Computers, Materials & Continua

Tech Science Press

DOI: 10.32604/cmc.2022.028680

ARTICLE

Detecting Android Botnet Applications Using Convolution Neural Network

Mamona Arshad[1], Ahmad Karim[1], Salman Naseer[2], Shafiq Ahmad[3], Mejdal Alqahtani[3], Akber Abid Gardezi[4], Muhammad Shafiq[5,*] and Jin-Ghoo Choi[5]

[1]Department of Information Technology, Bahauddin Zakariya University, Multan, 60000, Pakistan

[2]Department of Information Technology, University of the Punjab Gujranwala Campus, Gujranwala, 52250, Pakistan

[3]Industrial Engineering Department, College of Engineering, King Saud University, P.O. Box 800, Riyadh, 11421, Saudi Arabia

[4]Department of Computer Science, COMSATS University Islamabad, Islamabad, 45550, Pakistan

[5]Department of Information and Communication Engineering, Yeungnam University, Gyeongsan, 38541, Korea

*Corresponding Author: Muhammad Shafiq. Email: shafiq@ynu.ac.kr

Received: 15 February 2022 Accepted: 05 July 2022 Published: 29 November 2023

ABSTRACT

The exponential growth in the development of smartphones and handheld devices is permeated due to everyday activities i.e., games applications, entertainment, online banking, social network sites, etc., and also allow the end users to perform a variety of activities. Because of activities, mobile devices attract cybercriminals to initiate an attack over a diverse range of malicious activities such as theft of unauthorized information, phishing, spamming, Distributed Denial of Services (DDoS), and malware dissemination. Botnet applications are a type of harmful attack that can be used to launch malicious activities and has become a significant threat in the research area. A botnet is a collection of infected devices that are managed by a botmaster and communicate with each other via a command server in order to carry out malicious attacks. With the rise in malicious attacks, detecting botnet applications has become more challenging. Therefore, it is essential to investigate mobile botnet attacks to uncover the security issues in severe financial and ethical damages caused by a massive coordinated command server. Current state of the art, various solutions were provided for the detection of botnet applications, but in general, the researchers suffer various techniques of machine learning-based methods with static features which are usually ineffective when obfuscation techniques are used for the detection of botnet applications. In this paper, we propose an approach by exploring the concept of a deep learning-based method and present a well-defined Convolutional Neural Network (CNN) model. Using the visualization approach, we obtain the colored images through byte code files of applications and perform an experiment. For analysis of the results of an experiment, we differentiate the performance of the model from other existing research studies. Furthermore, our method outperforms with 94.34% accuracy, 92.9% of precision, and 92% of recall.

KEYWORDS

CNN; botnet applications; machine learning; image processing

1 Introduction

Nowadays, Android smartphones have become a convenient and ubiquitous part of our lives, are widely used by the majority of end users for everyday activities, business management, data sharing, social sites, and online banking. As compared to other Personal Computer (PC) platforms, smartphone devices are stored more sensitive information because of the full-featured operating system incorporated with a user-friendly environment that helps the end users for installing applications for everyday activities from unverified sources such as third-party Appstore. Therefore, cybercriminals can exploit the weakness of smartphone devices by installing the applications and embedding malicious codes into the mobile clean applications.

The most common technique to embed malicious codes into mobile clean applications is the Android permission system. Every android application needs specifically ask for permission from end-users to install in order to perform operations on the devices, such as sending an Short Messaging Service (SMS)/Multimedia Messaging Service (MMS) [1]. Unfortunately, many end users provide access to unknown mobile applications without considering the internal code of applications. As a consequence, cybercriminals employ the reverse engineering technique on apps, disassembling the apps, embedding malicious code into the original code, and then uploading the updated version to the android market. Using this concept, end users can be easy misleads because they are unable to recognize the difference between malicious and clean applications. Once malicious applications have been installed, the malicious code within applications can perform attacks in the background. Botnet applications are a type of such attacks in which the cybercriminals exploit a security hole in the platform i.e., Command and Control server (C&C) in order to command, inspect the applications and install malware, worms, trojans, threats, and botnets [2] on the android devices that can be used to launch malicious activities to the targets.

Botnets are a type of malware that uses a C&C channel to compromise a network of interconnected devices. Attackers or cybercriminals are known as botmasters that control the command-and-control channel to send, update, and obtain information from end-users [3]. In addition, such types of attacks can infect Android smartphones and turns into harmful bots. These malicious bots turn into a large botnet. The objective of our paper is the detection of executing malicious activities in devices through C&C servers that are arranged by botmasters [4].

From the security perspective, the exponential growth in malicious activities is a significant threat in educational and industry research. Botnets have their own set of characteristics and capabilities. The botnets are usually divided into subgroups such as botmasters, bot clients, and bot servers. The botmaster is incharge of the botnet's controller and operator known as malware management and launch an attack through interaction with the network of the command-and-control channel (C&C network) in order to establish a communication channel with the system as sown in Fig. 1.

