Android Malware Detection Method Based on App-Image Conversion

Nannan Xie[1], Hongpeng Bai[1(✉)], Yanfeng Shi[2], and Haiwei Wu[1]

[1] Changchun University of Science and Technology, Changchun 130022, China
2018100597@mails.cust.edu.cn

[2] School of Computer Engineering, Nanjing Institute of Technology, Nanjing 211167, China

Abstract. With the rapid development of mobile internet, Android has become the most widely used mobile terminal operating system and play an increasingly important role in users' lives. However, Android malware is also bringing privacy leaks and security threats that are causing troubles to third-party markets and users. What's more, malware uses code obfuscation and camouflage to hide itself to avoid detection. Traditional malware detection techniques based on machine learning and feature matching are usually difficult to deal with this type of malware. Considering about this problem, an Android malware detection method based on app-image conversion is proposed, which maps the Android installation files to grayscale images, and employs the deep learning algorithm, CNN (Convolutional Neural Networks), for malware detection. A detection framework for Android malware is presented, which includes three parts: data set construction, app-image conversion, and deep learning detection. In the experiments, the parameters of CNN are determined through comparative analysis. It achieves the detection accuracy of 95.23%, which shows the effectiveness and feasibility of the proposed method.

Keywords: Android malware · Grayscale image · Deep learning · Convolutional Neural Networks

1 Introduction

The openness and freedom of the Android operating system allow developers all over the world to conduct secondary development and upload their own applications to the application markets. These features have promoted the rapid growth of Android applications, but they have also led to the spread of malicious applications. For example, the Android malware family DroidKungFu [6], which appeared in 2011, is a typical Trojan horse with many variants. It opens the system back door to remotely access the infected mobile phone and uses the system vulnerability to root the system. The common functions of DroidKungFu include to execute file delete commands, execute web page opening commands, download other apks, open URLs, and start other programs. These malicious behaviors have caused serious economic losses and privacy leaks.

Y. Tian et al. (Eds.): ICBDS 2020, CCIS 1415, pp. 60–74, 2021.
https://doi.org/10.1007/978-981-16-3150-4_6

Smartphones with Android operating system accounted for 87% of global smartphone sales in 2019 [10]. Malware is becoming an important threat to privacy protection and network security. Kaspersky pointed out in the analysis report on the evolution of mobile malware [11], attacks on personal data have become frequent in 2019 with more Trojan horse attacks. From 2018 to 2019, the number of attacks of personal data on mobile devices increased by 50%. Public security incidents caused by mobile terminal security issues are appearing from time to time. For example, the positioning function leaks mobile phone location information, applications collect user information without notification, free wireless connection stealing device data, which resulting in the leakage of sensitive information and have caused serious security risks to social public safety and individuals' privacy. Therefore, how to detect malicious software efficiently and deal with malicious behaviors timely are still the problems that need to be solved urgently.

In the academic and industrial practice, the current Android malware detection methods mainly include rule matching and active detection. Rule matching detects the maliciousness of an application by matching its signatures with the rule base, and the rule base is maintained and updated on time. This method detects quickly but is difficult to detect new emerging malware. The active detection has high detection accuracy and can deal with unprecedented attacks by using unsupervised algorithms. Static or dynamic features are extracted, such as permissions, API calls, component information, and machine learning or other algorithms are employed to classify the malware and benign applications. There are many existing Android malware detection techniques and detection engines. However, due to the limitations of technology and regional cultures, the relevant researchers and security companies of malware detection have difficulty to share the detection results. Therefore, it is necessary to study fast and efficient detection methods.

CNN is the most widely used deep learning method with a wide range of applications in image processing since 2010. Its key concepts are "local receptive fields" and "shared weights and biases". The CNN algorithm is based on the correlation between local pixels, so the local receptive fields can be used to reduce the processed feature dimensionality and build high-level features, which make CNN more suitable for processing related features than independent features.

In this work, we propose an Android malware detection framework based on app-image conversion. The installation apk files of Android applications are converted into grayscale images in binary form, and then we employ CNN to classify the images to detect malicious applications. The theoretical basis of this method is that malware and normal applications have different behaviors in different classes while have similarities in the same class, and the grayscale images can reflect the difference between malware and normal applications to a certain extent. In addition, the detection method based on image classification try to solve the problem of malicious code deformation and confusion caused by code obfuscation and camouflage.

