Journal of Computer Virology and Hacking Techniques (2019) 15:29–37
https://doi.org/10.1007/s11416-018-0324-z

ORIGINAL PAPER

Analysis of ResNet and GoogleNet models for malware detection

Riaz Ullah Khan[1] · **Xiaosong Zhang**[1] · **Rajesh Kumar**[1]

Received: 6 April 2018 / Accepted: 18 August 2018 / Published online: 28 August 2018

Abstract
We have utilized two distinct models to identify the obscure or new sort of malware in this paper. GoogleNet and ResNet models are researched and tried which belong to two different platforms i.e. ResNet belongs to Microsoft and GoogleNet is the intellectual property of Google. Two sorts of datasets are utilized for training and validation the models. One of the dataset was downloaded from Microsoft which is the combination of 10,868 records and these records are binary records. These records are additionally isolated in nine diverse classes. Second dataset is considerate dataset and it contains 3000 benign files. The said datasets were initially in the form of EXE files and were changed over into opcode, after that changed over into images. We got a testing accuracy of 74.5% on GoogleNet and 88.36% precision on ResNet.

Keywords Malware detection · Malware classification · Opcode · ResNet · GoogleNet

1 Introduction

Malicious software which is additionally alluded as malware is the major threat for computer users. The principal focal point of malware is, to accumulate the individual's data without the consideration of clients and to exasperate the PC activities which makes issues for clients. There are numerous sorts of malware, for example, Trojan-horse, Virus, Rootkit, Backdoor, Worms, Spyware, Adware and so on. Yearly reports from antivirus organizations demonstrate that a large number of malicious software are made day by day. Attackers are making new software and the new software turn out to be more advanced that they could never be recognized by the traditional discovery procedures, for example, signature-based recognition and behavior-based recognition.

Signature-based identification looks for determined bytes groupings into an object so it can recognize extraordinarily a specific kind of a malicious software. The main disadvantage is that it can't identify zero-day malicious software since the new software signatures aren't stored in the database. Behavior-based recognition was created to fundamentally conquer the impediment of the signature-based method, in the way that it filters the framework's behavior to recognize the abnormal exercises rather than looking for the signature of the malware. The constraint of the behavior-based procedure is that it influences the execution time of the system and more storage is required. This method focuses on the conduct of the program when it executes. The program is marked as benign if it executes normally, otherwise, it is set apart as a malware. By breaking down this meaning of the behavior-based method, we can specifically presume that the disadvantage of this procedure is the generation of numerous false positives and false negatives, considering the way that a legitimate program can be slammed and be set apart as a malware or malware can execute as a normal program.

1.1 Motivation

Malware is developing in the enormous number consistently. Image pattern recognition procedure can play an important role to detect and prevent threats of malware so we used this methodology to enhance precision and execution. Image pattern recognition method dissects malicious code into images which are gray-scale images and these gray-scale images are created through opcode image strategy. Previous research by Nataraj, et al. [1] proposed a strategy of gray-scale images for malware classification. Image preparing procedures are generally utilized for object pattern recognition, for example, Taobao is a well known online shopping site in China which discovers the item utilizing image recognition system. In our investigation, we changed from a twofold code to gray-scale

✉ Riaz Ullah Khan
rerukhan@gmail.com

[1] Center of Cyber Security, School of Computer Science and Engineering, University of Electronic Science and Technology of China, Chengdu, China

pictures for perceiving malicious software which protects the similarity variation pictures. Two models that are GoogleNet model and ResNet model were looked to accomplish better execution regarding velocity and precision.

1.2 Structure of paper

Section 1 of this paper talks about the presentation, inspiration and fundamental commitments of this work. Section 2 talks about malware detection procedures, examination methods which were previously by different researchers. Section 3 gives a concise review to the approach of changing over .exe files into pictures. Section 4 proposes a malware identification procedure, likewise portrays the execution and trial brings about terms of speed and precision. Finally, this paper is concluded with Section 5.

2 Background

2.1 Deep Learning

Current accomplishments in deep learning innovative work draw in individual's consideration. Google released TensorFlow in 2015 [2], a structure of acknowledging deep learning. All the more particularly, deep learning is a counterfeit neural network system, in which numerous layers of neurons are connected to each other with various weights and enactment capacities to take in the shrouded connection among inputs and outputs. Instinctively, input information is encouraged to the primary layer that produces diverse mixes of the information [3]. These combinations, after the activation function, are fed to the second layer, and so on. Under the above procedures, different combinations of the outputs from the previous layer can be seen as a different representation of features. The weights on joins between layers are balanced by in reverse propagation, contingent upon the separation between obviously labeled output. Deep learning approach could be viewed as a neural system with an expansive layer. After the above learning process by means of numerous layers, we can determine a superior comprehension and portrayal of recognizable features, improving the recognition precision [4]. Additionally, see that the adequacy of deep learning increments by the system measure. The most understood profound systems are convolutional neural systems (CNN) in addition to the Neural Network system. The portrayal of CNN incorporates AlexNet, VGG, GoogleNet, and ResNet [5]. All the more particularly, CNN is made out of concealed layers, completely associated layers, convolution layers, and pooling layers. The shrouded layers are utilized to expand the many-sided quality of the model. In the event that a similar number of neural is related with the information picture, the number of parameters can be altogether diminished, receiving to the capacity structure much legitimately.

