Android Malware Detection with Deep Learning using RNN from Opcode Sequences

https://doi.org/10.3991/ijim.v16i01.26433

A. Lakshmanarao[1](✉), M. Shashi[2]
[1]Aditya Engineering College, Surampalem, India
[2]AU College of Engineering, Andhra University, Visakhapatnam, India
laxman1216@gmail.com

Abstract—Android is the most widely used operating system in smartphones. Mobile users can download and access apps easily from the play store. Due to lack of security awareness and risk associated with mobile apps, malware apps would be downloaded by normal users in general. The consequences after installing a malware app are unpredictable. Malware apps can gather user personal data, browsing history, user profiles, user sensitive data like passwords. Hence, android malware detection is essential for providing security to mobile users. Android malware detection using machine learning is done either by extracting static features (opcodes, permissions, intents, system commands) or by extracting dynamic features (log behavior, system calls, dataflow). In this paper, opcode sequences are extracted from malware and benign apps, and Recurrent Neural Networks are proposed on extracted sequences. Benign apps are collected from the play store, apkpure.com and malware apps are collected from the virus share website. The proposed Recurrent Neural Network model could achieve 96% accuracy for android malware detection.

Keywords—android, malware, opcodes, recurrent neural networks

1 Introduction

Around 85% of the smartphones in the world are using the android platform [1]. The number and variety of applications in the play store are increasing drastically. Since most of the apps are made available free of cost in the play store, they are getting installed into mobiles without any risk assessment. The attackers are exploiting this vulnerability by creating malware for android devices. The Android framework provides some security mechanisms to mitigate malicious app activities. Especially permission mechanism controls the installation of malicious apps. Before installing a specific mobile app, it prompts the user for permissions. But the normal user doesn't have much knowledge about all the details for identifying malware apps. Hence, the Permission mechanism may not always control the malware applications. If a malware app was installed by a naïve mobile user, malicious activities can happen. Due to the

wide spread use of smart phones among naïve people, there is a drastic increase in the proportion of malware apps over the years. Hence, android malware detection has become more and more important aspect of cybersecurity.

Android framework is based on a Linux environment. The bottom layer of the android architecture is the Linux kernel. On top of the kernel, it has a set of libraries (like WebKit, SQ Lite, SSL libraries) and android runtime. Android runtime contains Core java libraries. Mobile applications are built based on the java programming language. Every mobile application requires permissions to run on the android platform. If a classifier is developed to identify malware from the given mobile apps, the permission mechanism can be strengthened to automatically avoid malware being installed in the android devices. Since every mobile app is defined as a sequence of instructions/ opcodes, the basic nature of the app would be captured by the sequence of opcodes and hence provides the essential information for discriminating a malware app from the benign app. In this paper, a sequence of Dalvik opcodes is extracted from each android mobile app and a collection of such sequences with known labels are used for building a binary classifier to classify an unknown android app into either malware or benign apps. Sequential data is a type of data where the order of items plays a major role. Sequential data is also useful for solving classification problems. First, a model is trained with known sequences. Later, the trained model predicts the class label for the new input sequence. Sequential data can be a numerical sequence or text sequence. In this paper, sequential text patterns of opcodes are used for malware detection. The selection of datasets plays an important role in achieving good results in the case of machine learning and deep learning. The dataset was prepared from the raw apk files directly. This paper proposed a deep learning framework with Recurrent Neural Networks for android malware detection. The performance of the model is evaluated based on the classification accuracy. The remaining sections of the paper are organized as follows: Section-2 discusses related work in android malware detection, section-3 presents the proposed methodology, Section-4 presents Experimentation details with results and Section-5 has a conclusion and future scope.

2 Literature review

Applying machine learning techniques in the field of mobile technologies is not new [2]. Several authors applied machine learning and deep learning methods for malware detection. Android malware detection can be done in three different ways namely static analysis, dynamic analysis, hybrid analysis. In static analysis, static features like permissions, opcodes, API calls information are used for the detection of malware. Dynamic analysis involves running the android application in a virtual environment and extracting dynamic features. In hybrid analysis, apps can be converted to image representation and detection techniques are developed on images. Daniel Arp [3] et al. proposed a lightweight android malware detection model with machine learning algorithms. Static features like user permissions, Suspicious API calls, network addresses, intents are extracted from apk files and mapped to a vector space. Support Vector Machine applied on joint vector space and achieved an accuracy of 94%.