2 Related Work

2.1 Android Malware and Detection

As the most widely used mobile terminal system, in order to improve the security and deal with the problems caused by malware attacks, Android itself has three security mechanisms: program sandbox, software signature, and permission management. However, malware can bypass these security mechanisms to a certain extent by increasing code obfuscation, encrypting malicious payloads, or conducting secret commands communicates with remote servers, which increasing the difficulty of malware detection.

Android malware refers to applications developed for malicious purpose based on Android and circulating in the application markets. The malicious behaviors mainly include the following 6 types.

(1) Malicious deduction. It refers to the automatic deduction of fees without permission. For example, some applications automatically purchase paid services without notifying the user and deduct the fees from the user's phone account.
(2) Privacy theft. The application contains code to collect sensitive data, such as call records, contact information, location information, even account passwords, and upload them to a remote server or analyze them.
(3) Traffic data consumption. The application opens network download services in the background, which consumes data traffic and causes direct economic losses.
(4) Remote control. This kind of malicious application remotely controls the device and conducts remote operations, such as calling the camera to take pictures and upload them to a remote server.
(5) System damage. These applications delete important system files and maliciously occupy system resources, such as CPU computing resources or memory resources, which cause the system to fail to operate normally.
(6) Other behaviors. There are some other malicious behaviors, and new malware has emerged over time. For example, some applications bundle the installation of applications or automatically perform certain or illegal operations, such as playing audio. Although they do not directly damage the system, they cause confusion to users.

Android malware detection technologies mainly include static detection and dynamic detection. Static detection constructs the feature set by extracting features from system files or sentence sequences without running the source code, and use feature matching or machine learning methods to detect the maliciousness [23]. The advantages of the static detection are that it can cover the entire code and is suitable for large-scale detection, but the disadvantage is that the real operating environment cannot be simulated, so it is difficult to detect some specific behaviors. There are many studies on static methods, such as malware detection with permission features [7], behavior analysis with opcode [24], and detection with combined features of permissions and API calls [9].

Dynamic detection methods run the Android application in a sandbox or virtual environment and judge whether the application is malicious by monitoring the behaviors. This method is more practical and has a higher detection accuracy since it can simulate real running environment. Because techniques such as repacking, obfuscation, and encryption will not change the actual running behaviors of the application, dynamic method can effectively deal with the variants of malicious applications. However, dynamic detection has the disadvantage of large resource consumption. Since all working paths and functions of the application must be executed, the operation of the simulation program requires a lot of time and space.

A typical dynamic detection research is TaintDroid, which is a dynamic tool based on the detection of sensitive data flow paths in a sandbox environment [5]. By analyzing user-related activities to determine whether the user operation is an active behavior or passive one, RansomeProber [3] detected malicious behaviors in real time. In addition, Cai et al. proposed Droidcat [1], which detected the behavior of malware from the app-level, and used a combination of multiple dynamic features based on method calls and communication intentions between components to determine the maliciousness.

In addition to be used in feature matching, the features extracted by static and dynamic methods can be processed by machine learning algorithms. Machine learning algorithms, especially which based on probability statics and neural networks, are the most widely used detection methods. Probabilistic models represented by Naive Bayes have many practical applications, such as risk assessment of Android software through probability generation models [22], and construct probability discriminant models of permission features for malware detection [21].

Statistical machine learning algorithms represented by SVM and KNN are widely used because of their high accuracy and mature development. For example, Mu et al. [26] used SVM as a classifier to analyze malware risks introduced by permission features. Since a single algorithm has disadvantages, there are also researches that combine different algorithms to improve the detection result [15]. Probabilistic statistical methods have some limitations, for example, high dimensional features will take more running time and computational space. Therefore, some research combined feature selection, feature extraction or other feature dimensionality reduction methods with classification techniques, in order to achieve the balance of detection effect and detection consumption. In recent years, deep learning methods have made good attempts in this field. The correlations between Android applications are the theoretical basis for using deep learning in malware detection. Hao et al. [17] used CNN to process the extracted opcodes, while Lei et al. [2] employed Deep Belief Network to analyze multiple kinds of features.

