International Journal of Engineering Trends and Technology
ISSN: 2231 – 5381 / https://doi.org/10.14445/22315381/IJETT-V70I11P227

Volume 70 Issue 11, 249-257, November 2022

Original Article

Android Malware Detection using Multilayer Autoencoder and Random Forest

A. Lakshmanarao[1], M. Shashi[2]

[1,2]*Department of CS&SE, AU College of Engineering, Andhra University, Visakhapatnam, A.P, India*

[1]*Corresponding Author : laxman1216@gmail.com*

Received: 14 August 2022 Revised: 08 November 2022 Accepted: 17 November 2022 Published: 26 November 2022

Abstract *- One of the most challenging concerns in the world of operating systems and software is the presence of malicious software. The Android operating system is also experiencing the same issues. Because of the significant increase in the refinement of Android malware obfuscation and detection avoidance methods, a significant number of conventional malware investigative techniques have become outdated. The malware detection approach based on earlier signatures is ineffective for detecting unknown threats. In recent years, machine learning and deep learning techniques have proved promising for malware detection. A framework is proposed to extract several features like permissions, opcodes, api packages, system calls, intents, and api calls from Android malware and benign apps and to build a classifier for malware detection using the most suitable machine learning and deep learning algorithms. Based on the performance analysis Random Forest algorithm was identified as the suitable classifier as it produced the highest accuracy on raw input. In order to further improve the accuracy, this paper proposes a cascade of multilayer autoencoder for feature extraction followed by the random forest classifier for Android malware detection. A cascade of an autoencoder and random forest was applied to real-world datasets and achieved an accuracy of 99.1%. The proposed work also individually examines the impact of the six types of features to distinguish malware and benign apps.*

Keywords *- Android Malware, Random Forest, Multilayer Autoencoder.*

1. Introduction

People's mobiles are becoming increasingly essential in their day-to-day activities, such as digital payments, text messaging, e-shopping, etc. However, the challenge of smartphone security is becoming ever more severe as time passes. The research community has been focusing on developing malware detection methods by utilizing machine learning or deep learning algorithms with various features to protect users from newly developed malware. Because the Android OS is open source, creating malware that exploits the service flaws and security lapses is quite simple and economical. This is the primary reason for the significant rise in malware attacks on the Android platform.

Manual inspection and signature-based methods are two conventional malware detection approaches that could reveal some flaws but suffer from issues like slower detection speed and less accuracy. Various ways of detecting Android malware were developed in the previous research. They are broadly categorized into static, dynamic, and hybrid analysis techniques. Malware detection techniques based on static analysis do not have to execute the given app to detect if it contains malware. Instead, the static features are extracted and analyzed. Malware detection techniques based on dynamic analysis require executing the app in a virtual environment, extracting dynamic features, and using them for malware analysis. In the hybrid approach, both static & dynamic features are used in the analysis. Static analysis involves less complexity and risk.

Machine Learning and Deep Learning are emerging as prominent AI technologies that are being used in a variety of fields. Cybersecurity issues are also solved using ML and DL algorithms. This paper used ML and DL techniques to classify apks into malware and benign classes.

This research paper is structured as follows. Section 2 describes previous work. Section 3 discusses related concepts. Section 4 presents a proposed methodology for malware detection. The malware detection technique using ML classifiers and a combination of an autoencoder and the random forest is detailed in Section 5. Section 6 concludes and provides insights into future work.

