Convolutional Neural Network for Classification of Android Applications Represented as Grayscale Images

Meghna Dhalaria, Ekta Gandotra

Abstract*: A rapid dissemination of Android operating system in smart phone market has resulted in an exponential growth of threats to mobile applications. Various studies have been carried out in academia and industry for the identification and classification of malicious applications using machine learning and deep learning algorithms. Convolution Neural Network is a deep learning technique which has gained popularity in speech and image recognition. The conventional solution for identifying Android malware needs learning based on pre-extracted features to preserve high performance for detecting Android malware. In order to reduce the efforts and domain expertise involved in hand-feature engineering, we have generated the grayscale images of AndroidManifest.xml and classes.dex files which are extracted from the Android package and applied Convolution Neural Network for classifying the images. The experiments are conducted on a recent dataset of 1747 malicious Android applications. The results indicate that classes.dex file gives better results as compared to the AndroidManifest.xml and also demonstrate that model performs better as the image become larger.

***Keywords*: Android malware, Android malware grayscale images, Convolutional Neural Network, Deep learning, Feature engineering, Machine learning**

I. INTRODUCTION

In accordance with the worldwide smartphone sales report, Android is considered as one of the most prominent mobile operating system (OS) [1]. Due to the increase in the usage of various applications of Android such as education, lifestyle, banking and gaming, it has lured attention of the cyber attackers. Consequently, Android malware applications have been developed by the attackers in order to steal user's private information, money etc. Malware is a piece of software which aimed to harm the system, collects the important information, make fake premium calls etc. According to the report of McAfee, more than 31 million samples were discovered as Android malware and 1.9 million new samples are found every year [2]. However, most of the new malicious applications are installed as variants of earlier known ones [3], [4]. As a result, malicious programs can be grouped into their families having a set of similar characteristics and behaviors. These similarities between the samples of same family can be used for the classification of new Android applications.

Malware authors are making use of various techniques for evading detection of their applications. The most popular method used to detect malware is based on signature based approach which matches the signature of new application with the signatures available in the existing database. The drawback of this method is that is it not able to detect unknown malware. The current methods for Android malware detection are based on machine learning and deep learning techniques. For using the concept of machine leaning, there is a need of feature engineering which requires domain expertise. These features are used to train a machine learning model. The features are extracted by using two techniques i.e. static and dynamic malware analysis. Static malware analysis technique analyzes the source code of the application to identify malicious patterns without executing the code [5]. The dynamic malware analysis technique analyzes the behavior of application during runtime using the virtual environment (i.e. Sandbox). The features obtained from static and dynamic malware analysis can also be integrated to enhance the classification accuracy [6]. Nowadays, due to the availability of automated tools, a large number of malware variants can be created with few clicks.

Moreover, attackers are using innovative ways to avoid detection of their Android applications. In order to deal with both of these problems and to reduce the efforts and domain expertise involved in feature engineering, we have created the grayscale images of AndroidManifest.xml and classes.dex files of APK. A Convolutional Neural Network (CNN) is designed which is used to classify the Android malicious applications.

The following are the contributions of the paper:

1. We prepared the two different sets of graysacle image datasets based on AndroidManifest.xml and classess.dex file which consists of 1747 application with 13 malware families.
2. Convolutional Neural Network model has been designed for the classification of the Android malicious applications into their families.
3. A Comparative analysis of classification results obtained from AndroidManifest.xml and classess.dex files (represent as grayscale images) is performed on the basis of different image sizes i.e.

Revised Manuscript Received on October 31, 2019.

Meghna Dhalaria, Department of Computer Science and Engineering, Jaypee University of Information Technology Waknaghat, Solan, (H.P.) India. Email: meghna8aug@gmail.com

Ekta Gandotra*, Department of Computer Science and Engineering, Jaypee University of Information Technology Waknaghat, Solan, (H.P.) India.. Email: ekta.gandotra@gmail.com

32x32, 64x64, 128x128 and 256x256.

The rest of the paper is framed as follows: Section II provides an overview related to Android Package (APK) structure. Section III provides the work related to identification and classification of Android application using machine learning and deep learning algorithms. Section IV discusses the methodology used and experimental result analysis for the Android malware family classification. Section V concludes the paper.

II. BACKGROUND

This section gives an overview related to APK structure:

Android applications are introduced by Google [7] which are developed using Java programming language. The applications of Android are packed as APK file which contains various files/sub-folders like Asset, lib, res, META-INF, classes.dex, resources.arsc and Android Manifest.xml [8]. Structure of an APK file is shown in Fig. 1. A brief description of its files and sub-folders is given as follows.

