until a ransom is paid, ransomware has targeted some of the biggest organizations in the world today — with expensive results.

1.3 Malware Analysis

Malware analysis involves mainly two techniques: static analysis and dynamic analysis. Static analysis consists of examining the code or structure of a program without executing it. This kind of analysis can confirm whether a file is malicious, provide information about its functionality and can also be used to produce a simple set of signatures.

The most common static analysis approaches are:

- Finding sequences of characters or strings. Searching through the strings of a program is the most simple way to obtain hints about its functionality. For instance, you can find strings related to printed messages, URLs accessed by the program, the location of files modified by the executable and names of common Windows dynamic link libraries (DLLs).
- Analysis of the Portable Executable File Format. The Portable Executable(PE) file format is used by Windows executables, object code and DLLs. Among the information it includes, the most useful pieces of information are the linked libraries and functions as well as the metadata about the file included in the headers.
- Searching for packed/encrypted code. Malware writers usually use packing and encryption to make their files more difficult to analyze. Software programs that have been packed or encrypted usually contain very few strings and higher entropy compared to legitimate programs.

– Disassembling the program, i.e. recovering the symbolic representation from the machine code instructions.

Dynamic analysis involves executing the program and monitoring its behavior on the system. Unlike static analysis, dynamic analysis allows to observe the actual actions executed by the a program. It is typically performed when static analysis has reached a dead end, either due to obfuscation and packing, or by having exhausted the available static analysis techniques. Some techniques are:

– Function Call Monitoring. The behavior of the program is analyzed by using the traces containing the sequence of functions invoked by the executable under analysis.
– Function Parameter Analysis. Consists of tracking the values of parameters and function return values.
– Information Flow Tracking. Analyze how a program processes data and how data is propagated through the system.
– Instruction Trace. Analysis of the complete sequence of machine instructions executed by the program.

Both methods have its own advantages and disadvantages. On the one hand, static analysis is faster but suffers from code obfuscation, techniques used by malware authors to conceal the malicious purpose of the program. On the other hand, code obfuscation techniques and polymorphic malware fails at dynamic analysis because it analyses the runtime behavior of a program by monitoring it while in execution. However, each malware sample must be executed in a safe environment for a specific time for monitoring its behavior, which is a very time-consuming process. In addition, the environment might be quite different from the real runtime environment and malware may behave in different ways in the two environments, and under some conditions the actions of malware might not be triggered and thus, not logged.

1.4 Malware Analysis Using Machine Learning Techniques

The use of machine learning algorithms to address the problem of malicious software detection and classification has increased during the last decade. The success of these approaches has increased thanks to: (1) the rise of commercial feeds of malware; (2) the reduction in cost of computing power; and (3) the advances made in the machine learning field. Several methods have been applied based on features extracted from both static and dynamic analysis. Then the features extracted are used to train a model. The type of features can be broadly divided into two groups: (1) static features and (2) dynamic features.

Static features are those extracted from the binary of the malware without executing it. Dynamic features are those extracted by executing and observing the behavior of malware.

1.5 Visualizing Malwares as Images:

To visualize a malware sample as an image, every byte has to be interpreted as one pixel in an image. Then, the resulting array has to be organized as a 2-D array and visualized as a gray scale image. Values are in the range [0,255] (0:black, 255:white). The main benefit of visualizing a malicious executable as an image is that the different sections of a binary can be easily differentiated. In addition, malware authors only used to change a small part of the code to produce new variants. Thus, if old malware is re-used to create new binaries the resulting ones would be very similar. Additionally, by representing malware as an image it is possible to detect the small changes while retaining the global structure of samples belonging to the same family. In most cases, when observed in detail, one can notice several sections in the program, which usually have distinct feature patterns. In addition, with this representation you can detect where zero-padding has been applied. Zero-padding is mainly used for block alignment but malware authors also use it to reduce the overall entropy of an executable.

A given malware binary file is first read in a vector of 8-bits unsigned integers. Next, the binary value of each component of this vector is converted to its equivalent decimal value (e.g. the decimal value for [00000000] in binary is [0] and for [11111111] is [255]) which is then saved in a new decimal vector representative of the malware sample. At last, the resulting decimal vector is reshaped to a 2D matrix and visualized as a grayscale image. Selecting width and height of the 2D matrix (i.e. the spatial resolution of the image) mainly depends on the malware binary file size.

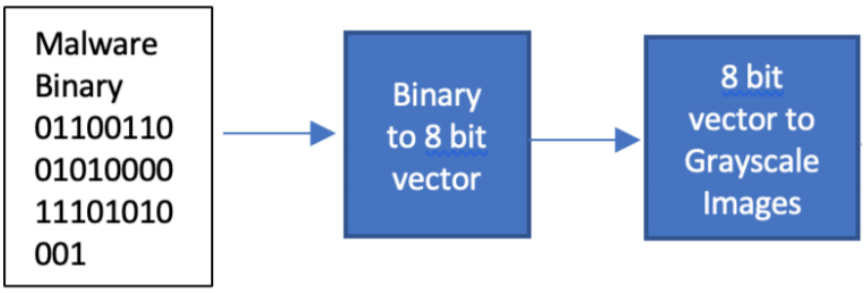

Figure 1.1: Converting malware binary codes to Grayscale images.

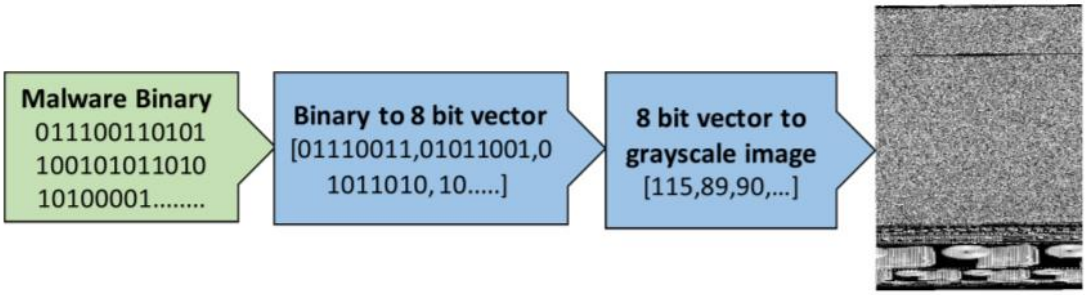

Fig. 1.2. Overview of malware visualization process. The malware binary file is divided into 8-bit sequences which are then converted to equivalent decimal values. This decimal vector is reshaped and gray-scale image is generated that represent the malware sample.

Malware variants belonging to same family usually have similar texture (i.e. visual appearance). Fig. 2.3 provides some examples that support this observation.

