

# ANDROID MALWARE DETECTION USING MULTIMODAL DEEP LEARNING METHOD

<sup>1</sup> AFSANA, <sup>2</sup> Dr. SAYED ABDULHAYAN

<sup>1</sup> Student of M.tech <sup>2</sup> Associate Professor

<sup>1</sup> Dept. Of TCE, <sup>2</sup> Dept. Of TCE,

<sup>1&2</sup> Dayananda Sagar College of Engineering, Bangalore, Karnataka, India

**ABSTRACT:** With the widely uses of smart phones, the number of malware has been increasing exponentially. Among android devices, smart devices are the most targeted devices by malware because of their growing popularity. In this paper proposes a novel framework for Android malware detection. The framework utilities various kinds of features to reflect the properties of Android applications from various aspects, and the features are refined using the similarity-based feature extraction or existence based method for effective feature indication on malware detection. Besides, a multimodal learning method is proposed to be used as a malware detection model.

**Keywords :** Android malware, Machine learning , intrusion detection , malware detection, neural network.

## I. INTRODUCTION

With the popularly growing of mobile devices such as tablets or smart phones, attacks on the mobile devices have been increasing. Mobile malware is one of the most dangerous and damaging threats which cause various security incidents also financial damages. In 2017 according to the G DATA report [1], security experts are discovered almost 750,000 new Android malware during the first quarter of 2017. It is expected that a large number of mobile malware will keep spread to commit various cybercrimes on mobile system or devices. And android is a mobile operating system that is most targeted by mobile malware or virus because of the popularly growing of Android devices. In the addition to the number of Android devices, there is one more reason that leads malware authors to develop Android malware.

So far to mitigate the attacks by Android malware, various research approaches have been proposed. The malware identification approaches can be classified in two categories;1: static analysis based detection [2-19],2: dynamic analysis based detection [20-24]. The static analysis based methods are used syntactic features that can be extracted without executing an application, and the dynamic analysis based methods use semantic features and that can be monitored when an application is executed in a controlled environment. Static analysis has a uses that it is unnecessary to set the execution environments, and then the computational overheads for static analysis are relatively less. Dynamic analysis has a uses that it is possible to handle the malicious applications which use some obfuscation techniques such as code packing or encryption.

Because the reason is that the Android operating system allows users can install applications and downloaded from third-party markets and attackers can seduce or mislead Android users to suspicious applications from attackers’ servers or download malicious.

## II. PROPOSED FRAMEWORK

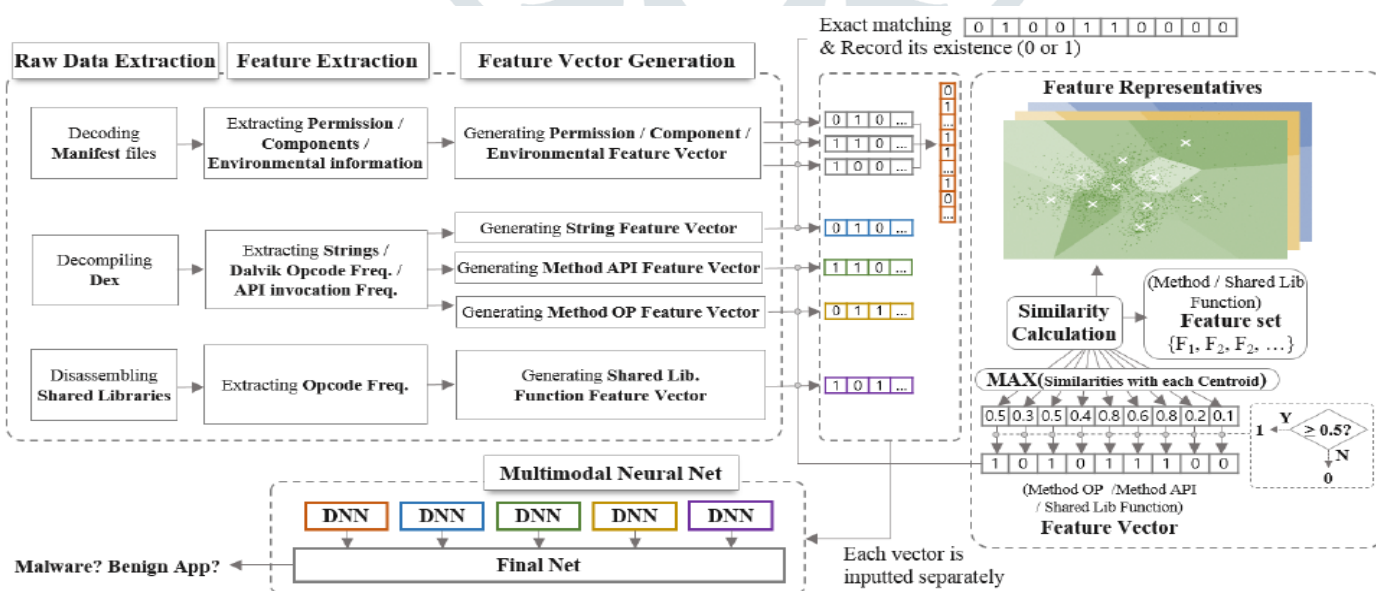


Figure 1 : The overall architecture of the proposed framework

Fig.1 shows overall architecture of the framework, and the framework uses seven kinds of the feature; Method opcode feature, string feature, method permission feature, shared library function opcode feature, API feature, environmental feature, and component feature. Using these features, three corresponding feature vectors are generated first then, among them, permission/component/predefined setting feature vectors are merged in one feature vector. Finally, five feature vectors are fed into classification model for malware detection. The

framework conducts four major processes for detection; the raw data extraction process, feature extraction process, feature vector generation process, and detection process. These processes are explained as follows,

#### A. Raw Data Extraction Process.

This process is performed to make Android APK (Android Package Kit) files interpretable. To extract the raw data, an APK file is unzipped, and manifest file, a dex file, and shared library files are extracted first. Manifest file and dex file are decoded or disassembled by APK tool [32], and then shared library files (i.e. .so files) in the package can be disassembled by IDA Pro [33].