Moreover, a C&C network is a large server with a lot of resources such as Central Processing Unit (CPU), memory, and bandwidth responsible for delivering commands to the victims for configuration and updating purposes. Furthermore, the victims or service provider has no idea what malware has been installed on it or if it is part of the botnet. The attackers infect the targets' systems in order to carry out malicious operations such as data breaches, remote monitoring, online or mobile banking, click fraud, hacking, malware dissemination, spam, phishing, and unauthorized information [5].

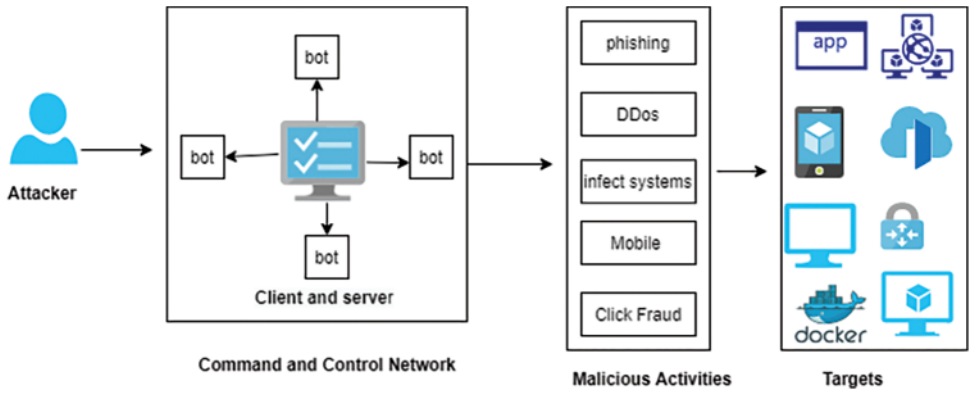

Figure 1: Workflow of communication channel

Therefore, above all explanation, there is a critical situation to develop innovative methods for the detection of botnet applications. In recent years, the detection of mobile botnets is challanging for researchers because of feature representations, extraction or by dissembling the program codes through the static [6,7] and dynamic analysis [8]. To solve this issue, we present an image-based approach through visualization representation methods for android botnet application detection in our paper.

The following are the main contributions of this paper:

- This paper proposed an effective approach for detecting botnet applications in android devices by learning feature maps extraction through the Convolutional Neural Network (CNN) model.
- The most challenging task in CNN is to transform the android applications into images. For this task, we use the feature representations method in the form of colored images using the visualization technique on bytecode files of android applications.
- The proposed approach to represent the extracted features through colored images, a CNN model is trained to detect android botnet applications from benign ones.
- We compare the effectiveness of our approach to other existing studies that significantly improve detection accuracy mainly due to the simple image processing of android applications. To the best of authors' knowledge, this is the first CNN model that uses image processing for android botnet detection applications.

The rest of this paper is structured as follows. Section 2 summarizes the related works. Section 3 inclides the proposed work of the CNN model for the detection of botnet applications. Section 4 represents results and discussion. In the last section, we have presented the conclusion.

2 Related Works

In recent years, several research studies have proposed several methodologies for the detection and prevention of Android Botnet Applications. Additionally, the existing methodologies can be categorized into two kinds such as static and dynamic analysis. In static analysis, applications are decompiled to detect malicious features of Application Programming Interface (API) calls, Permissions, and lines of codes. On the other hand, in the dynamic analysis, the Applications are installed and executed on devices to investigate the malicious behavior. It should be noted that it uses special security techniques to inspect and stop malicious activities on the devices. i.e., sandbox

Kadir et al. [9] proposed a fully automated and comprehensive analysis for a better understanding of malicious activities and their aspects in their study. The researchers combine static and dynamic analysis approaches and demonstrate numerous families of an Android botnet. Moreover, the author's study on C&C channel and embedded Uniform Resource Locators (URLs) uncover the malicious behavior of applications with the basic infrastructure of botnet families and analyze the relations. Furthermore, the approach uses the ISCX [10] dataset consists 1929 samples of android botnet applications belonging to 14 different families.

Anwar et al. [6] proposed a static feature-based methodology to detect Android botnet activities. The most important features, such as permissions, MD5 signatures, broadcast, and background services were extracted. In order to extract these features, the researchers used 1400 ISCX [10] botnet applications and 1400 benign applications. The Machine Learning (ML) Classifiers were applied to perform the experiment and examine the accuracy with respect to the feature length. The authors claimed that they obtained an accuracy of 95.1%, recall of 0.827%, and precision of 0.97% by the consideration of respective features.

Another approach [11] was designed for analysis of the mobile platform to distinguish malicious activity by investigating permissions and protections of the applications. The authors introduced 138 attributes of permissions initially and were extended to 145 when protection applications were used as unique attributes. The designed approach is then classified based on four different Machine Learning (ML) boxes of Multi-Layer Perceptron (MLP), Decision Tree, Random Forest, and Naïve Bayesian were performed on 1635 benign and 1635 botnet features from the ISCX dataset [10]. However, Random Forest achieved the best results of 97.3% accuracy, recall of 0.987%, and 0.985 precision among four ML classifiers.