- *Assets:* Assets folder holds the media files which could be acquired by the Asset Manager.
- *lib:* It contains the information about the compiled code of Android application. The lib comprises of multiple sub folders which contains compiled code for particular hardware architecture. These are armeabi (holds compiled code for Advanced RISC Machine (ARM) based processor), arm64- V8a (holds compiled code for ARMv8 and above processor), armeabi v7a (holds compiled code for ARMv7 and above processor), X86 (holds compiled code for X86 processor), X86-64 (holds compiled code for X86-64 processor) Mips (holds compiled code for MIPS processor).
- *res and resource.arsc folder:* Resources are found in res and resource.arsc file. It contains XML file, string file, fonts, icons and images etc.
- *META_INF:* This folder contains meta information about the contents of APK file. This folder comprises of three parts i.e. CERT.RSA, CERT.SF and MANIFEST.MF. CERT.RSA holds public certificate of the Android application. CERT.SF holds the application security certificate and the records of all files present in the APK file with their hash (SHA-1 digest). MANIFEST.MF is a manifest file which comprises of vital information about the application. It generally holds the information about the permissions (which application needs from other apps or OS), package name of the application and the features related to hardware and software which are required by the application.
- *Classes.dex:* It includes the application bytecode in Dalvik Virtual Machine (DVM) format. It also contains the information about the API calls or methods, class name, class path and the package name.
- *AndroidManifest.xml:* This file contains information about the permissions, minimum level of API and intents.

III. LITERATURE REVIEW

Various techniques and methods have been introduced for the identification or classification of Android malware. This section presents the review related to Android malware detection on the basis of features extracted from static and dynamic malware analysis and the visualization of malware using machine learning and deep leaning techniques.

Sugunan et al. [9] performed an analysis on the behaviour of benign and malicious applications using its dynamic and static features. The authors proposed a system to detect Android malicious applications. The results indicate that the combinations of features (dynamic and static) are found to be more accurate in detecting the Android applications. In [10], the author presented an OmniDroid, which is an automated framework for both static and dynamic analysis of Android applications. They employed ensemble classifiers over this dataset for detection of Android malware. The experimental outcomes indicate the usability and feasibility of their publically available dataset and automated framework. Yuan et al. [11] introduced an online malware detection engine named as Droid detector which is based on deep learning techniques. It automatically detect the Android applications as either benign or malware. The results show that Droid detector achieves 96.76% accuracy in comparison with existing machine learning methods. In [12], the author proposed a detection system named as service monitor that dynamically detects the malware applications on the Android devices. They employed Markov chain model to depict how services are used to access system resources. Markov chain is considered for feature vector form and is classified by using Random Forest algorithm. The accuracy obtained is 96%. Cen et al. [13] presented a probabilistic discriminative model which is based on logistic regression for the identification of malware. They extracted API calls and permissions from decompiled source code. The results indicate that their proposed algorithm works well with combining both API calls and permissions. In [14], the authors designed a technique named as DREBIN for the detection of Android malware. They extracted eight different types of static features and applied machine learning algorithms to identify the malicious application. The experimental outcomes indicate that the DREBIN provide results with 94% accuracy. In [15], the author presented a technique to increase the user's mobile security. The author implemented deep learning technique for identification of Android malware. They extracted API calls from APK file and employed CNN for detection of malware. The accuracy obtained by CNN algorithm is 90%. Dhalaria et al. [16] performed a comparative analysis on Malgenome dataset on the basis of different machine learning and ensemble techniques to detect the Android application. The experimental outcomes indicate that stacking ensemble technique perform better as compared to other machine learning classifiers.

The above mentioned research works are related to extracting features using static and dynamic malware analysis to classify Android application. In order to remove the hand-feature engineering, the concept of CNN came into being which automatically extract the features.

The following are the research works related to the classification and detection of malware after visualizing these as images.

Nataraj et al. [17] was the first who worked on the malware images. They suggested a technique based on visualized classification of windows malware through image processing. The authors converted malware binaries into grayscale images and applied K-Nearest Neighbour (K-NN) technique for classifying the malware. The results indicate that the accuracy obtained by K-NN model is 98%. Yan et al. [18] introduced a novel technique known as MalNet which uses Long Short Term Memory (LSTM) and CNN. These networks learn from opcode sequence and grayscale images and applied stacking ensemble technique for classification of windows malware. The results demonstrate that MalNet obtained 99.88% accuracy for identifying malware. Cui et al. [19] presented a novel technique in order to efficiently detect the variants of malware. They transformed malicious code into grayscale images and classified these images using CNN. The results indicate the model achieves better accuracy i.e. 94.5% than other existing models. In [20], the authors designed a detection system for Android malware based on color inspired CNN. They converted the classes.dex to rgb color image and store the image with fixed size. The results indicate that the designed system give accurate results in terms of security. Zhao and Qian [21] introduced a static technique which does not require knowing the source code of application. With the help of decompilation they map the opcodes, API function into RGB image. They employed CNN to detect Android malware families. The results are demonstrated on the basis of accuracy, precision, recall, and f-measure which were 90.67%, 93.36%, 93.95% and 93.56% respectively. Hasegawa and Iyatomi [22] presented a method which is highly efficient and accurate in detection of malware. The proposed method considered a very small part of APK file as a target and examines it with 1-D CNN. They applied the proposed technique on two datasets. The results demonstrate that the accuracy obtained by two datasets is 95.40% and 97.04% respectively.