#### B. Feature Extraction Process.

This process is conducted to obtain the essential feature data from the raw data. The method opcode features and method API features are extracted from smali files which are disassembled results of a dex file. The smali file is separated into method blocks, and by scanning Dalvik bytecodes, Dalvik opcode frequency of each method is calculated. In addition, during the byte code scanning, it is checked whether the invocation of dangerous APIs exists in the method, and then dangerous API invocation frequency of each method is calculated. In case of string features, strings are simply collected from whole smali file without considering the method separation. Shared library function opcode features are extracted from instruction sequences of the disassembled code of .so files. And instruction sequence of each function is scanned to extract information of the assembly opcode frequency. The permission features, component features, and the environmental features are extracted from manifest XML file. While visiting XML tree nodes, each nodes are checked to confirm whether node contains information about permissions, application components, and so on.

#### C. Feature Vector Generation Process.

The extracted features in previous process are used to compose the feature vectors. Seven kinds of the feature vector are produce from extracted features. The seven feature vectors are divided in two types according to their feature representations: 1:existence-based feature vectors and 2:similarity-based feature vectors. The existence-based feature vector is feature vector whose elements only represent the existence of features in malicious feature database, and examples strings, permission, component, and environmental feature vectors. And the similarity-based feature vector is the feature vectors whose elements are similar to the malware representatives in the malicious feature database, and the method opcode, the method API and the shared library function feature vectors are the similarity-based feature vectors. The malicious feature database here is repository that contains features and malware representatives of known malicious applications. To improve the performance of the framework, we clarified the feature vector with the predefined threshold value. The similarity values are exceeding the predefined similarity threshold value become one. Otherwise, it will set to zero. This refinement removes the features that are not similar to the certain malware representative but have small similarity values, and it is simplifies the computation in the learning process.

#### D. Detection Process

After that all the seven feature vectors are generated in previous process and detection process is conducted to determine whether the given application is malicious or not. Before testing the feature vectors with the detection model, the permission feature vector, the component feature vector, and the environmental feature vector are mixed into a single feature vector. Therefore, this model gets the five feature vectors and the performs mathematical operations in each layer. Then if all the operations are conducted completely, then the model produces the estimated label for the given input application.

### III. THE DEFINITION OF FEATURES

Diverse features could be useful to reflect the characteristics of an application. And some features like environmental information are indirectly related to malicious activities, those features may contribute to defining the application characteristics. Our proposed framework uses the following features: String feature, Method opcode feature, Method API feature. In this framework, the learning algorithm is utilized to classify the unknown samples into the malware class or the benign class. The learning algorithm creates a neural network model that can be derive the better classification accuracy by updating the weight of each neuron input. The degree of influence of the characteristic on classification is find according to the weight of the neurons affected by the feature. If there is an insignificant feature in the classification, the weight of the relevant neurons is reduced. Therefore, each feature can be used differently by their contributions. The next sections explain each and every feature type that is used in this framework. It is denoted that the characteristics are converted into the feature vectors to apply them to the neural network.

#### A. String Feature

In this feature extracted from a set of string values in smali files. The feature extraction module collects all of the operand values with types of const-string and const-string/jumbo. And there are also the Dalvik opcodes that move a reference to the string into a specific register. The number of strings in this application spans a wide range. And then if the number of applications increases, then the number of strings from those applications also increase explosively. So that, strings are hashed, and then hashed values of strings are applied to this modular operation. And the hash function applied in the framework as the hash function.

## B. Method opcode and API Feature

In Dalvik opcode frequency and API invocation frequency of methods may also imply application behaviors and coding habits of the developer. Due to this reason, Dalvik opcode frequency and API invocation frequency of methods are utilized to define the method features. The method opcode frequency can be calculated by scanning the byte code in each and every method. In case of the API invocation frequency and the byte codes for API invocation are checked to count the API invocations in each and every method. To capture malicious behaviors, invocations of only the selected APIs are counted. And the APIs that might be utilized in malicious activities are investigated manually using the Android Developer reference pages [50]. And additionally, the APIs that were introduced in [35] are also included to the selected API list. According to [35], these selected APIs are useful to distinguish malware and benign applications.

## C. Shared Library Function Opcode Feature

Android provides the Java Native Interface (JNI) and then allows applications to incorporate native libraries. It is also known that native code defeats Android security mechanisms because the native code is not covered by using security model. For example, shared library files can be used to avoid countermeasure against attacks or to hide malicious behaviors. This is the reason why many malicious applications used in the native code to attack the Android system. To prevent these kind of malwares with native code from hiding its behaviors, this framework defines and utilizes the shared library function features in the detection. In the same way to the method feature extraction, The ARM opcode frequency and the system call invocation frequency are extracted from native code. When it is scanning the disassembled code of each function, the opcodes and system call invocations in each and every function are counted.

