



A Novel Approach for Android Malware Detection and Classification using Convolutional Neural Networks

Ahmed Lekssays¹ ^a, Bouchaib Falah¹ ^b and Sameer Abufardeh² ^c

¹*School of Science and Engineering, Al Akhawayn University in Ifrane, Ifrane, Morocco*

²*Math, Science, and Tech. Dept., University of Minnesota Crookston, Crookston, MN, U.S.A.*

Keywords: Malware, Android, Machine Learning, Classification, Convolutional Neural Networks.

Abstract: Malicious software or malware has been growing exponentially in the last decades according to antivirus vendors. The growth of malware is due to advanced techniques that malware authors are using to evade detection. Hence, the traditional methods that antivirus vendors deploy are insufficient in protecting people's digital lives. In this work, an attempt is made to address the problem of mobile malware detection and classification based on a new approach to android mobile applications that uses Convolutional Neural Networks (CNN). The paper suggests a static analysis method that helps in malware detection using malware visualization. In our approach, first, we convert android applications in APK format into gray-scale images. Since malware from the same family has shared patterns, we then designed a machine learning model to classify Android applications as malware or benign based on pattern recognition. The dataset used in this research is a combination of self-made datasets that used public APIs to scan the APK files downloaded from open sources on the internet, and a research dataset provided by the University of New Brunswick, Canada. Using our proposed solution, we achieved an 84.9% accuracy in detecting mobile malware.

1 INTRODUCTION


According to Statista, the number of Android devices in the world is 2.53 Billion devices. This number is expected to reach 2.87 Billion in 2020 (Statista, 2020). On the other hand, Kaspersky Lab detected more than 6 million malware package in 2018 (Lab., 2019). These numbers show the criticality of mobile security and the challenges it has been facing in the last years. It is an open war between malware writers and AV vendors.


Due to the limitations of signature-based detection, AV vendors rely on some heuristic methods to detect malware. These methods are based on some rules defined by the AV security experts where they state the definition of restricted behaviors or operations that a software can do or execute. The main drawback of heuristic-based methods is that they generate many false positives since not all software that access certain files are malware. However, they help in detecting new malware without comparing it with


a signature that was saved in AV vendors' databases (Michie et al., 1994).

From the two discussed detection techniques, AV vendors used a hybrid technique that combines both signature-based and heuristic-based methods to detect malware. There are two types of analyses that are done on any software to detect if it is malware or not: static analysis and dynamic analysis (Zhao and Liu, 2007). They are both done in a sandboxed environment for security purposes. For static analysis, it is a phase in malware analysis where the malware analyst does not execute the malware. He or she tries to reverse engineer the malware to get its source code. Then, the malware analyst checks the string signature, byte-sequence, opcodes, etc. For dynamic analysis, it has two main components: memory and network. It is a technique in malware analysis where the malware analyst executes the malware and watch its behavior in the sandboxed environment. The malware analyst checks the behavior in the run time and the API endpoints and calls that the malware tries to access. It gives an idea about the severity of the malware and the ways that security professionals can implement to stop the malware (Zhao and Liu, 2007).

Nowadays, the probability of success of machine

^a  <https://orcid.org/0000-0001-5783-8638>

^b  <https://orcid.org/0000-0001-5086-0808>

^c  <https://orcid.org/0000-0002-9893-8923>

learning in classification problems has increased thanks to three main components: (1) the increase of commercial feeds that helped new malware to exist, (2) computing power has become cheaper, and (3) machine learning as an independent computer science discipline has evolved, and big companies are investing in it which help researchers by providing the tools to innovate in the field.

Machine learning approaches and results are encouraging to achieve a high malware detection rate without any human interaction. As a result, AV vendors and their research and development teams began deploying some machine learning classifiers such as neural networks, decision trees, and logistic regression (Krizhevsky et al., 2012).

Malware analysis, as an independent discipline in cybersecurity, has been facing the problem of malware classification or detection as a binary classification. So, any file shall be analyzed to detect if it is malware or not. If it is malware, it is labeled according to its type and family based on its behavior by using a classification mechanism. The main purpose behind this work is introduce a different approach for malware detection in Android by using visual characteristics of malware and deep learning for pattern recognition.

Contributions. In this work, we have:

- Combined and preprocessed a dataset containing benign and malicious Android applications;
- Developed a machine learning model based on CNN to detect and classify mobile applications samples as benign or malware.
- Experimented the suggested model based on comparisons with other defined models.