R. Hemanth [4] et al. proposed an android malware detection model with a classification and clustering algorithm. Feature Vector space is segregated using clustering and later random forest applied and achieved good results. Y. Zhang [5] proposed a Graph Convolutional Network for android malware detection. A tool named "SOOT" was used for analyzing and getting data flow chains. Similar nodes in the data flow chains are combined to form a graph and on top of it GCN applied and achieved an accuracy of 95%. A. Lakshmanrao [6] et al. proposed convolutional neural networks for android malware classification. Two different grayscale image datasets were generated from whole Android apps and only dex files in apks. Convolutional Neural Networks were applied on two image datasets and achieved an accuracy of better accuracy rate with dex images. Wei Wang [7] et al. proposed a hybrid model based on Deep Auto Encoder and Convolutional Neural Networks for malware detection. The authors used Autoencoder as a pre-training model to extract the best features and later CNN applied and achieved good results. The authors also concluded that the training time with DAE & CNN model was reduced by 83% compared to only the CNN model. Mahindru. A [8] et al. extracted a set of 123 permissions from android apps and applied various machine learning classifiers and achieved good accuracy with logistic regression classifier. Y. Yang [9] et al. proposed a deep graph Convolutional Network for malware detection. The call graph is generated from decompiled android app and then the function subgraph containing sensitive APIs extracted. The proposed network achieved an accuracy of 98%. Xiang Li [10] et al. applied the Naïve Bayes classifier after extraction of the android manifest file and achieved an accuracy of 85%. Abikoye Oluwakemi Christiana [11] et al. applied ensemble learning techniques for android malware detection and achieved good accuracy rate.

Dan Li [12] et al. proposed convolutional neural networks for malware detection. Apk files are decompiled using APK tool and smali files are extracted. From the smali files, Dalvik opcode sequence information was collected for malware detection and achieved a good accuracy rate. Unver. H [13] converted the android apps into grayscale images. Local and Global features are extracted from grayscale images. Later several classification algorithms were applied and achieved a good accuracy. A. Roy [14] et al. proposed a malware detection model using feature aggregation. The frequency-based feature vector is created from the API calls extracted from smali files. Random Forest is used with all extracted features (209) and achieved an accuracy of 93.77% and an f1-score of 91.73%. NMF (Non-negative matrix factorization) has been applied to reduce the size of the feature set from 209 to 50. Support Vector Machine applied on reduced set and achieved an accuracy of 88.7% and f-score of 85%. P. Agrawal [15] et al. discussed the pros and cons of android malware detection with machine learning. Detection techniques that uses dynamic analysis were resource consuming and the techniques that use static analysis suffers from tracking runtime behavior. H. Zhu [16] et al. proposed a stacking ensemble model for malware detection. Static features are extracted from android apps and a feature vector is generated. Principal Component Analysis applied on feature vector and Multilayer Perceptions used as base classifiers. Support Vector Machine employed as final fusion classifier and achieved an accuracy of 89%.

Oluwakemi Christiana Abikoye [17] et al. discussed different machine learning approaches applied for android malware detection.

All the previous works are based on either static or dynamic features. Extracting a large set of static and dynamic features is a complex task. The feature extraction (either static or dynamic features) requires special tools. So, in this paper authors extracted the most relevant static feature named "opcode sequences" for android malware detection. As opcode sequences of malware and benign apks are having lots of variation, we selected opcode sequences for android malware detection. The complexity of the feature extraction process is reduced to one specific tool. For extracting opcode sequences a python tool named "androguard" was used. Later RNN proposed on obtained features. Authors achieved good accuracy rate with only opcode sequences. The authors proposed a light weight malware detection model with good detection rate.

3 Proposed methodology

The proposed model aims at the recognition of static feature sequences. Android applications are developed in java. The android application is available in compressed form as zip file with apk extension. Extraction of a zip file results in several files that include meta-inf, res, android manifest XML file, dex file, assets. A dex file can be decompiled to produce a smali file. opcodes information is extracted from smali files. The authors used a python tool named "Androguard" to extract the features [18]. Androguard provides several commands to operate with android apks. The command "androguard decompile" creates control flow graphs for the given android app. It also generates .ag files (smali like format) for all the methods in the decompiled classes (.java file). The .ag files are used for extracting opcode sequences. After obtaining Dalvik opcode sequences, the Recurrent Neural Network model is applied for deep learning from these sequences. The proposed architecture for Android Malware Detection using RNN from opcode sequence is shown in Figure 1.