Due to the openness of the Android system, any organizations and individuals can carry out secondary development. Malware developers may evade the detection by repackaging, resigning, and obfuscating the codes. At present, the researchers are taking attentions to the new techniques, such as knowledge

graphs, artificial intelligence, to improve detection performance. In the industry, with the continuous emergence of new malware, it is important to develop detection systems with high accuracy and low consumption.

2.2 Grayscale Image and Android Malware

Grayscale image has only one sampled color per pixel and displays from the darkest black to the brightest white, with multiple levels of color depth between black and white. A complete image is usually composed of three channels of red, green, and blue, with different gray scales, which can be used to express the proportion of the three colors in the image. Grayscale image reduces the amount of original image information, but it still retains important features of the original image for processing.

There are studies attempt to combine Android malware detection with visualization, such as Android malware binary file analysis based on visualization methods. The theoretical basis of these research is that most variants of malware are generated by using automated techniques or reusing some important modules, so they have similarities in the binary codes.

In 2011, NATARAJ et al. [16] proposed a method to visualize and classify malware by image processing. Later, researchers tried different methods to visualize and detect malware, which include the method that transformed malware into images by direct conversion and opcode conversion [27]. Direct conversion is to convert the PE file of the malware into binary codes, and then generate the malware image from the binary codes, which are converted into grayscale and RGB. Opcode conversion decompiles the installation file to obtain the operation codes at first, and then the operation codes are selected and converted into a malware image. Fu et al. [8] proposed a method to convert malware into RGB color images to solve the problem of insufficient information in gray images. Zhang et al. [25] converted the opcodes into an image and achieved a better malware detection effect. Similarly to traditional malware detection, Android malicious behaviors have certain regularity, which can be distinguished from the perspective of images during the visualization process.

2.3 Convolutional Neural Networks

CNN is a deep learning algorithm with a multi-layer structure, and its related research was proposed in 1989 [12]. In the 1980s and 1990s, some researchers published related work on CNN and achieved good recognition results in several pattern recognitions and handwritten digit recognitions [13]. Utile 2012, Krizhevsky et al. used the extended depth CNN and achieved the best classification effect in the ImageNet Large Scale Visual Recognition Challenge, which make CNN to attract growing attentions. In addition, many other algorithms have emerged in deep learning, including denoising autoencoders [19], DCN [4], and sumproduct [18].

CNN is composed of input layer, convolutional layer, pooling layer, and fully connected layer. It is based on multilayer neural network structure and constructs

high level abstract features by learning the original features. CNN extracts features from the input data layer by layer. After the data being operated by convolution, pooling, activation function mapping, it finally realizes the classification. The 4 layers are described as following.

(1) Convolutional layer. Through the convolution operation, the convolution kernel is used to extract the features in the local area. It uses a sliding window to extract features from the image to obtain the local information of the overall data. The size of the convolution kernel is the main parameter in this layer.
(2) Pooling layer. It performs dimensionality reduction of the up layer, and it controls the over-fitting of the model to a certain extent and optimizes the entire network. The pooling window size is the main parameter.
(3) Fully connected layer. This layer maps the features learned by the model to the classification space. In image processing, the neurons of the input are connected to the neurons of the output, and the input is the stitching of the feature maps.
(4) Activation function. The features of activated neurons are preserved and mapped out to enhance the nonlinear characteristics of convolutional neural networks. The common activation functions include Sigmoid and ReLU.

One of the characteristics of CNN is the introduction of "local perception" and "weight sharing" concepts. Local perception is a learning method that extends from local feature learning to global learning, which divides the image into several regions. Since the local features of the image are relatively stable, the overall information can be obtained after the local information is summarized. Thus, the number of connections between neurons can be greatly reduced, in other words, the model parameters are reduced. Weight sharing means to that all convolution kernels in the same layer use the same convolution kernel weights, with the purpose to effectively reduce the number of parameters required in the model training. The basic process of CNN is shown in Fig. 1.