Outline. The rest of the paper is organized as follows. In Section II, different malware types are introduced. Then, in Section III, we introduce the different malware analysis methodologies, whereas in Section IV we discuss the related work. After that, in Section V, we present in details our methodology. Then, we explain the process of processing malware as an image in Section VI. We share our results in Section VII. And finally, we present out conclusions and future work in Section VIII.

2 MALWARE TYPES

Malware is a compound word of two words: malicious and software. Malware are software programs that are designed and implemented to damage or execute some malicious commands on a system which may lead to some unwanted actions for the user such

as gathering sensitive information, disrupting normal computer operations, gain control over the computer system, spying on the user's daily activities by gaining access to mobile sensors, and destroying the mobile system. The word malware is the general terminology used to describe any malicious software. However, they can be technically divided into the following categories depending on their goal.

- **Adware:** It is a type of malware that automatically displays advertisements. It is used to gather data about users' interests and to get revenue from the displayed advertisements.
- **Spyware:** It is a type of malware that tracks the daily activities of the users without them knowing. It is dependent on the data that it gets from mobile sensors and other running applications. Spyware can collect sensitive data, including keystrokes, data harvesting, and monitoring activities.
- **Virus:** It is a type of malware that can copy itself and spread on the mobile system. Viruses can be transported on any medium, including but not limited to email attachments, social media messages, malicious links, etc.
- **Worm:** It is a type of malware that can spread on the network by exploiting operating system vulnerabilities. The major difference between worm and virus is that virus depends on human action to spread while the worm can replicate itself and spread without any human interaction.
- **Trojan:** it is a type of malware that makes itself appear as a normal file or application to trick users into downloading and installing the trojan. A trojan can give unauthorized remote access to the infected mobile phone. It is usually designed in a client-server architecture where the server is installed on the attacker's machine, and the client is the trojan itself. It is used to steal private information including but not limited to logins, financial data, cryptocurrencies wallets, etc. In addition, it is used to enable some devices on the victim's mobile phone such as front camera, spying on users' activities such as keystrokes and files.
- **Rootkit:** It is a type of malicious software designed to access other mobile phones remotely and control them without being detected.
- **Backdoors:** It is a computer software that allows access to compromised mobile phones. It allows the attacker to have an entry point to the mobile phone without the consent of the user.
- **Ransomware:** It is a malicious software that restricts the user from accessing his or her files by encrypting them. The decryption happens after

the attacker receives money from the victim. The payment is usually done using cryptocurrencies like Bitcoin or Ethereum.

Malware detection and classification problem is one of the key issues that AV vendors face since their foundation. They have been detecting malware using signature-based methods and heuristic methods. A malware signature is a hash that identifies a specific malware uniquely. This property helped AV vendors to detect all the malware that happen to be from a certain family through a generic signature.

From experiments, researchers found out that all malware in a certain family share properties, behaviors, and a generic signature of that family. This idea is a fundamental concept that this thesis is built upon. However, when AV started to detect such malware, malware authors tried to bypass the security mechanisms implemented by AV vendors. They started to write polymorphic and metamorphic malware to avoid matching generic signatures. Polymorphic malware uses a polymorphic engine to mutate where the original algorithm stays intact. One of the main techniques used in writing polymorphic malware is encryption. Metamorphic malware translates their binary code into a temporary representation. Then, they edit their temporary representation, and they translate then the edited version back to machine code again. This mutation form can be achieved by several techniques that may go to the architectural level of the mobile phone by inserting NOP instructions or changing the machine instructions completely.

3 MALWARE ANALYSIS METHODOLOGIES

3.1 Static Analysis

Malware static analysis is based on the analysis of the source code. It is called static because the analysis is done without running the malware in a sandboxed environment. It is also called code analysis. It is basically about examining the code without executing the program. It helps in having an overall idea about the code structure and auditing the code to check if it adheres to industry standards. Automated tools help security analysts and developers in performing static analysis. The main advantage of static analysis is that it can reveal bugs that do not manifest themselves. Nevertheless, static analysis is just a first step towards analyzing the behavior and the effects of a certain application (Schmidt et al., 2009).