3.1 Recurrent Neural Networks

Recurrent Neural Networks (RNNs) were introduced as a variant of ANNs by [19] in the year 1986. RNNs overcome the limitations of Feed Forward Neural Networks to extract sequential patterns. In feed-forward neural networks, inputs and the activation produced are passed in the forward direction only. The neurons in each layer function independently by receiving inputs from the same set of sources to produce different activations based on the weights of the incoming edges at various hidden layers to extract different features of an entity and finally the activation produced at the output layer predicts the class label. Then, a cost function is used to calculate the error, and this error information is passed backward to update the weights in the network. Back-propagation and gradient descent algorithms are used to update the weights and train the feed forward network for classification efficiently. Feed forward networks are not designed to handle sequentially ordered inputs to extract sequential features. Alternative architecture is the Recurrent Neural Networks wherein the hidden neurons captures the sequence of states as they have recurrent connections to the previous hidden states. These recurrent connections help handle sequential information. It has memory to store

previous data and based on previous information, it can predict the future. As RNNs have the property of remembering information through time, they are widely used to extract sequential patterns from sequence datasets like time series data and textual data. "Backpropagation Through Time (BPTT)" algorithm is developed to train the RNNs on sequential training dataset. When the recurrent network is trained to perform a task that requires predicting the future based on the past, the network typically learns to use the state information, s(t) as a kind of lossy summary of the task-relevant aspects of the past sequence of inputs up to t. This summary might selectively keep some aspects of the past sequence with more precision/attention than others.

