Using Transfer Learning for Malware Detection

A Thesis Presented to the Faculty of Natural

and Applied Sciences at Notre Dame

University-Louaize

In Partial Fulfillment of the

Requirements for the Degree

Master of Science

by VERONICA RAMMOUZ

22 DECEMBER 2021

Notre Dame University - Louaize

Faculty of Natural and Applied Sciences

Department of Computer Science

We hereby approve the thesis of

Veronica Rammouz

Candidate for the degree of Master of Science in Computer Science

Dr. Hikmat Farhat

HIKMAT FARHAT

Supervisor, Chair

Dr. Khalil Challita

Khalil Challita

Committee Member

Acknowledgements

I want to thank everyone who stood by me through the most difficult times and the times I needed the support. Through the unprecedented times, they are the reason for all the beautiful things I do, and the reason this beautiful piece of my knowledge was created.

Firstly, I want to thank Dr Hikmat Farhat, for his continuous support through my journey. I want to give credit to the faculty members that I truly admire: Dr Hoda Maalouf, for her generous personality and dedication. I want to extend my appreciation towards Dr Khalil Challita, for keeping me curious about fascinating topics and believing in great things, as well as Dr Hicham Hage, for his inspiring character. I will never forget all the beautiful things I have learned from the amazing Dr's I am lucky to have crossed paths with at Notre Dame University. M oreover, I certainly will not forget the individuals, whom I proudly call my family and friends, who stood by me and supported me at my lowest. I want to thank Lily, Samir, Margarita, Chris, Floki, Hiba, Nanor, Mario, Christina, Joelle, Elias and Jad, among others. Lastly, I want to express my gratitude for each day, for every chance and for the blessings I have.

Contents

1 Introduction 1
1.1 Overview 1
1.2 Problem Description 2
1.3 Objectives 3
1.4 Limitations 3
1.5 Outline 4
2 Theoretical Background 5
2.1 Malware 5
2.2 Machine Learning 5
2.2.1 Classification 6
Static Methods 6
Dynamic Analysis 7
2.3 Deep Learning 8
2.3.1 Convolution Neural Networks 9
2.3.2 Transfer Learning 13
3 Related Work 15
4 Proposed Method 18
4.1 Dataset 19
4.2 Strategy 21
4.2.1 Classification through Base Model 22
4.2.2 Classification through Transfer Learning 23
4.3 Experiment Technology Ecosystem 24
5 Results 24

5.1 True vs. Predicted Class 25
5.2 Training Accuracy per Epoch and Model 27
5.3 Fine-Tuning Accuracy per Epoch and Model 28
6 Discussion and Analysis 29
6.1 True vs. Predicted class 29
6.2 Training Accuracy per Epoch and Model 30
6.3 Fine-Tuning Accuracy per Epoch and Model 31
6.4 Method Evaluation 33
7 Closing remarks 34
7.1 Conclusion 34
7.2 Future work 34

Appendix **i**

A Gabor Filtering **i**

References **iii**

List of Figures

1 A CNN sequence to classify handwritten digits 10

2 High-Level Diagram of the MalConv Architecture 16

3 Inception V3 Confusion Matrix 25

4 ResNet 50 Confusion Matrix 26

5 Training Accuracy Plot 27

6 Fine-Tuning Accuracy Plot 28

7 Feature Representation Detected by Gabor filters i

List of Tables

1 Microsoft Malware Classification Dataset (BIG 2015) Samples 20

Abstract

The internet has made room for lots of unwanted activity to propagate through computers. In response, many methods were established to detect a certain computer executable as malicious. However, there were still loopholes for hackers within traditional systems. Some methods use machine learning others use deep learning. There are some drawbacks to each method, such as reverse analysis and restricted simulation on different execution paths, as well as long execution time. Some methods cannot generalize well and cannot scale to large amounts of data. Moreover, anti-viruses, using signature-based classification, have proven to be insufficient in certain instances, as certain malware has been developed in a way to include a signature beyond the available malware datasets. For this reason, deep learning techniques with different architectures were introduced to select features automatically, identify and classify malware programs. Specifically, using transfer learning to classify malware binaries has proven to be an improvement on the current deep learning methods which take days to execute. Transfer learning speeds up the process by using much less epochs in fitting the models.

1 Introduction

1.1 Overview

We have reached an age where the internet has become an essential part of living. It is the driving force of all modern workflow, communication, organization and so on. Starting from the concept of existence of antimatter as opposed to matter, it is in the basic forms of creation that one recognizes that for every bit of evolution/innovation, there is some form of downside, or opposition to what is there. Stemming from this ideology, one pays attention to the fact that the internet has allowed malicious intent to propagate by causing unwanted activity to be executed through the said helpful means. For this reason, action is needed to be taken against malicious activity through cybersecurity systems with the intent to prevent, detect and/or combat malicious software; therefore, the need to evaluate the current established methods for malware detection has increased, for it is necessary to ensure the most efficient and optimal strategies are to be taken place to promote a more secure environment for our daily usage. Anti-viruses have proven to be insufficient with their signature-based means. Additionally, when it

comes to Artificial Intelligence methods, there were still limitations when it comes to the accuracy and/or insufficient data to train on, with the rapid increase of malware variants. Moreover, sometimes it is required for a rapid incident response to take place, if a system is under attack. Therefore, it is necessary for a deep learning system to run in a fast manner; however, this isn't the case in some deep learning methods, where the execution time takes days. This is a problem from different angles that needs to be addressed to ensure security.

1.2 Problem Description

Anti-viruses aren't enough nowadays, for malware has been adapted to evade classical detection techniques that use file signature matching. Another problem is that malware variants have increased using a variety of methods such as compressing and encrypting file data [4]. To counter these workarounds, several machine learning were proposed. For most of the deep learning methods proposed in the literature, training takes a long time. The purpose of this thesis is to improve on the training time of such methods.