Abdullah et al. [12] specified a mechanism to track permissions in android applications that limits functionality and code and employ an ML box of Decision Trees, Naive Bayes and Random Forest for evaluation of the results. Overall, in these approaches, Random Forest was performed best approach and gave results of 94.6% accuracy and 0.099% with a false-positive rate.

Another mobile botnet De-Droid approach was proposed by Karim et al. in [7]. De-Droid is a static and lightweight technique to identify the botnet-specific properties. Additionally, the dataset consists of 5064 malicious and 14865 benign applications, which is the largest dataset to detect suspicious binaries by decoding the API Calls and Permissions. Through the comparison between malicious and benign applications, the researchers claimed that applications could be classified as botnets with 35% of malicious binaries. However, results reveal that the applications are confirmed as botnet with 90% accuracy.

Jadhav et al. [8] proposed a dynamic and cluster analysis approach to detect Android botnets by using a virtual environment. The approach employs cloud-based analysis systems to detect the behavior of the security systems with redefined attributes such as tcp dump, strace, sysdump, logcat, and netflow. However, there were no experimental results in their study for the usefulness of the proposed cloud-based solution. Additionally, the virtual environment needs the installation of a JAVA program on the user's smartphone for a security analysis report.

MBotCS another approach presented by Meng et al. [13], was designed for botnet applications detection in network parameters like source/destination IP address, frame duration, and packet size. The researchers identify malicious network traffic through the cooperation of ML algorithms. By contrast, their method suffers from a significant rate of false positive value. Another study [14], illustrated a comprehensive study for the detection of Android Botnets. The authors utilized API call permissions from different Android Botnet families and performed an experiment using various

machine learning approaches. As a result, the high false positive rate was the main drawback of their method.

Nataraj et al. [15] designed an image processing classification method for visualizing and identifying malware that first converts malware binaries to grayscale images. The visualization approach has been widely used due to fast processing for achieving high accuracies and detections of malware binaries. The authors concluded that the malware binary must first be converted to "image" before executing the classification method.

Three-branch embedding network (TBE-Net) is another approach with complementary learning maps proposed in [16]. They used global appearance and local region features in the form of images for feature learning to differentiate the vehicle from others. Another features map approach presented by Sun et al. [17], was presented for Unmanned Aerial Vehicles (UAVs) in automobile systems encourage the establishment of object detection techniques for collecting real-time traffic data.

An image-based mobile botnet detection was proposed CNN in [18]. Detection is achieved by analyzing applications and extracting permissions that belong to malicious applications. In the suggested methodology, the authors transformed the features into images and trained a CNN model. Furthermore, the proposed model can be classified as an initial classifier to discriminate between benign and botnet image samples with a precision of 0.955%, recall of 0.96%, f-measure of 0.957, and obtained 97.2% accuracy. Recently, another approach [19] was presented based on the feature extraction phase for an android botnet detection. These features are API calls, Commands, Intents, and extra files and trained a CNN model. In contrast, we have used learning features maps of image processing using visualization technique on bytecode files of android applications, whereas research studies are based upon some specific features such as API Calls, Permissions, strings, and intents for detection of the Botnet Applications. Moreover, we used the reverse engineering technique only for dataset preparation in the form of images. Furthermore, our approach is of particular use for the detection and classification of android applications with the help of image processing and the CNN model.

3 Methodology

We propose an image-based CNN model detection approach to avoid employing sophisticated dataset analysis techniques to extract the feature representation of android applications.

3.1 Dataset

For detection of botnet android applications from benign ones, we have collected datasets for both benign and botnet android applications. For the implementation of the CNN model, initially, it is important to observe the dataset which contains a significant number of samples for the analysis. The dataset largely consists of two parts: the first one is an android botnet application and the other one is a benign applications dataset. Furthermore, we have collected 1288 Benign Android Packages (APKs) from CICAND Malware Dataset [20] which comprised 6,500 benign APKs obtained from the Google Play store. In addition, to ensure that the APKs do not consist any malicious code, we have used the website Virus Total and obtained results with the secure APK files that we require. For Botnet APK files, we used ISCX [10] android botnet dataset which includes 1,929 botnet apps from 14 families. Additionally, we have selected 612 botnet APKs for our experiment.

3.2 APK Files Reverse Engineering

After the collection of benign and botnet APK files, we used the reverse engineering technique to unzip the files as shown in Fig. 2. An android package normally contains the following files:

- Classes. dex: android opcode file, which can be compiled by the Dalvik Virtual machine.
- AndroidManifest.xml: includes important the applications such as android compatibility, platform, permissions and API Calls, etc. that must be executed before any of the opcode files.
- The Resource and lib folder contains the binary compiled code and needs by the APK files.