2.2 Malware analysis

2.2.1 Static analysis

Analysis of the bad code segment or malicious characteristics when code is not executing is called static analysis [6–8]. There are many patterns to detect the static malware e.g. n-grams, string signature, control flow graph, bytes-sequence etc.

2.2.2 Dynamic analysis

Analysis of the bad code or malicious characters when software is running environment i.e. emulator, Virtual Machine, simulator etc. is called dynamic analysis. Dynamic analysis approach applied for monitoring and tracking system. Dynamic analysis is a better approach as compared to static analysis but dynamic analysis consumes more resources e.g. time and memory and has scalability issues.

2.2.3 Statistical and content analysis

This technique is based on a verity of techniques e.g. n-gram, n-perms, hash based, file structure.

2.2.4 Hybrid analysis

Analysis of the bad code or malicious characters while performing static and dynamic analysis in an offline mode [9,10]. It is also useful for the android application. In the first step of this method, the static analysis takes the image of the applications into small pieces which are also known as binary code and search suspicious patterns among the binary codes. In the second step, the code is executed by dynamic analysis in an Android emulator and logs its system calls. There are many patterns to detect the static malwares e.g. n-grams, string signature, control flow graph, bytes-sequence etc.

2.3 Related work

2.3.1 Deep learning-based malware detection

Deep learning once it seemed as the cure for the above problem. However, a preprocessing step, such as feature engineering, is still needed before the model is learned. Furthermore, the dataset for training the model usually cannot reflect real-world malware accurately. For example, [11] proposed a malware detection in which the Windows API inquiry generates a corresponding ID, which is treated as the input of the deep learning architecture (eg. stack of Auto-Encoders),

and then it fine-tunes the model parameters. [12] is a method that it works on features extracting first, such as contextual byte features, PE import features, string 2d histogram features, and PE metadata features. Then, the extracted features are fed into the deep neural network (DNN). With the training of two hidden layers, it is categorized. [13] uses static analysis to extract features such as required permission, sensitive API, and also uses dynamic analysis to extract features such as "action dex class load", "action recurrent" and "action service-start", from 500 samples for about 200 features as the input for the deep belief network (DBN). There is a similarity between the execution logic of Android malicious apps and the order of functions being called. Thus, in addition to the aforementioned solutions that apply DNN to malware analysis based on "exploit attack" and "privileged escalation", another category of malware detection relies on n-gram analysis on byte-code or op-code. For example, [14] and [15] first calculate the n-grams on the binary byte-code and then perform the malware detection based on k nearest neighbor. [16] proposes to do reverse-engineering first and then analyze op-code. In addition, one more category of the malware detection relies on transforming malware into the images. For example, [17] proposes to first transform binary byte-code into the gray-scale image and then applies pattern recognition to the gray-scale image. All of the above methods achieve a certain level of detection accuracy. However, as mentioned in the introduction, the number of malware increased dramatically. Even worse, more and more anti-debugging techniques are discovered. The size of the dataset used for training the model also has significant impacts on the detection accuracy and the computing efficiency in the training process. Here, we particularly note that despite the detection accuracy of the n-gram approach, Ngram approach consumes substantial computing resources and time for handling the dynamic growth of the model parameters required, implying the impracticality [18] However, if we have limited computing resources and time, CNN is able to handle the explosive data growth because the increased number of parameters does not imply the growth of computing resource and time required. Recently, [10] also proposes deep learning-based malware detection, where the sequences of the op-code are encoded as one-hot vectors for the input of CNN. However, this method needs to disassemble the Android apps via reverse-engineering tools for deriving small source code from classes.dex, and therefore cannot handle malware that with encryption and obfuscation.

2.3.2 Machine learning-based malware detection

Various machine learning algorithms are proposed for classification and detecting unknown codes into their families, for instance, Naive Bayes, Support Vector Machine, Clustering, and Association Rule. In this section, we will discuss a few of the good researchers who have worked in the field of malware detection and classification. M. Damshenas et al. [19] proposed a technique for detecting malware in mobile devices. This technique is comprising a server analyzer and a lightweight client agent. The server analyzer generates a signature for every application. The proposed technique is capable of generating standardized mobile malware signatures based on their behavior and this is the main contribution of their research. It compares the generated signature and previously blacklist of malware signature. N. Milosevic et al. [20] proposed the static analysis approach to detect and analyze malicious behavior within the code in Android apps. They use the machine learning approach to detect the malware families, this is also signature based anti-malware solution. They used Service Vector Machine (SVM) for finding the accuracy, the results were 95.6

Siddiqui at al. [21] provided a technique for malware detection. They used malware detection approaches by using data mining on file features. They categorized into analysis type, the file properties and the detection stages also used variable length instruction sequence. They used the decision tree and random forest algorithm for classification of the malware.

Egele et al. [22] analyzes the behavior of malware. They have designed the binary obfuscation methods, which transform the malware binaries into self-compressed. They have also designed a technique that uniquely identifies binary files which restricted the reverse engineering.