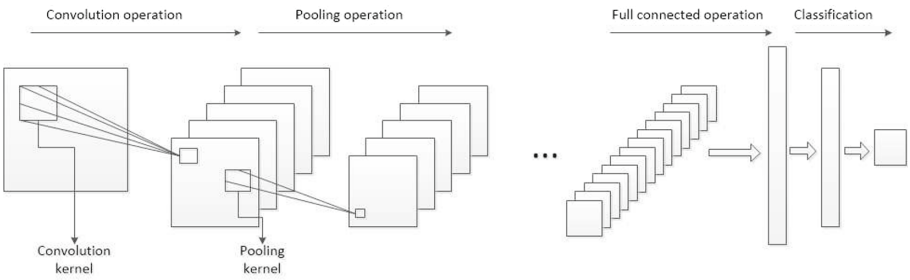

Fig. 1. Convolution Neural Networks structure.

The applications of CNN in the network security have gradually increased in recent years. Wang et al. [20] presented a hybrid Android malware detection

model based on deep autoencoder and CNN, which effectively reduced the running time of the algorithm while improving the detection accuracy. Li et al. [14] proposed an Android malware detection system, which employed optimized deep Convolutional Neural Network to learn from opcode sequences. CNN is mainly employed in Android malware detection to process the extracted static or dynamic features. Considering that CNN has advantages in processing images, we employs CNN to process the converted grayscale images to achieve the malware detection in this work.

3 Malware Detection Based on App-Image Conversion

3.1 Malware Detection Framework

The proposed Android malware detection method is to convert binary Android application installation files into grayscale images, and then classify them by CNN. The detection framework is in Fig. 2, which includes data set construction, app-image conversion, and CNN detection.

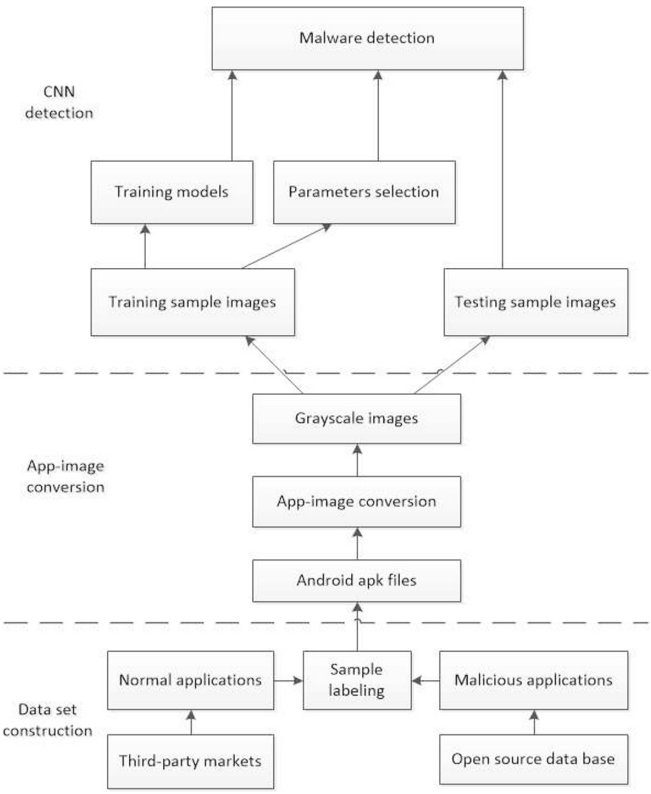

Fig. 2. Malware detection framework.

(1) Data set construction. The data set includes malicious samples and normal samples. The normal samples are collected from third-party markets, and the malicious samples come from public virus databases. The constructed data set is used to train and test CNN. In order to ensure the label purity and classification accuracy, the applications obtained are scanned by VirusTotal, the open source malware detection tool, to label the samples as accurate as possible.
(2) App-image conversion. The main work of this step is to convert the collected normal samples and malicious samples to grayscale images. At first, the apk file is read as a binary file, and then the binary file is mapped to a grayscale image. The pixels in the image represent the original Android sample.
(3) Malware detection. The Android application data set constructed above is divided into training samples and testing samples. The converted images are used to train the CNN model and test the classification results. In this process, the parameters of the model are selected, and we use the test samples to evaluate the malware detection results.