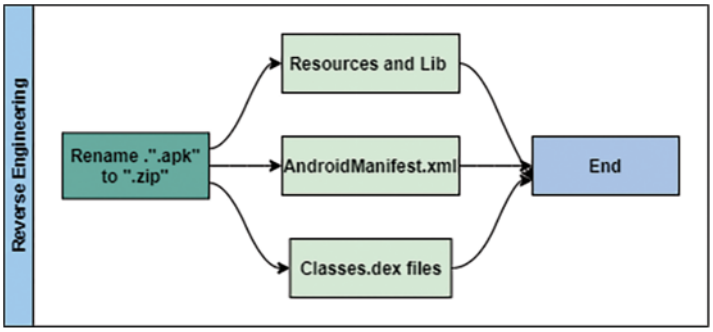

Figure 2: Reverse engineering

For our work, we have utilized the classes.dex files from android package files because all android application code contains the byte code of the application, in addition, to initializing and executing the apps.

3.3 Representation of Classes.dex Files

After the extraction of required classes.dex files for each android application, we need to represent the Classes.dex in byte series of sequence. For this purpose, we have adopted a hexadecimal approach to the representation of bytes series. The key idea for the representation of the hexadecimal view is taken from research study in [21]. The reason to choose the hexadecimal approach is that the byte series is displayed in the consequence of sequential 16-byte blocks, as shown in Table 1, in which each byte file is processed as a binary byte. Precisely, each byte highlights the important information of classes.dex files, such as instruction codes or data.

Table 1: Hexadecimal view of bytes

Address of machine code	Hexadecimal view
10918000	E6 01 00 00 20 0C 75 07 8B 40 00 00 21 0C CA 08 . . .

3.4 Files Visualization as Images

In the proposed study, the CNN-based model plays a vital role in image classification. As the model only deals with two-dimensional data such as images, therefore, we need to transform the bytecode files of android applications into images. For this purpose, we studied the bytecode files and these files contain 2-digit hexadecimal values that go from 00 to FF and are equivalent to 0–255

in decimal format. Additionally, we have evaluated the bytecode code files and converted them into RGB images by using the python script of a pillow as shown in Fig. 3.

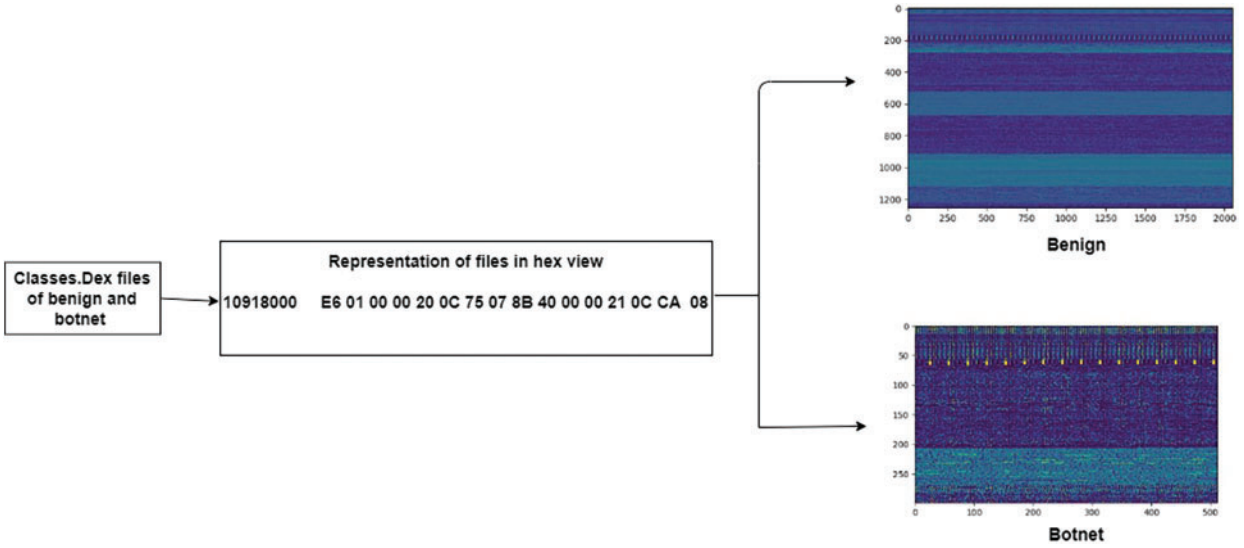

Figure 3: File visualization

3.5 Convolutional Neural Network

Initially, we selected classes.dex files from botnet and benign applications to detect the malicious patterns of applications with the help of image processing discussed in file visualization as images. In order to correctly differentiate botnet applications from benign, we need to train an image dataset. For this purpose, we have adopted a CNN-based model to classify the images dataset and achieve high accuracy.