## IV. MULTIMODAL NEURAL NETWORK

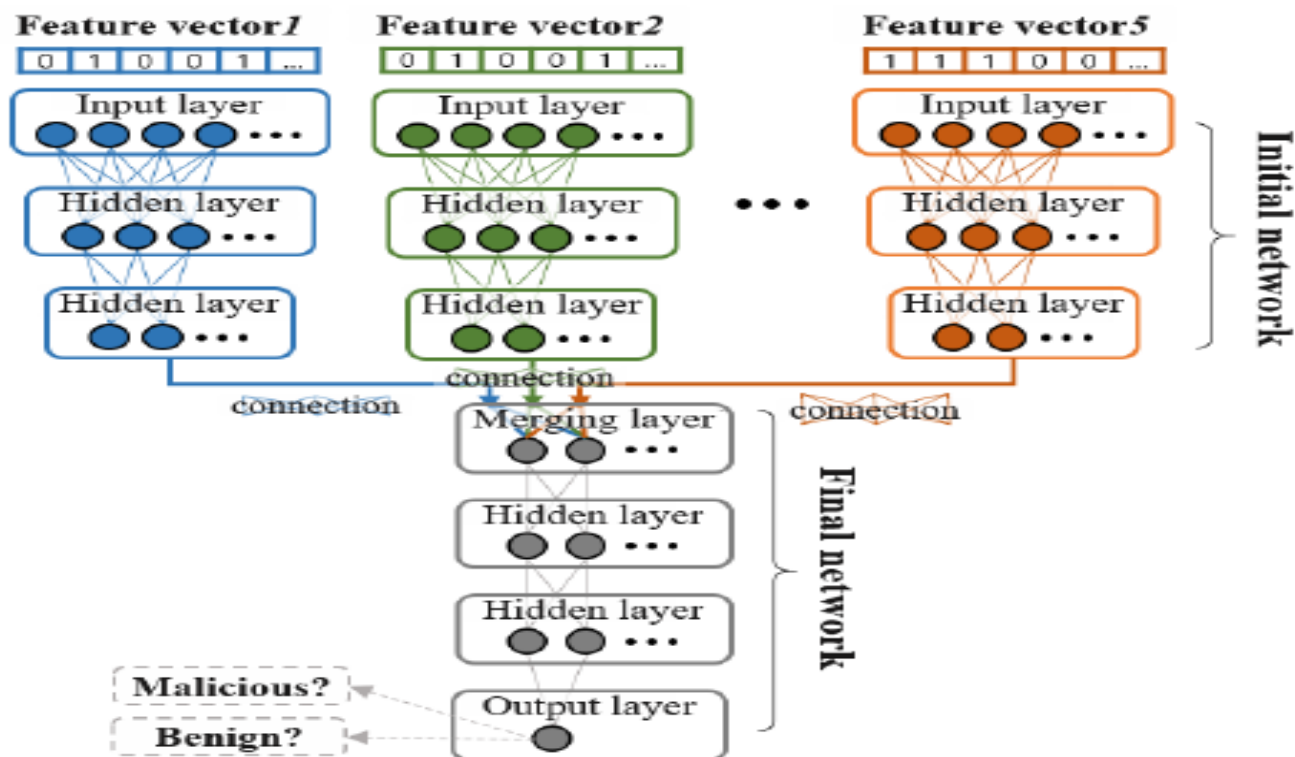
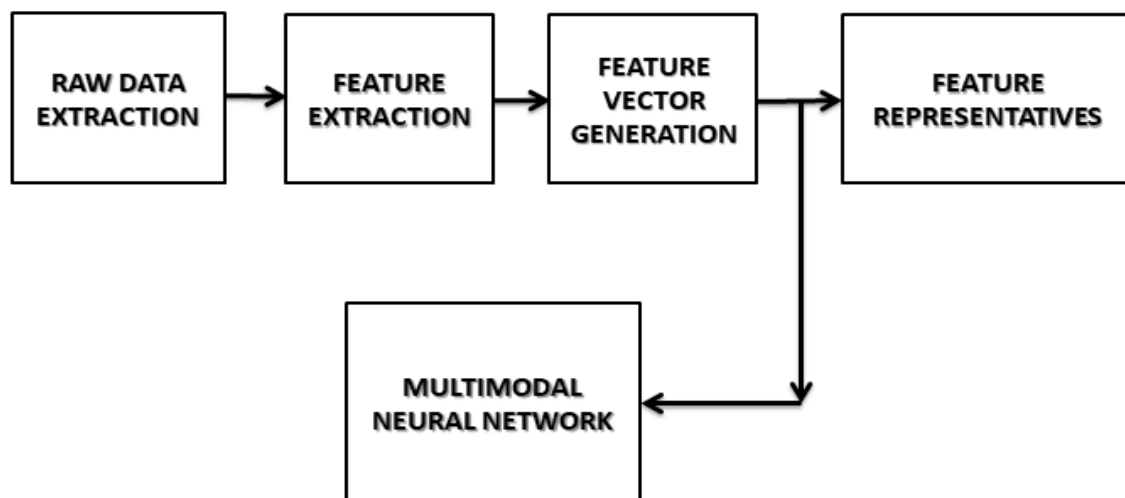


Figure 2 : Multimodal deep neural network

Fig. 2 indicates the architecture of the multimodal deep neural network for malware detection in this framework. This project proposed neural network model utilizes five feature vectors, and each and every vector is inputted separately to the initial networks which consist of five DNNs (Deep Neural Network). The initial networks are not connected to one another, and then the last layers of the initial networks are connected to the merging layer that is the first layer of the last network. The last network is a DNN, and it produces the classification outputs. And each DNN of the initial networks includes of an input layer and two hidden layers, and all layer only receives connections from the previous layer. All layers are fully-connected, and then the activation functions used in this DNNs and they are the rectified linear units (ReLU) activation function. The ReLU activation functions are utilized to prevent the vanishing gradient problem in training, and it also makes our model computationally efficient.

In the final network is a same shape of the DNNs to the initial network except for the first and the last layers. The merging layer, is connected with the last layers of the DNNs of the initial networks. The last layer of the final network, that is output layer, produces the classification output. In the output layer, there is only one neuron that uses the sigmoid function to label the input application as benign or a malware application.