It is clear that the traditional methods are based on the pre-extracted features to detect the Android malicious application which requires domain expertise. Moreover, attackers are making use of innovative ways to evade detection of their Android applications, In order to deal with these problems, we have generated the grayscale images classes.dex and AndroidManifest.xml files which are extracted from the part of APK and applied CNN for classifying the images. The experiments are conducted on a recent dataset of 1747 malicious Android applications. The results show that classes.dex file performs better results as compared to the AndroidManifest.xml.

IV. METHODOLOGY USED

This section describes the workflow of the methodology used for comparing the Android family classification results obtained using AndroidManifest.xml and classes.dex files represented using grayscale images. Firstly, we collected the Android applications from virusshare.com [23]. These applications are then decompiled using AXMLPrinter2 [24] and Baksmali Disassembler [25] to extract the AndroidManifest.xml and classes.dex files. The AndroidManifest.xml and classes.dex files are transformed into their grayscale images. Afterward, CNN is applied to classify the malware images. Fig. 2 demonstrates the workflow of the methodology used.

A. Data collection and Preparation

- *Collection of Android Applications*: A total of 2200 Android applications are collected from virusshare. These are downloaded after getting registered with their website (www.virusshare.com).
- *Renaming to MD5*: MD5 hash algorithm has been employed to rename the applications with their hash value and to remove the duplicate applications. After this step, we left with 1747 applications.
- *Labelling of Data:* All these Android applications are scanned using Avira antivirus (AV) tool to identify the families of malware. The Avira AV tool detects the name of families which the particular application belongs to (shown in Table- I). It is found that, 1747 malicious applications belong to 13 different families of Android malware.

Table- I: Number of applications belonging to a particular family.

Malware Family	Number of applications
Android/AdLoad.A.Gen	50
Adware/ANDR.AdMogo.FAN.Gen	55
Adware/ANDR.AdsWo.CG.Gen	163
Adware/ANDR.Kuguo.K.Gen	263
Adware/ANDR.Mobwin.A.Gen	114
Android/Mseg.E.Gen	36
Android/MTK.F.Gen	52
Android/Plankton.C.Gen	60
Android/SmsAgent.AAV.Gen	402
Android/TrojanSMS.Boxer.B.Gen	335
Adware/ANDR.Waps.I.Gen	179
Adware/ANDR.Dianjin.A.Gen	18
Adware/ANDR.Fengvi.B.Gen	20

- *Decompilation process:* The applications are decompiled using python script which uses automated tool named as AXMLPrinter2 and Baksmali Disassembler. Through this, we extracted the AndroidManifest.xml and classes.dex.
- *Formation of Grayscale images:* After the process of decompilation, the classes.dex and AndroidManifest.xml are transformed to grayscale images using python script. In our work, a malware is read as binary as a vector of 8 bit unsigned integer and then arranged into 2D array as shown in Fig. 3. This can be represented as a grayscale images which has pixel values ranging from 0 to 255. Here 0 means black and 255 means white. Here, the width of the image is kept fix using a square root function and the height of the image is dependent on the size of the file.

- *Resizing the image*: The grayscale images generated are of different sizes and dimensions depending on the respective size of each Android application. Therefore, these malware images can't be directly provides as the input for CNN. To overcome this problem, we reshaped the images to a fixed size i.e. 32x32, 64 x 64, 128 x128 and 256 x 256. In this way, the images are resized and could be directly provided to the CNN for classification. Table- II shows the algorithm of the methodology used.

B. Classification using CNN

CNN or ConvNet is a feed forward neural network where the association pattern between the neuron is encouraged by the structure of an animal visual cortex [26]. It is used in various areas such as audio, image recognition, identifying faces etc. We have used CNN because it can be applied to whole image at a time and it is more reliable. The purpose of convolution layer is to extract features from original image. It makes use of different filters which acts as feature detectors and produce different feature maps for the same image. Different activation functions such as ReLU, sigmoid, tanh etc are used to convert the input to an output. The output is used as an input for the next layer. Pooling layer is responsible for reducing the dimensionality of each feature map. It is known as down sampling and sub sampling. The fully connected layer is used for classifying the original images into different classes.

In the proposed work, we have designed 4-layer convolutional neural network for classifying the Android applications into 13 malware families (shown in Fig. 4). We first employed convolution layer and after every layer we applied ReLU non linear activation function to perform operation. We employed ReLU because it helps in vanishing gradient problem and is much faster than other activation function such as sigmoid and tanh (described in equation 1).