CNN is a normally used neural network to detect botnet images from benign through the trained model. Initially, we use bytecode images to meet the requirements of the CNN model and detect botnets during the training period. Additionally, CNN layers such as ConvNet, MaxPooling, and Fully Connected layer patterns, are linked together to produce accurate predictions from the training period. For accurate predictions, the CNN model adjusts the Convolutional layer which consists several kernels and analyzes the features of images in the form of pixels. Each kernel is made up of a layer of connection weights with the width and height of the input patch during the forward phase and builds a two-dimensional feature map of the kernel as shown in Fig. 4. Pooling layer uses to decrease the dimensions images of the previous convolutional layer and summaries the computational weightage to reduce the overfitting problem in the model.

For our work, we use the LeNet-5 architecture and which is comprised of 3 conventional layers, two Max Pooling layers, and 2 Fully connected layers. For the input layer of CNN model, the height and width of the input images are $28 * 28 * 3$ with 32 feature maps. In addition, the kernel size of each feature map is $5 * 5$ strides and uses ReLU function to measure the dimensions of images and reduce the overfitting problem. After the Convolutional layer, an additional layer known as max pooling is used to overcome the downsampling of the model and its size is $2 * 2$. In addition, the computational weights are calculated through the max pooling layer on the input layer of CNN model. Next, a fully connected layer with 1024 units is passed with a flattening layer which is used to convert the data into

a 1-dimensional array for inputting it to the output layer value. For the last output layer, another fully connected layer is used with binary classes and passes the SoftMax function used for the accurate classification of images. In our last step, we feed the images to train the CNN Model and the dataset is split into 65% training samples and 35% test sample images. During the training phase of the model, we keep track of the validation and training sets accuracy. The model will be terminated and weights will be restored for classifying the test images with an accuracy of 94.34%. The Construction parameters are shown in Table 2.

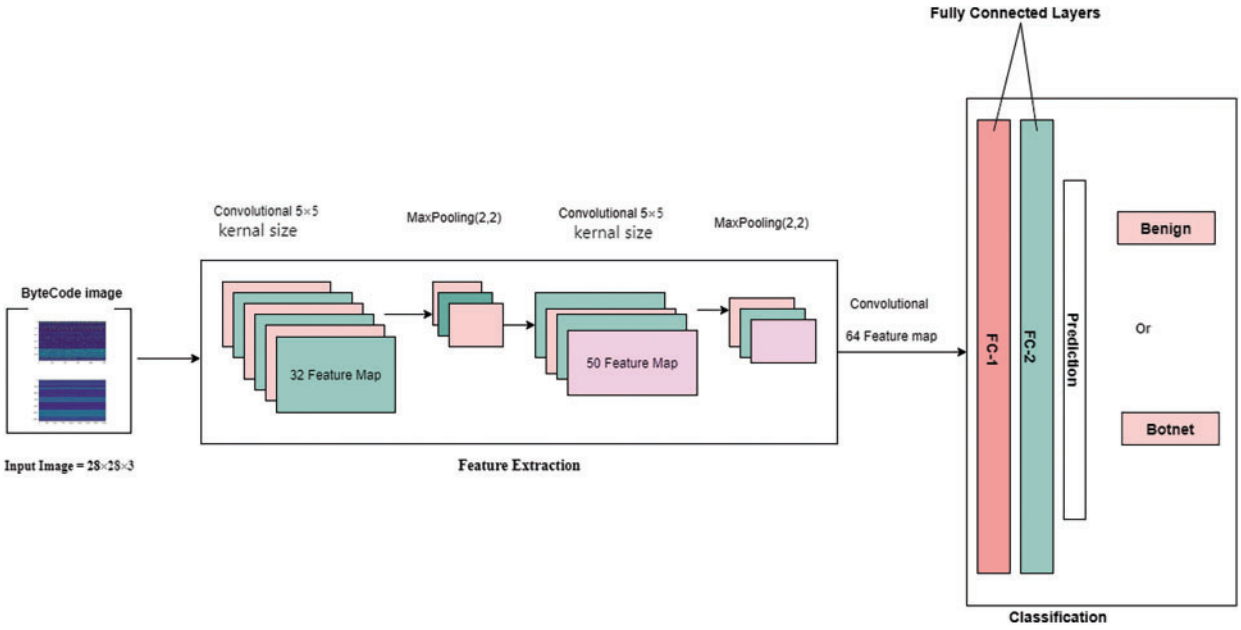

Figure 4: Proposed framework of the CNN model

Table 2: Construction of CNN model

Layers	Parameters	Function	Filter	Filter size	Strides
1	Conv2D	ReLU	32	$5 * 5$	$1 * 1$
2	MaxPooling2D	-	-	$2 * 2$	$2 * 2$
3	Conv2D	ReLU	50	$5 * 5$	$1 * 1$
4	MaxPooling2D	-	-	$2 * 2$	$2 * 2$
5	Conv2D	ReLU	64	$5 * 5$	$1 * 1$
6	Dense (Fully connected-1)	ReLU	1024	-	-
7	Dense (Fully connected-2)	Softmax	-	-	-

4 Experimental Results and Discussion

All experiments were performed on the windows server 2019 operating system and using 128 GB RAM and GPU with Keras [22] library and TensorFlow backend. Tensorflow is a machine learning backend library that can be used for a wide range of applications, whereas Keras is generally a part of the deep learning and self-contained structure. For our work, the convolutional neural network

is built using Keras and TensorFlow, including the OpenCV classes architecture. To examine the suggested method, we acquired malicious and benign Android applications and investigated the method's performance in other existing studies.