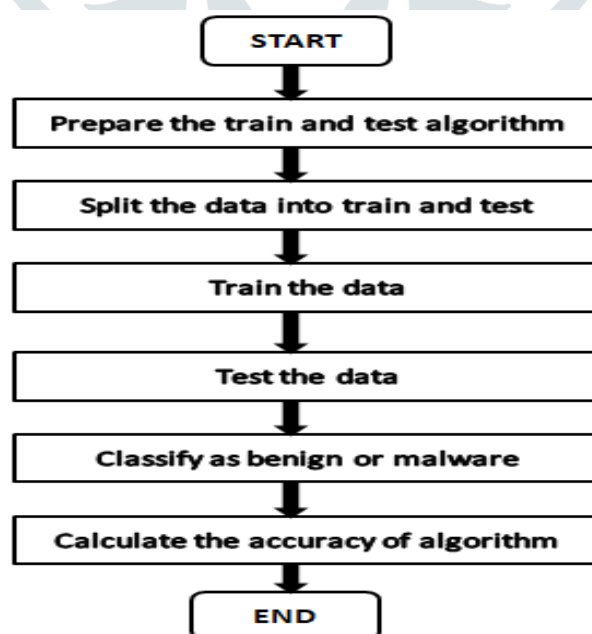
## V. BLOCK DIAGRAM



**Figure 3 : Block diagram of proposed framework.**

The above figure shows the block diagram of the framework, and the framework uses five kinds of the stages; Raw data extraction, feature extraction, feature vector generation, feature representatives and multimodal neural network. Raw data extraction process will be performed to make the Android APK files to interpretable. Then the feature extraction process will be conducted to obtain essential feature data from the given raw data. Then the extracted features are in the previous process will be use to compose the feature vectors. After all of those feature vectors are generated in previous process, then the detection process will be conducted to determine whether that the given application is malicious or not. Then before examining that of the feature vectors with detection model, permission feature vector, component feature vector, and environmental feature vector are merged into the single feature vector. Therefore, model gets the five feature vectors to prevent detected malicious.

## VI. FLOW CHART



**Figure 4 : Flow chart of the proposed framework.**

The above shown figure is the flow chart of the proposed framework that includes the working flow of the proposed algorithm. At first we need to collect the data set which includes the benign and the malware data set and then we need to prepared the train and test

algorithm. After that the collected data will be splitted according to the test and train purpose. The data which is considered for the training purpose is fed to the algorithm and it will be trained after that the data which is splitted for the testing purpose that will be taken and fed for the testing purpose. After testing the data the proposed feature vector generation method will be make out the difference between the malware and the benign applications and it will describe that how many malware and how many benign data are present. So on the basis of the predicted value and the actual value got the accuracy of the model will be concluded.

### VII. RESULTS

The multimodal deep neural network is used for the android malware detection which is helpful for the more accuracy as it includes the more number of iterations so it is possible to get the more accurate results. The feature vector generation method which is proposed here is very much helpful for the detection of the features of the malware and the benign applications because as compare to their characteristics the malware and the benign applications are having the more number of common properties so as considering the each bit of the collected data sample it is possible to get the proper result. Here the result will be analyzed in the form of confusion matrix , bar graph ,pie chart , precision and recall , by evaluating all these features it is possible to get the 100% accurate algorithm.