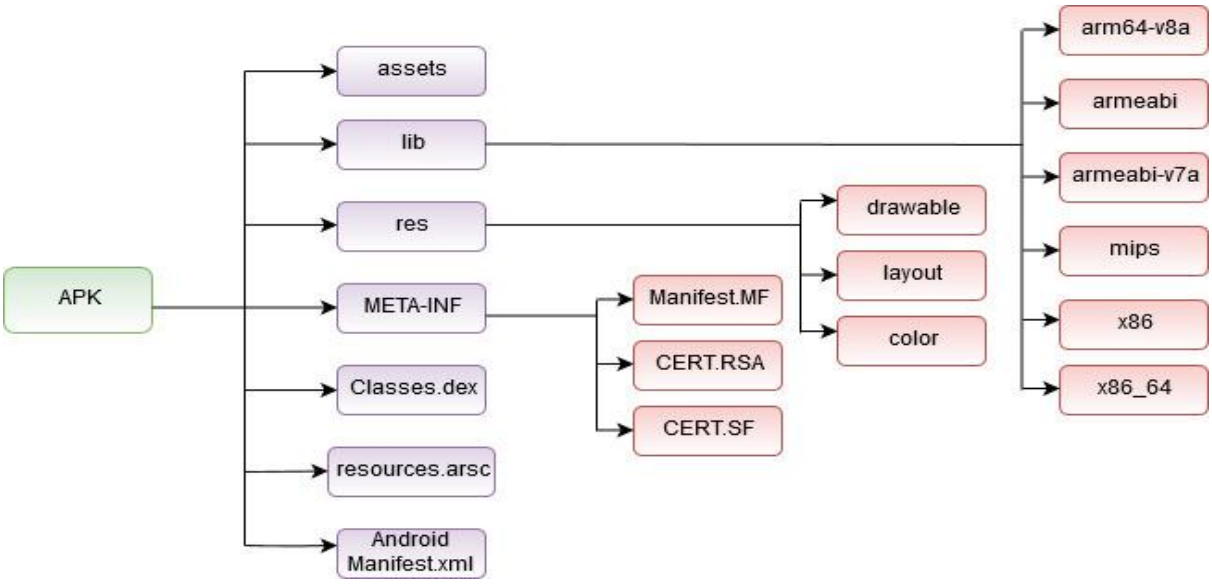

Fig. 1. APK Structure

We have used max pooling of 2x2 to reduce the dimensionality of feature maps. It takes filter size of 3x3 and stride=1. After employing all the layers, we have four dimensional vectors of arrays. To transform these vectors into single layer of 1-D vector which is called as fully connected layer. There may be loss of information while downsampling all vector into 1-D. For this purpose, we have employed two fully connected layers using softmax non linear activation function. It is represented in equation (2). Finally, this layer classifies the original images into 13 different malware families. The loss is calculated using cross entropy loss function which is mainly used for multiclass classification problem. For optimization purpose, we have used Adam optimizer.

$$f(y) = \begin{cases} 0 \ for \ y < 0 \\ y \ for \ y \geq 0 \end{cases} \quad (1)$$

$$\sigma(x)_i = \frac{e^{x_i}}{\sum_{j=1}^{J} e^{x_j}} \quad (2)$$

Here x is a vector of the inputs to the output layer. i indexes the output units, so $i = 1, 2, ..., J$.

The lists of the parameters used in the proposed model of CNN are shown in Table- III.

Table- III: List of CNN network parameters

Model Parameters	CNN
Kernel size	3x3
Stride	1
Activation function	ReLU
Max Pooling	2x2
Optimizer	Adam
Loss function	Cross entropy
Dropout probability	0.5
Epoch	20
batchsize	16
Fully connected (Activation function)	Softmax