3.2 Dynamic Analysis

Dynamic analysis is a technique used in computer forensics and software testing in order to test and evaluate an application by executing it in a sandboxed environment in real-time. The malware analyst keeps an eye on the behavior of the application in terms of CPU, memory, and network usage. Automated tools can help in this process by raising alerts in case of suspicious activity by the application (Schmidt et al., 2009).

4 RELATED WORK

Researchers have been exploring the field of malware detection in android applications from different perspectives and angles. In this work, we are exploring the field of malware detection using deep learning in the light of convolutional neural networks. Many approaches were used to detect malware based on their network traffic, permissions, memory behavior, and CPU usage. However, in our research, we followed a static approach which is about detecting malware by converting malware to images.

Ahmadi et al. have worked on novel feature extraction, selection and fusion for Windows malware families classification. Their work tried to keep up with the involvement of modern malware which is designed with mutation characteristics such as polymorphism and metamorphism. These characteristics lead to an exponential growth in the number of variants for each malware. In their research, they developed a novel paradigm that is effective in classifying malware variants using feature extraction. They group malware based on feature extraction, selection and fusion (Ahmadi et al., 2016).

Saxe and Berlin discussed the problem of malware detection from a different approach. They used deep neural networks to detect malware based on two-dimensional binary program features. They take the Windows binaries of malware and they designed a system that learns from the binary features. They introduced an approach that achieves a usable detection rate at a low false-positive rate (Saxe and Berlin, 2015).

Yajamanam et al. have chosen a different approach. They integrated a computer vision approach by calculating GIST descriptors for image-based malware classification. GIST descriptors are image features that have been recently used a lot in the field of malware classification. In their research, they implemented, tested, and analyzed a malware score based on GIST descriptors. It is a potential advantage for

the field of malware classification. Their research was based on Windows malware (Yajamanam et al., 2018).

Li et al. opted for a hybrid malicious code detection using deep learning. They suggested a new Android classification method called HADM, which stands for Hybrid Analysis for Detection of Windows Malware. They start with static and dynamic information extraction. Then, they convert it into vector-based representations. The method is based on combining features extracted from deep learning with the original features which resulted in an increase in detection rate (Li et al., 2015).

Tong et al. have developed a hybrid approach for mobile malware detection in Android. They adopted both static and dynamic analysis. They collected execution data of sample malware and benign applications using a *netlink* technology to generate patterns of system calls. They have built up a malicious pattern set and normal pattern set in order to compare the patterns of malware and benign applications. For detecting unknown applications, they have followed a dynamic method to collect system calls data. Then, they compare them with the patterns that were built up before (Tong and Yan, 2017).

Narudin et al. have evaluated machine learning classifiers for mobile malware detection. They used various network traffic features, and they group them into four categories selected based on basic information such as content-based, time-based, and connection-based. They have used their own dataset. They conducted multiple experiments and they found that k-nearest neighbor is the efficient classifier for malware detection (Narudin et al., 2016).

5 METHODOLOGY

5.1 General Overview of the Solution

The suggested solution of this paper is inspired by previous research that has been done on malware detection and classification on the Windows platform (Ahmadi et al., 2016). The idea is to convert malware to images based on the observation that images of different malware samples from the same malware family appear to be the same as shown in Fig. 1. They have common visual characteristics. The idea suggested in this paper is not common in research since it was tested only on Windows. In addition, if the malware was embedded in another application, it saves the same visual characteristics, so it produces a similar image to its family.

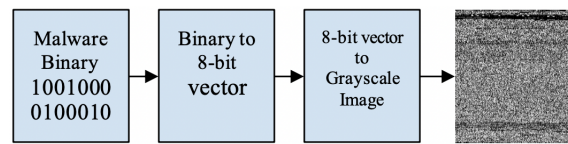


Figure 1: Visualizing Malware as Grayscale Image.



Figure 2: Charger Malware Family Visualization.

The work in (Ahmadi et al., 2016) focused on computing image-based features to characterize malware precisely. They used GIST descriptors to calculate texture features without going through the process of segmentation. This step resulted in feature vectors for each malware in the size of 900 features. However, their work used just 320 features because based on research, they found that the 320 feature is the optimal number of features needed to identify malware. In addition, they suggested that just 60 feature is the minimum that can be used with an error of 30%.