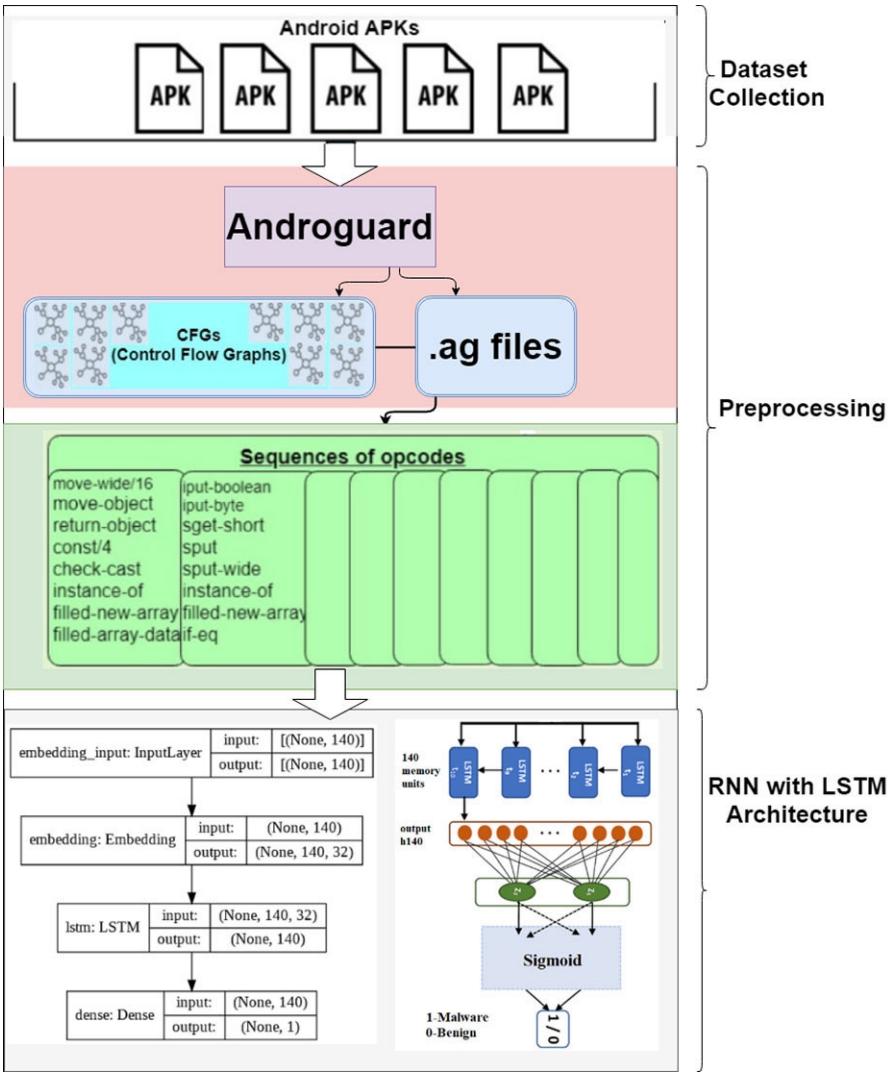

Fig. 1. Proposed framework

3.2 Long Short Term Memory (LSTM)

RNNs work with the principle of "current information is dependent on previous information in previous time steps". If the length of the sequence is very long, then RNN suffers from a problem known as the "Vanishing gradient problem". In such cases, RNN may take a long time or it may not work. If the sequences in the datasets are long, then RNNs leave out some important information in before time steps. In the vanishing gradient problem, the gradients (weights) become too small and do not contribute much to learning thus model performance is reduced. The vanishing gradient problem can be solved by Long Short-term memory. LSTM was introduced by [20] in 1997. LSTM block contains input, output, forget gates for controlling memorizing operation. Forget gate can be used to decide whether information can be stored or discarded. Input gate used to pass the relevant information from the present state, Output gate decides the next hidden state. LSTMs are a type of RNNs for handling longer dependencies. LSTMs are useful for sequential problems with longer sequences. In this paper, the authors used LSTM for malware detection. The architectures of ANN, RNN, LSTM are shown in Figure 2.

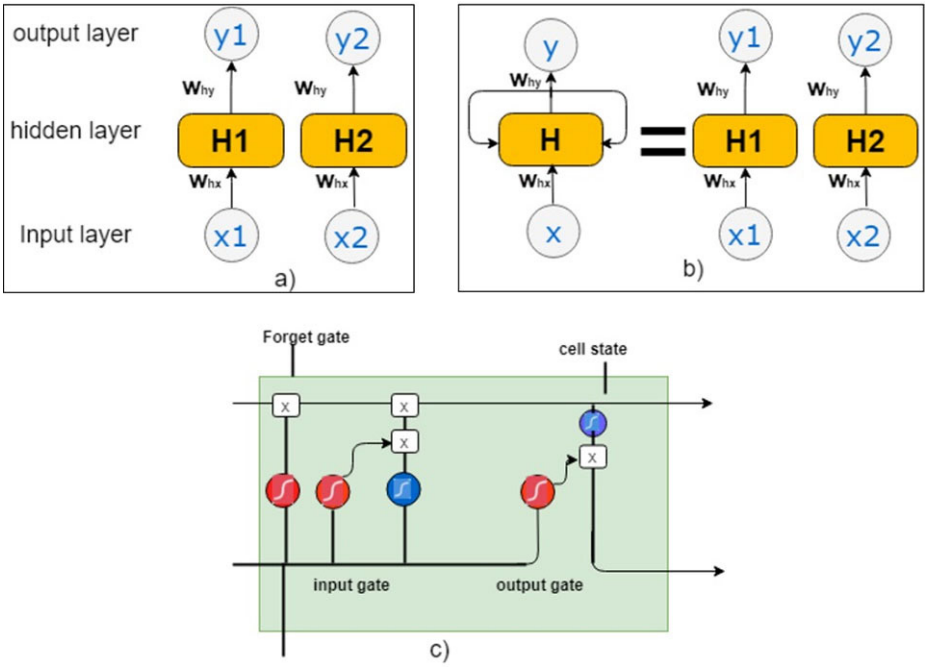

Fig. 2. a) architecture of ANN, b) architecture of RNN and c) architecture of LSTM

4 Experimentation and results

4.1 Dataset details

Malware apks are collected from virusshare.com website. Benign (non malware) apks are collected from CICAndMal2017 [21], Google play store, apkpure.com. The details of the dataset are shown in Table 1.

Table 1. Details of dataset

Number of Malware Apks	Number of Benign Apks
750	750

4.2 Extraction of opcodes from apks (creation of dataset)

Android apk is a zip file.Apk produces several files like classes.dex, manifest file, assets, ref file.classes.dex files contain the code part of the android application. So, classes.dex files contain important information for differentiating malware, benign apps. Dex files are decompiled to generate Smali files. One android appl contains several java programs. To generate opcodes from a single app, all these java programs can be analyzed. The number of opcode sequences differs from one app to another app. A tool named "Androguard" is used to decompile the app. After decompilation, an app produces several Control Flow Graphs (CFGs).