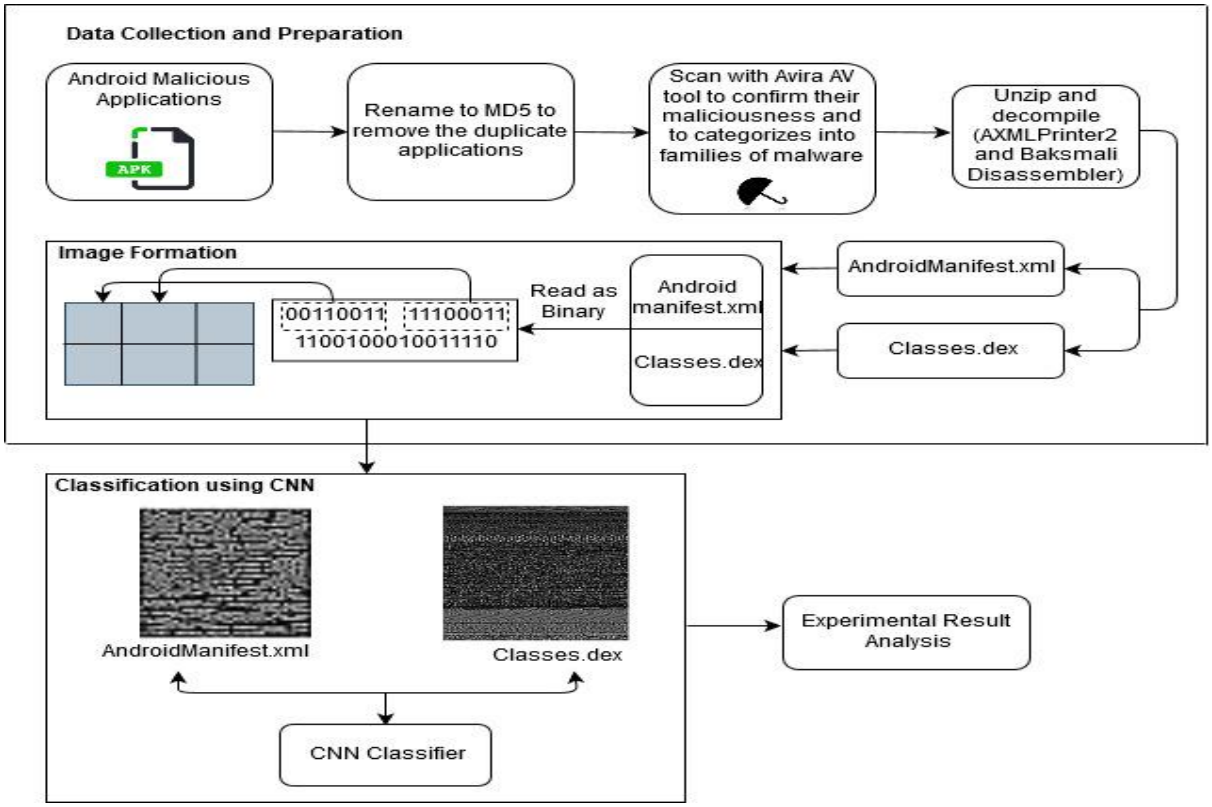

Fig. 2. Workflow of Methodology Used

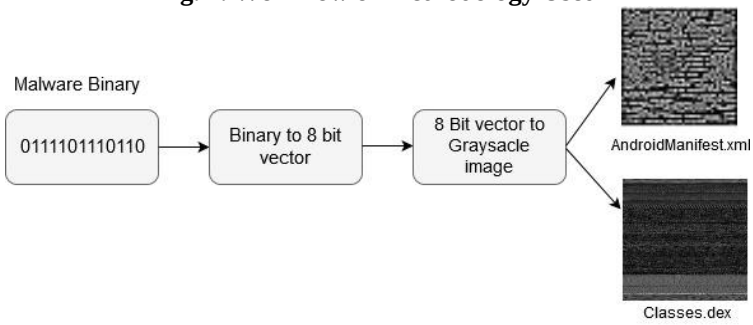

Fig. 3. Formation of grayscale image

Table- II: Algorithm of the methodology used

Input: Android malware Applications *(X)* in the form of APK
Begin
1. *Z*=MD5(*X*) //rename all the file to MD5 to remove duplicates
2. Gpscan.log=Scan_Avira(*Z*) //to categorize the families of Android malware
3. Unzip each application $x \in Z$, use //to unzip.APK
   AXMLPrinter2 tool for disassembling AndroidManifest.xml (*A*) //to extract AndroidManifest.xml file
   Baksmali disassembler for disassembling classes.dex (*C*) //to extract Classes.dex file
4. *for* every file $f \in A \text{ and } C$, repeat the following //transform A and C into grayscale images
   Read *f* as a binary //as a vector of 8 bit unsigned integer
   Set width of image as Sqrt (*f*) //fixed the width of the image
   Set height of image as length (*f*) //height of the image
   Resize(32x32) //create the images of size 32x32
   Resize(64x64) //create the images of size 64x64
   Resize(128x128) //create the images of size 128x128
   Resize(256x256) //create the images of size 256x256
5. *for* both type of images of all sizes, apply CNN algorithm (shown in table 4)

End
Output: Android malware family classification.

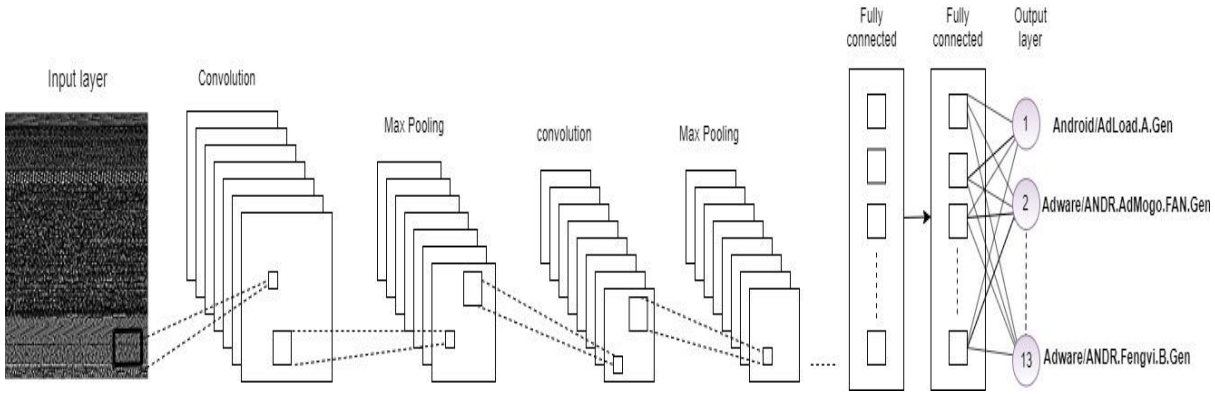

Fig. 4. Illustration of Android family classification using CNN

Table- IV: Algorithm for CNN

Input: Image of size w x h x d where w is the width of an image, h is the height of an image and d is the number of channel for grayscale image.
Begin:
Step 1: Convolution operation
1. Pass the input as an image (i.e. matrix with pixel value).
2. Select K as 3x3x1 matrix where K is the kernel/filter with stride =1.
3. Through this extract the feature from the original image (i.e. called Convolved feature).