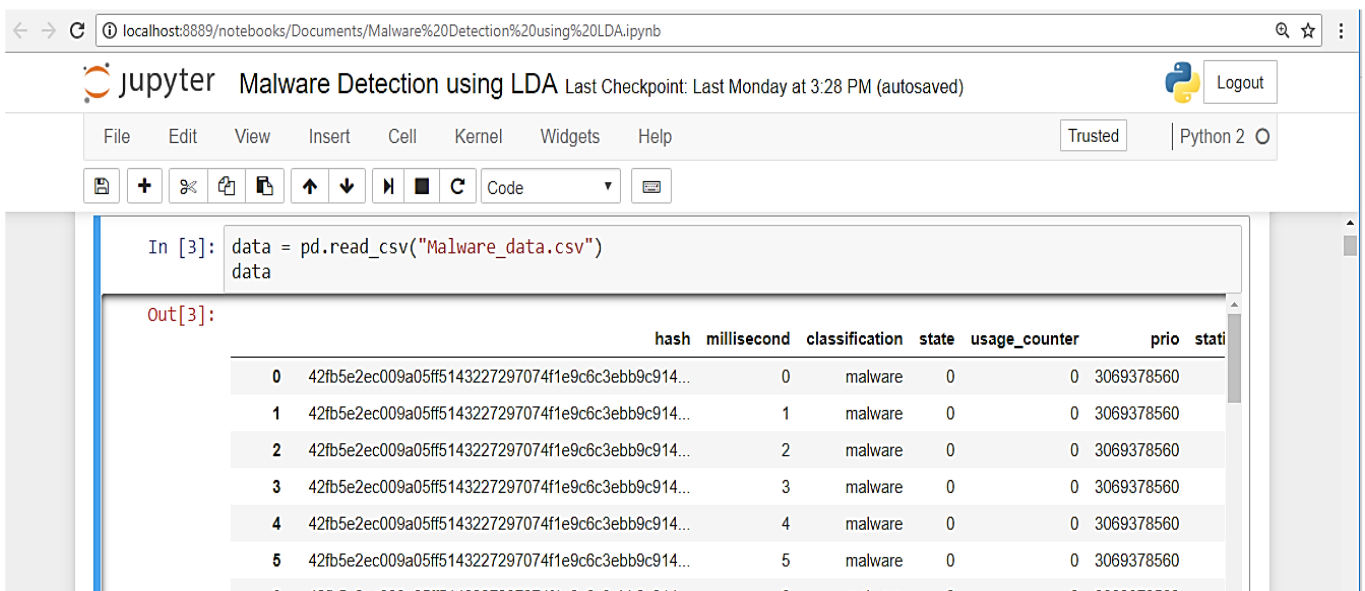


Figure 5 : Storing CSV file into variable called data

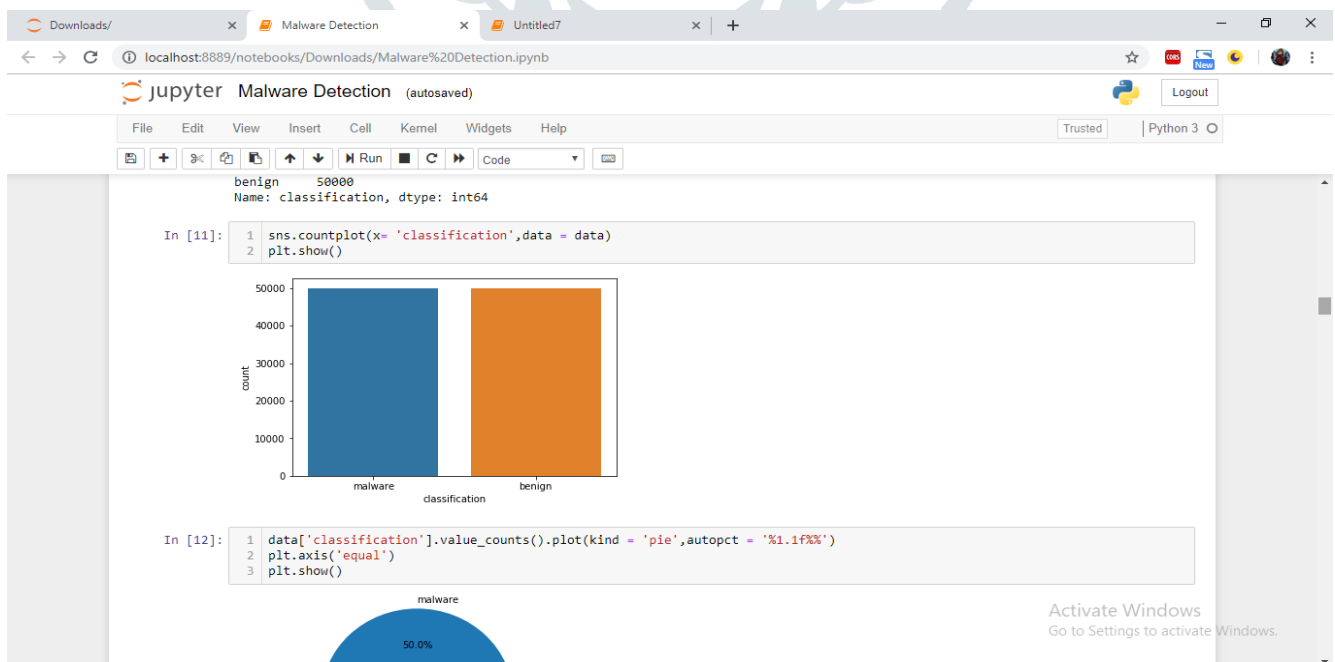


Fig 6: Accuracy in bar graph for equal amount of benign and malware samples.

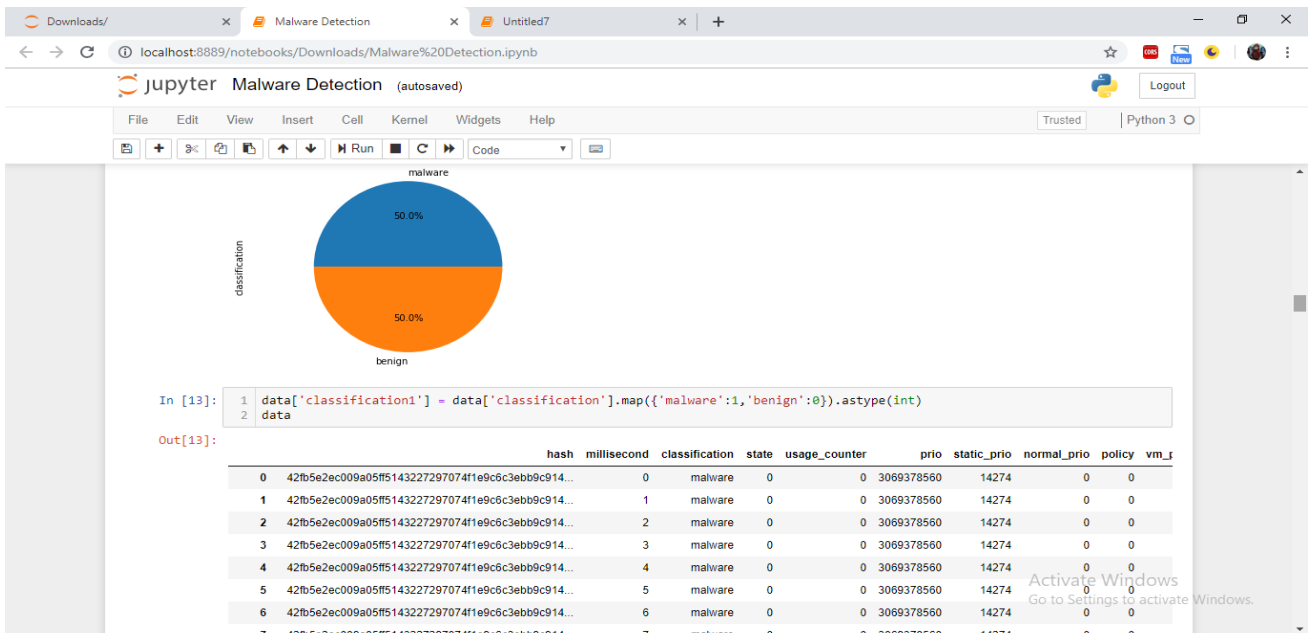


Fig 7: Accuracy in pie chart for equal amount of benign and malware samples.