The feature vectors are used to train a K-nearest neighbor classifier with Euclidean distance. The conversion process expects a malware binary file, in our case the APK file, that is read as a vector of 8-bit unsigned integers and structured as a 2D array. This array is, then, visualized as a grayscale image in the range [0, 255].

Visualizing malware as an image as the example shown in Fig. 2 has many benefits especially that sequences of a binary can be identified easily. Moreover, malware authors tend usually to change some small portions of the code in order to write new variants. Those small pieces of code are usually changed after the malware is caught by anti-viruses. So, images are a good tool to detect the changes while having a global structure of the malware. Hence, different malware can be regrouped into families based on their visual properties, so they can be easily identified from images.

5.2 Dataset

The data set used in this research is called Android Malware Dataset (CICAndMal2017) (Lashkari et al., 2018). The approach used to build this dataset is to run both malware and benign applications on real android smartphones in order to ensure the exact running behavior of the applications. Research has shown that simulators often result in inconsistent behaviors of the applications, which might change the

end results. In addition, some malware is smart enough to detect emulated environments. The dataset is composed of 10,854 samples (4,354 malware and 6,500 benign) from different sources. The benign applications are downloaded from Google Play in the period between 2015 and 2017. However, the dataset runs 5,491 applications (426 malware and 5,065 benign). Due to storage and computational power limitations, we have used in this research a sample of 852 applications (426 malware and 426 benign). This step was also done to balance the dataset from any bias. The malware samples in this dataset are classified into four categories:

- Adware
- Ransomware
- Scareware
- SMS Malware

The samples consist of 42 unique malware families within the four mentioned categories above.

6 MALWARE IMAGES PREPROCESSING

6.1 Labeling

During this research, we have been dealing with a binary classification problem using convolutional neural networks. In order to build a balanced and unbiased dataset, we downloaded benign android applications from Play Store. The process of identifying if the application is benign or malware was done using VirusTotal API. VirusTotal is a web framework that provides malware analysis as a service based on file signatures since it has a large database (VirusTotal, 2020).

6.2 Data Pre-processing

In order to feed the CNN, a normalization step is needed in order to decrease the sizes of the images and to make them unified. CNN expects a set of labeled images of the same size. However, this was a problem in this research since images have different dimensions. In order to overcome this problem, we used mean subtraction and normalization.

6.2.1 Mean Subtraction

For the mean subtraction, it is a widely used technique in preprocessing images for CNN. It is about subtracting the mean across every individual feature

in the data. It can be interpreted geometrically by centering the data cloud around the origin of every dimension. In our implementation, we used NumPy arrays. We have implemented this operation as: $X - = np.mean(X)$ where X is our NumPy array that holds the data assuming that we have grayscale images.

6.2.2 Normalization

For the normalization step, it refers to normalizing the data dimensions so they are approximately the same scale. We have implemented this step by dividing the standard deviation after making the data zero-centered. The implementation using NumPy was done as follows: $X / = np.std(X)$ where X is our NumPy array that holds the data assuming that we have grayscale images.

6.2.3 GIST Feature Extraction

In this research, we have worked with two types of data. We have used images after performing the pre-processing phase and feed them to the CNN. In addition, we have extracted the GIST vector features. We have used the first 320 values. This was implemented using LearGist Python Wrapper since the official python library seems to be dead [15]. The idea is to give a preprocessed image with the same size of computing its GIST which results in a feature vector of 960 values in which we used 320 values because researchers concluded that 320 is enough to get optimal results. In addition, researchers stated that only 60 values will give accurate results up to 60% (Douze et al., 2009).

6.2.4 Malware Images Classification

In this paper, we have used the k-nearest neighbor algorithm. KNN or k-Nearest Neighbor is a supervised learning algorithm that is used widely in machine learning and data mining. It is a classifier algorithm where the learning is based on how a vector is similar to another. It does not compare the unclassified data with all the other data. It performs a mathematical calculation to measure the distance between the data to make the classification. The main distance calculations that are used in k-NN are Euclidean Distance and Manhattan Distance (Cover and Hart, 1967). In this research, we have used the Euclidean Distance between two points p and q with the following formula:

$$d(p, q) = \sqrt{\sum_{i=1}^n (q_i - p_i)^2} \quad (1)$$

The algorithm of KNN is as follows:

1. Receives an unclassified data;
2. Measures the distance (Euclidean, Manhattan, Minkowski or Weighted) from the new data to all other data that is already classified;
3. Sort the distances;
4. Gets the K smaller distances (nearest neighbors) where K is the number of neighbors that should be selected. The default value for K is 1;
5. Gather the cluster of the nearest neighbor;
6. Classifies the new data with the cluster that has been chosen in Step 5.