(The main purpose of the convolution layer is to extract feature from the original image)
Step 2: Apply activation function
1. Rectified Linear Unit (ReLU) activation is applied.

(The main purpose of the activation function is to convert the input to an output and that output become input for the next layer)
Step 3: Pooling layer
1. Max pooling of window size 2x2 is applied over convolved feature.
2. Each window size takes maximum value.

(The main purpose of this layer is to reduce the size and also retains the important data)
Step 4: Flattening
1. Convert the output of the previous layer (which is matrix form) into 1-D feature vector (or we can say column vector).
2. Feature vector become input for the next layer (i.e. fully connected layer)

Step 5: Fully connected layer
1. Fed flattens output as an input to ANN (Artificial Neural network)
2. Softmax activation function is used to classify the families of malware.

End
Output: Classifying the images into their families

The various parameters used in the proposed design of CNN are shown in Table- IV.

C. Experimental Result Analysis

This section describes the dataset, evaluation parameters used for evaluating and comparing the proposed design of CNN model on different sizes of images created from AndroidManifest.xml and classes.dex files. It also presents the analysis of results obtained.

Dataset

As discussed in sub-section 4.1., the dataset used in this work is created from 1747 malicious Android application containing 13 malware families. AndroidManifest.xml and classes.dex files of each Android application are represented grayscale image of different sizes i.e. 32x32, 64x64, 128x128 and 256x256. The proposed CNN model is applied on all the images.

Evaluation Parameters

70% of total data is used for training purpose and remaining 30% is used for testing purpose. The experiments are conducted on Intel core (i5 processor) CPU (3.3 GHZ) with 8 GB RAM. The performance of proposed CNN classifier is evaluated on the basis of different evaluation parameters as discussed below:

- *Precision:* It is defined as what amount of positive identifications is correctly classified. It is calculated using equation (3)

$$Precision = \frac{TP}{TP+FP} \quad (3)$$

- *Recall:* It is defined as what amount of actual positive cases is classified correctly. The recall is computed using equation (4)

$$Recall = \frac{TP}{TP+FN} \quad (4)$$

- *Accuracy:* It is defined as the ratio of number of correct predictions to the total number of prediction. It is computed using equation (5)

$$Accuracy = \frac{TP+TN}{TP+TN+FN+FP} \quad (5)$$

Here false positive (FP) and true positive (TP) are the number of samples wrongly and correctly classified as malware respectively. Similarly, false positive (FP) and true negative (TN) are correctly classified as benign respectively.

- *F-measure:* It is defined as the harmonic mean of both precision and recall. It is also known as F-score. The F-measure is computed using equation (6)

$$F-measure = \frac{2\times(Precision\times Recall)}{(Precision + Recall)} \quad (6)$$

Result Analysis

Table- V presents the comparative analysis of classification results obtained on applying the proposed CNN model on the images (of size 32x32, 64x64, 128x128 and 256x256) created from AndroidManifest.xml and classes.dex files of APK

Table- V: Classification results of AndroidManifest.xml and classes.dex images of APK on different image sizes

Image	Size of image	Precision	Recall	F-measure	Accuracy (%)
AndroidManifest.xml	32x32	0.631	0.629	0.629	63.0
	64x64	0.667	0.663	0.664	66.5
	128x128	0.684	0.680	0.681	68.2
	256x256	0.701	0.700	0.700	70.0
Classes.dex	32x32	0.642	0.641	0.641	64.1
	64x64	0.671	0.670	0.670	67.0
	128x128	0.722	0.720	0.720	72.1
	256x256	0.746	0.745	0.745	**74.5**

It shows that for every image size considered in our dataset, classes.dex files give better classification accuracy as compared to AndroidManifest.xml. It clearly indicates that classes.dex file contains more relevant information about the Android application for classification purpose. It also demonstrates that for both types of files, as the image size increases, the classification accuracy increases. It may be due to the reason that as the image size is decreased, some of the features get destroyed. For the larger size image, the features remain intact and thus the larger size image gives better results. However, the larger size images take more training time.

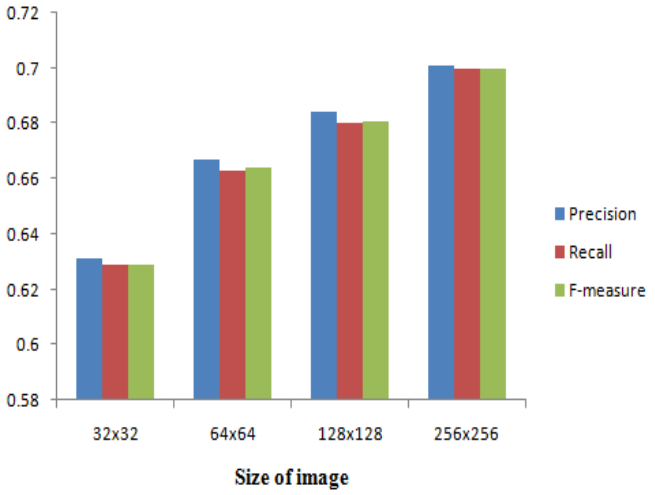

Fig. 5. Comparison of Precision, Recall, F-measure of AndroidManifest.xml images of different sizes

Fig. 5 demonstrates the comparison of Precision, Recall, F-measure of AndroidManifest.xml images of different sizes. It shows that as the size of images become larger it gives more accurate results. The images with size 256x256 perform better in terms of precision, recall and f-measure i.e. 0.701, 0.700 and 0.700 respectively.