1.3 Objectives

The goal of this thesis is to use transfer learning to speedup the learning phase in the case of malware detection/classification.. Finding the right dataset is essential, since the method includes using a CNN architecture. For this reason, a dataset including data that is transformable to images is necessary. To be specific, uniformly transformable data is essential, meaning that binary files need to be uniformly considered if they are chosen. Malware data of different formats cannot be used for comparison in that case. Images are then to be placed as input to existing CNN models. By using publicly available pre-trained models from Keras to account for limited datasets and using different seeds for training, it is ensured that high accuracy levels are obtained, and fast training takes place.

1.4 Limitations

One of the limitations is finding enough datasets for malware detection. Since datasets exist in very different structures, the method proposed in this paper was tailored to fit the Microsoft Malware Classification Challenge dataset (BIG 2015), which includes binary and assembly files. Dataset size is another limitation, for

malware variants are ever-increasing and accounting for all of them is a never ending process.

1.5 Outline

First, a theoretical background will be mentioned, including overviews and definitions per the keywords mentioned in the text, such as malware, machine learning and deep learning techniques. Next, some related work will be presented based on previous findings and implementations, right before a discussion on the proposed method will be mentioned. A description of the dataset used, as well as the implementation strategy for the experimental simulations and the architectures used. Then, the stage is set for the results of the proposed method, including overall performance of each model used. Finally, the paper is concluded with analysis and conclusions of the study undertaken.

2 Theoretical Background

2.1 Malware

Malware (malicious software) is any computer program that compromises any of the levels of architecture of a computer (physical level, data transfer layer, network layer by hijacking packets for instance, or application layer etc.) Different types of malware include Trojans, Spyware, Adware, Rootkits, Ransomware, Worms, Key loggers, Grayware, fileless malware, Adware, Malvertising, Botnet, Backdoor, and so on. The method of malware propagation differs, like the extracted malware data formats. Malware can be extracted as binary files, assembly code, and Portable Executables.

2.2 Machine Learning

Machine learning (ML) is a branch of Artificial Intelligence (AI) that enables computers to learn from patterns in data, extracting useful knowledge that would predict a specific outcome from learned data – a white box model; a pre-programmed one. ML has facilitated the prediction process by learning from

gathered data and improving itself. However, in some cases, the machine learning process is rather ineffective in the face of huge datasets and time consumption, as well as it includes its weaknesses in specific cases.

2.2.1 Classification

Classification is a supervised ML method, and there are various algorithms to achieve this use case, such as Logistic Regression, Naive Bayes, K-Nearest Neighbors (KNN), Decision Trees, and Support Vector Machines (SVM). Some ML methods introduced by various researchers, as traditional methods, are characterized as static and dynamic methods.

Static Methods

Static methods have relied on feature extraction from disassembly code, bytecode, file structure, file signatures etc.

Compared to dynamic analysis, static analysis has the advantage of efficiency and fast performance since there is no execution to be taken place. Relative work was that of Anderson and Roth in 2018, who have created the labeled benchmark EMBER dataset used for the models for malware classification, specifically static analysis of Windows portable executable (PE) files [1]. The dataset was a product of extraction of 1.1 million malware binary PE files, which included 81% training

samples and 19% test samples. Training samples included 30% benign labels, 30% malignant and 30% unlabeled data whereas test data included 50% benign by 50% malignant only. Their work has allowed for the experimenting of static analysis by many others whose work will be listed.

All in all, static analysis refers to the unpacking and/or decryption of the executable in question, then checking whether the code contains malicious patterns in control flow. However, this entails problems like code obfuscation in many instances.

Dynamic Analysis

Dynamic analysis consists of running executable code in what is called a ”sandbox environment.” The dynamic method is different than the static method in the sense that it saves the time of unpacking and decryption by not requiring to do so. Thus, time is saved but not at the expense of scalability, where the issue of intensive resource consumption is in question. Additionally, this type of analysis has its problems as well; In some cases, it cannot trigger the malicious behavior to surface, for it needs to set the right conditions for it, which isn’t always feasible.[8]

2.3 Deep Learning

Deep learning is a complex structure of algorithms inspired by the human brain, as the structure is formed of Neural Networks. Deep learning allows for the processing of unstructured data, such as image, text, and so on. It is a subset of machine learning. This method has solved lots of problems, including malware classification. However, the performance of proposed deep learning methods for malware classification still consume a lot of time.

Artificial Neural Networks are defined as “a mathematical model that is based on biological neural networks and therefore is an emulation of a biological neural system”, as stated by Singh and Chauhan [9].

Deep learning uses the Neural Network architecture (NN) to learn and train on specific data to be able to predict specific outcomes; however, it’s rather a black box model. The NN architecture is inspired by the human brain, as it is formed of ”neurons”, namely fully connected layers which take data in a tailored format as input. The NN structure is comprised of an input layer which is connected to hidden layer(s) that include weights and biases to set probabilities for the data.

The hidden layer(s) are fully connected to output layers which depict the outcome of the prediction. The layers require a specific format for the data to be accepted uniformly and lead to the desired outcome. For example, if it is intended to classify dog images according to their breed, the input layers, which accept images of a pre-defined size, for example, require specific dimensions for the images for the learning procedure to take place from layer to layer, as well as classified in the output layer, which includes output nodes according to the number of dog breeds that exist. The output, for instance, may include "one-hot encoded" values, indicating a value of 1 next to the predicted breed and a 0 from the other nodes that indicate different breeds. In general, when using NNs data can be text, image, PDFs, audio and so on. As per the scope of this thesis, the intent is to use images as input, thus requiring the architecture of Convolutional Neural Networks (CNNs) which will be discussed in the following paragraph.