4.1 Experimental Results

Through the trained CNN, quantitative measurements have been performed to examine the performance of the model using image classification of benign and botnet image samples. For this purpose, we define measurement parameters as described in Table 3. in which the learning rate is defined as a floating point value. For accuracy of android images applications, we adapt the Adam optimizer and loss calculated as binary cross-entropy. Furthermore, the proposed method's quality has been measured using accuracy, recall, precision, and F1_Measure. The metrics can be defined as follows [18]:

$$\text{Recall} = \frac{TP}{TP + FN} \tag{1}$$

$$\text{Precision} = \frac{TP}{TP + FP} \tag{2}$$

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{3}$$

$$\text{F1\_Measure} = \frac{2 \times Presision \times Recall}{Precision + Recall} \tag{4}$$

Table 3: Measurement parameters of the proposed methodology

Parameters	Measurement values
Learning rate	1e-3
Test split	0.35
Optimizer	Adam
Loss	Binary cross entropy
Classifier	Binary (0 or 1)

In Eqs. (1)–(3) FP, TN, TP, and FN are respectively denoted as False Positive, True Negative, True Positive, and False Negative. From the measurement evaluation, our model shows an accuracy of 94.34%, F1 score of 92%, Precision of 92.9%, and Recall of 92% with cross-entropy loss of 0.139.

4.2 Variations of Accuracy and Loss on Network Training

As shown in Fig. 5, we have examined the performance of the model at 85, 90, 95, and 100 epochs and analyzed that no major change in both train_ac and val_acc after 90 epochs. Moreover, the model returns the best prediction results with 94.34% accuracy, 92% of recall, and a precision of 92.9%. Precisely, the training accuracy is improving as the model processes and is varying passage time. Similarly, training loss has a specific amount in the completion, while validation loss varies. After completing the training phase, the statistical analysis is saved and used in the testing process after several training and testing iterations.

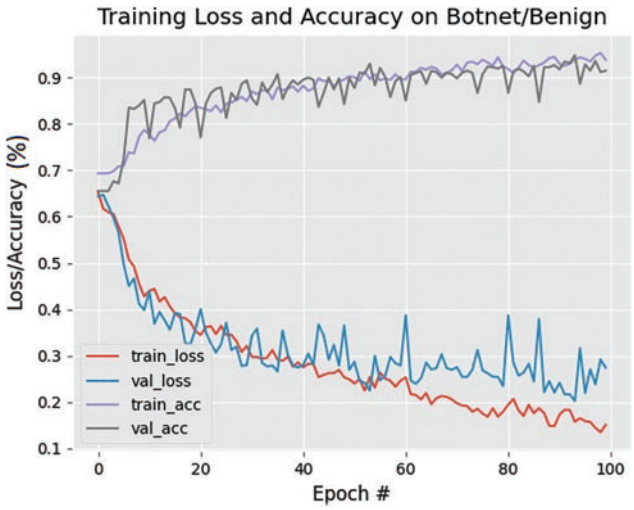

Figure 5: Loss/Accuracy over 100 iterations of the training phase

4.3 Comparative Analysis with Existing Work

To emphasize the importance of our study, we evaluate the proposed method to existing neural model approaches in this section. Existing methods use dynamic, static, or hybrid analysis to detect malware in general. As a result, the exact comparison is not possible. Nevertheless, we can compare the performance results in terms of neural network models, Feature source application characteristics and accuracy.

The static analysis approach presented in [23] used a Recurrent Neural Network (RNN) model for training and testing the applications and obtained 0.897% accuracy. The researchers used a feature analysis approach to extract permissions from androidmanifest.XML files for the classification of malicious applications. Another study in [24] was based on dynamic analysis and the authors employed reverse engineering technique and used permissions from Androidmanifest.XML files with two different models such as RNN and Long Short-Term Memory (LSTM) models that have acquired 0.93% accuracy. In addition, another research [25] proposed a general android malware detection using the CNN model. The authors concluded that the CNN model is feasible for detecting malicious applications with the usage of APK file dissembler and extracting android manifest.xml files. Then, the extract android manifest.xml file is converted into feature vectors and these feature vectors are fed into the CNN model for training. The authors experimented on 200 malicious samples with 500 clean samples and obtained 0.93% accuracy.

In [26] and [27], authors investigated malicious android applications using CNN model which assists the user to predict maliciousness in applications by generating the images for each android malware detection. The authors used the reverser engineering technique and extract the classes.dex files then transformed into images. The study in [26] is somewhat related to our approach. However, our objective of this study has presented an approach to detecting android botnet applications from benign ones. However, this approach generaly applies for mobile malware detection. Moreover, the researcher used Convolutional Neural Network Model for training and testing the android malware applications and obtained 0.93% accuracy.