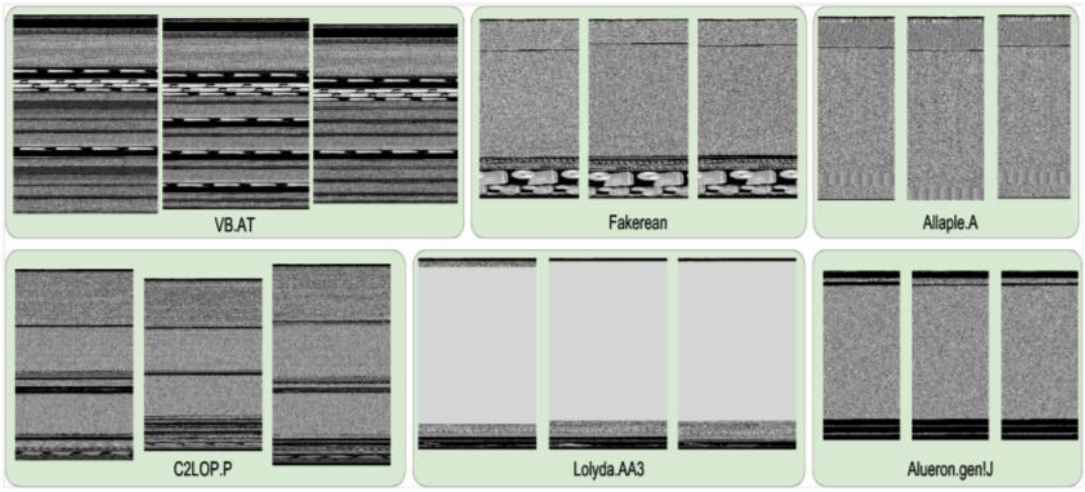

Fig. 1.3. Samples of malware grayscale images belonging to different malware families. These images are acquired from Malimg Dataset. We can see that malware samples from same family are visually similar. Note that images shown above are rescaled for better visualization.

1.6 Conclusion:

In this chapter, we introduced malwares along with a few of their types. In addition, we introduced the concept of malware analysis: static, dynamic and machine learning based analysis. And finally, we presented the idea of visualizing malwares as grey scale images.

CHAPTER 2: CONVOLUTIONAL NEURAL NETWORKS

2.1 Definition

Deep learning is a sub-branch of the machine learning field, inspired by the structure of the brain. Deep learning techniques used in recent years continue to show an impressive performance. A convolutional neural network (CNN) is a class of deep neural networks used in image recognition problems. Convolutional neural network (CNN) is a powerful tool which is extensively utilized for image classification. The hierarchical structure and efficient feature extraction characteristics from an image make CNN a dynamic model for image classification. Coming to how CNN works, the images given as input must be recognized by computers and converted into a format that can be processed. For this reason, images are first converted to matrix format. The system determines which image belongs to which label based on the differences in images and therefore in matrices. It learns the effects of these differences on the label during the training phase and then makes predictions for new images using them. CNN consists of three different layers that are a convolutional layer, pooling layer, and fully connected layer to perform these operations effectively. The feature extraction process takes place in both convolutional and pooling layers. On the other hand, the classification process occurs in fully connected layer.

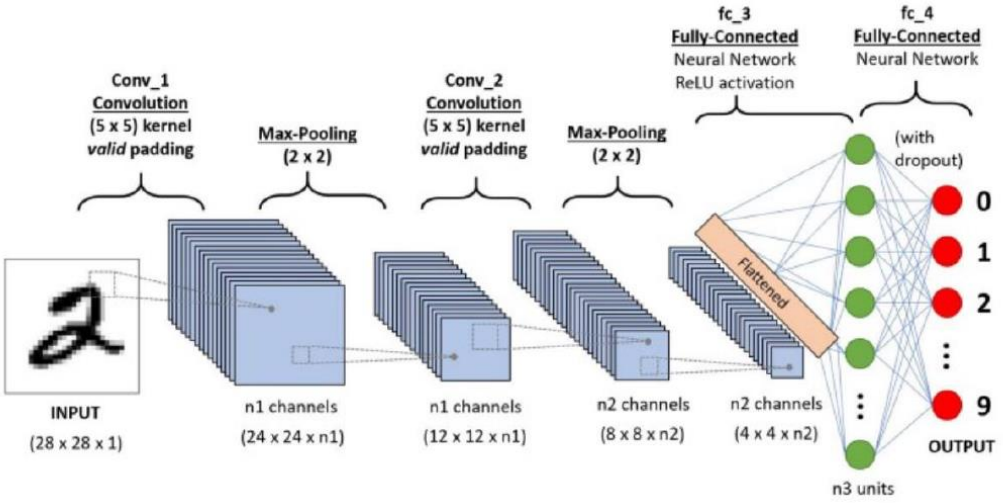

Figure 2.1: CNN Sequence to Classify Handwritten Digits

2.2 Convolution Layer

Convolutional layer is the base layer of CNN. It is responsible for determining the features of the pattern. In this layer, the input image is passed through a filter. The values resulting from filtering consist of the feature map. This layer applies some kernels that slide through the pattern to extract low- and high-level features in the pattern.

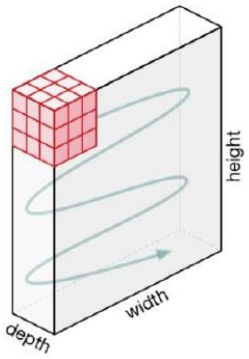

Figure 2.2. Movement of Kernel

The kernel is a 3x3 or 5x5 shaped matrix to be transformed with the input pattern matrix. Stride parameter is the number of steps tuned for shifting over input matrix. The output of convolutional layer can be given as:

$$x_j^l = f\left(\sum_{a=1}^{N} w_j^{l-1} * y_a^{l-1} + b_j^l\right)$$

where x_{jl} is the j-th feature map in layer l, w_{jl}-1 indicates j-th kernels in layer l-1, yal-1 represents the a-th feature map in layer 1-1, b_j l indicates the bias of the j-th feature map in layer l, N is number of total features in layer l-1, and (*) represents vector convolution process. Figure shows how kernel/filter can extract potential features by using the stride.

The objective of the Convolution Operation is to extract the high-level features such as edges, from the input image. CNN is not limited to one Convolutional Layer. Conventionally, the first Conv-Layer is responsible for capturing the Low-Level features such as edges, color, gradient orientation, etc. With added layers, the architecture adapts to the High-Level features as well.