3.2 Grayscale Image Conversion

The process of grayscale image conversion adopts the function in the OpenCV library, which is an open source computer vision library that contains hundreds of computer vision algorithms of image processing. The steps of conversion are as following.

(1) The apk installation file of the Android application is read in a binary format. Every 8 bits are treated as an unsigned integer.
(2) By setting a fixed line width in advance, the binary file is converted into a two-dimensional array, and the value range of each element in the array is in [0,255].
(3) In the single channel image processing, "0" represents black and "255" represents white. The value in the array is used as the value of each pixel to generate a grayscale image. A part of the generated grayscale image is shown in Fig. 3.

4 Experiments

4.1 Environment and Data Set

The experimental part discusses the three parameters of CNN: batch size, convolution kernel size, and the number of hidden layers, and finally achieves malware detection according to the selected parameters.

The configurations used in this experiment are: Linux system, Intel Core Processor 2.4 GHz CPU, 4 GB memory, 80G hard disk, and the deep learning framework is TensorFlow-cpu 1.7.0. The data set contains 533 malware and 561

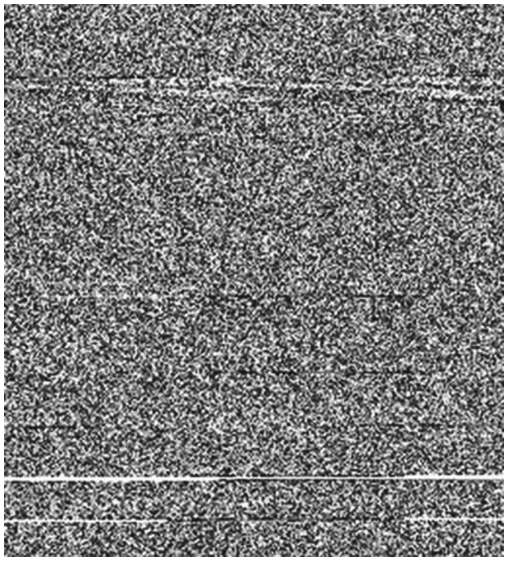

Fig. 3. Grayscale image generated by binary file (partial).

normal applications with a total of 1094 samples. The following experiments adopt five-fold cross validation.

We use accuracy to evaluate the classification results. Set P denotes the number of positive samples and N denotes the negative samples. Define the following 4 parameters: (1) TP (True Positives): the number of samples that are correctly divided into positive samples. (2) FP (False Positives): the number of samples that are wrongly divided into positive samples. (3) FN (False Negatives): the number of samples that are wrongly divided into negative samples. (4) TN (True Negatives): the number of samples that are correctly divided into negative samples.

The accuracy is defined as: Accuracy $=(\mathrm{TP}+\mathrm{TN})/(\mathrm{P}+\mathrm{N})$. The higher the accuracy, the better the classification effect.

4.2 Parameters of CNN

CNN needs several parameters during its execution. These parameters are related to the actual data set and operating environment. In practical, there is no effective and unified method for specific data set and environment, and experiments are usually needed to find proper parameters. Three parameters are mainly considered in the following experiments: batch size, convolution kernel size, and hidden layer.

(1) **Batch size**

Batch size is an important learning parameter in machine learning. In the gradient algorithm, increasing batch size within a reasonable range can improve

memory utilization. It reduces the algorithm iterations, speeds up the processing of the data, and can determine the directions of decline. But the choice of batch size is not as large as possible. When it is too large, because the number of iterations required is too small, the time spend will increase greatly. Reasonable batch size selection is related to the format and quantity of data, and it is usually determined by expert experience.

In this experiment, the values of batch size are set to 10, 20, 30, 40, and the numbers of training iteration are set to 500, 1000, 2000. The experimental results are shown in Table 1 and Fig. 4.