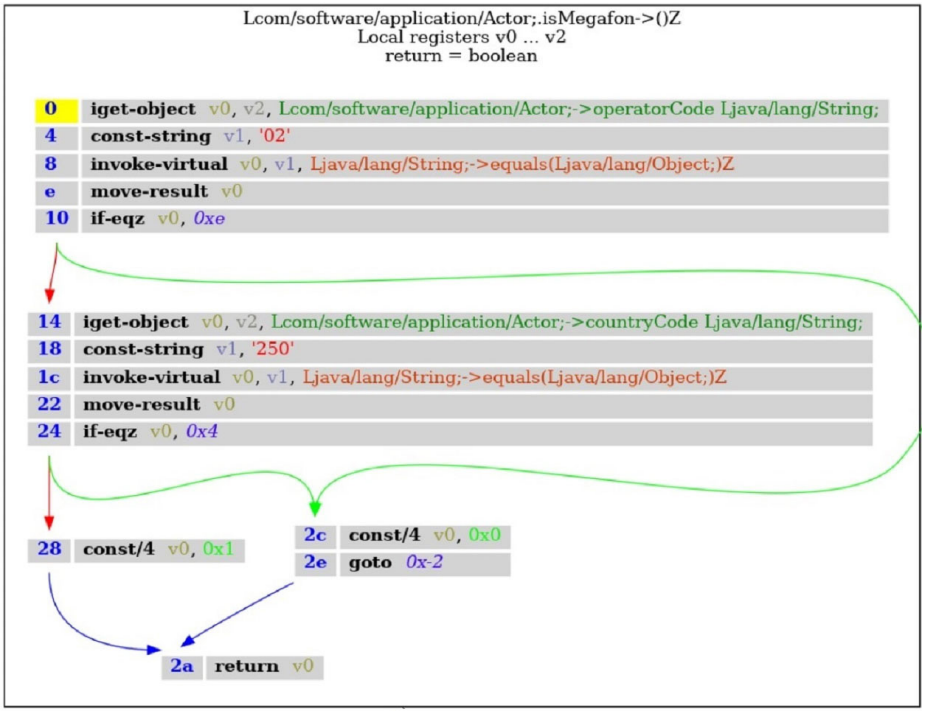

Fig. 3. Control flow graph

In addition, to control flow graphs, it also generates a .ag file(smali like format) for every method class used in the application. The single app can generate n number of .ag files. Each .ag file represents one java method. These .ag files are used for the extraction of opcode sequences. The sample CFG and .ag file for one method are shown in Figures 3 and 4. The opcode sequence "iget-object, const-string, invoke-virtual, move-result, if-eqz, iget-object, const-string, invoke-virtual, move-result, if-eqz, const/4, return, const/4, goto" is generated from Figure 4. The procedure for extracting all opcode sequences from android apk is shown in algorithm-1. The apk file was decompiled with androguard tool. It produces one output folder with different subfolders with several CFGs and .ag files. The extraction process of opcodes from all .ag files is as follows: Initially, two lists are created. One list is empty (for adding opcodes) and another list is a list of Dalvik opcodes [22].

```
# Lcom/software/application/Actor;->isMegafon()Z
[access_flags=public]
#
# Parameters:
# local registers: v0...v2
#
# - return:boolean

isMegafon-BB@0x0 : [ isMegafon-BB@0x14 isMegafon-BB@0x2c ]
      0       (00000000) iget-object          v0, v2,
Lcom/software/application/Actor;->operatorCode Ljava/lang/String;
      1       (00000004) const-string         v1, '02'
      2       (00000008) invoke-virtual       v0, v1,
Ljava/lang/String;->equals(Ljava/lang/Object;)Z
      3       (0000000e) move-result          v0
      4       (00000010) if-eqz               v0, +e

isMegafon-BB@0x14 : [ isMegafon-BB@0x28 isMegafon-BB@0x2c ]
      5       (00000014) iget-object          v0, v2,
Lcom/software/application/Actor;->countryCode Ljava/lang/String;
      6       (00000018) const-string         v1, '250'
      7       (0000001c) invoke-virtual       v0, v1,
Ljava/lang/String;->equals(Ljava/lang/Object;)Z
      8       (00000022) move-result          v0
      9       (00000024) if-eqz               v0, +4

isMegafon-BB@0x28 : [ isMegafon-BB@0x2a ]
      10      (00000028) const/4              v0, 1

isMegafon-BB@0x2a :
      11      (0000002a) return               v0

isMegafon-BB@0x2c : [ isMegafon-BB@0x2a ]
      12      (0000002c) const/4              v0, 0
      13      (0000002e) goto                 -2
```

Fig. 4. .ag file format

If the .ag file contains a string that matches with the opcode string, then it is added to the opcode sequence list (empty). The opcode sequences with lengths less than 10 are discarded. The number of .ag files are also restricted as 3 to reduce the complexity

of the model. This process is repeated for all .ag files to produce the final list of lists with all opcode sequences of that apk.