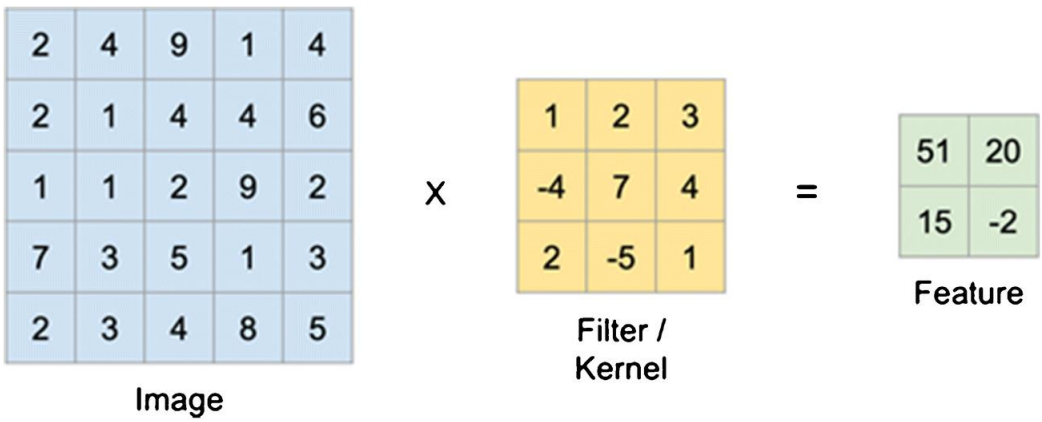

Figure 2.3 - Convolution operator with kernel size 3 and stride 2

2.3 Pooling Layer

The second layer after the convolutional layer is the pooling layer. Pooling layer is usually applied to the created feature maps for reducing the number of feature maps and network parameters by applying corresponding mathematical computation. In this study, we used maxpooling. The max-pooling process selects only the maximum value by using the matrix size specified in each feature map, resulting in reduced output neurons. It is connected to the fully connected layer after global average pooling layer. The other intermediate layer used is the dropout layer. The main purpose of this layer is to prevent network over fitting and divergence.

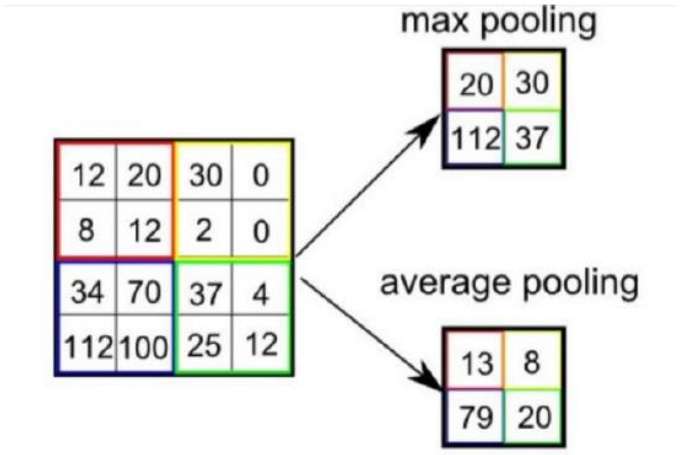

Figure 2.4. Types of Pooling

The Convolutional Layer and the Pooling Layer, together form the i-th layer of a Convolutional Neural Network. Depending on the complexities in the images, the number of such layers may be increased for capturing low-levels details even further, but at the cost of more computational power. After going through the above process, the model is successfully enabled to understand the features. Moving on, the final output is going to be flattened and fed to a regular Neural Network for classification purposes.

2.4 Non-Linearity (ReLU)

ReLU stands for Rectified Linear Unit for a non-linear operation. ReLU purpose is to introduce non-linearity in the CNN. (ReLU) activation function is used to learn complex functional mappings between the inputs and response variables. It is a linear function that produces the input directly if it is positive; otherwise, it will output zero.

$$f(x) = \max(0, x)$$

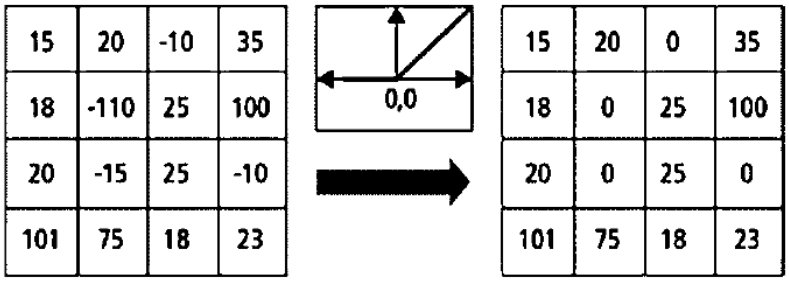

Figure 2.5: ReLU Operation

2.5 Classification – Fully Connected Layer

The feature map matrix will be flattened into column vector (x1, x2, x3, …). The flattened output is fed to a feed-forward neural network and backpropagation is applied to every iteration of training. Over a series of epochs, the model is able to distinguish between dominating and certain low-level features in images and classify them using the Softmax Classification technique.

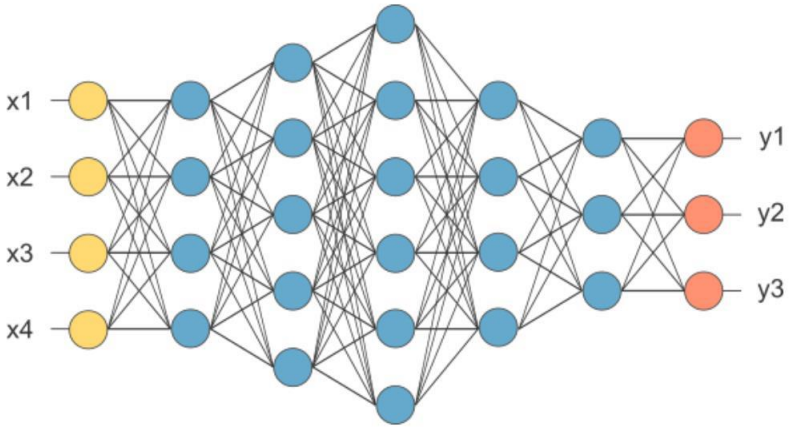

Figure 2.6. Feed-Forward Neural Network

2.6 Classification – Softmax

Softmax Classifier takes as input a vector z of K real numbers, and normalizes it into a probability distribution consisting of K probabilities proportional to the exponentials of the input numbers. That is, prior to applying Softmax, some vector components could be negative, or greater than one; and might not sum to 1; but after applying Softmax, each component will be in the interval (0, 1) and the components will add up to 1, so that they can be interpreted as probabilities. Furthermore, the larger input components will correspond to larger probabilities. Softmax maps the non-normalized output of the network to a probability distribution over predicted output classes. The standard (unit) Softmax function is defined by the formula:

$$\sigma(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \text{ for } i = 1, \ldots, K \text{ and } \mathbf{z} = (z_1, \ldots, z_K)$$

2.7 Performance Metrics

One criteria was used for the performances of the CNN deep learning models:

Accuracy = (TN + TP) / (TN + TP + FN + FP)

TP, FP, TN and FN given in the previous Equation represent the number of True Positive, False Positive, True Negative and False Negative, respectively.

2.8 Conclusion

In this Chapter, we introduced the concept of deep learning, neural networks and CNNs. We also highlighted the layers we used in our implementation which are the most common layers in CNNs. And Finally we exposed the performance metrics we're going to be using to evaluate our CNN models.