2. Related Works

M. Kumaran [1] et al. proposed ML algorithms with manifest files for malware detection. Permissions and intents are retrieved from mobile apps, and a feature set is created with 182 features. After experimenting with 1000 apps, cubic SVM was given 91.7% accuracy. A. Fatima [2] et al. proposed a genetic algorithm for selecting the best features for malware detection. The number of features is reduced from 99 to 40 using a genetic algorithm, and later neural networks and SVM algorithms are applied. Sirisha. P [3] et al. proposed android malware detection using deep learning algorithms from permission features—a dataset with 400 apps with 331 extracted features applied with ANN and achieved 85% accuracy. M. Gohari [23] et al. applied an ensemble of CNN and LSTM for malware detection. The proposed model was applied to 86 network features and achieved good results. X. Su [5] et al. proposed deep learning algorithms for malware detection. More than 30,000 features are extracted from malware, and benign apps and deep Belief Networks (DBN) are applied. T. Kim [6] et al. proposed a multimodal with multiple features for malware detection. Different features like "permissions", "strings", "API calls", "opcodes", etc., are extracted from malware, and non-malware apps and ANN are applied. Android apps are transformed into grayscale images, applied CNNs and ensemble learning techniques and achieved good results [7],[8]. Omar N. Elayan [9] et al. proposed GRU for malware detection through permission and api call features. A dataset with 650 mobile apps (both benign and malware) is tested with the proposed GRU and achieved good accuracy. Tianliang Lu [10] et al. applied hybrid DL techniques for android malware detection. The combination of DBN and GRU was used with 351 hybrid features extracted from 13000 apks and achieved good accuracy.

Meghna Dhalaria [26] et al. proposed feature selection-based Malware detection with ML techniques for android malware detection and classification. Gianni D'Angelo [12] et al. translated api calls data into api images, and an autoencoder was applied to api images. Later, machine learning classifiers were applied to the features vector and achieved good results. X. Xing [13] et al. converted android apps into android images, and later autoencoder was applied to classify malware and benign apps. The autoencoder structure has an external MLP network to help in categorization and testing. H.J Zhu [24] et al. used android permission data, system event data, and API calls to create a feature vector. Later, the proposed random forest classifier for android malware detection and 89.9% accuracy was achieved. Parnika Bhat [16] et al. applied multi-layered feature selection for android apps. The features used for malware detection are Permissions, API call data, Intents, Network features and Hardware features. Several feature selection techniques like Feature frequency, discrimination and gain are used for selecting the best features. After creating a final dataset with relevant features, five ML classifiers were applied and achieved 96.2% accuracy with a random forest classifier.

Weiqing Huang [25] et al. proposed a bagging classifier for malware prediction. Opcodes, App permissions, and android images are combined input features and bagging with several ML classifiers applied at the training stage. Atika Gupta [18] et al. applied shallow learning algorithms J48, NB and random forest for the "Drebin" malware dataset and achieved better results with the random forest classifier. Shohel Rana [19] et al. proposed supervised ML techniques for malware detection. Several classifiers like KNN, NB, SVM and DT were applied and achieved good accuracy of 96% with KNN. In [20], authors extracted opcode sequences from ag files (retrieved from CFGs) of malware and benign apps and applied the LSTM model for malware detection.

3. Related Concepts

3.1. Classification Algorithms

Android malware detection is a classification problem. There are a variety of classification algorithms available, and depending on the task being carried out, some of them will be more suitable than others. Among the suitable classification algorithms for a specific task, the most suitable one can be selected based on the performance analysis of the classifiers built for specific task-related datasets. A brief short note on some of the classifiers involved in this research is presented below:

3.1.1. Naive Bayes Classifier

A classification method that relies on the Bayes theorem with a naive assumption on class conditional independence of individual features describing an entity is known as the Naive Bayes classifier. It makes learning easier by estimating the likelihood of a feature vector in a class as the product of individual conditional probabilities of the constituent features given the class.

3.1.2. Support Vector Machine

SVM constructs a decision boundary called the hyperplane for distinguishing data samples of different classes. For non-linearly separable problems, an appropriate kernel is used to translate the input data into a high-dimensional implicit feature space to make the dataset linearly separable; it identifies specific training data points on the decision boundary called support vectors and finds the hyperplane with maximum margin from them. Thus, SVM can also successfully classify data into complex, non-linearly separable decision regions. Proper regularization of SVM avoids over-fitting, even with small data.

3.1.3. Logistic Regression

The logistic regression method is a common statistical technique that is primarily useful for classification. It uses a logit function for classifying samples into several classes.