2.3.1 Convolution Neural Networks

Convolution Neural Networks (CNNs) are a type of deep learning network architecture. Note that the images are to be in uniform shape, namely normalized in

order to be accepted and provide an objective prediction as an outcome. The process of building the CNN architecture, as shown in Figure 1, requires several layers to be structured. The layers include an input layer taking an image of a set size, Max-Pooling layers, convolution layers and fully connected layers for the output, which consist of a finite set of nodes denoting the finite number of outcomes we can have for the prediction. The fully connected layer must also include an activation function, depending on the problem at hand.

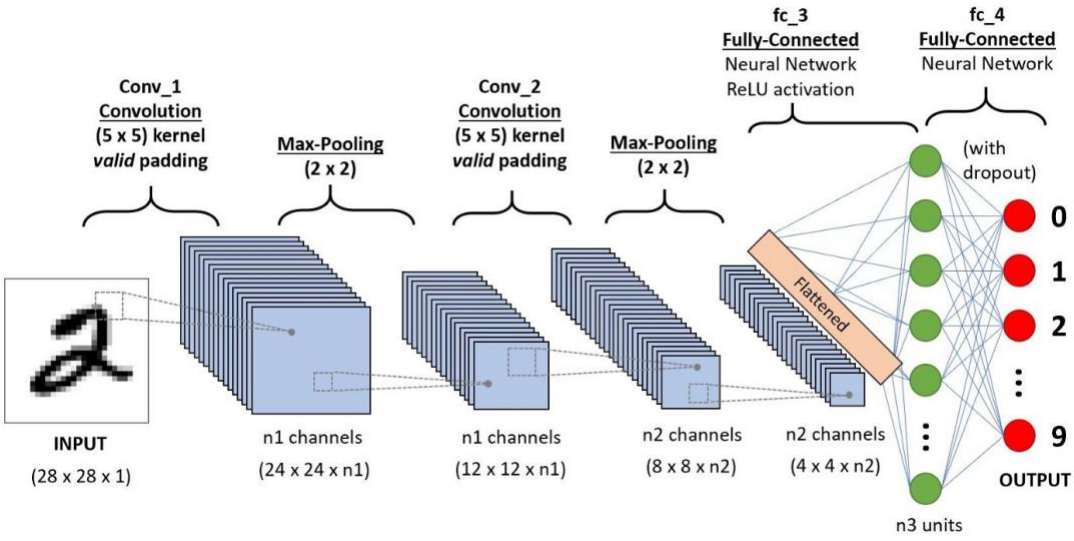

Figure 1: A CNN sequence to classify handwritten digits

Convolution Neural Networks use Kernel Convolution, which is used in a variety of image processing applications. Kernel convolution denotes a small matrix of

numbers, which is also called the kernel or filter that transforms the images based on the filter values and mapped as features according to the following formula

$$G[m,n] = (f * h)[m,n] = \sum_j \sum_k h[j,k] f[m - j, n - k]$$

where f is the input image, h is the kernel, and m and n being the indexes of the rows and columns of the matrix obtained, respectively. With each convolution, the image shrinks. Therefore, the number of passes done by the filter should be limited. Moreover, the loss of information should be accounted for, using the following equation to pad the image

$$p = (f - 1)/2$$

where f is the filter dimension and p is the padding. Depending on each image's size, it is decided whether the original image is kept or one with a border, to ensure all images are of equal sizes.

The Convolution Neural Network consists of a forward propagation, backpropagation and a gradient descent. Forward propagation is characterized by calculating an intermediate Z value obtained by the convolution with W filters and bias b. A non-linear activation function g is then applied.

$$Z_{[l]} = W_{[l]} * A_{[l-1]} + b_{[l]}$$

$$A_{[l]} = g_{[l]}(Z_{[l]})$$

Backpropagation then occurs within the neural network by calculating derivatives to ensure the parameters are updated during gradient descent. A simplified version of the notation is represented as follows.

$$\partial A^{[l]} = \frac{\partial L}{\partial A^{[l]}}$$

$$\partial Z^{[l]} = \frac{\partial L}{\partial Z^{[l]}}$$

$$\partial W^{[l]} = \frac{\partial L}{\partial W^{[l]}}$$

$$\partial b^{[l]} = \frac{\partial L}{\partial b^{[l]}}$$

dW and db are associated with the current layer. dA-1 is passed to the previous layer and dA is the input. Note that dW, db, W, b, dA and A are of the same dimension. By applying a derivative of the activation function to the input, we obtain the intermediate.

$$\partial Z_{[l]} = \partial A_{[l]} * g_0(Z_{[l]})$$

Full convolution then takes place. The matrix operation

$$\partial A+ = \sum_{m=0}^{n_h} \sum_{n=0}^{n_w} w * \partial Z[m,n]$$

is then used, after a 180 degree rotation of the kernel to handle the backpropagation, where dZ[m,n] is obtained from the previous layer.

Then, pooling layers are used to reduce tensor sizes including forward propagation and backpropagation to distibute the gradients.

2.3.2 Transfer Learning

With different pre-trained neural network architectures, there is room for editing the architecture according to the requirements. For instance, the last classification layer might need removal to add a layer that fits the number of outcomes that are related to the dataset. The neural network models are then considered to be "recyclable." For this reason, Keras has deployed a library of pre-trained models of very deep neural networks to be used publicly for a variety of AI applications. The models used, which are available through Keras, help with a variety of classification use cases. This thesis aims to improve on the current

methods of malware classification using Transfer Learning by using several pre-trained models. Transfer learning proves to be helpful for small dataset sizes as the models already contain some information to aid in the learning procedure.

In the proposed method, the following pre-trained models were used and evaluated; MobileNetV2, VGG16, ResNet101, ResNet152, ResNet50, InceptionResNetV2 and InceptionV3.

3 Related Work

Various methods for malware classification were proposed and established using static and dynamic features as stated in [11] and defined in 2.2.1. Despite their effective results while being classical techniques, more accurate methods have been achieved to accommodate for the large amount of malware variants as well as performance, as discussed further ahead.