On the other hand, our proposed model uses 1288 benign and 612 botnet applications for classification. Initially, we used the reverse engineering technique to dissemble the APK application

and extract the classes.dex files. After extraction of the classes.dex files, these files are transformed into images and trained in a CNN-based model. As a result, we achieve the highest accuracy of 0.9434% as compared to other existing studies. As shown in Table 4 compares the proposed method to other Neural Models. According to the table, the proposed method is entirely effective in detecting botnet applications.

Table 4: Comparative analysis of existing studies

Reference	Neural model	Platform	Application characteristics	Features source	Accuracy
Vinayakumar et al. [23]	RNN/LSTM	Android	Android permission sequence	AndroidManifest.xml	0.897%
Vinayakumar et al. [24]	LSTM	Android	Permissions, dynamic behaviors	AndroidManifest.xml dynamic analysis	0.939%
Ganesh et al. [25]	CNN	Android	Android permission Sequence	Permissions	0.93%
Vinayakumar et al. [23]	R2-D2: CNN	Android	Android bytecode	Classes.dex files	0.93%
Our model	CNN	Android	Android bytecode	Classes.dex files	0.9434%

5 Conclusions

In recent years, there has been a rising security concern in all android and handheld devices because of internet facilities. Nowadays, smartphones and handheld devices have become analogous to personal computers due to the same facilities of battery power and memory capacities of such devices. Mobile devices are usually linked to the Internet at all times which are more susceptible to cyberattacks. Botnet attacks have proven to be a significant concern for internet security. Therefore, developing a model of botnet application detection is necessary for security admins. In this paper, we proposed a CNN approach as a learning model and used images sample of android applications for the detection of botnet applications. In the first step, the reverse engineering is used on 1288 benign applications from CICAND Malware Dataset and 612 Botnet applications from the ISCX dataset to extract the classes.dex files. In the second step, we used the classes.dex files as a features representation method in the form of hexadecimal view and transformed into images using the visualization technique. At last, we trained and tested a proposed CNN model using the images to detect botnet applications from benign ones. As a result, the model gives the best prediction results and obtained an accuracy of 94.34%, 92% of recall, and a precision of 92.9%. Furthermore, experimental results showed the best accuracy result when we compare the performance of the model to other existing research studies. In the future, we plan to provide an internet service that allows users to check if an application is benign or botnet before downloading it. This measure would go a long way toward ensuring the security of android smartphones.

Acknowledgement: The authors extend their appreciation to King Saud University of Saudi Arabia for funding this work.

Funding Statement: This works was supported by King Saud University for funding this work through Researchers Supporting Project Number (RSP2022R426), King Saud University, Riyadh, Saudi Arabia.

Author Contributions: The authors confirm contribution to the paper as follows: study conception and design: M. Arshad, A. Karim, S. Naseer, S. Ahmad, M. Shafiq; data collection: M. Alqahtani, A.A. Gardezi, M. Shafiq, J.G. Choi; Data curation, analysis and interpretation of results: M. Arshad, A. Karim, S. Naseer, S. Ahmad, M. Shafiq; draft manuscript preparation: M. Alqahtani, A.A. Gardezi, M. Shafiq, J.G. Choi. All authors reviewed the results and approved the final version of the manuscript.

Availability of Data and Materials: The data underlying this article will be shared (after the patent of the underlying research is filled) upon reasonable request to the corresponding author.

Conflicts of Interest: The authors declare that they have no conflicts of interest to report regarding the present study.

References

[1] G. Geng, G. Xu, M. Zhang, Y. Yang and G. Yang, "An improved SMS based heterogeneous mobile botnet model," in *Proc. of Int. Conf. on Information and Automation*, Shanghai, China, pp. 198–202, 2011.

[2] H. Singh and A. Bijalwan, "A survey on malware, botnets and their detection," *International Journal of Advanced Engineering Research & Science*, vol. 3, no. 3, pp. 258842, 2016.

[3] H. Pieterse and M. S. Olivier, "Android botnets on the rise: Trends and characteristics," in *Proc. of Information Security for South Africa*, Johannesburg, South Africa, pp. 1–5, 2012.

[4] I. Letteri, M. Del Rosso, P. Caianiello and D. Cassioli, "Performance of botnet detection by neural networks in software-defined networks," in *Proc. of Italian Conf. on Cyber Security*, Milan, Italy, 2018.

[5] A. Karim, S. A. A. Shah and R. Salleh, "Mobile botnet attacks: A thematic taxonomy," in *New Perspectives in Information Systems & Technologies*, 1st ed., vol. 1. NY, USA: Springer International Publishing, pp. 153–164, 2014.

[6] S. Anwar, J. M. Zain, Z. Inayat, E. U. Haq, A. Karim *et al.,* "A static approach towards mobile botnet detection," in *Proc. of Int. Conf. on Electronic Design*, Phuket, Thailand, pp. 563–567, 2016.