Table 1. Training accuracy of different batch sizes

Group No.	Batch size	500 iterations	1000 iterations	2000 iterations
1	10	59.90%	54.53%	58.99%
2	20	55.00%	59.50%	79.00%
3	30	56.74%	59.97%	82.03%
4	40	56.49%	60.04%	82.60%
5	50	56.38%	59.99%	82.58%

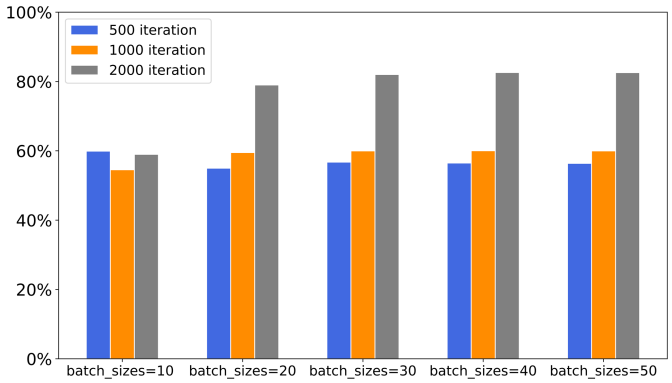

Fig. 4. Training accuracy of different batch sizes.

It can be seen from Fig. 4 that when the batch size is small, it may be insufficient training and is difficult to achieve the satisfied effect. With the batch size increasing, the improvement of accuracy is slowing down. When training 2000 iterations, batch size equals to 30 and 40, the training accuracy is 82.03% and 82.60%, and the test accuracy is 83.10% and 83.83%. When the value is 50, the accuracy drops slightly. Therefore, in the subsequent experiments, batch size is selected as 40, and the iteration is set to 2000.

(2) **Convolution kernel size**

The size of the convolution kernel is another key parameter of CNN. Set 5 groups of convolution kernels: 3 * 3, 5 * 5, 9 * 9, 11 * 11, 15 * 15, and the training accuracy and test accuracy are shown in Table 2 and Fig. 5.

Table 2. Accuracy of different convolution kernel sizes

Group No.	Kernel size	Training accuracy	Test accuracy
1	3 * 3	80.56%	55.23%
2	5 * 5	83.25%	55.40%
3	9 * 9	83.00%	63.12%
4	11 * 11	86.99%	94.88%
5	15 * 15	88.50%	44.30%

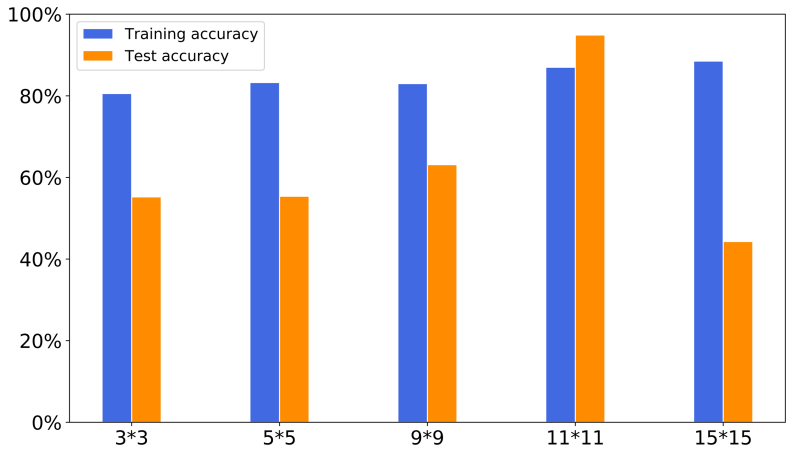

Fig. 5. Accuracy of different convolution kernel sizes.

From Table 2, we can see that as the convolution kernel size increases, the accuracy of training is gradually increasing, and the accuracy of testing also shows the same trend. However, when the convolution kernel is 15*15, the test accuracy is significantly reduce. Therefore, for the data set of this experiment, the convolution kernel of 11*11 achieves relatively best result.

(3) **Hidden layers**

A typical characteristic of deep learning is the multi-layer neural network, which can provide high level abstraction of data features. A hidden layer includes a convolutional layer and a pooling layer as in Fig. 1. The following 5 groups of

experiments are set with the hidden layers of 1,2,3,4,5, and with the same of pooling layers. "1c-1p" refers to "1 convolution layer and 1 pooling layer". The experimental results are shown in Table 3 and Fig. 6.