Nataraj et al. [1] used image processing technique to classify the malwares. They converted binary malware to gray-scale images. The proposed method of Nataraj et al. [1] represents executable binary files into gray-scale bitmap images.

Kong et al. [23] build a model to classify the malware, based on structural information. For the structural information, they use the function call graph, this function extracts the features of each malware sample. they used the discriminate distance metric learning method which clusters the malware samples belongs to the same family and also used assemble of a classifier that classifies malware into their respective families.

Tian et al [24,25] used the Weka [26] library to classify the Trojans malware by using frequency length function. This is measured by the number of bytes in the executable code. It is observed from their results that the malware family is identified by frequency function and can be joined with other features for classification of the malicious code.

Santos et al [27] used the semi-supervised learning for unknown malware detection. They used Learning with Local and Global Consistency (LLGC) semi-supervised algorithm for reducing the required number of instances while minting the high precision and also determining the optimal number of labeled instances which affect the model's accuracy. Further, Santos et al, [28] proposed a collective learning

technique to detect malicious code. It is a semi-supervised learning type which presents the methodology for optimization of the partially labeled data classification. To build different machine learning classifiers, collective classification algorithms are used with a set of unlabeled and labeled instances.

Zolkipli et al. [29] used the security tools e.g., HoneyClients, Amun for analysis the behavior of malwares. They analysed of the malware behavior of each sample by executing Cw-sandbox [30] and Anubis. The malware divided into two families (i) Worms and (ii) Trojans. The main disadvantage of this research is customization is not possible.

3 Environment setup and data preparation

3.1 Dataset

Datasets were downloaded from various sources. Malicious software dataset (malware) was downloaded from Microsoft. Similarly, we downloaded 3000 benign software's from open source websites. In the accompanying discussion, the datasets are described in details.

3.1.1 Dataset from Microsoft Malware Classification Challenge

The dataset which is collected from Microsoft contains 9 classes. 500GB of data incorporates 21741 malware samples. 10868 of tests are utilized for preparing, and the rest of the samples are utilized for testing.

a) Bytes Files: Byte files in Microsoft dataset include 10,868 training data and 10873 testing data. Each byte file contains a hexadecimal representation of binary content.

b) Asm Files: Asm files in Microsoft dataset include 10,868 training data and 10873 tasting data. Each Asm file extracted by the IDA dis-assembler tool and it contains metadata manifest. This information includes assembly command sequences, strings, function calls and so on.

c) Training Labels: MD5 Hash is the file name in the actual program and this name is used as a training label. The file of training label contains each MD5 hash and class of malware which it maps to. No training labels were provided for the test data input files.

d) Sample Submission: The sample submission file illustrates the valid submission format for 10,873 sample records.

e) Data Sample: The data sample file includes a preview of the test and training data (Table 1).

3.1.2 Benign files

We gathered 3000 of clean coded files from various sources.

Table 1 Microsoft malware classification challenge dataset

No	Family name	No of samples
1	Ramnit	1541
2	Lollipop	2478
3	Kelihos_ver3	2942
4	Vundo	475
5	Simda	42
6	Tracur	751
7	Kelihos_ver1	398
8	Obfuscator.ACY	1128
9	Gatak	1013

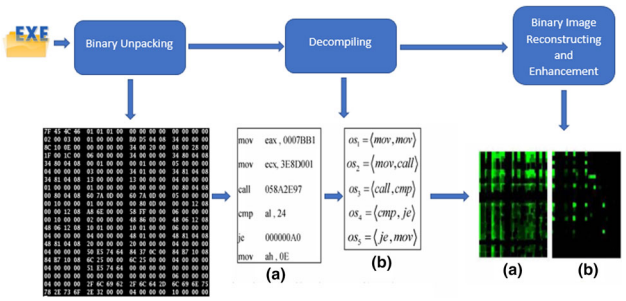

Fig. 1 Overview architecture of Preparation Dataset using opcode

3.2 Data preparation technique

The accompanying strategy to process the information is proposed in this paper.

3.2.1 Opcode to images

1) We utilized two tools to extract hidden code from binary files.
   a) PEID: This tool is utilized for static code
   b) Poly Unpack: This tool is utilized for dynamic code
2) Decompile Opcode: We have decompiled Opcode succession from assembly code and after that change over 2-tuple opcode grouping.
3) Opcode Sequence: The binary image matrices are reconstructed by these opcode sequences with their probabilities and information gains.The matrix is shown in Figure 1, each opcode sequences of length 2 can be matched to one of the elements in the matrix according to $os_i =< op_j, op_k >$,as shown in def 3 , def.. The element value $val(os_i \mid x_j)$ of the image matrix $im(x_j)$ is calculated by the probabilities $p(os_i \mid x_j)$ and the information gains $w(os_j)$ of os_iin binary x

$$val(os_i|x_j) = p(os_i|x_j)w(os_j) \quad (1)$$

The probabilities $p(os_i \mid x_j)$ and information gains $w(os_i)$ are calculated by the frequencies $freq(os_i \mid x_j)$ of the opcodes sequences of length 2 , as shown in Eq 2 and 3, where $p(os_i \mid y_1)$ be the probability of os_i in the training malware binaries, $p(os_i)$ be the probability of os_i in the whole training binaries, and $p(y_1)$ be the probability of training malware binaries.

$$p(os_i|x_j) = \frac{freq(os_i|x_j)}{\sum_{os_t Ex_j} freq(os_i|x_j)} \tag{2}$$

$$w(os_i) = p(os_i|x_j) \log\left(\frac{p(os_i|x_j)}{p(os_i)p(y_i)}\right) \tag{3}$$

1) Binary Image Re-construction and Enhancement: binary opcode frequency constructs images. Histogram normalization, dilation, and erosion techniques are used to enhance the Opcode sequences.