One proposed architecture for malware classification using transfer learning, is the MalConv architecture used by Mohamad Al Kadri, Mohamed Nassar, and Haidar Safa [7]. MalConv has performed better than many other architectures including RNN. It was designed based on a few principles such as accounting for high generalization, large sequences, sparse features, and overfitting layers. The MalConv architecture is depicted in Figure 2.

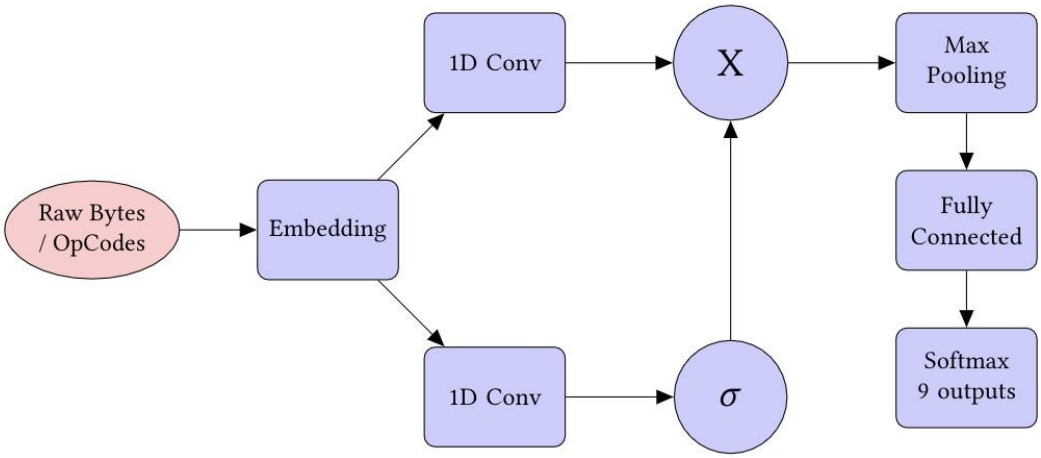

Figure 2: High-Level Diagram of the MalConv Architecture

As for some performed methods for image classification, several steps are taken for image texture analysis, such as using the (A)Gabor filtering or using the traditional methods, such as feature extraction, transformation and classification. The work in [5] suggests a Principal Component Analysis (PCA) for dimensionality reduction. It is a statistical method that converts a set of n possible correlated variables into uncorrelated values (m, where $n \leq m$) using orthogonal transformation. The use of GIST descriptors including Gabor filtering, as well as Haralick method for image classification and Local Binary Pattern are also techniques to be used for texture analysis.

As for other proposed methods for classification, the work in [4] suggests a

K-Fold Cross-Validation technique by dividing the dataset into K equal-sized folds. One of these samples is kept for validation and the whole process executes according to the number of folds. This method included similar limitations such as similarity of malware images and compressed files.

Some methods for image visualizations, resorted to visualizing malware binary executable files as grayscale images, by reading the files as sequences of 8-bit unsigned integers, reshaped into 2D matrices using the equation

$$A = a_0 * 2^0 + a_1 * 2^1 + a_2 * 2^2 + a_3 * 2^3 + a_4 * 2^4 + a_5 * 2^5 + a_6 * 2^6 + a_7 * 2^7$$

as in [3].

Other work [10] for malware classification included the usage of deep autoencoders (DAEs) for digital signatures and notations, as well as features, each including different frameworks. The signatures are extracted using feed-forward neural networks. It is trained for 800 epochs, at a learning rate ranging between 0.1 and 0.21. It achieves a near-100% accuracy. However 800 epochs is still a big number.

Additional work related to malware classification for Android [12], depicted a specific method by decompiling APK files and API feature vector construction using One-hot encoding for the classification. Deep Auto-Encoders were then used to reduce feature vector dimensions. A Logistic Regression binary classification model is then used as comparison, represented by the following function:

$$h(x) = g(w * x) = (1/1 + e^{-w*x})$$

where x is the feature vector of the input and w is the trainable parameter. Other models, including a finite Gaussian Distribution and CNN model were used as comparison to the proposed method. The results showed that proposed method performed best, rating a high F1 score of 0.93 and 0.643, making it a remarkable approach.

4 Proposed Method

The proposed method dictates the transformation of binary files into grayscale images to be fed as input to the NN. By transforming the files to images, one can immediately observe how similar those ”malware visualizations” belonging to the same family are; this can be explained by the fact that malicious code is perhaps edited, reused and plugged into target files.

4.1 Dataset

The selected dataset for malware classification used is Microsoft Malware Classification Dataset (BIG 2015) avaliable on Kaggle. This dataset is about half a terrabyte big. It includes raw binary malware files, representing 9 malware families, depicted in Table 1. Each malware sample has 2 associated files to it; an '.asm' file and a '.byte' file. The target files of the experiment conducted were those having a '.byte' extension. The binary files are chosen for visualization and normalization purposes. The identifiers of the binary files, uniformly, are strings composed of an ID, 20-character hash values, a class and an integer indicating one of the families that the file belongs to. Each binary file contains hexadecimal information about the file's content. Portable Executable (PE) headers are excluded for normalization purposes. In this dataset, metadata is provided through a manifest of function calls, strings and other code-level information. As stated from the source, a reverse engineering tool was used to collect this data (IDA disassembler).

Family	Samples	Type
Gatak	1013	Backdoor
Kellihos ver1	398	Backdoor
Kellihos ver3	2942	Backdoor
Lollipop	2478	Adware
Obfuscator	1228	Obfuscator Malware
Ramnit	1541	Worm
Simda	42	Backdoor
Tracur	751	Trojan
Vundo	475	Trojan

Table 1: Microsoft Malware Classification Dataset (BIG 2015) Samples

There are lots of datasets which are plausible for the same experiment, due to similar structures, such as IoTPOT [6] and Drebin [2] dataset.