[7] A. Karim, R. Salleh and S. A. A. Shah, "DeDroid: A mobile botnet detection approach based on static analysis," in *Proc. of Int. Conf. on Autonomic and Trusted Computing*, Toulouse, France, pp. 1327–1332, 2015.

[8] S. Jadhav, S. Dutia, K. Calangutkar, T. Oh, Y. H. Kim *et al.,* "Cloud-based android botnet malware detection system," in *Proc. of Int. Conf. on Advanced Communication Technology*, Pyeongchang, South Korea, pp. 347–352, 2015.

[9] A. F. A. Kadir, N. Stakhanova and A. A. Ghorbani, "Android botnets: What urls are telling us," in *Proc. of Int. Conf. on Network & System Security*, Cham, Springer, pp. 78–91, 2015.

[10] Android Botnet Dataset. [Online]. Available: https://www.unb.ca/cic/datasets/android-botnet.html

[11] J. F. Alqatawna and H. Faris. "Toward a detection framework for android botnet," in *Proc. of Int. Conf. on New Trends in Computing Sciences*, Amman, Jordan, pp. 197–202, 2017.

[12] Z. Abdullah, M. M. Saudi and N. B. Anuar, "Mobile botnet detection: Proof of concept," in *Proc. of Control & System Graduate Research Colloquium*, Shah Alam, Malaysia, pp. 257–262, 2014.

[13] X. Meng and G. Spanoudakis, "MBotCS: A mobile botnet detection system based on machine learning," in *Proc. of Int. Conf. on Risks & Security of Internet and Systems*, Cham, Springer, pp. 274–291, 2015.

[14] M. Yusof, M. M. Saudi and F. Ridzuan. "A new mobile botnet classification based on permission and API calls," in *Proc. of Int. Conf. on Emerging Security Technologies*, Amman, Jordan, pp. 122–127, 2017.
[15] L. Nataraj, S. Karthikeyan, G. Jacob and B. S. Manjunath, "Malware images: Visualization and automatic classification," in *Proc. of Int. Symp. on Visualization for Cyber Security*, PA, USA, pp. 1–7, 2011.
[16] W. Sun, G. Dai, X. Zhang, X. He and X. Chen, "TBE-Net: A three-branch embedding network with part-aware ability and feature complementary learning for vehicle re-identification," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 9, pp. 1–13, 2021.
[17] W. Sun, L. Dai, X. Zhang, P. Chang and X. He, "RSOD: Real-time small object detection algorithm in UAV-based traffic monitoring," *Applied Intelligence*, vol. 52, pp. 1–16, 2011.
[18] S. Hojjatinia, S. Hamzenejadi and H. Mohseni. "Android botnet detection using convolutional neural networks," in *Proc. of Iranian Conf. on Electrical Engineering*, Tabriz, Iran, pp. 1–6, 2020.
[19] S. Y. Yerima and M. K. Alzaylaee. "Mobile botnet detection: A deep learning approach using convolutional neural networks," in *Proc. of Int. Conf. on Cyber Situational Awareness, Data Analytics and Assessment*, Dublin, Ireland, pp. 1–8, 2020.
[20] Android Malware Dataset, 2017. [Online]. Available: Available from: https://www.unb.ca/cic/datasets/andmal2017.html
[21] M. Ahmadi, D. Ulyanov, S. Semenov, M. Trofimov and G. Giacinto, "Novel feature extraction, selection and fusion for effective malware family classification," in *Proc. of ACM Conf. on Data & Application Security and Privacy*, New Orleans, USA, pp. 183–194, 2016.
[22] Keras, 2020. [Online]. Available: https://keras.io/
[23] R. Vinayakumar, K. P. Soman and P. Poornachandran, "Deep android malware detection and classification," in *Proc. of Int. Conf. on Advances in Computing, Communications and Informatics*, Udupi Karnataka, India, pp. 1677–1683, 2017.
[24] R. Vinayakumar, K. P. Soman, P. Poornachandran and S. Sachin Kumar, "Detecting android malware using long short-term memory (LSTM)," *Journal of Intelligent & Fuzzy Systems*, vol. 34, no. 3, pp. 1277–1288, 2018.
[25] M. Ganesh, P. Pednekar, P. Prabhuswamy, D. S. Nair, Y. Park *et al.,* "CNN-based android malware detection," in *Proc. of Int. Conf. on Software Security & Assurance*, PA, USA, pp. 60–65, 2017.
[26] T. Hsien-De Huang and H. Y. Kao, "R2-d2: Color-inspired convolutional neural network (CNN)-based android malware detections," in *Proc. of Int. Conf. on Big Data*, WA, USA, pp. 2633–2642, 2018.
[27] X. Wang, S. Yin, M. Shafiq, A. A. Laghari, S. Karim *et al.,* "A new V-Net convolutional neural network based on four-dimensional hyperchaotic system for medical image encryption," *Security & Communication Networks*, vol. 2022, no. 4260804, pp. 1–14, 2022.