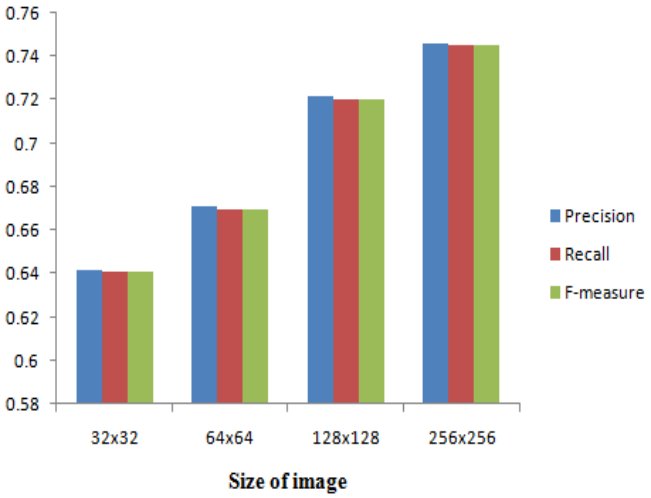

Fig. 6. Comparison of Precision, Recall, F-measure of classes.dex images of different sizes

Fig. 6 demonstrates the Comparison of Precision, Recall, F-measure of classes.dex images of different sizes. It indicates that as the size of image become larger it gives more accurate results. The images with size 256x256 perform better in terms of precision, recall and f-measure i.e. 0.746, 0.745 and 0.745 respectively.

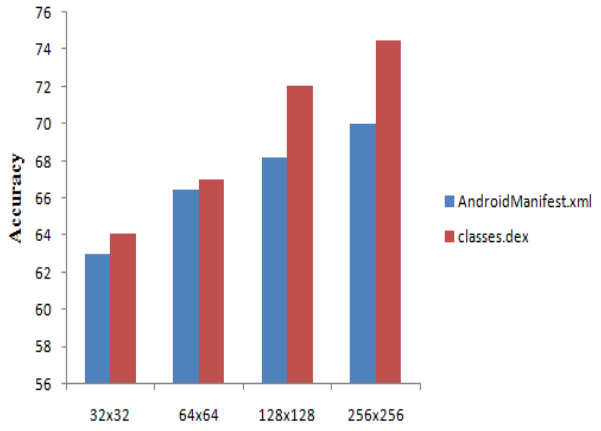

Fig. 7. Comparison of classification accuracy of AndroidManifest.xml and classes.dex images of different sizes

Fig. 7 indicates the comparison of AndroidManifest.xml and classes.dex images of different sizes on the basis of accuracy. It is concluded that the images created from the classes.dex file with size 256x256 provides the highest classification accuracy i.e. 74.5% followed by 128x128 i.e. 72.1%.

V. CONCLUSION

The risk of Android malware is increasing exponentially due to the increase in the dependency of Android users on different applications. At present, the malware generated are complex and sophisticated that can't be easily recognized. Most of the new malicious applications are installed as variants of earlier known ones. For better classification of Android malicious applications, the concept of machine learning and deep learning is being used where the domain expertise is required to select the features to be used. Therefore in this paper, we have proposed a design of CNN model which is applied on the grayscale images (of different sizes) obtained from AndroidManifest.xml and classes.dex file which are extracted from APK. CNN automatically extracts features from the images. The experiments are conducted on the dataset consisting of grayscale images of 1747 Android application with 13 malware families. We claim that our datasets consist of recent Android malicious applications. The results demonstrate that the images created through classes.dex files give better classification accuracy as compared to the images created using AndroidManifest.xml files. It is also concluded that for both types of files, as the image size increases, the classification accuracy increases. Thus images created from the classes.dex file with size 256x256 provide the highest classification accuracy i.e. 74.5%.