To improve the complexity between malware variation pictures and benign pictures, we utilize histogram standardization. enlargement and disintegration techniques to upgrade the binary pictures. Through picture upgrade, the difference of these uncommon opcode pictures would be improved.

Let $val_{enhance}(os_i \mid x_j)$ be the pixel-value of the enhanced image, the histogram normalization method is according to the equation 4

$$val_{enhance}(os_i|x_j) = \alpha \frac{val(os_i|x_j)}{max(val(os_i|x_j))} 255 \tag{4}$$

In this strategy of data preparation, we can without much of a stretch recognize malware and favorable records by visual investigation as appeared in Figure 1.

3.3 Environmental settings

We have used Ubuntu 64bit operating system and RAM of 8 GB. To play out the experiment we used Python programming language with Python libraries, for example, Tensor Flow, Docker Server, Anaconda. The Tensor Flow Library utilizes the architecture CNN.

3.4 Proposed model

This segment is separated into two parts. Initial part is about information planning or processing the data and the second part is tied in with training/testing the model. Following steps describe the model briefly, followed by Figure 2:

1) Download malware and benign softwares from open source databases and then convert into assembly code
2) Convert assembly code to 2-tuble code
3) Opcode sequence generate the matrix

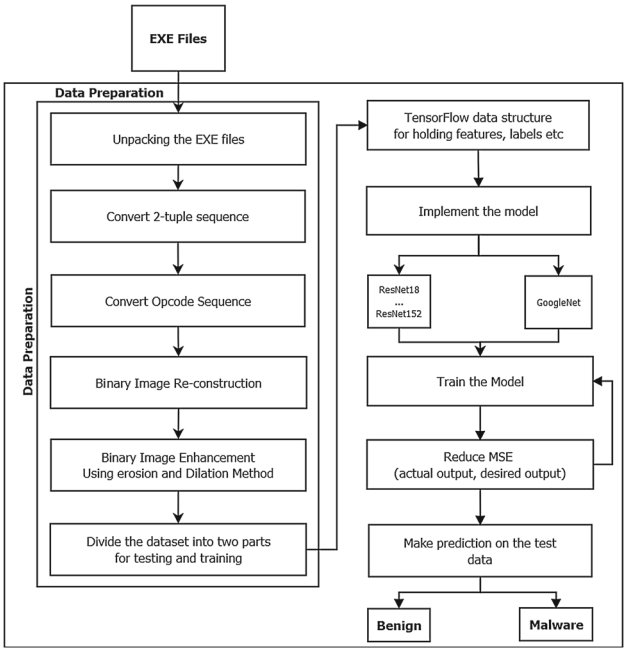

Fig. 2 Architecture of Proposed Model i.e. from data preparation to prediction

4) Generate images from binary code matrix
5) For the training and testing, divide dataset into two parts i.e. training and testing phases
6) Use tensor flow models i.e. GoogleNet and ResNet which hold the features and labels
7) Implement the relevant model
8) Optimize the model to reduce RMSE
9) Prediction on test dataset

Further, in our design, we partitioned our model in two stages, that is training and detection. For the preparation and the discovery of malicious code, we utilized CNN, as appeared in Figure 3. The outputs of the information arrangement were “images”. Pictures have paired marks that is malware or benign. We utilized CNN demonstrate which extricate the highlights naturally. The recognition stage appears in Figure 3. In other words, .exe (executable) files were converted into images and the classifier is used to detect the malicious code.

In the followings, we explain the procedure of malware detection methodology. First of all, we collected the software and they are classified as benign and malware. We decompile the software to the assembly files, finally, we constructed gray-scale images. We further applied CNN based models i.e. GoogleNet model and ResNet model to detect malware and compute the accuracy. The following sections describe in both models.

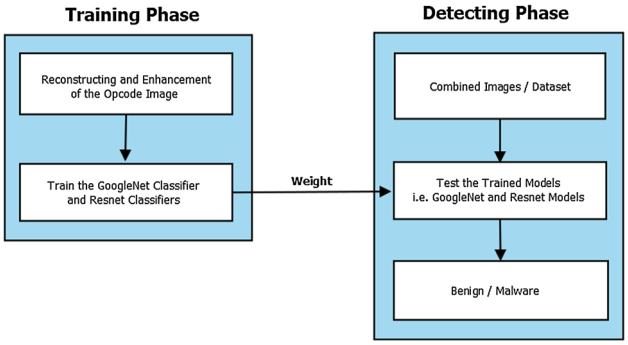

Fig. 3 Implementation of deep learning

3.5 Malware detection by GoogleNet model

The GoogleNet shows work by Google association in 2014, which contains 22 concealed layers. We have utilized the GoogleNet CNN system since it is solid and it can be connected to the whole picture at once and after that, we expect n effective feature extraction. However the model evolving in Inception v2, v3, v4. In this approach, we used inception v4 model. We also found that two approaches might be used to malware detection.