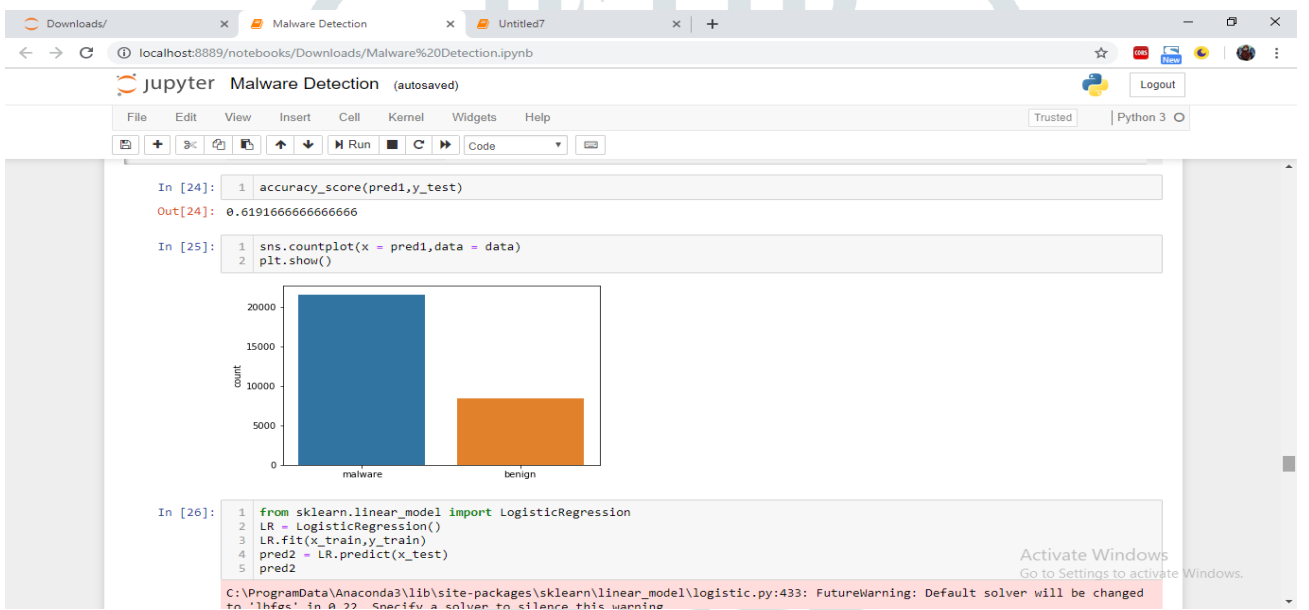


Fig 8: Accuracy in bar graph for unequal amount of benign and malware samples.

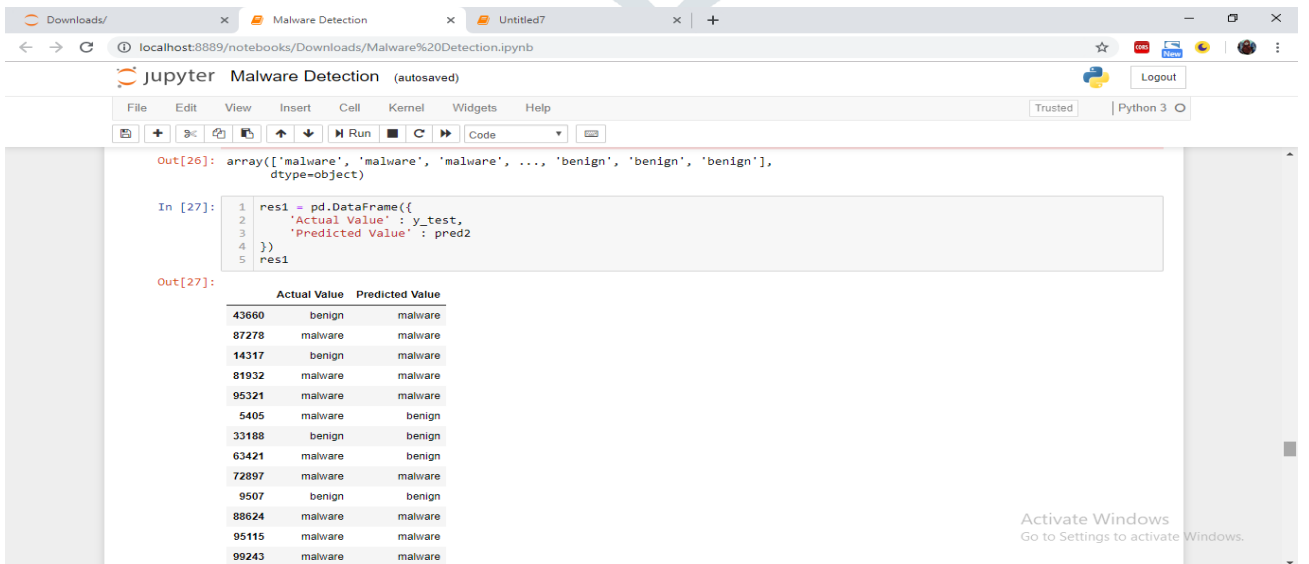
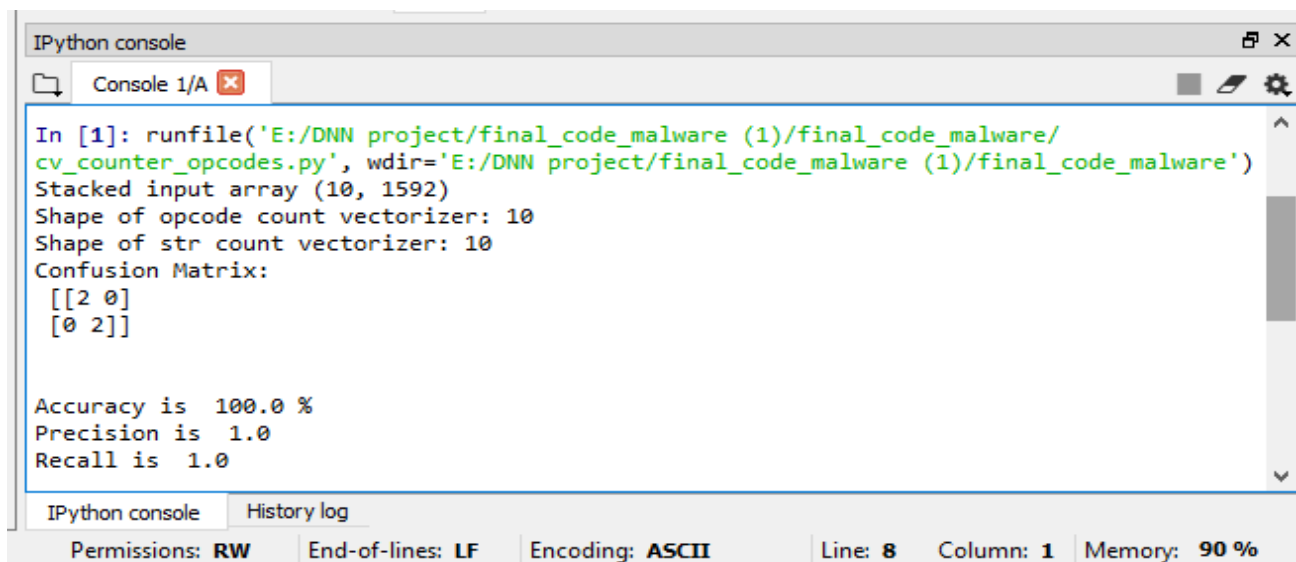