CHAPTER 3: CLASSIFICATION OF MALWARES USING CONVOLUTIONAL NEURAL NETWORKS

3.1 Introduction

After going through the basics on CNNs and Malware analysis and detection, we now move on to our application where we plan on comparing the results of 3 different implementations of CNNs: the first is a basic CNN taken from a Towards Data Science article, the second was taken from a paper by Daniel Gibert, Carles Mateu, Jordi Planes, Ramon Vicens entitled: "Using convolutional neural networks for classification of malware represented as images ", the third is from a paper by Mahmoud Kalash, Mrigank Rochan, Noman Mohammed, Neil D. B. Bruce, Yang Wang, Farkhund Iqbal entitled: "Malware Classification with Deep Convolutional Neural Networks".
It is worth noting that all 3 CNNs were built for the MalIMG dataset and not on MaleVIS, meaning that part of our contribution in this project was to test the effectiveness of these CNNs on the MaleVIS dataset as well as compare its results on the MalIMG dataset.

3.2MalIMG and MaleVIS Datasets:

3.2.1 MalIMG Dataset:

This dataset has a total of 9,339 malware samples that are represented as grayscale images. Each malware sample in the dataset belongs to one of the 25 malware families/classes. Also, the number of samples belonging to a malware family vary across the dataset. In our experiments, we randomly select 90% of malware samples in a family for training and the remaining 10% for testing. At the end, we have 8,394 malware samples for training and 945 samples for testing.

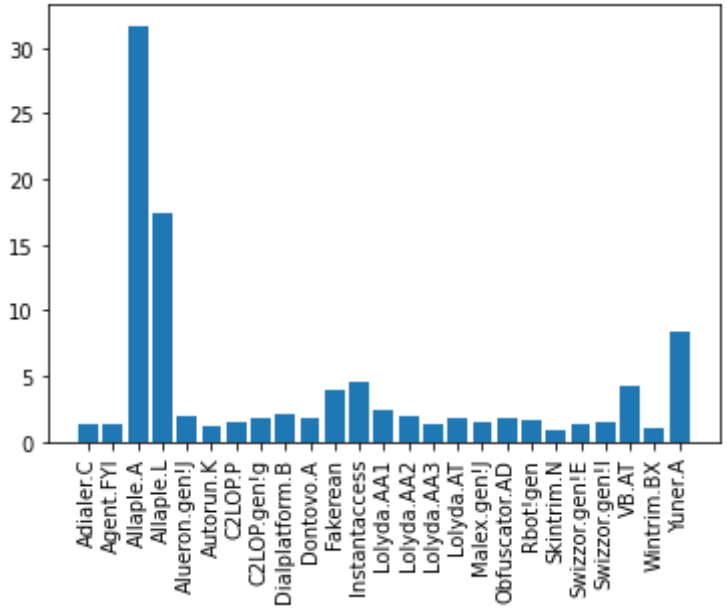

Fig 3.1. Percentage of Distribution of each Malware Family in the MalIMG dataset.

#	Class	Family	#
1.	Worm	Allaple.L	1591
2.	Worm	Allaple.A	2949
3.	Worm	Yuner.A	800
4.	PWS	Lolyda.AA 1	213
5.	PWS	Lolyda.AA 2	184
6.	PWS	Lolyda.AA 3	123
7.	Trojan	C2Lop.P	146
8.	Trojan	C2Lop.gen!g	200
9.	Dialer	Instantaccess	431
10.	TDownloader	Swizzot.gen!I	132
11.	TDownloader	Swizzor.gen!E	128
12.	Worm	VB.AT	408
13.	Rogue	Fakerean	381
14.	Trojan	Alueron.gen!J	198
15.	Trojan	Malex.gen!J	136
16.	PWS	Lolyda.AT	159
17.	Dialer	Adialer.C	125
18.	TDownloader	Wintrim.BX	97
19.	Dialer	Dialplatform.B	177
20.	TDownloader	Dontovo.A	162
21.	TDownloader	Obfuscator.AD	142
22.	Backdoor	Agent.FYI	116
23.	Worm:AutoIT	Autorun.K	106
24.	Backdoor	Rbot!gen	158
25.	Trojan	Skintrim.N	80

Table 3.1. Table showing the type, name and number samples of each malware family in the MalIMG Dataset.

3.2.2 MaleVIS Dataset

This dataset has a total of 14,226 evenly distributed malware samples that are represented as RGB images. Each malware sample in the dataset belongs to one of the 25 malware families/classes. In our experiments, we randomly select 90% of malware samples in a family for training and the remaining 10% for testing. At the end, we have 12,803 malware samples for training and 1423 samples for testing.

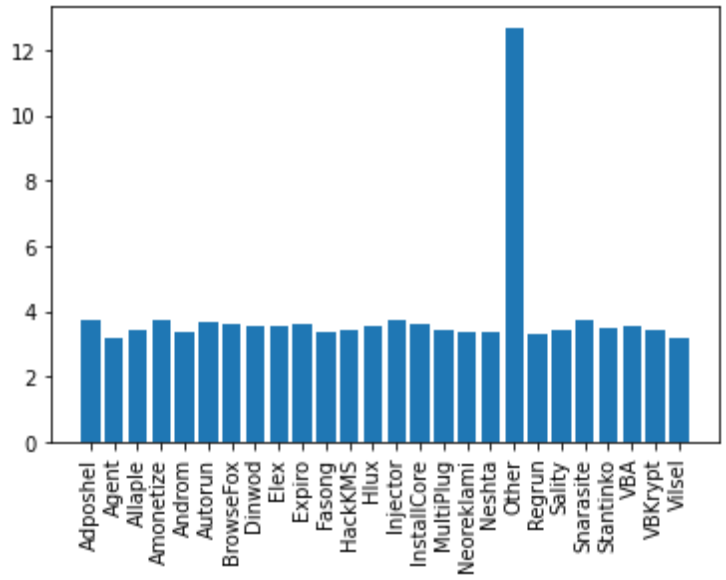

Fig 3.2. Percentage distribution of each malware family in the MaleVIS Dataset

As we can see from the figure above, the MaleVIS dataset also contains a 26th class called “other” which contains non malicious software.

1	Win32/Adposhel	Adware
2	Win32/Agent-fyi	Trojan
3	Win32/Allaple.A	Worm
4	Win32/Amonetize	Adware
5	Win32/Androm	Backdoor
6	Win32/AutoRun-PU	Worm
7	Win32/BrowseFox	Adware
8	Win32/Dinwod!rfn	Trojan
9	Win32/Elex	Trojan
10	Win32/Expiro-H	Virus
11	Win32/Fasong	Worm
12	Win32/HackKMS.A	Trojan
13	Win32/Hlux!IK	Worm
14	Win32/Injector	Trojan
15	Win32/InstallCore.C	Adware
16	Win32/MultiPlug	Adware
17	Win32/Neoreklami	Adware
18	Win32/Neshta	Virus
19	Win32/Regrun.A	Trojan
20	Win32/Sality	Virus
21	Win32/Snarasite.D!tr	Trojan
22	Win32/Stantinko	Backdoor
23	VBA/Hilium.A	Virus
24	Win32/VBKrypt	Trojan
25	Win32/Vilsel	Trojan

Table 3.2. Table showing the type and name of each malware family in the MaleVIS Dataset.