- Since the traditional filter size of CNN is 3*3 or 5*5, the uncorrelated byte-code might become correlated when we transform them into images. we replaced a few filters with a smaller Perceptron layer with a mixture of 1x1 and 3x3 convolutions. In this way, we reduce the dimensions inside the inception module.
- Pooling is a common approach in the CNN model to reduce the computation overhead significantly in traditional image recognition. The detection engine in our original research inherently uses pooling to achieve the speedup. However, Gray scale images are not natural images; instead, they are formed from EXE source code.
- ReLU was used rather than non-linearity work since it is quicker than sigmoid or tanh and helps in vanishing inclination issue which emerges in lower layers of the system.
- It takes a channel and a walk of a similar length at that point applies it to the information volume and outputs the most extreme number in sub-district that the channel convenes around. The intuition behind this was the fact that our malware image is a gray-scale and the layers like average max pooling may not help much because there is a lot of dark space in the image and they don't contribute much in the model.

3.6 Malware detection by ResNet model

ResNet is owned by Microsoft and it was introduced in 2015. Latest ResNet contains 152 hidden layers. ResNet convolutional neural networks are used in our experiments and the results show that it is accurate as compare to GoogleNet and it can be applied to all the images at a time. ResNet model is a good choice for extracting the features from images. The dubbing and innovation cost is increased in terms of time and memory but this also a fact that it gives us a high accuracy. Here dubbing means the transfer or copying of previously recorded structure of the same or a different type. In this experiment, we applied the Tensor Flow ResNet Library which is easy to deploy and achieve more accuracy than GoogleNet.

4 Implementation and evaluation

We run our experiments on openly sourced datasets from Microsoft Malware Classification Challenge. The images contained Malware and benign samples. We adopted the Deep Neural network model i.e. GoogleNet and ResNet model for comparison. It was observed that ResNet model has tremendous performance as compare to the GoogleNet model. We are tempted to believe that, if the data sets were larger than we have experimented, ResNet152 would have performed better. However, ResNet (18, 50 and 101) really performed better in terms of prediction accuracy which is observed in Table 2 and Figures 4, 5, 6, 7 and 8. It however performed poorly in terms of running time on malware dataset which is observed in Table 2 and Figure 9. Be that as it may, when just a restricted measure of preparing information is accessible, all the more capable models are required to accomplish an improved learning capacity. It is along these lines of incredible essentially to think about how to plan deep models to gain from less preparing information, partic-

Table 2 Comparison results of GoogleNet and ResNet models

Model	Training accuracy	Testing acuracy	Loss	Validation loss	Time
GoogleNet	0.84	0.745	0.389	NA	1000s
ResNet18	0.83	0.87	1.0436	1.006	2701s
ResNet34	0.8651	0.8519	1.5983	1.7510	4800s
ResNet50	0.8662	0.8095	4.3967	4.5914	5580s
ResNet101	0.8594	0.7884	8.4274	8.7475	6112s
ResNet152	0.8798	0.8836	11.943	12.05	9248s