Fig 9 : List of actual value and predicted value



```

IPython console
Console 1/A
In [1]: runfile('E:/DNN project/final_code_malware (1)/final_code_malware/
cv_counter_opcodes.py', wdir='E:/DNN project/final_code_malware (1)/final_code_malware')
Stacked input array (10, 1592)
Shape of opcode count vectorizer: 10
Shape of str count vectorizer: 10
Confusion Matrix:
[[2 0]
 [0 2]]

Accuracy is 100.0 %
Precision is 1.0
Recall is 1.0

IPython console History log
Permissions: RW End-of-lines: LF Encoding: ASCII Line: 8 Column: 1 Memory: 90 %

```

Fig 10 : Accuracy of equal amount of benign and malware samples in the form of confusion matrix precision and recall.

## VIII. CONCLUSION

In this paper, the project proposes a novel Android malware detection framework and that utilizes so many static features to reflect the properties of applications in the various aspects. Total three kinds of feature are extracted by comparing files such as a manifest file, a dex file, and a .so file from APK file, and those features enrich the extracted information to the express applications' characteristics. In all addition, this project suggested the effective feature vector generation form which is appropriate to detect malware that is similar to benign applications. Through this project proposed feature representation, it is possible to prevent feature vector of malware from including the common properties that appear in the benign applications. Finally, in this project uses the multimodal learning method, which are designed to deal with various kinds of feature type. Different types of the feature are mainly used to train the initial networks, and the results of the initial networks are subsequently used to train the last network. This architecture of the model is suitable for the framework to improve the malware detection exact accuracy.

## IX. REFERENCES

- [1] G DATA Report, "8,400 new android malware samples every day".
- [2] Ch.-Y. Huang, Y.-T. Tsai, and C-H. Hsu, "Performance evaluation on permission-based detection for android malware," *Advances in Intelligent Systems and Applications*, vol. 2, pp. 111-120, 2018.
- [3] A. Zarni, and W. Zaw, "Permission-based android malware detection," *International Journal of Scientific and Technology Research*, vol. 2, no. 3, pp. 228-234, 2016.
- [4] D. Luke, V. Notani, and A. Lakhotia, "Droidlegacy: Automated familial classification of android malware," In *Proc. of the ACM SIGPLAN on Program Protection and Reverse Engineering Workshop*, pp. 3, 2017.
- [5] M. Zhang, Y. Duan, H. Yin, and Z. Zhao, "Semantics-aware Android malware classification using weighted contextual API dependency graphs," In *Proc. of the ACM Conference on Computer and Communications Security (CCS)*, pp. 1105-1116, 2017.
- [6] M. Grace, Y. Zhou, Q. Zhang, S. Zou, X. Jiang, "Riskranker: scalable and accurate zero-day android malware detection," In *Proc. of the ACM International Conference on Mobile Systems, Applications, and Service (Mobisys)*, pp. 281-294, 2017.
- [7] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, K. Rieck, "DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket," In *Proc. the Network and Distributed System Security Symposium (NDSS)*, vol. 14, pp. 23-26, 2017.
- [8] D-J. Wu, C-H. Mao, T-E. Wei, H-M. Lee, K-P. Wu, "Droidmat: Android malware detection through manifest and api calls tracing," In *Proc. of the Asia Joint Conference on Information Security (Asia JCIS)*, pp. 62-69, 2016.
- [9] W. Zhou, Y. Zhou, X. Jiang, P. Ning, "Detecting repackaged smartphone applications in third-party android marketplaces," In *Proc. of the ACM conference on Data and Application Security and Privacy*, pp. 317-326, 2015.
- [10] S. Hao, B. Liu, S. Nath, W. G. Halfond, and R. Govindan, "PUMA: Programmable UI-automation for Large-scale Dynamic Analysis of Mobile Apps," In *Proc. of the ACM International Conference on Mobile Systems, Applications, and Services (MobiSys)*, pp. 204-217, 2014.
- [11] Y. Zhongyang, Z. Xin, B. Mao, L. Xie, "DroidAlarm: an all-sided static analysis tool for Android privilege-escalation malware," In *Proc. of the 8th ACM SIGSAC symposium on Information, computer and communications security*, pp. 353-358, 2013.
- [12] E. Chin, A. P. Felt, K. Greenwood, D. Wagner, "Analyzing inter-application communication in Android," In *Proc. of the international conference on Mobile systems, applications, and services*, pp.239-252, 2016.
- [13] L. Lu, Z. Li, Z. Wu, W. Lee, G. Jiang, "Chex: statically vetting android apps for component hijacking vulnerabilities," In *Proc. of the ACM conference on Computer and communications security*, pp. 229-240, 2016.
- [14] P. PF Chan, L. CK Hui, SM. Yiu, "Droidchecker: analyzing android applications for capability leak," In *Proc. of the ACM conference*

on Security and Privacy in Wireless and Mobile Networks, pp. 125-136, 2015.