3.3 Overview of CNN Architectures

Now that we've taken a look at the Datasets we're going to use in our implementation, let's look at the architectures of the 3 CNNs we're going to be using to classify the images in these datasets.

3.3.1 Basic CNN from Towards Data Science:

Our first CNN was taken directly from a towards data science article online and will hereinafter be referred to as CNN-1. It contains 2 convolution layers, 2 maxpooling layers, 2 dropout layers, 2 dense layers and one final softmax layer for classification. The following is a summary of the layers

of the CNN when applied on the MalIMG dataset as well as the output from each layer with the number of trainable weights and biases(params).

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 222, 222, 30)      840
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 111, 111, 30)      0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 109, 109, 15)      4065
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 54, 54, 15)        0
_________________________________________________________________
dropout (Dropout)            (None, 54, 54, 15)        0
_________________________________________________________________
flatten (Flatten)            (None, 43740)             0
_________________________________________________________________
dense (Dense)                (None, 128)               5598848
_________________________________________________________________
dropout_1 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_1 (Dense)              (None, 50)                6450
_________________________________________________________________
dense_2 (Dense)              (None, 25)                1275
=================================================================
Total params: 5,611,478
Trainable params: 5,611,478
Non-trainable params: 0
_________________________________________________________________
```

3.3.2 CNN from "Using convolutional neural networks for classification of malware represented as images"

Our second CNN was taken directly from a paper "Using convolutional neural networks for classification of malware represented as images" and will hereinafter be referred to as CNN-2. It contains 3 convolution layers, 3 maxpooling layers, 3 normalization layers, 1 dense layers and one final softmax layer for classification. The following is a summary of the layers of the CNN when applied on the MalIMG dataset as well as the output from each layer with the number of trainable weights and biases(params).

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_2 (Conv2D)            (None, 220, 220, 50)      3800
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 110, 110, 50)      0
_________________________________________________________________
layer_normalization (LayerNo (None, 110, 110, 50)      220
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 108, 108, 70)      31570
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 54, 54, 70)        0
_________________________________________________________________
layer_normalization_1 (Layer (None, 54, 54, 70)        108
_________________________________________________________________
```

```
conv2d_4 (Conv2D)              (None, 52, 52, 70)        44170
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 26, 26, 70)        0
_________________________________________________________________
layer_normalization_2 (Layer (None, 26, 26, 70)        52
_________________________________________________________________
flatten_1 (Flatten)            (None, 47320)             0
_________________________________________________________________
dense_3 (Dense)                (None, 256)               12114176
_________________________________________________________________
dense_4 (Dense)                (None, 25)                6425
=================================================================
Total params: 12,200,521
Trainable params: 12,200,521
Non-trainable params: 0
_________________________________________________________________
```

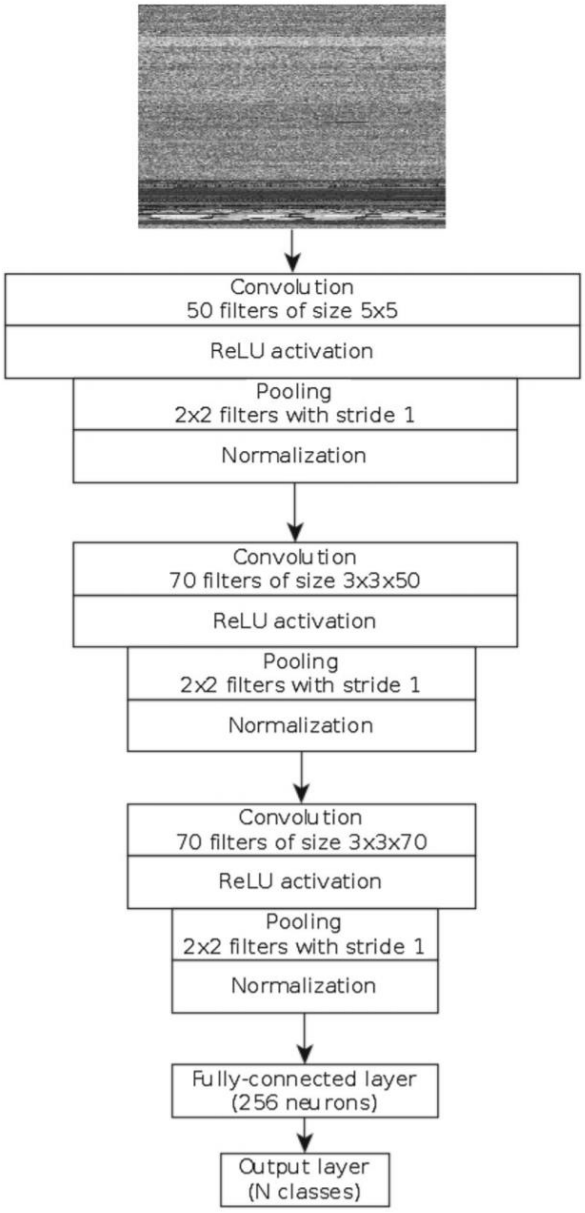

Fig. 3.3. Convolutional neural network for classification of malware represented as gray-scale images. It is composed by 3 convolutional layers followed by one fully-connected layer. The input of the network is a malicious program represented as a gray-scale image. The output of the network is the predicted class of the malware sample.

3.3.3 CNN from "Malware Classification with Deep Convolutional Neural Networks"

Our third CNN was taken directly from a paper "Malware Classification with Deep Convolutional Neural Networks" and will hereinafter be referred to as CNN-3. It contains 5 convolution layers, 5 maxpooling layers, 2 dense layers and one final softmax layer for classification. The following is a summary of the layers of the CNN when applied on the MalIMG dataset as well as the output from each layer with the number of trainable weights and biases(params).