4.2 Strategy

The strategy proposed is as follows. First, to get some meaning out of the binary files, they are labeled and structured according to each file's name - which includes an identifier in the file name's string - and assigned a category, one of the 9 malware families. Then, each binary file is parsed using its hexadecimal content, then resized to a uniform, fixed size of 256x256 grayscale image to fit the input layer of the NN architecture.

The architectures used were pre-trained on the ImageNet dataset and were modified for the sake of the experiment. The last classification layer of the models used was removed, and the whole other part was used connected to another set of layers, consisting of a fully connected layer having 1024 neurons and a classification layer of 9 nodes for each class of malware samples.

The image set is split into training and testing sets through Keras's functions. As a test before the transfer learning, the Yuan model [11]is built and trained, then used to predict the unclassified images in the testing dataset. The performance is then evaluated and the model is saved.

After this procedure transfer learning phase takes place using the different models from Keras (InceptionV3, VGG16, InceptionResNetV2, ResNet50, ResNet101, ResNet152, and MobilenetV2), which are loaded, frozen, and run for 15 epochs, with only the two added layers optimized. The same data is split into 90% training and 10% validation, loaded as RGB input to be normalized for the imported models and given a different seed - one of 10 different ones, to make sure the whole variety of classes are shuffled in experimentation. An activation function (ReLU) for the dense layer is then used, with a softmax activation for the prediction layer. The model is then trained, evaluated and unfrozen, respectively. It is then retrained to ensure the fine-tuning operation, lasting for 10 epochs with a batch size of 32 and learning rate of 10^{-5}.

All the results are recorded and will be brought to light further ahead and evaluated.

4.2.1 Classification through Base Model

The base model is structured as follows: it is a sequential model consisting of a 256x256 input layer with a grayscale color channel, two Conv2D layers of size 64, a MaxPooling2D layer, two Conv2D layers of size 128, another MaxPooling2D, then an

additional three sets of 3 layers of Conv2D of size 256 and 256 then 512 respectively, followed by a MaxPooling2D layer. A flattening layer is then introduced followed by a dense layer using the Relu activation then another dense layer using softmax activation for the output. The model is then optimized using the Adam optimizer - an adaptive gradient descent algorithm. A Sparse categorical Crossentropy computation is used for the loss since the output is categorical, and not one-hot encoded. The optimization is set to not compute from logits for the reason that it would then occur after the softmax computation. The model is then trained and evaluated.

4.2.2 Classification through Transfer Learning

For the transfer learning phase, images are prepared using the same method as that of the base model. From Tensorflow, different models are imported and the seed for each training and testing dataset is 1 of 10 different ones, for each model training to be simulated. Since the color channel of Keras models' input is RGB, we parse the grayscale images to be read in an RGB fashion. Each model is frozen, trained, Adam-optimized, unfrozen, fine tuned and evaluated.

4.3 Experiment Technology Ecosystem

The environment used to run the conducted experiments was through Kaggle's workspace, which includes an NVIDIA Tesla P100 GPU having 16GB of RAM. Python language was used, with the major library Tensorflow being used.

5 Results

When fitting the pre-trained models, and right after freezing, 15 epochs were used with a sparse categorical cross entropy loss. Then, after fine-tuning the model, the prediction score is recorded and the results are shown in confusion matrices below for the seed 254, per model. An interpretation is then stated along with discussions on the accuracy recorded per epoch of fine-tuning.