REFERENCES

1. http://gs.statcounter.com/os-market-share/mobile/worldwide.
2. McAfee Labs Threat Predictions Report, McAfee Labs, Santa Clara, CA, USA, Mar. 2018.
3. S. Singla, E. Gandotra, D. Bansal, and S. Sofat, "Detecting and classifying morphed malwares: A survey," *International Journal of Computer Applications*, vol. 122, no. 10, 2015.
4. E. Gandotra, S. Singla, D. Bansal, and S. Sofat, "Clustering morphed malware using opcode sequence pattern matching," *Recent Patents on Engineering*, Vol. 12, no. 1, 2018, pp. 30-36.
5. E. Gandotra, D. Bansal, and S. Sofat, "Malware analysis and classification: A survey," *Journal of Information Security*, Vol. 5, no. 02, 2014, p. 56.
6. E.Gandotra, D. Bansal, and S. Sofat, "Integrated framework for classification of malwares," In *Proceedings of the 7th International Conference on Security of Information and Networks*, 2014, p. 417.
7. Google. Android. 2014. URL, www.android.com.
8. *Application Fundamentals | Android Developers". Android Developers. Retrieved 2018-12-03.*
9. K. Sugunan, T. G. Kumar, and K. A. Dhanya, "Static and dynamic analysis for android malware detection," In *Advances in Big Data and Cloud Computing*, Springer, Singapore, 2018, pp. 147-155.
10. A. Martín, R. L. Cabrera, and D. Camacho, "Android malware detection through hybrid features fusion and ensemble classifiers: The AndroPyTool framework and the OmniDroid dataset," *Information Fusion*, 52, 2019, pp. 128-142.
11. Z. Yuan, Y. Lu, and Y. Xue, "Droiddetector: android malware characterization and detection using deep learning," *Tsinghua Science and Technology*, 21, no. 1, 2016, pp. 114-123.
12. M. Salehi, and M. Amini, "Android Malware Detection using Markov Chain Model of Application Behaviors in Requesting System Services," *arXiv preprint arXiv:1711.05731*, 2017.
13. L. Cen, C. S. Gates, L. Si, and N. Li, "A probabilistic discriminative model for android malware detection with decompiled source code," *IEEE Transactions on Dependable and Secure Computing*, 12, no. 4, 2014, pp. 400-412.
14. D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, K. Rieck, and C. E. R. T. Siemen,. "Drebin: Effective and explainable detection of android malware in your pocket," In *Ndss*, vol. 14, 2014, pp. 23-26.
15. Y. U. Ding, W.G. Zhao, Z. P. Wang, and L. F. Wang, "Automaticlly Learning Featurs Of Android Apps Using CNN," In *2018 International Conference on Machine Learning and Cybernetics (ICMLC)*, vol. 1, 2018, pp. 331-336.
16. M. Dhalaria, E. Gandotra, and S. Saha, "Comparative Analysis of Ensemble Methods for Classification of Android Malicious Applications," In *International Conference on Advances in Computing and Data Sciences*, Springer, Singapore, 2019, pp. 370-380.
17. L. Nataraj, S. Karthikeyan, G. Jacob, and B. S. Manjunath, "Malware images: visualization and automatic classification," In *Proceedings of the 8th international symposium on visualization for cyber security*, 2011, p. 4.
18. J. Yan, Y. Qi, and Q. Rao, "Detecting malware with an ensemble method based on deep neural network," *Security and Communication Networks*, 2018.
19. Z. Cui, F. Xue, X. Cai, Y. Cao, G. Wang, and J. Chen, "Detection of malicious code variants based on deep learning," *IEEE Transactions on Industrial Informatics*, 14, no. 7, 2018, pp. 3187-3196.
20. T. T. Hsien-De Huang, and H. Y. Kao, "R2-D2: color-inspired convolutional neural network (cnn)-based android malware detections," In *2018 IEEE International Conference on Big Data (Big Data)*, 2018, pp. 2633-2642.
21. Y. Zhao, and Q. Qian, "Android Malware Identification Through Visual Exploration of Disassembly Files," *International Journal of Network Security*, 20, no. 6, 2018, pp. 1061-1073.
22. C. Hasegawa, and H. Iyatomi, "One-dimensional convolutional neural networks for Android malware detection," In *2018 IEEE 14th International Colloquium on Signal Processing & Its Applications (CSPA)*, 2018, pp. 99-102.
23. www.virusshare.com
24. android4me: J2ME port of Google's Android, 2011. https://code.google.com/p/android4me/downloads/list.
25. W. Enck, D. Octeau, P. D. McDaniel, and S. Chaudhuri, "A study of android application security," In *USENIX security symposium*, vol. 2, 2011, p. 2.
26. R. Yamashita, M. Nishio, R. K. G. Do, and K. Togashi, "Convolutional neural networks: an overview and application in radiology," *Insights into imaging*, *9*(4), 2018, pp.611-629.

AUTHORS PROFILE

Meghna Dhalaria is pursuing Ph.D in computer Science Department, Jaypee University of Information and Technology, Waknaghat. She received her Bachelor"s degree from Baddi University of Emerging Sciences and Technologies. She completed her Master's degree from Thapar Institute of Engineering and Technology, Patiala. Her current research includes the applications of Machine learning and Deep learning.

Ekta Gandotra is currently working as Assistant Professor in the Department of Computer Science & Engineering at Jaypee University of Information Technology, Waknaghat, Solan, Himachal Pradesh. She has around 12 years of teaching and research experience. She has completed her Ph.D. in Computer Science & Engineering from PEC University of Technology, Chandigarh. Her research areas include Network & Cyber Security, Malware Threat Profiling, Cyber Threat Intelligence, Machine Learning and Big Data Analytics.