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_5 (Conv2D)            (None, 220, 220, 64)      4864
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 110, 110, 64)      0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 108, 108, 128)     73856
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 54, 54, 128)       0
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 52, 52, 256)       295168
_________________________________________________________________
max_pooling2d_7 (MaxPooling2 (None, 26, 26, 256)       0
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 24, 24, 512)       1180160
_________________________________________________________________
max_pooling2d_8 (MaxPooling2 (None, 12, 12, 512)       0
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 10, 10, 512)       2359808
_________________________________________________________________
max_pooling2d_9 (MaxPooling2 (None, 5, 5, 512)         0
_________________________________________________________________
flatten_2 (Flatten)          (None, 12800)             0
_________________________________________________________________
dense_5 (Dense)              (None, 4096)              52432896
_________________________________________________________________
dense_6 (Dense)              (None, 4096)              16781312
_________________________________________________________________
dense_7 (Dense)              (None, 25)                102425
=================================================================
Total params: 73,230,489
Trainable params: 73,230,489
Non-trainable params: 0
```

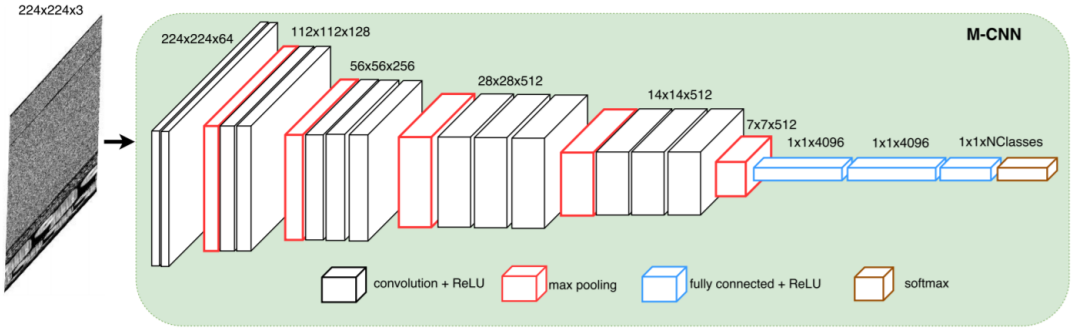

Fig. 3.4. Overview of CNN-3 (M-CNN). The network takes a malware image as input and it produces a set of scores of size equal to the number of malware classes (NClasses) in the dataset as the output.

3.4 Implementation

For our implementation, we chose to work on the google collab notebooks platforms as it provides us with great amounts of virtual RAM, Storage and a GPU for acceleration. Working on google collab notebooks is the same as working with the very famous Jupyter Notebooks (also known as IPython notebooks).

We also chose to work with the Tensorflow Keras API as it provided an easy learning curve and has a wide range of documentation and support by google. **Keras** is an open-source software library that provides a Python interface for artificial neural networks. Keras acts as an interface for the TensorFlow library. **TensorFlow** is a free and open-source software library for machine learning. It can be used across a range of tasks but has a particular focus on training and inference of deep neural networks.

We first started off by uploading both the MalIMG and MaleVIS dataset zip files onto our google drive which allows us to connect our notebook to our drive and extract the data from the zipfiles into our working directory. Following that, we used the keras preprocessing library to automatically extract the images and labels. Then, using the sklearn package we split each dataset into a train set and a test set. The training set is what is used to train the neural network and adjust the weights and biases accordingly, while the testing set is used to validate the accuracy of our CNN. And finally after training and testing each CNN we used the Matplotlib and Seaborn packages for python to plot the results and the confusion matrix respectively.