5.1 True vs. Predicted Class

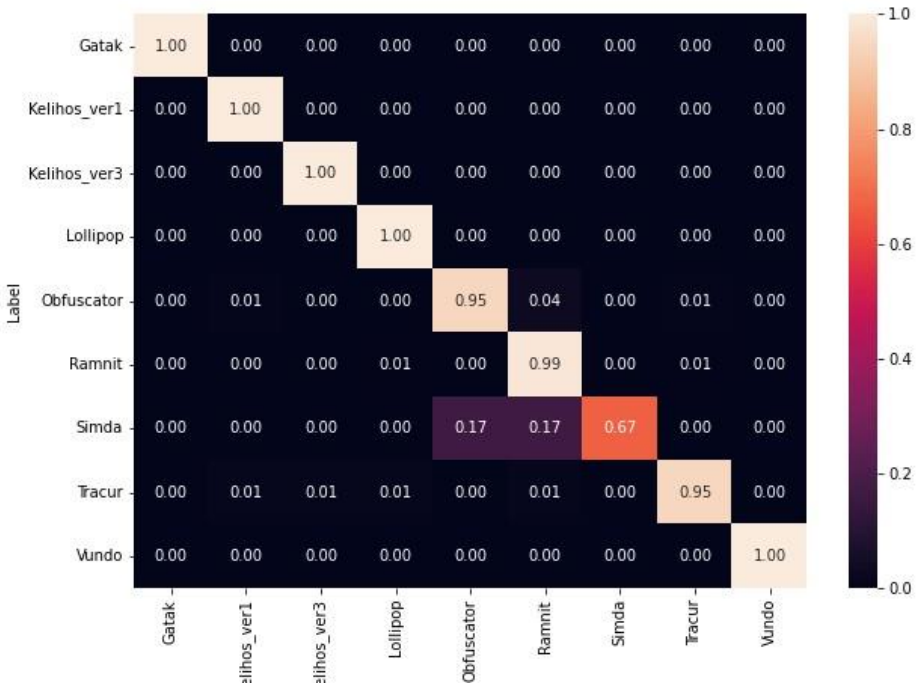

Figure 3: Inception V3 Confusion Matrix

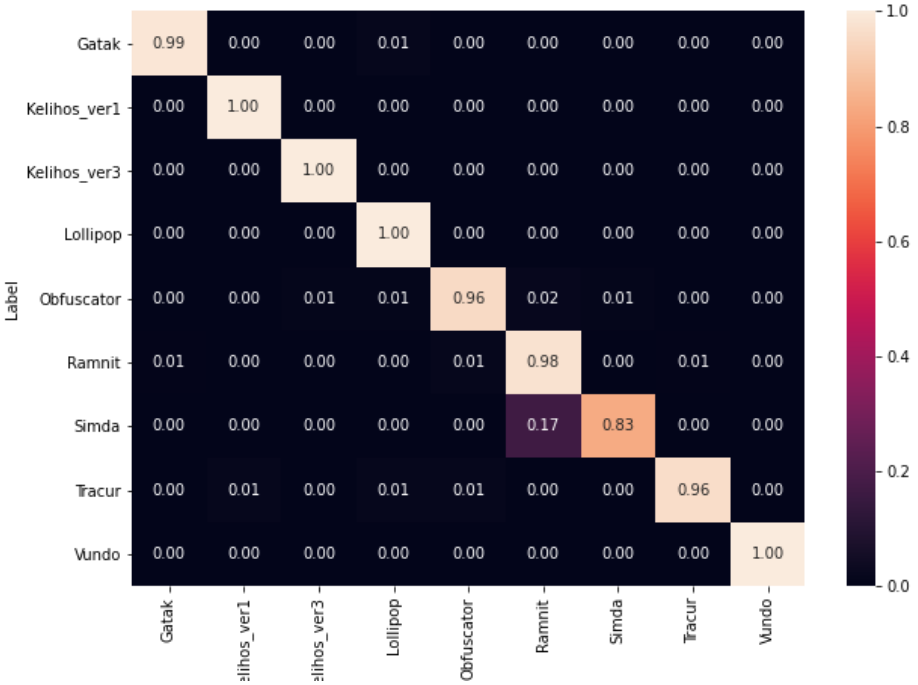

Figure 4: ResNet 50 Confusion Matrix

5.2 Training Accuracy per Epoch and Model

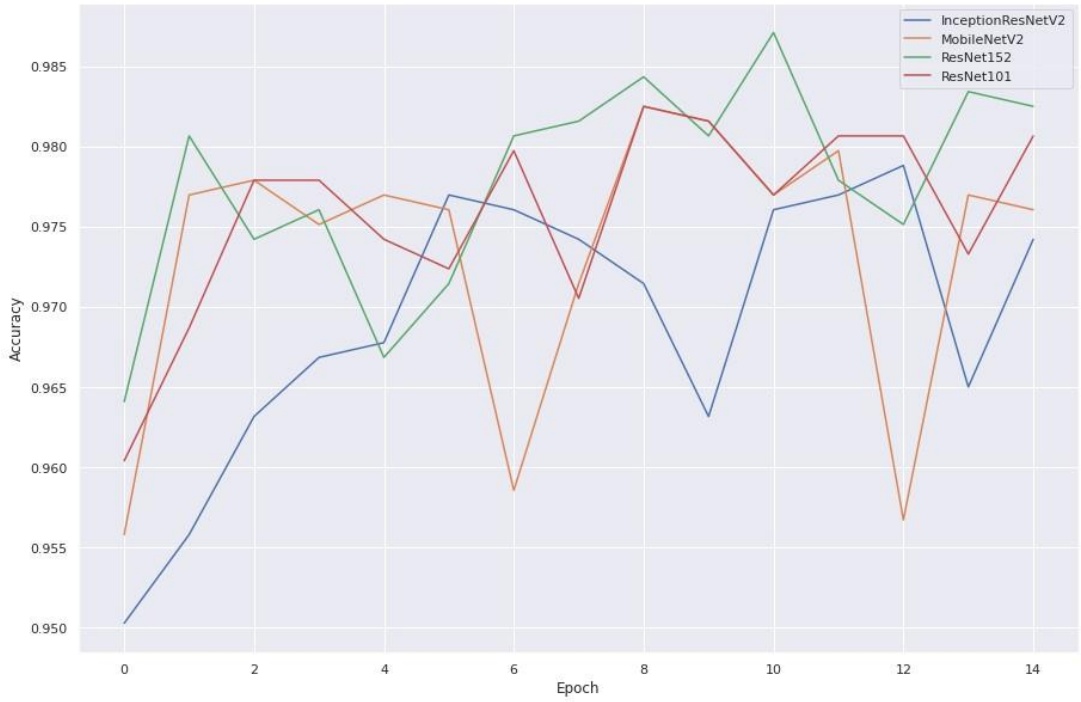

Figure 5: Training Accuracy Plot

5.3 Fine-Tuning Accuracy per Epoch and Model

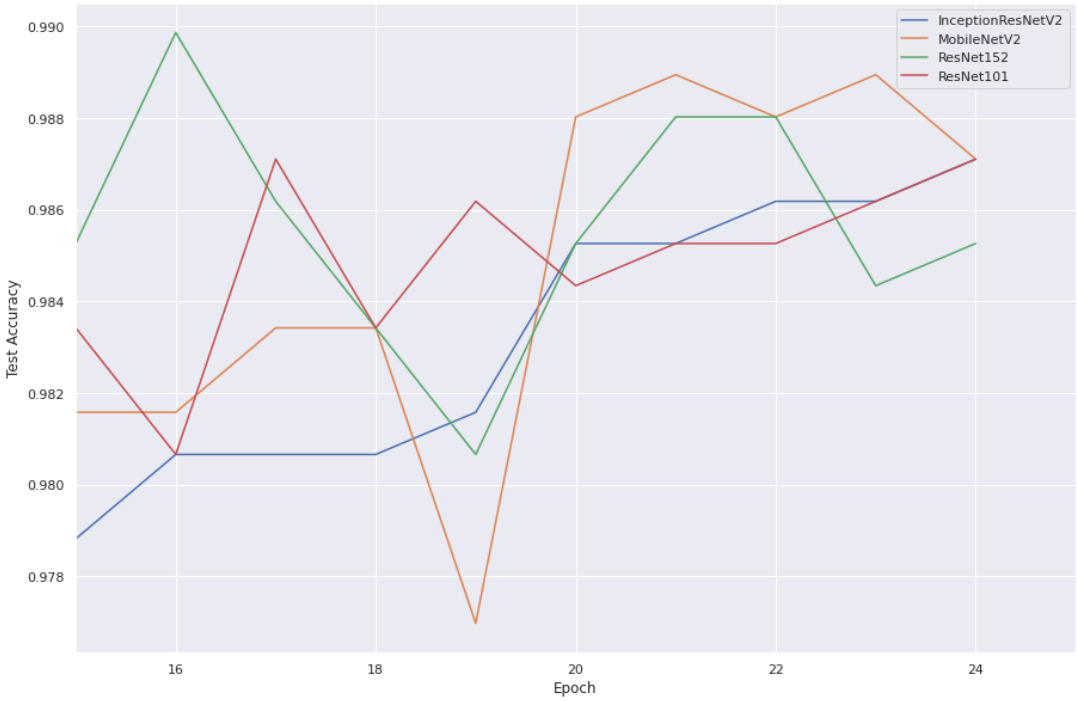

Figure 6: Fine-Tuning Accuracy Plot

6 Discussion and Analysis

6.1 True vs. Predicted class

It is observed in Figure 3 and Figure 4 that InceptionV3 and ResNet50 have a generally accurate prediction for most classes.

For InceptionV3, we can see that the diagonal portrays a true positive instance in 100% of cases for the classes Gatak, Kelihos ver1, Kelohis ver3, Lollipop, and Vundo. As for Obfuscator and Tracur, there was a 95% accuracy for the prediction. Ramnit was true in 99% of cases. However, the lowest recorded accuracy was for Simda, which was 67% accurate. As for the rest of the matrix, Simda was falsely identified as Obfuscator and Ramnit in 17% of cases and only 1% of cases for a few other classes were false.

All in all, InceptionV3 was able to predict 5 classes with a 100% accuracy, 2 classes with 95% accuracy, 1 class with 99% and one least identified in 83% of predictions. We can also deduce that the worst accuracy was recorded for the class Simda.

As for ResNet50, we can see that the classes Kelihos ver1, Kelihos ver3, Lollipop, and Vundo were 100% correctly predicted. Obfuscator and Tracur recorded 96%, Gatak was 99% and Ramnit was 98%. Also, for this model, Simda

showed the lowest accuracy result of 83%. However, in this model, it was falsely identified as Ramnit in 17% of cases, but not for Obfuscator. We analyze that ResNet50 showed similar results to InceptionV3, but a better prediction for Simda. Simda has shown low true prediction rates. This makes sense because the dataset does not contain many samples for this malware class. Overall, the heatmaps show an overall satisfactory prediction across the diagonals as well as the surroundings, where false predictions were somewhat negligible.

6.2 Training Accuracy per Epoch and Model

In figure 5, during the first epoch, we can observe that all models' performance curves spike upwards, with that of ResNet152 starting the highest at approximately 96.5% accuracy, ResNet101 at 96%, MobileNetV2 at 95.5% and InceptionResNetV2 at 95%. InceptionResNetV2's curve gradually increases to 97.56% between epochs 2 and 5, while those of MobileNetV2 and ResNet101 decrease slightly, with ResNet152's performance curve fluctuating more sharply downwards. MobileNetV2's curve decreases sharply to 96% during epoch 6, then spikes upwards until 98.3%, similar to that of ResNet101. From epoch 4 till 10, ResNet152's accuracy increases, reaching 99.2%, making it the highest reached value of all models, during all epochs. However, it decreases slightly and ends at