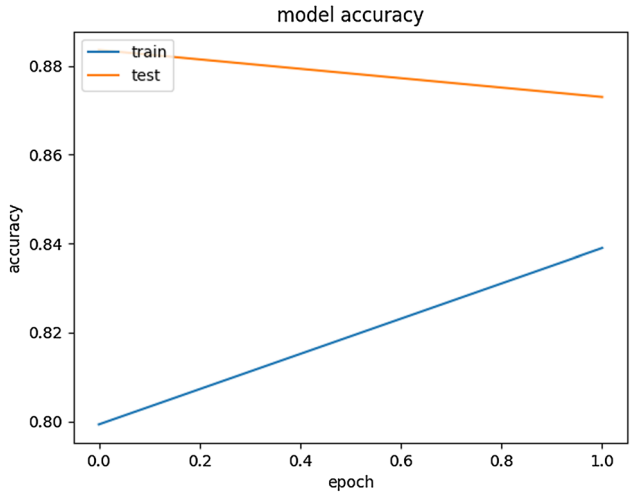

Fig. 4 Model accuracy and model loss of ResNet 18

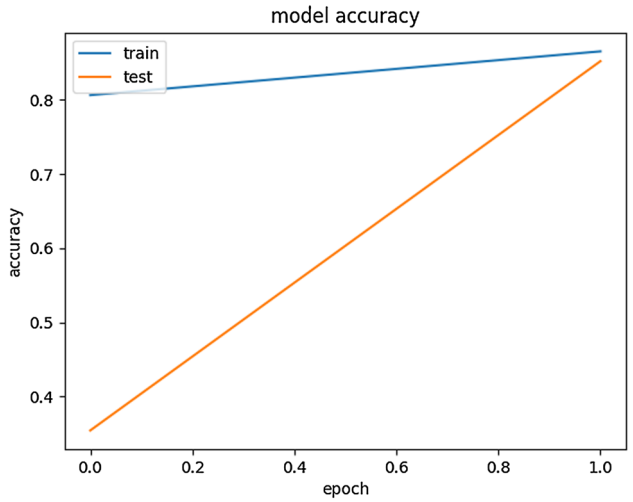

Fig. 5 Model accuracy and model loss of ResNet 34

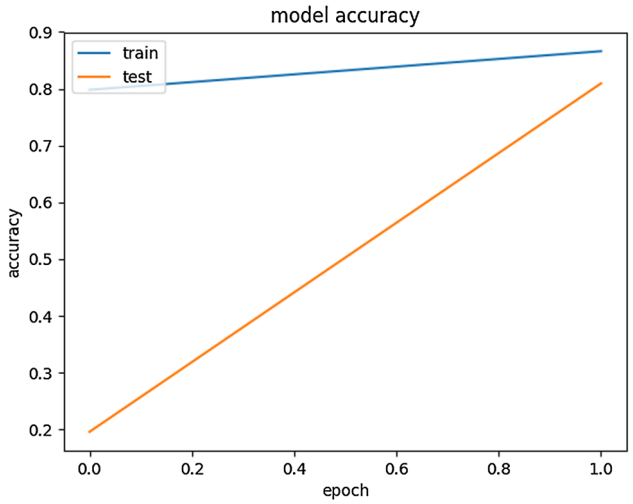

Fig. 6 Model accuracy and model loss of ResNet 50

ularly for discourse and visual acknowledgment frameworks. This was apparent in our explore different avenues regarding

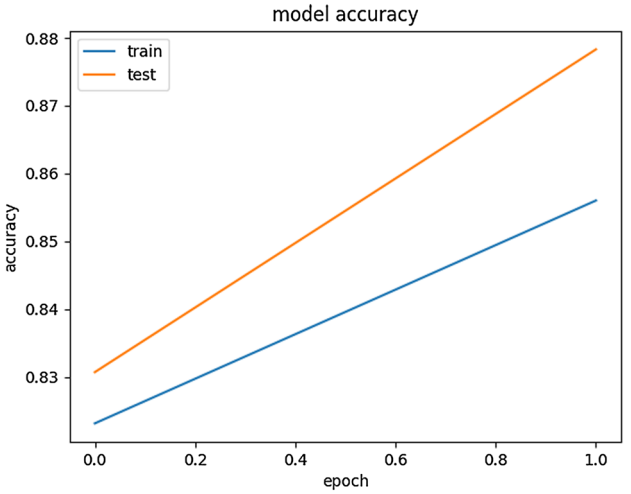

Fig. 7 Model accuracy and model loss of ResNet 101

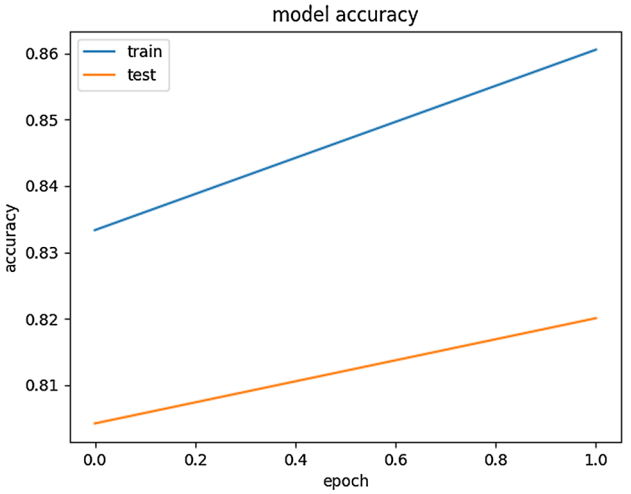

Fig. 8 Model accuracy and model loss of ResNet 152

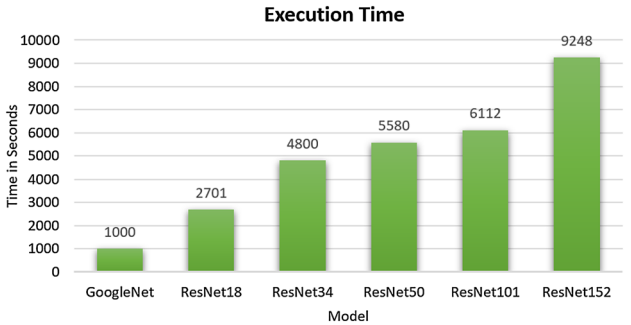

Fig. 9 Execution time of different models

ResNet uses of optimization algorithms to adjust the network parameters: The technique to modify the parameters in machine learning calculations is a rising theme in software engineering. In DNNs, an extensive number of parameters should be balanced. Additionally, with an expanding number of shrouded hubs, the calculation is more probably get caught in the nearby ideal. Enhancement procedures, for example, the PSO, are hence required to maintain a strategic distance

from this issue. The proposed preparing calculation ought to have the capacity to extricate the highlights naturally and diminish the loss of data to moderate both the scourge of dimensional and the local optimum.