As the android app uses several java methods (say k), k number of sequences are generated from single apk. If the class label of an apk with opcode sequences: S1, S2, S3, S4……Sk (Here each Si represents one opcode sequence) is 'malware/benign', then all Si's are assigned with the same class label ('malware/benign'). Algorithm-1 is applied on malware and benign apps separately. The algorithm-1 is applied n number of times (n is number of apps). After applying algorithm n times, n number of output files are created. These output files produces m number of sequences (m can be any number). All the sequences with attached class labels are combined and the final dataset is generated.

Algorithm-1: Generation of opcode sequences from apk:

```
Input: One apk, Two lists: dalvik opcode list, opcode
sequence list (Initially it is empty list).
Output: List of lists with opcode sequences.

Step 1: Apply androguard (with decompile command) on
     android apk to generate output folder (contains
     several cfgs, .ag files)

Step 2: for all files in output folders

  Step 2.1: if file ends with .ag

    Step 2.1.1: read each line of String (text) from the
           .ag File

    Step 2.1.2: If the string (text) in .ag file matches
           with dalvik opcode list, add the text (op
           code) to opcode sequence list.

    Step 2.1.3: Discard the sequence if its length less
           than 10.
```

4.3 Applying RNN/LSTM

The total number of opcode sequences created are 2, 39, 650. The opcode sequences generated from benign apps are more than the sequences generated by malware apps. To balance the dataset, Stratified K-Fold cross validation technique applied for experimentation. LSTM is applied to the dataset. All experiments are conducted on the Google Colab-Keras GPU environment. The deep learning architecture (LSTM) for android malware detection is shown in Figure 5.

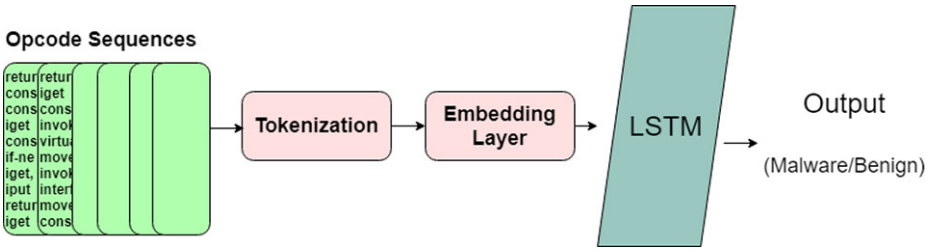

Fig. 5. Deep learning architecture for malware detection

The opcode sequences are in text format. So, before applying LSTM, text processing techniques should be applied. But algorithm-1 generates clean opcode sequences. So, there is no need to apply text preprocessing techniques (like removing stop words, stemming, removing unnecessary symbols). Keras Tokenizer API applied for converting the list of opcodes into a list of integers. Later Keras Embedding layer was applied. The embedding layer converts the integer representation of opcodes into word embedding. The maximum length of the sequence is set to 200. After that, an LSTM layer is added. 'Adam' optimizer is used for balancing binary cross-entropy. The model is trained with 100 epochs. The accuracy obtained after applying LSTM is 96%.

4.4 Comparison with previous work

Table 2 shows the accuracy comparison of proposed framework with the previous work. In [24], the authors applied LSTM for system call sequences for malware detection and achieved an accuracy of 93.10%.

Table 2. Accuracy comparison with previous work

Method	Accuracy
LSTM with system call sequences [24]	93.10%
CNN+LSTM [23]	91.42%
Proposed method	96%

In [23], the authors applied CNN+LSTM model on features collected from control flow graphs and achieved an accuracy of 91.42%. In this paper, the authors applied Recurrent Neural Networks with LSTM framework on extracted opcode sequences in byte code files (.ag files) and achieved an accuracy of 96% (Figure 6).