Table 3. Accuracy of different hidden layers

Group No.	Hidden layers	Training accuracy	Test accuracy
1	1c-1p	85.02%	61.87%
2	2c-2p	86.99%	84.17%
3	3c-3p	87.54%	94.88%
4	4c-4p	87.62%	95.12%
5	5c-5p	87.68%	95.23%

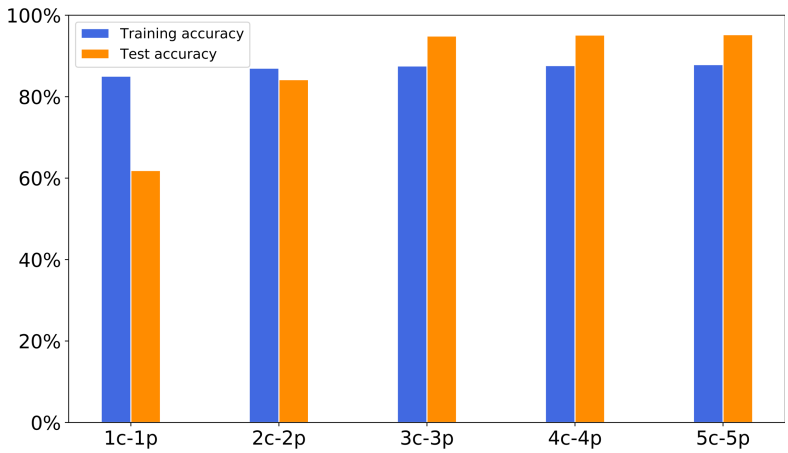

Fig. 6. Accuracy of different hidden layers.

With the number of hidden layers increasing, the training accuracy and test accuracy are both rising. However, when the number of hidden layers increases from 3 to 5, the accuracy increasing tends to be flat, reaching a relatively similar accuracy. When the hidden layer is 5, the highest test accuracy of 95.23% is achieved. However, as the hidden layer increases, the time consumption is increasing significantly. Therefore, the accuracy and time consumption need to be balanced in practical application.

5 Conclusion

Android malware detection is not only for the safety of the third-party application markets, but also for the safety of individual users who download and install the applications. With the continuous emergence of new malicious applications and the continuous updating of malicious code camouflage and obfuscation, Android malware detection is the main technology to ensure security.

An Android malware detection framework based on app-image conversion is presented in this work. It focuses on mapping the installation files of Android applications to grayscale images in binary form, and then classify the images by CNN. This method can solve the problem of malicious code deformation and confusion to a certain extent.

The experiments compare and select three parameters of CNN: batch size, convolution kernel size, and the number of hidden layers. By the constructed data set, the proposed method finally achieves a classification accuracy of 95.23%, which shows the effectiveness and feasibility of the presented method. In the future work, we will apply the proposed method to specific types of malware detection, especially malware with disguise and confusion, to verify the effectiveness and scalability if the method.

Acknowledgments. This work was supported in part by the 13th Five-Year Science and Technology Research Project of the Education Department of Jilin Province under Grant No. JJKH20200794KJ, the Innovation Fund of Changchun University of Science and Technology under Grant No. XJJLG-2018-09, the fund of Key Laboratory of Symbolic Computation and Knowledge Engineering of Ministry of Education (Jilin University) under Grant No. 93K172018K05.

References

1. Cai, H., Meng, N., Ryder, B., Yao, D.: Droidcat: effective android malware detection and categorization via app-level profiling. IEEE Trans. Inf. Foren. Secur. **14**(6), 1455–1470 (2018)
2. Cen, L., Gates, C.S., Si, L., Li, N.: A probabilistic discriminative model for android malware detection with decompiled source code. IEEE Trans. Dependable Secur. Comput. **12**(4), 400–412 (2014)
3. Chen, J., Wang, C., Zhao, Z., Chen, K., Du, R., Ahn, G.J.: Uncovering the face of android Ransomware: characterization and real-time detection. IEEE Trans. Inf. Foren. Secur. **13**(5), 1286–1300 (2017)
4. Deng, L., Yu, D.: Deep convex net: A scalable architecture for speech pattern classification. In: Twelfth Annual Conference of the International Speech Communication Association (2011)
5. Enck, W., et al.: Taintdroid: an information-flow tracking system for realtime privacy monitoring on smartphones. ACM Trans. Comput. Syst. (TOCS) **32**(2), 1–29 (2014)
6. F-Secure: Trojan: andriod/droidkungfu.c. [EB/OL] (2020). https://www.f-secure.com/vdescs/trojan_android_droidkungfu_c.shtml
7. Fang, Z., Permission based android security: Permission based Android security: issues and countermeasures. Comput. Secur. **43**, 205–218 (2014)