3.5 Results

3.5.1 Validation Accuracy

The following is a table showing the average results for the validation accuracies of each CNN on each Dataset:

Dataset	CNN-1	CNN-2	CNN-3
MalIMG	97.79 %	98.00 %	98.60 %
MaleVIS	86.59 %	86.80 %	87.20 %

Table 3.3. Average results for the Validation Accuracy on the MalIMG and MaleVIS Datasets

The following Figures show the results of the training and validation accuracy evolution per epoch for the last training iteration:

MalIMG:

(a) (b) (c)

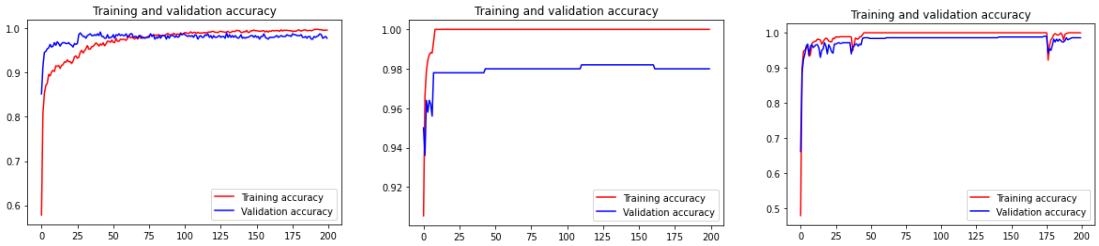

Fig 3.5. Graphs showing the evolution of the Training accuracy(red) and Validation accuracy(blue) over the number of epochs on the MalIMG dataset

(a): Results from CNN-1
(b): Results from CNN-2
(c): Results from CNN-3

MaleVIS:

(a) (b) (c)

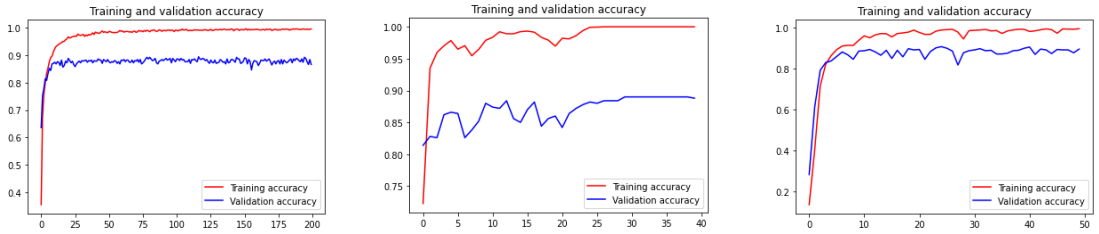

Fig 3.6. Graphs showing the evolution of the Training accuracy(red) and Validation accuracy(blue) over the number of epochs on the MaleVIS dataset

(a): Results from CNN-1
(b): Results from CNN-2
(c): Results from CNN-3

3.5.2 Confusion Matrix

Confusion matrix is a table with rows and columns that reports the number of false positives, false negatives, true positives, and true negatives. This allows more detailed analysis than simple proportion of correct classifications (accuracy). Our algorithm releases the following confusion matrices that is analyzed as following:

- The rows correspond to the predicted class (Output Class) and the columns correspond to the true class (Target Class). The diagonal cells correspond to observations that are correctly classified.
- The off-diagonal cells correspond to incorrectly classified observations.
- The column on the far right of the plot shows the percentages of all the examples predicted to belong to each class that are correctly and incorrectly classified. These metrics are often called the precision (or positive predictive value) and false discovery rate, respectively.

– The row at the bottom of the plot shows the percentages of all the examples belonging to each class that are correctly and incorrectly classified. These metrics are often called the recall (or true positive rate) and false negative rate, respectively.

MalIMG:

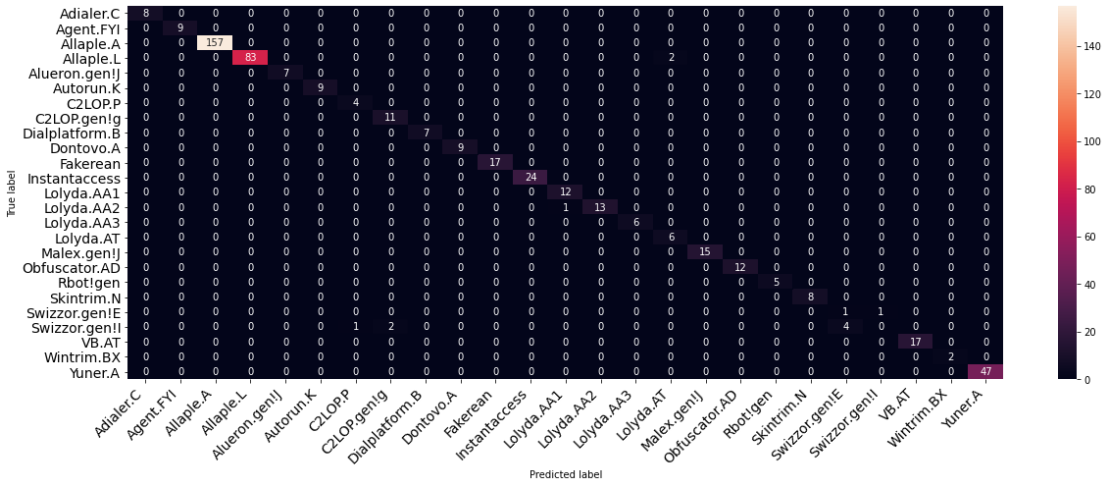

Fig 3.7. Confusion matrix of CNN-1 on MalIMG Dataset

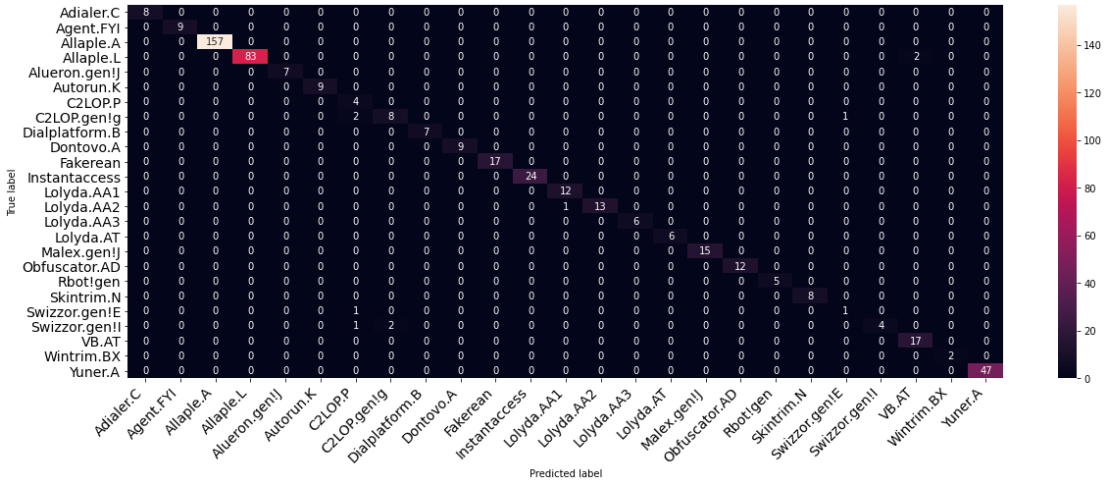

Fig 3.8. Confusion matrix of CNN-2 on MalIMG Dataset

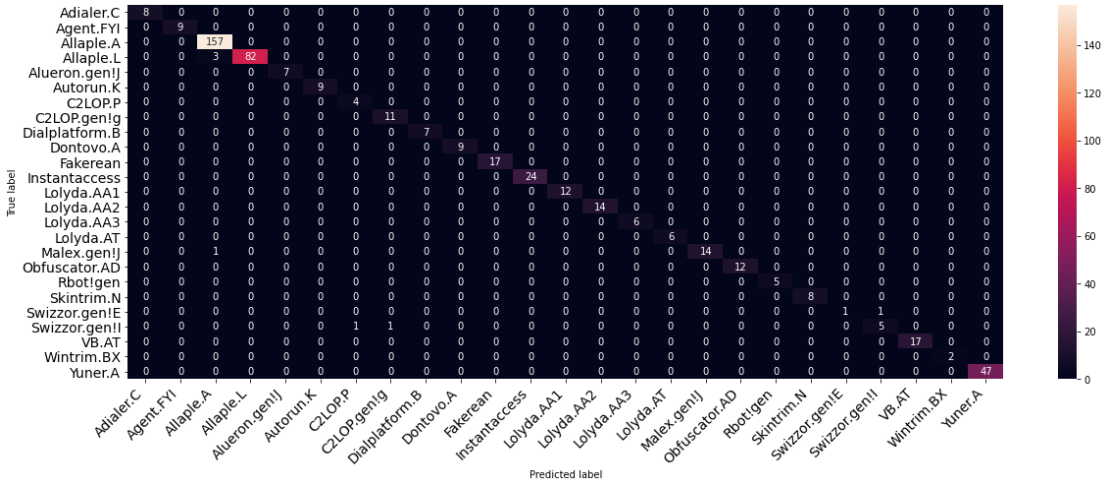

Fig 3.9. Confusion matrix of CNN-3 on MalIMG Dataset

MaleVIS:

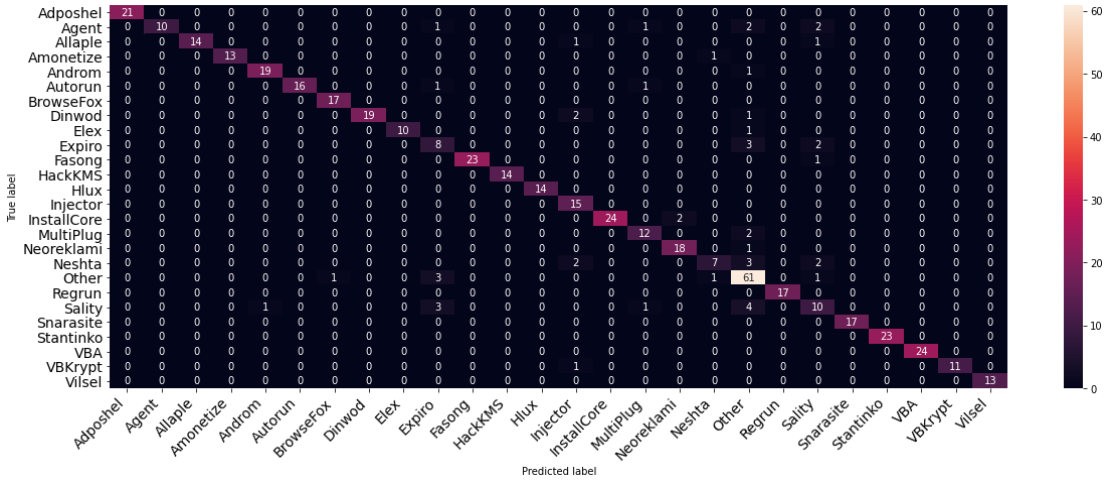

Fig 3.10. Confusion matrix of CNN-1 on MaleVIS Dataset

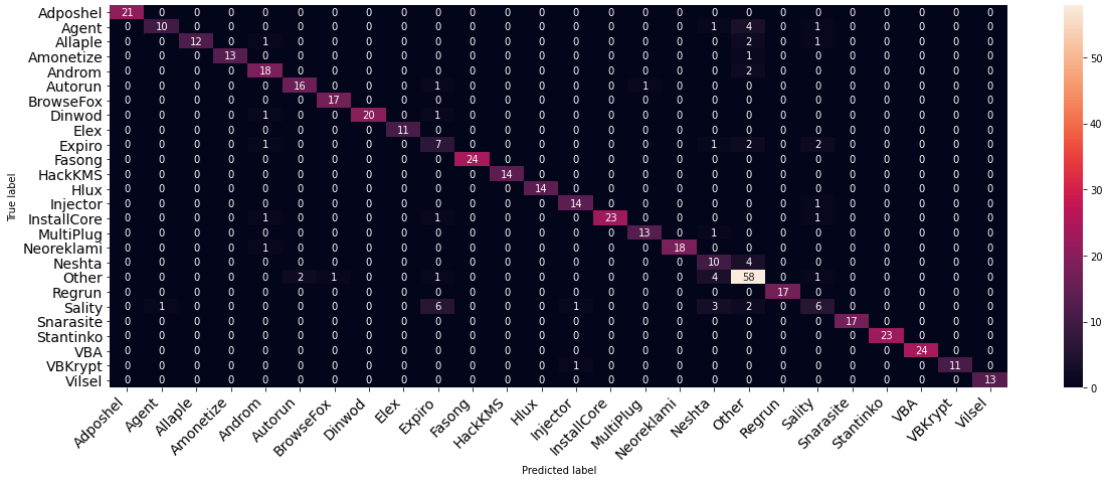

Fig 3.11. Confusion matrix of CNN-2 on MaleVIS Dataset

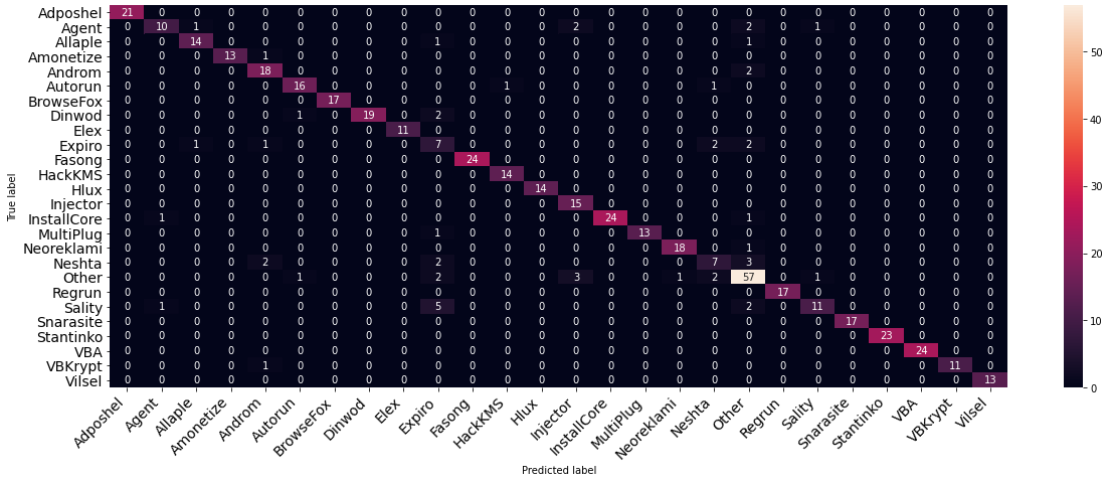

Fig 3.12. Confusion matrix of CNN-3 on MaleVIS Dataset

3.5.6 Discussion of Results:

As we can see from the accuracy table, the results for the MalIMG dataset are very close yet it is also apparent that the CNN-3 architecture performs best on this dataset. It is also apparent that the same holds true for the MaleVIS dataset which is extremely interesting considering the differences between the MalIMG and MaleVIS datasets notably the fact that MaleVIS images are RGB. Yet we can very clearly notice that even though the results for the MaleVIS dataset are close, the accuracy is very far from what it is in the MalIMG dataset, this can maybe be attributed to the "Other" class in the MaleVIS dataset as we can see from the Confusion matrices that it is responsible for the most false positives.