7 RESULTS

7.1 Environment

The experiments were run on Ubuntu 18.04 LTS with built-in GPUs. We used Deep Learning Virtual Machine provided by Microsoft Azure with SSD storage. The programming language that we used is Python because it supports many machine learning libraries such as TensorFlow, Keras, Anaconda, Jupyter, NumPy, matplotlib, SciPy, Pandas, and scikit learn. The usefulness of each library is as follows:

- TensorFlow is “an open-source software library for numerical computation using data flow graphs.” The graph nodes represent mathematical operations, while the graph edges represent the multidimensional data arrays (tensors) that flow between them (TensorFlow, 2020).
- Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano (Keras, 2020).
- Anaconda is a free and open-source distribution of the Python and R programming languages for scientific computing to simplify package management and deployment (Anaconda, 2020).
- Jupyter Project exists to develop open-source software, open-standards, and services for interactive computing across dozens of programming languages (Jupyter, 2020).
- NumPy, or Numerical Python, is the most universal and versatile library both for pros and beginners (NumPy, 2020).
- Matplotlib is a flexible library for creating graphs and visualization (Matplotlib, 2020).
- Pandas is a well-known and high-performance tool for presenting data frames (Pandas, 2020).

- Scikit Learn implements a wide-range of machine-learning algorithms and makes it comfortable to plug them into actual applications (kit Learn, 2020).

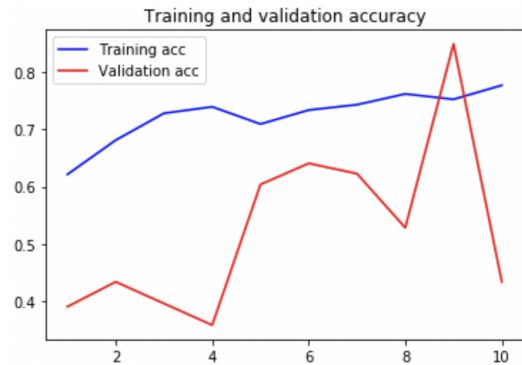


Figure 3: Training and Validation Accuracy vs. Epochs with 10 Epochs.

7.2 Architectures

In order to achieve high accuracy, we have built several architectures to detect malware using convolutional neural networks. We have changed various parameters such as the learning rate, batch size, and the number of epochs for each architecture. We have implemented different architecture with different layers and activation functions.

7.2.1 CNN A: 3c 2D

The architecture consists of:

1. Input layer $N \times N$ pixels ($N = 128$)
2. Convolutional Layer (32 filters of size 3×3)
3. Max Pooling layer
4. Convolutional Layer (64 filters of size 3×3)
5. Max Pooling layer
6. Convolutional Layer (128 filters of size 3×3)
7. Max Pooling layer
8. Flatten Layer
9. Densely-connected layer (64 neurons)
10. Densely-connected layer (1 neuron)

We have used *ReLU* activation function for all the layers except for the last one, we used *sigmoid*. We have fixed a learning rate of 0.01, batch size of 16, the number of epochs of 10, and a loss function to be binary cross-entropy. The accuracy that we achieved with this architecture is 84.9%. Fig. 3 describes the behavior of the model in terms of accuracy. In addition, Fig. 4 shows the training vs. validation loss.



Figure 4: Training and Validation Loss vs. Epochs with 10 Epochs.

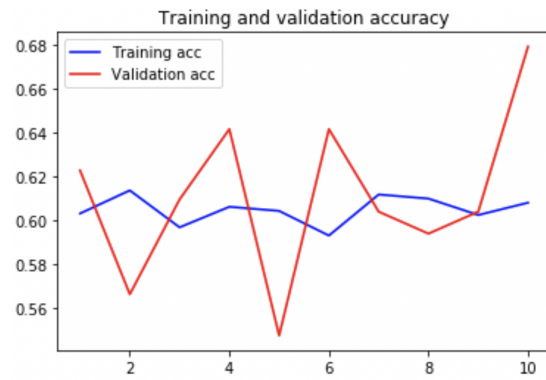


Figure 7: Training and Validation Accuracy vs. Epochs with 10 Epochs.