8. Fu, J., Xue, J., Wang, Y., Liu, Z., Shan, C.: Malware visualization for fine-grained classification. IEEE Access **6**, 14510–14523 (2018)
9. Hou, S., Ye, Y., Song, Y., Abdulhayoglu, M.: Hindroid: an intelligent android malware detection system based on structured heterogeneous information network. In: Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 1507–1515 (2017)
10. IDC: Smartphone challenges continue in 2019. [EB/OL] (2019). https://www.idc.com/getdoc.jsp?containerId=prUS45487719
11. Kaspersky: Mobile malware evolution 2019. [EB/OL] (2020). https://securelist.com/mobile-malware-evolution-2019/96280
12. LeCun, Y., et al.: Backpropagation applied to handwritten zip code recognition. Neural Comput. **1**(4), 541–551 (1989)
13. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proc. IEEE **86**(11), 2278–2324 (1998)
14. Li, D., Zhao, L., Cheng, Q., Lu, N., Shi, W.: Opcode sequence analysis of android malware by a convolutional neural network. Concurr. Comput. Pract. Exp. **32**(18), e5308 (2020)
15. McLaughlin, N., et al.: Deep android malware detection. In: Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy, pp. 301–308 (2017)
16. Nataraj, L., Karthikeyan, S., Jacob, G., Manjunath, B.S.: Malware images: visualization and automatic classification. In: Proceedings of the 8th International Symposium on Visualization for Cyber Security, pp. 1–7 (2011)
17. Peng, H., et al.: Using probabilistic generative models for ranking risks of android apps. In: Proceedings of the 2012 ACM Conference on Computer and Communications Security, pp. 241–252 (2012)
18. Poon, H., Domingos, P.: Sum-product networks: a new deep architecture. In: 2011 IEEE International Conference on Computer Vision Workshops (ICCV Workshops), pp. 689–690. IEEE (2011)
19. Vincent, P., Larochelle, H., Lajoie, I., Bengio, Y., Manzagol, P.A., Bottou, L.: Stacked denoising autoencoders: learning useful representations in a deep network with a local denoising criterion. J. Mach. Learn. Res. **11**(12), 3371–3408 (2010)
20. Wang, W., Zhao, M., Wang, J.: Effective android malware detection with a hybrid model based on deep autoencoder and convolutional neural network. J. Ambient Intell. Humaniz. Comput. **10**(8), 3035–3043 (2018). https://doi.org/10.1007/s12652-018-0803-6
21. Wei, F., Roy, S., Ou, X.: Amandroid: a precise and general inter-component data flow analysis framework for security vetting of android apps. In: Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, pp. 1329–1341 (2014)
22. Xu, K., Li, Y., Deng, R.H.: ICCDetector: ICC-based malware detection on android. IEEE Trans. Inf. Foren. Secur. **11**(6), 1252–1264 (2016)
23. Zhandi, W.: Research and application of Android malware detection based on deep learning. Guizhou Normal University (2019)
24. Zhang, H., Xiao, X., Mercaldo, F., Ni, S., Martinelli, F., Sangaiah, A.K.: Classification of Ransomware families with machine learning based on n-gram of opcodes. Fut. Gener. Comput. Syst. **90**, 211–221 (2019)
25. Zhang, J., Qin, Z., Yin, H., Ou, L., Hu, Y.: IRMD: malware variant detection using opcode image recognition. In: 2016 IEEE 22nd International Conference on Parallel and Distributed Systems (ICPADS), pp. 1175–1180. IEEE (2016)

26. Zhang, M., Duan, Y., Yin, H., Zhao, Z.: Semantics-aware android malware classification using weighted contextual api dependency graphs. In: Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, pp. 1105–1116 (2014)
27. Zhang, J., Chen, B., Gu, L.: Research on malware detection technology based on image analysis. Netinfo. Secur. **19**(10), 24–31 (2019)