3.6 Conclusion

In this chapter, we discussed our implementation of the 3 different CNN architectures and compared the results on each dataset. We noticed that all 3 CNNs perform very similarly on the MalIMG dataset with CNN-3 having the highest final average accuracy score. We also noticed that even though the results were relatively close for the MaleVIS dataset, they were very far in accuracy from the MalIMG accuracy numbers which is most probably due to the "other" class in the MaleVIS dataset.

CONCLUSION

In this project, we introduced the idea of malwares and malware analysis in both traditional techniques and more modern machine learning techniques. Then we introduced the concept of Deep learning and more specifically Convolutional Neural Networks which are deep learning networks used mostly for image classification. Finally, we discussed our implementation of 3 different CNN architectures on the 2 different datasets and analyzed the results.

In our opinion the topic of utilizing the very rapidly growing domain of AI in cyber security is an extremely interesting and attractive idea which is what encouraged us to choose the project in the first place. Further developments for this project could include the implementation of a new CNN architecture specifically for the MaleVIS dataset and trying to increase the accuracy. One other idea could be to try and remove the "other" class from the dataset and perform the same classification using the same CNNs.

REFERENCES

- M. Kalash, M. Rochan, N. Mohammed, N. D. B. Bruce, Y. Wang and F. Iqbal, "Malware Classification with Deep Convolutional Neural Networks," *2018 9th IFIP International Conference on New Technologies, Mobility and Security (NTMS)*, 2018, pp. 1-5, doi: 10.1109/NTMS.2018.8328749.
- Gibert, Daniel & Mateu, Carles & Planes, Jordi & Vicens, Ramon. (2019). Using convolutional neural networks for classification of malware represented as images. Journal of Computer Virology and Hacking Techniques. 15. 10.1007/s11416-018-0323-0.
- Bozkir, A. S., Cankaya, A. O., & Aydos, M. (2019). Utilization and Comparision of Convolutional Neural Networks in Malware Recognition. *2019 27th Signal Processing and Communications Applications Conference (SIU)*. https://doi.org/10.1109/siu.2019.8806511
- hugom1997. (n.d.). *Malware_Classification/Malware_Classification.ipynb at master · hugom1997/Malware_Classification*. GitHub. https://github.com/hugom1997/Malware_Classification/blob/master/Malware_Classification.ipynb.
- LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature*, *521*(7553), 436–444. https://doi.org/10.1038/nature14539
- Mallet, H. (2020, May 28). *Malware Classification using Convolutional Neural Networks-Step by Step Tutorial*. Medium. https://towardsdatascience.com/malware-classification-using-convolutional-neural-networks-step-by-step-tutorial-a3e8d97122f.
- Nataraj, L., Karthikeyan, S., Jacob, G., & Manjunath, B. S. (2011). Malware images. *Proceedings of the 8th International Symposium on Visualization for Cyber Security - VizSec '11*. https://doi.org/10.1145/2016904.2016908