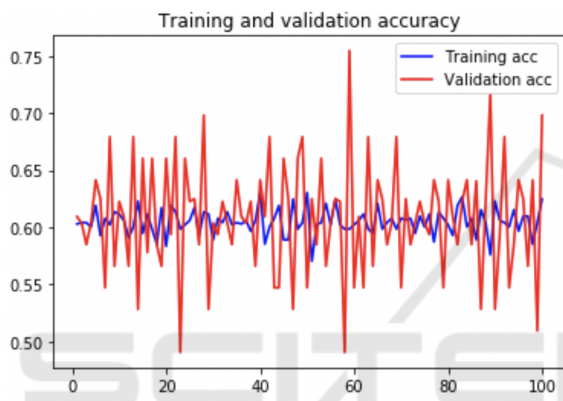


Figure 5: Training and Validation Accuracy vs. Epochs with 100 Epochs.



Figure 8: Training and Validation Loss vs. Epochs with 10 Epochs.



Figure 6: Training and Validation Loss vs. Epochs with 100 Epochs.

We have tried to experiment with the architecture by increasing the number of epochs to reach 100. The model kept its good detection performance. The graphs in Fig. 5 and Fig. 6 describe the accuracy and the loss of the model in 100 epochs.

7.2.2 CNN B: 2c 2D

The architecture consists of:

1. Input layer $N \times N$ pixels ($N = 128$)
2. Convolutional Layer (32 filters of size 3×3)
3. Max Pooling layer
4. Convolutional Layer (64 filters of size 3×3)
5. Max Pooling layer
6. Flatten Layer
7. Densely-connected layer (1 neuron)

We have used ReLU activation function for all the layers except for the last one, we used softmax. We have fixed a learning rate of 0.001, batch size of 16, number of epochs of 10, and a loss function to be binary cross-entropy. The accuracy that we achieved with this architecture is 68.1%. The graph in Fig. 7 describes the behavior of the model in terms of accuracy. In addition, the graph in Fig. 8 shows the training vs. validation loss.

We have tried to experiment with the architecture by increasing the number of epochs to reach 100.

The model kept its good detection performance. The graphs in Fig. 9 and Fig. 10 describes the accuracy and the loss of the model in 100 epochs:

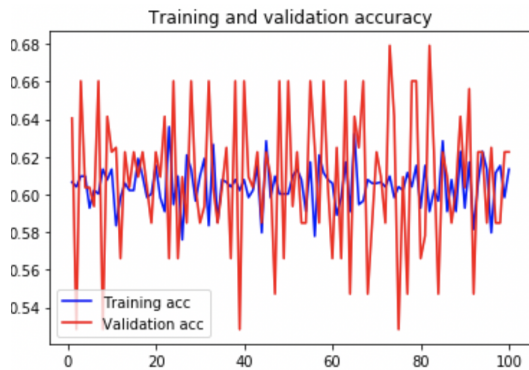


Figure 9: Training and Validation Accuracy vs. Epochs with 100 Epochs.

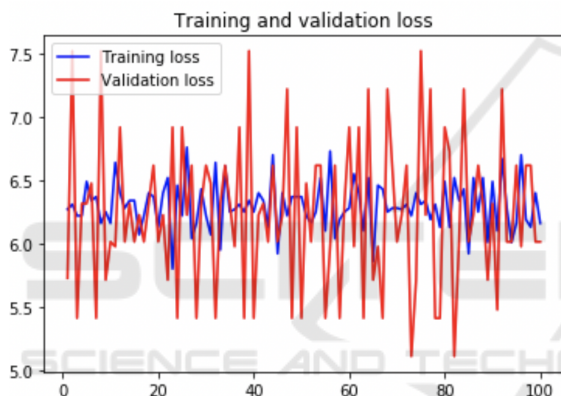


Figure 10: Training and Validation Loss vs. Epochs with 100 Epochs.

8 CONCLUSION AND FUTURE WORK

In this research, we focused on studying the feasibility of detecting and classifying mobile malware by treating them as images. It presents a static analysis approach using deep learning and convolutional neural networks. It is a different approach in mobile malware detection since the research focus is on how to detect and classify mobile malware based on their behavior. In addition, this work presented challenges of applying such approach on mobile applications since they are usually large files. We used the proposed approach to detect and classify malware from the binary files of Android applications. We used different architectures to test our approach and compared our results. We have achieved an accuracy of 84.9% using one architecture, and in another, we reached an accuracy of

68.1%. So, both architectures are useful for malware detection. Our model can detect variants of malware or another unknown malware based on the training data. This is an added value to overcome some of the problems facing signature-based detection systems.