From the plot in Figure 4, we observed that the model could likely be prepared more, as the drift for accuracy on both datasets is rising for the last couple of epochs. We also observed that the final training accuracy of the model is 0.83 and the testing accuracy is 0.87. Further. we observed that the model has different execution results on training and validation datasets i.e. labeled test. If these parallel plots begin to depart reliably, it may be an indication to quit training at a prior epoch. In this analysis, final training loss is noted as 1.0436 and validation loss is noted as 1.006. The execution time of ResNet18 model was noted as 2701 seconds.

From the plot in Figure5, we observed that the model could likely be prepared more, as the drift for accuracy on both datasets is rising for the last couple of epochs. We also observed that final training accuracy of the model is 0.8651 and the testing accuracy is 0.8519. Further. we observed that the model has different execution results on training and validation datasets. The final training loss is noted as 1.5983 and validation loss is noted as 1.7510. The execution time was noted as 4800 Seconds.

From the plot in Figure 6, we observed that the model could likely be prepared more, as the drift for accuracy on both datasets is rising for the last couple of epochs. Further, We observe that the model's last preparing accuracy is 0.8662 and testing accuracy is 0.8095. We also observed that the model has equivalent execution on both training and testing datasets. If these parallel plots begin to depart reliably, it may be an indication to quit training at a prior epoch. In this analysis, final training loss is 4.3967 and validation loss is 4.5914. Add up to execution time was noted as 5580 Seconds.

From the plot in Figure 7, we observed that the model could likely be prepared more, as the drift for accuracy on both datasets is rising for the last couple of epochs. Further, We observe that the model's last preparing accuracy is 0.8594 and testing accuracy is 0.7884. Further. we observed that the model has different execution results on training and validation datasets. If these parallel plots begin to depart reliably, it may be an indication to quit training at a prior epoch. In this analysis, final training loss is 8.4274 and validation loss is 8.7475. Add up to execution time of ResNet101 was noted as 6112 Seconds.

From the plot of model accuracy in Figure 8, We can see that the model's final training accuracy is 0.8798 and validation accuracy is 0.8836. In this experiment, training loss is 11.943 and validation loss is 12.05. The execution time for ResNet152 was noted as 9248 Seconds (Fig. 10).

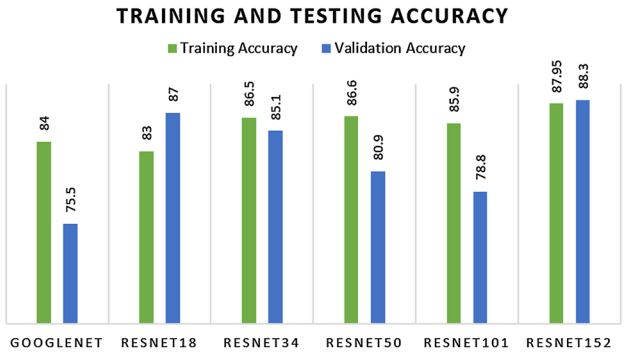

Fig. 10 Comparison of the training and testing accuracy of different models

5 Conclusion

Having the capacity of visualizing the vindictive code as images has been an awesome accomplishment. Numerous analysts have been utilizing this procedure for the errand of malware grouping and identification. We have shown in this works how a small change in the image could lead to miss-classification of images and how a small change in the image could lead an effective classification. The greatest test is to locate a proficient method to defeat the vulnerability of Neural Networks. This could be accomplished via precisely malware binaries. We observed in our study that the GoogleNet Model took less time in execution but ResNet152 model is more accurate. The execution time of ResNet152 model is the most highly taken time as compare to GoogleNet and other models of ResNet family.

Acknowledgements Funding was provided by National Natural Science Foundation of China (Grant No. 61572115).

References

1. Nataraj, L., Yegneswaran, V., Porras, P., Zhang, J.: A comparative assessment of malware classification using binary texture analysis and dynamic analysis. In: Proceedings of the 4th ACM Workshop on Security and Artificial Intelligence, pp. 21–30 (2011). https://doi.org/10.1145/2046684.2046689
2. Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G.S., Davis, A., Dean, J., Devin, M. Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mane, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viegas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., Zheng, X.: TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems (2016). arXiv:1603.04467
3. Dahl, G., Stokes, J., Deng, L.: Large-scale malware classification using random projections and neural networks. Acoust. Speech (2013). http://ieeexplore.ieee.org/abstract/document/6638293/
4. Adebayo, O.S., Aziz, N.A.: Static code analysis of permission based features for android malware classification using a priori algorithm with particle swarm optimization. J. Inf. Assur. Secur. **10**(4), 152–163 (2015)