98.3%. MobileNetV2 decreaes sharply at epoch 12 but increases again, ending at 97.6%. ResNet101 reaches an accuracy of 98% at epoch 15. InceptionResNetV2 shows a significant decrease from epoch 5 to 9, reaching 96.4%, but increases again to 97.8% at epoch 12, decreases again to 96.5% at epoch 13 but ends at 97.4% in the last epoch.

It is clearly observed that the overall range of accuracy recorded is quite favorable, considering the number of epochs it took to train the Keras models. The end range of accuracy for all models is within the percentages 97.4 and 98.3, which is high, indicating good performance.

6.3 Fine-Tuning Accuracy per Epoch and Model

We can observe in Figure 6 that there were a few fluctuations when the models were being tested. A few remarkable points in the plot will be analyzed and interpreted for each model across the 10 epochs.

For InceptionResNetV2, there was a brief incline in accuracy during the 15th epoch, a rectilinear record until the third epoch, then we observe a spike between epoch 18 and 20. Afterwards, a smooth increase, reaching a 98.7% accuracy in the final epoch is achieved.

For MobileNetV2, a start from ~= 98.2% increases then drops to 97.7% in the 19th epoch. A spike then occurs to reach 98.8% in the 20th epoch then a minor increase then drop to 98.7% in the final epoch.

ResNet152 records the highest of model prediction starts at ~= 98.5% accuracy, spikes to 99% at epoch 16 then drops to 98.1% at epoch 19. It reaches 98.8% during epoch 21, then drops to 98.5% at the end.

ResNet101 accuracy fluctuates between 98.4% and 98.7% from start till epoch 20, where it gradually increases and ends at 98.7%.

We can observe that InceptionResNetV2, MobileNetV2 and ResNet101 all meet at an end of 98.7%. ResNet152, on the other hand, ends at 98.5%. all the models encountered fluctuations, but overall, it can be seen in the graph that from epoch 20 till the last, all models tended towards an increase. Therefore, 10 epochs have proven to be enough for improving accuracy.

6.4 Method Evaluation

The proposed method seems to be suitable for the malware classification use case and leaves room for a lot of optimization and improvement as a starting point. However, it is important to take into account other types of datasets and being able to accommodate for different types of input. A need to check for generalization is raised as a concern that must be tackled to ensure that the proposed method is more reliable and scalable across different datasets.

The obtained results beyond this paper were based on 10 different seeds for each of 7 different models from Keras for iteration of training, testing and fine-tuning to ensure there is no bias in the output metrics.