Although we have 84.9% accuracy, we still have limitations and challenges. The first limitation is the dataset. Our model learned from previous sample malware. The dataset that we used had a limited number of sample malware. Also, because the majority of Android malware datasets and some malware families and benign applications are private, we had to collect our own data. The second limitation is the processing power. Processing images needs a variety of high-performance GPUs that are not available at the university lab.

This work is an attempt to make an advanced malware detection system. In the future, we plan to increase the accuracy of our system and reduce the number of false positives by using new image feature extraction and new computer vision techniques that can help in the problem of malware classification. In addition, we plan to introduce multi-level classification instead of binary classification to precisely determine the type of the family of malware.

ACKNOWLEDGEMENTS

We would like to thank all fellow students, faculty, and staff for supporting this research. Special thanks to Mr. Saad Taame for his help in data pre-processing.

REFERENCES

- Ahmadi, M., Ulyanov, D., Semenov, S., Trofimov, M., and Giacinto, G. (2016). Novel feature extraction, selection and fusion for effective malware family classification. In *Proceedings of the sixth ACM conference on data and application security and privacy*, pages 183–194.
- Anaconda (2020).
- Cover, T. and Hart, P. (1967). Nearest neighbor pattern classification. *IEEE transactions on information theory*, 13(1):21–27.
- Douze, M., Jégou, H., Sandhawalia, H., Amsaleg, L., and Schmid, C. (2009). Evaluation of gist descriptors for web-scale image search. In *Proceedings of the ACM International Conference on Image and Video Retrieval*, pages 1–8.
- Jupyter, P. (2020).
- Keras (2020).
- kit Learn, S. (2020).

- Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105.
- Lab., K. (2019). Kaspersky lab detected more than 6 million malware package.
- Lashkari, A. H., Kadir, A. F. A., Taheri, L., and Ghorbani, A. A. (2018). Toward developing a systematic approach to generate benchmark android malware datasets and classification. In *2018 International Carnahan Conference on Security Technology (ICCSST)*, pages 1–7. IEEE.
- Li, Y., Ma, R., and Jiao, R. (2015). A hybrid malicious code detection method based on deep learning. *International Journal of Security and Its Applications*, 9(5):205–216.
- Matplotlib (2020).
- Michie, D., Spiegelhalter, D. J., Taylor, C., et al. (1994). Machine learning. *Neural and Statistical Classification*, 13(1994):1–298.
- Narudin, F. A., Feizollah, A., Anuar, N. B., and Gani, A. (2016). Evaluation of machine learning classifiers for mobile malware detection. *Soft Computing*, 20(1):343–357.
- NumPy (2020).
- Pandas (2020).
- Saxe, J. and Berlin, K. (2015). Deep neural network based malware detection using two dimensional binary program features. In *2015 10th International Conference on Malicious and Unwanted Software (MALWARE)*, pages 11–20. IEEE.
- Schmidt, A.-D., Bye, R., Schmidt, H.-G., Clausen, J., Kiraz, O., Yuksel, K. A., Camtepe, S. A., and Albayrak, S. (2009). Static analysis of executables for collaborative malware detection on android. In *2009 IEEE International Conference on Communications*, pages 1–5. IEEE.
- Statista (2020). Number of smartphone users worldwide from 2014 to 2020 (in billions).
- TensorFlow (2020).
- Tong, F. and Yan, Z. (2017). A hybrid approach of mobile malware detection in android. *Journal of Parallel and Distributed computing*, 103:22–31.
- VirusTotal (2020). Analyze suspicious files and urls to detect types of malware, automatically share them with the security community).
- Yajamanam, S., Selvin, V. R. S., Di Troia, F., and Stamp, M. (2018). Deep learning versus gist descriptors for image-based malware classification. In *ICISSP*, pages 553–561.
- Zhao, Z. and Liu, H. (2007). Spectral feature selection for supervised and unsupervised learning. In *Proceedings of the 24th international conference on Machine learning*, pages 1151–1157.