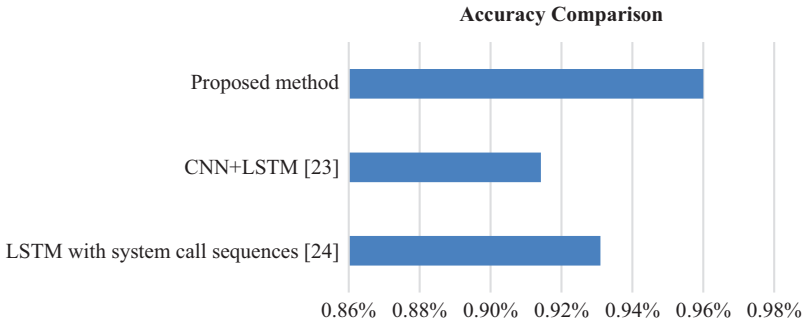

Fig. 6. Accuracy comparison with previous work

5 Conclusion & future scope

In this paper, a deep learning Recurrent Neural Networks model was proposed for android malware detection. Dalvik opcode sequences are extracted from raw byte code files and trained Recurrent Neural Networks with Long Short-Term Memory. The opcode sequences are extracted from malware and benign samples and a feature vector is created. Finally, RNN with LSTM framework applied on the feature vector sequences and achieved an accuracy of 96%. Experimental results have shown that Recurrent Neural Networks outperform traditional Machine Learning models for android malware detection. Although the proposed model achieved good accuracy compared to previous works, malware apps with similar code as benign apps may be detected as benign apks. So, there is a need to add some more features to differentiate malware and benign apps. But adding more features introduces complexity in the malware detection model. In this work, the authors applied the proposed methodology to 1500 apks (750 malware apps and 750 benign apps). The performance of the proposed model may vary if the number of samples in the dataset increases. In future, We intend to work with more android features. It is also planned to apply the proposed methodology to larger datasets.

6 References

[1] https://www.statista.com/statistics/272307/market-share-forecast-for-smartphone-operating-systems/

[2] Dr. S. V. Manikanthan, Dr. T. Padmapriya, "Artificial Intelligence Techniques for Enhancing Smartphone Application Development on Mobile Computing", International Journal of Interactive Mobile Technologies (iJIM), vol. 4, no. 17, 2020. https://doi.org/10.3991/ijim.v14i17.16569

[3] A. George Daniel, M. Spreitzenbarth, M. Hubner, H. Gascon, K. Rieck, "DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket", NDSS (Network and Distributed System Security Symposium), 3–26 February 2014, 10.14722/ndss.2014.23247

[4] R. Hemant, Sanjay K Sahay, T. Shivin, S. Mohit, "Detection of Malicious Android Applications: Classical Machine Learning vs. Deep Neural Network Integrated with Clustering", In: Gao H., J. Durán Barroso R., Shanchen P., Li R. (eds) Broadband Communications, Networks, and Systems. BROADNETS 2020. LNICSSITE, vol. 355, pp. 109–128. https://doi.org/10.1007/978-3-030-68737-3_7

[5] Y. Zhang, B. Li, "Malicious Code Detection Based on Code Semantic Features", IEEE Access, vol. 8, pp. 176728–176737, 2020, doi: https://doi.org/10.1109/ACCESS.2020.3026052

[6] A. Lakshmanarao, M. Shashi, "Android Malware Detection Using Convolutional Neural Networks", In: Data Engineering and Intelligent Computing. Advances in Intelligent Systems and Computing, volume 1. Springer, 2021, Singapore, pp. 151–162. https://doi.org/10.1007/978-981-16-0171-2_15

[7] W. Zoabi Wang, M. Zhao, J. Wang, "Effective android malware detection with a hybrid model based on deep autoencoder and convolutional neural network", Journal of Ambient Intelligence and Humanized Computing, vol. 10, 3035–3043, 2019. https://doi.org/10.1007/s12652-018-0803-6

[8] A. Mahindru, P. Singh, "Dynamic Permissions Based Android Malware Detection Using Machine Learning Techniques", In Proceedings of the 10th Innovations in Software Engineering Conference, February-2017, Association for Computing Machinery, pp. 202–210. https://doi.org/10.1145/3021460.3021485

[9] Y. Yang, X. Du, Z. Yang, X. Liu, "Android Malware Detection Based on Structural Features of the Function Call Graph", Electronics 2021, 10, 186. https://doi.org/10.3390/electronics10020186