Based on the previous discussions, Transfer learning has indeed proven its effectiveness, yet it still includes some weaknesses when it comes to accurate predictions of some classes, such as Simda and test performance. Moreover, the models performed well during the final phases of testing, as they have shown an increase in the plots and the ratio of accuracy to epoch is remarkable as it solves the problem of long running time.

7 Closing remarks

7.1 Conclusion

In conclusion, the proposed method has proven to be effective in the use case of malware classification of binary files. The ratio of accuracy to epoch is remarkable. Transfer learning is indeed helpful when a dataset is relatively small; However, the dataset needs to be prepared and normalized according to the input layer of the neural network. The overall performance is adequate and the strategy used is straightforward. Transfer learning has saved a lot of time by using less epochs than usual to obtain an optimal accuracy. To each method its weaknesses; in this case some classes were predicted better than others, mainly due to sample size. Overall, this method is a step forward towards faster classification.

7.2 Future work

Where there is a contribution, there is always a door for further investigation and improvement; Therefore, future possible work extending from this research may include finding different variants and formats of malware and parsing them for classification. Other possible work worth looking into would be checking if normalizing the input image files does affect training when it comes to loss of data

by compression. Furthermore, another starting point would be further optimization and development of neural network architectures to achieve top-tier performance levels.

Appendix

A Gabor Filtering

The Gabor filter method is a computer vision use case inspired by mammals' visual cortex cells. It is characterized by a sinusoidal signal, having its own frequency and orientation and regulated by a Gaussian signal. A bank of Gabor filters is denoted by a matrix of different recorded orientations, which are used to identify different textures and/or orientations in a given image.

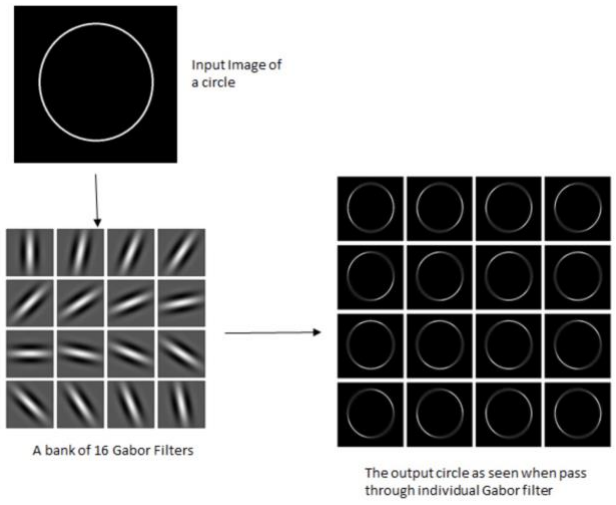

Figure 7: Feature Representation Detected by Gabor filters

The Gabor filtering technique relies two components, representing orthogonal directions by real and imaginary equations as follows, and a complex number formation is optional:

Real: $g(x,y;\lambda,\theta,\psi,\sigma,\gamma) = exp(-(x^{02} + \gamma^2 y^{02})/2\sigma^2)exp(i(2\pi x^0/\lambda + \psi))$

Complex: $g(x,y;\lambda,\theta,\psi,\sigma,\gamma) = exp(-(x^{02} + \gamma^2 y^{02})/2\sigma^2)cos(2\pi x^0/\lambda + \psi)$

Imaginary: $g(x,y;\lambda,\theta,\psi,\sigma,\gamma) = exp(-(x^{02} + \gamma^2 y^{02})/2\sigma^2)sin(2\pi x^0/\lambda + \psi)$

where

$$x^0 = xcos\theta + ysin\theta$$

and

$$y^0 = -xsin\theta + ycos\theta$$

References

[1] Hyrum S Anderson and Phil Roth. Machine Learning Models.

[2] Daniel Arp, Michael Spreitzenbarth, Malte Hu¨bner, Hugo Gascon, and Konrad Rieck. Drebin: Effective and explainable detection of android malware in your pocket, 2014.

[3] Omer Aslan and Abdullah Asim Yilmaz. A new malware classification framework based on deep learning algorithms. *IEEE Access*, 9:87936–87951, 2021.

[4] Daniel Gibert, Carles Mateu, Jordi Planes, and Ramon Vicens. Using convolutional neural networks for classification of malware represented as images. *Journal of Computer Virology and Hacking Techniques*, 15:15–28, 2019.

[5] Hotelling H. Analysis of a complex of statistical variables into principal components. J. Educ. Psych. 24, 417–441 (1933).

[6] Yin Minn, Pa Pa, Shogo Suzuki, Katsunari Yoshioka, Tsutomu Matsumoto, Takahiro Kasama, and Christian Rossow. Iotpot: Analysing the rise of iot compromises.

[7] Barath Narayanan Narayanan, Ouboti Djaneye-Boundjou, and Temesguen M. Kebede. Performance analysis of machine learning and pattern recognition algorithms for malware classification. *Proceedings of the IEEE National Aerospace Electronics Conference, NAECON*, 0:338–342, 2016.

[8] L. Nataraj, S. Karthikeyan, G. Jacob, and B. S. Manjunath. Malware images: Visualization and automatic classification. *ACM International Conference Proceeding Series*, 2011.

[9] Dr Yashpal Singh and Alok Singh Chauhan. Neural networks in data mining, 2005.

[10] IEEE Communications Society, Institute of Electrical, and Electronics Engineers. *2020 International Conference on Communication Systems Networks (COMSNETS).*

[11] Baoguo Yuan, Junfeng Wang, Dong Liu, Wen Guo, Peng Wu, and Xuhua Bao. Byte-level malware classification based on markov images and deep learning. *Computers and Security*, 92, 2020.

[12] Qian Zhu, Yuanqi Xu, Chenyang Jiang, and Wenling Xie. An android malware detection method based on native libraries. *Advances in Intelligent Systems and Computing*, 1168:233–246, 2021.