- [15] K. Lu, Z. Li, V. P. Kemerlis, Z. Wu, L. Lu, "Checking More and Alerting Less: Detecting Privacy Leakages via Enhanced Data-flow Analysis and Peer Voting," In *Proc. of the Network and Distributed System Security Symposium (NDSS)*, 2017.
- [16] W., Fengguo, S. Roy, X. Ou. "Amandroid: A precise and general inter-component data flow analysis framework for security vetting of android apps," In *Proc. of the 2014 ACM Conference on Computer and Communications Security*, pp. 1329-1341, 2017.
- [17] J. Sum, K. Yan, X. Liu, Ch. Yang, Y. Fu, "Malware Detection on Android Smartphones using Keywords Vector and SVM," In *Proc. of the IEEE/ACIS Conference on Computer and Information Science*, pp. 833-838, 2018.
- [18] A. Narayanan, L. Yang, L. Chen, L. Jinliang, "Adaptive and Scalable Android Malware Detection through Online Learning," In *Proc. Of the International Joint Conference on Neural Networks (IJCNN)*, pp. 2484-2491, 2018.
- [19] A. Demontis, M. Melis, B. Biggio, D. Maiorca, D. Arp, K. Rieck, I. Corona, G. Giacinto, F. Roli, "Yes, Machine Learning Can Be More Secure! A Case Study on Android Malware Detection," *IEEE Transaction on Dependable and Secure Computing*, 2016.
- [20] L. K. Yan, H. Yin, "DroidScope: Seamlessly Reconstructing the OS and Dalvik Semantic Views for Dynamic Android Malware Analysis," In *Proc. of the USENIX Security Symposium*, pp. 569-584, 2015.
- [21] W. Enck., P. Gilbert, S. Han, V. Tendulkar, B-G. Chun, L. P. Cox, J. Jung, P. Macdaniel, A. N. Sheth, "TaintDroid: an information-flow tracking system for realtime privacy monitoring on smartphones," *ACM Transaction on Computer Systems*, vol. 32, no. 5, 2014.
- [22] T. Bläsing, L. Batyuk, A-D. Schmidt, S. A. Camtepe, S. Albayrak, "An android application sandbox system for suspicious software detection," In *Proc. of the Malicious and unwanted software (MALWARE)*, pp. 55-62, 2016.
- [23] A. Shabtai, U. Kanonov, Y. Elovici, C. Glezer, Y. Weiss, "Andromaly: a behavioral malware detection framework for Android devices," *Journal of Intelligent Information Systems*, vol. 38, no. 1, pp. 161-190, 2012.
- [24] A- D.Schmidt, F. Peters, F. Lamour, C. Scheel, S. A. Camtepe, S. Albayrak, "Monitoring smartphones for anomaly detection," *Mobile Network sand Applications*, vol. 14, no. 1, pp. 92-106, 2009.
- [25] R. Pascanu, J. W. Stroke, H. Sanossian, M. Marinescu, A. Thomas, "Malware classification with recurrent networks," In *Proc. of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 1916-1920, 2015.
- [26] O. E. David, N. S. Netanyahu, "Deepsign: Deep learning for automatic malware signature generation and classification," In *Proc. of the International Joint Conference on Neural Networks (IJCNN)*, pp. 1-8, 2017.
- [27] J. Saxe, K. Berlin, "Deep neural network based malware detection using two dimensional binary program features," In *Proc. of the 10<sup>th</sup> International Conference on Malicious and Unwanted Software (MALWARE)*, pp. 11-20, 2015.
- [28] Z. Yuan, Y. Lu, Y. Xue, "Droiddetector: android malware characterization and detection using deep learning," *Tsinghua Science and Technology*, vol. 21, no. 1, pp. 114-123, 2018.
- [29] W. Yu, L. Ge, G. Xu, X. Fu, "Towards Neural Network Based Malware Detection on Android Mobile Devices," *Cybersecurity Systems for Human Cognition Augmentation*, pp. 99-117, 2016.
- [30] N. Mchaughlin, J. Martinez del Rincon, B-J. Kang, S. Yerima, Y. Safaei, E. Trickel, Z. Zhao, A. Doupe, G. Joon Ahn, "Deep Android Malware Detection," In *Proc of the ACM on Conference on Data and Application Security and Privacy (CODASPY)*, pp. 301-308, 2018.
- [31] H. Fereidooni, M. Conti, D. Yao, A. Sperduti, "ANASTASIA: ANdroid mALware detection using STATic analysis of Applications," In *Proc. of the IFIP International Conference on New Technologies, Mobility and Security*, pp. 1-5, 2018.
- [32] APKtool. (September , 2017) [Online]. Available: <https://ibotpeaches.github.io/Apktool/>
- [33] IDA pro. (September , 2017) [Online]. Available: <https://www.hex-rays.com/products/ida/>
- [34] Released Code (September, 2017) [Online]. Available: <https://github.com/cloudio17/A-Multimodal-Deep-Learning-Method-for-Android-Malware-Detection>.
- [35] A. Yousra, W. Du, H. Yin, "DroidAPIMiner: Mining API-level features for robust malware detection in android," In *Proc. of the International Conference on Security and Privacy in Communication Systems*, pp. 86-103, 2016.