[10] X. Li, J. Liu, Y. Huo, R. Zhang and Y. Yao, "An Android malware detection method based on Android Manifest file," 2016 4th International Conference on Cloud Computing and Intelligence Systems (CCIS), 2016, pp. 239–243, doi: https://doi.org/10.1109/CCIS.2016.7790261

[11] Abikoye Oluwakemi Christiana, Benjamin Aruwa Gyunka, "Optimizing Android Malware Detection Via Ensemble Learning," International Journal of Interactive Mobile Technologies (iJIM), vol. 14, no. 9, 2020. https://doi.org/10.3991/ijim.v14i09.11548

[12] D. Li, L. Zhao, Q. Cheng, N. Lu, W. Shi, Opcode Sequence Analysis of Android Malware by a Convolutional Neural Network. Concurrency Computat Pract Exper. 2020; 32:e5308. https://doi.org/10.1002/cpe.5308

[13] H.M. Unver, K. Bakour, "Android Malware Detection Based on Image-Based Features and Machine Learning Techniques", SN Appl. Sci. 2, 1299 (2020). https://doi.org/10.1007/s42452-020-3132-2

[14] A. Roy, S.J. Divjeet, J. Gitanjali, S. Kapil, "Android Malware Detection based on Vulnerable Feature Aggregation. Procedia Computer Science", vol. 173, 2020, ISSN 1877-0509, pp. 345–353. https://doi.org/10.1016/j.procs.2020.06.040

[15] P. Agrawal, B. Trivedi, "A Survey on Android Malware and their Detection Techniques", In: IEEE International Conference on Electrical, Computer and Communication Technologies (ICECCT), 2019, pp. 1–6. https://doi.org/10.1109/ICECCT.2019.8868951

[16] H. Zhu, Y. Li, R. Li, J. You, H. Song, "SEDMDroid: An Enhanced Stacking Ensemble of Deep Learning Framework for Android Malware Detection", IEEE Transactions on Network Science and Engineering, doi: 10.1109/TNSE.2020.2996379-2020

[17] Oluwakemi Christiana Abikoye, Benjamin Aruwa Gyunka, "Android Malware Detection through Machine Learning Techniques: A Review", International Journal of Online and Biomedical Engineering (iJOE), vol. 16, no. 2, 2020. https://doi.org/10.3991/ijoe.v16i02.11549

[18] https://github.com/androguard/androguard

[19] David E. Rumelhart, Geoffrey, E. Hintont, Ronald J. Williams, "Learning representations by back-propagating errors", Nature, 1986, vol. 323, pp. 533–536. https://doi.org/10.1038/323533a0
[20] S. Hochreiter, J. Schmidhuber, "Long Short-Term Memory. Neural Computation", vol. 9, no. 8, November 15, 1997, pp. 1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735
[21] online: unb.ca/cic/datasets/andmal2017.html
[22] https://developer.android.com/reference/dalvik/bytecode/Opcodes.html
[23] A. Pektaş, T. Acarman, "Learning to detect Android malware via opcode sequences. Neurocomputing", 396, 599–608, 2020, ELSEVIER. https://doi.org/10.1016/j.neucom.2018.09.102
[24] S. Xiao, X., Zhang, S., Mercaldo, F. et al. Android malware detection based on system call sequences and LSTM. Multimed Tools Appl., vol. 78, 3979–3999 (2019). https://doi.org/10.1007/s11042-017-5104-0

7 Authors

A. Lakshmanarao is currently working as Associate Professor in Aditya Engineering College, Surampalem.He completed his B. Tech in CSIT and M.Tech in Software Engineering. He is pursuing Ph.D. in Andhra University, Visakhapatnam. His areas of interest are Machine Learning, Cyber Security, Deep Learning. He is a member of Computer Society of India (CSI).

Prof. M Shashi received her B.E in Electrical and Electronics Engineering and M.E in Computer Science Engineering. She received her Ph.D. in 1994 from Andhra University and obtained best PhD thesis award. She is currently working as a Professor in the Department of Computer Science and Systems Engineering, Andhra University, Visakhapatnam, Andhra Pradesh, India. Her areas of interest are Data Analytics, Data Warehousing & Mining, AI, and Data Structures. She is a member of IEEE, ISTE, CSI and Fellow of Institute of Engineers (India).

Article submitted 2021-08-23. Resubmitted 2021-10-07. Final acceptance 2021-10-18. Final version published as submitted by the authors.