5. Bennasar, H., Bendahmane, A., Essaaidi, M.: An Overview of the State-of-the-Art of Cloud Computing Cyber-Security, pp. 56–67. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-55589-8_4
6. Barrera, D., Kayacik, H.G., van Oorschot, P.C.: A methodology for empirical analysis of permission-based security models and its application to android. *17th Proceedings* (2010). http://dl.acm.org/citation.cfm?id=1866317
7. Enck, W., Ongtang, M., McDaniel, P.: On lightweight mobile phone application certification. In: 16th ACM Conference on Computer (2009). http://dl.acm.org/citation.cfm?id=1653691
8. Felt, A.P., Greenwood, K., Wagner, D.: The effectiveness of application permissions. In: Conference on Web Application (2011)
9. Afifi, F., Anuar, N.B., Shamshirband, S., Choo, K.K.R.: DyHAP: Dynamic Hybrid ANFIS-PSO Approach for Predicting Mobile Malware. PLoS ONE (2016). http://journals.plos.org/plosone/article?id=10.1371/journal.pone.0162627
10. Tong, F., Yan, Z.: A hybrid approach of mobile malware detection in Android. J. Parallel Distrib. Comput. (2017). http://www.sciencedirect.com/science/article/pii/S074373151630140X
11. Hardy, W., Chen, L., Hou, S., Ye, Y., Li, X.: DL4MD: A Deep Learning Framework for Intelligent Malware Detection
12. Saxe, J., Berlin, K.: Deep neural network based malware detection using two dimensional binary program features. In: 2015 10th International Conference on Malicious and Unwanted Software (MALWARE), pp. 11–20. IEEE (2015). http://ieeexplore.ieee.org/document/7413680/
13. Yuan, Z., Lu, Y., Xue, Y.: Droiddetector: Android Malware Characterization and Detection Using Deep Learning. Tsinghua Science and Technology (2016). http://ieeexplore.ieee.org/abstract/document/7399288/
14. Abou-Assaleh, T., Cercone, N., Keselj, V.: N-Gram-based Detection of New Malicious Code. ieeexplore.ieee.org (2004). http://ieeexplore.ieee.org/abstract/document/1342667/
15. Reddy, D., Pujari, A.: N-gram analysis for computer virus detection. J. Comput. Virol. (2006). http://www.springerlink.com/index/9H321858271V2720.pdf
16. Moskovitch, R., Feher, C., Tzachar, N., Berger, E.: Unknown Malcode Detection Using Opcode Representation. Springer, Berlin (2008). http://www.springerlink.com/index/B6H4KR787186P460.pdf
17. Nataraj, L., Karthikeyan, S., Jacob, G., Manjunath, B.S.: Malware images. Proceedings of the 8th International Symposium on Visualization for Cyber Security—VizSec'11, pp. 1–7 (2011). http://dl.acm.org/citation.cfm?id=2016904.2016908
18. Zhang, X., Zhao, J., LeCun, Y.: Character-Level Convolutional Networks for Text Classification. papers.nips.cc. http://papers.nips.cc/paper/5782-character-level-convolutional-networks-fo
19. Damshenas, M., Dehghantanha, A., Choo, K.-K.R., Mahmud, R.: M0Droid: An android behavioral-based malware detection model. J. Inf. Privacy Secur. **11**(3), 141–157 (2015). https://doi.org/10.1080/15536548.2015.1073510
20. Milosevic, N., Dehghantanha, A., Choo, K.K.R.: Machine learning aided Android malware classification. Comput. Electr. (2017). http://www.sciencedirect.com/science/article/pii/S0045790617303087
21. Siddiqui, M., Wang, M.C., Lee, J.: Detecting internet worms using data mining techniques. J. Syst. Cybern. (2009). http://www.iiisci.org/Journal/CV%7B$%7D/sci/pdfs/QI505RM.pdf
22. Egele, M., Scholte, T., Kirda, E., Kruegel, C.: A survey on automated dynamic malware-analysis techniques and tools. ACM Comput. Surv. (CSUR) (2012). http://dl.acm.org/citation.cfm?id=2089126
23. Kong, D., Yan, G.: Discriminant malware distance learning on structural information for automated malware classification. In: Proceedings of the 19th ACM SIGKDD International (2013). http://dl.acm.org/citation.cfm?id=2488219
24. Tian, R., Batten, L.M., Versteeg, S.C.: Function length as a tool for malware classification. In: 3rd International Conference on Malicious and Unwanted Software (MALWARE 2008), pp. 69–76 (2008). http://ieeexplore.ieee.org/abstract/document/4690860/
25. Tian, R., Batten, L., Islam, R., Versteeg, S.: An automated classification system based on the strings of trojan and virus families. In: 2009 4th International Conference on Malicious and Unwanted Software (MALWARE 2009), pp. 23–30 (2009). http://ieeexplore.ieee.org/abstract/document/5403021/
26. Hall, M., Holmes, F.E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I.A.: The WEKA data mining software: an update. SIGKDD Explor. **11**(1) (2009). http://dl.acm.org/citation.cfm?id=1656278
27. Santos, I., Laorden, C., Bringas, P.G.: Collective classification for unknown malware detection. The International Conference (2011). http://ieeexplore.ieee.org/abstract/document/6732395/
28. Santos, I. Devesa, J., Brezo, F., Nieves, J.: Opem: a static-dynamic approach for machine-learning-based malware detection. Joint Conference CISIS (2013). https://doi.org/10.1007/978-3-642-33018-6_28
29. Zolkipli, M.F., Jantan, A.: An approach for malware behavior identification and classification. Comput. Res. Dev. (2011). http://ieeexplore.ieee.org/abstract/document/5764001/
30. Willems, C., Holz, T., Freiling, F.: Toward automated dynamic malware analysis using cwsandbox. IEEE Secur. Privacy **5**(2), 32–39 (2007)