3.1.4. KNN Classifier

KNN assigns an unlabeled sample to the class with the greatest number of samples in its nearest K neighbors. The

value 'K' is an integer that the user specifies. Higher values of K reduce the impact of noisy points. Nearest neighbors are identified by calculating distances on the fly using measures like Euclidean, Minkowski etc. KNN classifiers are inherently incremental as they are not model-based classifiers.

3.1.5. Decision Tree Classifier

A decision tree is a tree-structured classifier. In this tree-structured classifier, internal nodes show different dataset options, branches show selection criteria, and each leaf node shows the output.

3.1.6. Random Forest Classifier

Random forest is a way for machines to learn by using many different decision trees. During the process of creating a tree, the best way to split each node is to choose from a set of candidate variables that are chosen at random. Random Forests can be used to predict the results of classification and regression analyses, but they can also be used to pick the most important variables and groups and better understand how variables relate to each other. The performance of classifiers varies with the input set of features available for analysis. Modern approaches to classification involve Deep learning techniques like autoencoders for feature extraction, especially dealing with high-dimensional datasets. A brief discussion on autoencoders is given below:

3.1.7. Autoencoder

An autoencoder adopts an unsupervised learning strategy to learn the appropriate weights of a neural net that transforms the set of input features into a smaller set of latent features such that the original data can be approximated in terms of the latent features in the reconstruction phase. Thus the autoencoder is trained with an encoder and a decoder pair; the encoder compresses input into latent space, and the decoder reconstructs the essence of the input from the latent space representation. There are several types of autoencoders. The most widely used types are vanilla autoencoders, multilayer or deep autoencoders, regularised autoencoders (Sparse autoencoders, Denoising autoencoders), convolutional autoencoders, variational autoencoders etc.

Vanilla Autoencoder (Simple Autoencoder)

It is an unsupervised learning strategy for neural nets that compresses input into a latent space representation. The Vanilla autoencoder is a three-layer network with one input layer, one output layer, and one hidden layer. This is also called an under-complete autoencoder. Under complete autoencoders have smaller latent spaces than input dimensions.

Deep Autoencoder (Multilayer Autoencoder)

It is possible to implement the autoencoder's encoder and decoder using a stack of layers rather than a single layer. These autoencoders are called multilayer or deep autoencoders. In this paper, the authors used a multilayer autoencoder for android malware detection.

Regularized Autoencoder

The regularized autoencoders employ a loss function that aids in giving the model characteristics other than duplicating input to output. There are two different forms of regularized autoencoders, the sparse autoencoder and the denoising autoencoder. Sparse autoencoders provide an alternate way to introduce an information bottleneck, and they do so without necessitating a reduction in the number of nodes in hidden layers. Denoising autoencoders corrupt the input by adding noise. This prevents the autoencoders from just copying the input and producing the output without taking into account any of the unique properties of the data. Training a model to reproduce with a sparsity penalty can result in it learning valuable features. Instead of adding a penalty to the loss function, we can get an autoencoder that learns anything beneficial by modifying the reconstruction error term of a loss function.

Convolutional Autoencoder

In a convolutional autoencoder, convolutions are used instead of fully connected layers. In this model, the encoders are convolution layers, and the decoders are deconvolution layers.

4. Research Methodology

The authors proposed a multilayer autoencoder for android malware detection. Malware and benign apps are extracted to create a dataset with 1063 features. A multilayer autoencoder is applied to get a reduced feature set. The reduced feature set is applied with random forest and achieves 99.1% accuracy. The random forest was selected after applying various classifiers on the original dataset with all 1063 features.

The proposed method is shown in figure-1. Android apps of malicious and benign types are collected. The collected mobile applications are archived files with apk extensions. All these applications are applied with a tool named "andropytool" to extract static features. A feature vector is formed by combining several features, namely permissions, opcodes, intents, system commands, api calls, and api packages. Later, a multilayer autoencoder was applied for feature reduction. The reduced feature set is applied with the ML classifier. Several ML classifiers were applied to the raw dataset for selecting an ML classifier. After applying various ML classifiers, Random Forest was identified as the best classifier with 98.7% accuracy. Later, the random forest was applied with a reduced feature set and achieved an accuracy of 99.1%.

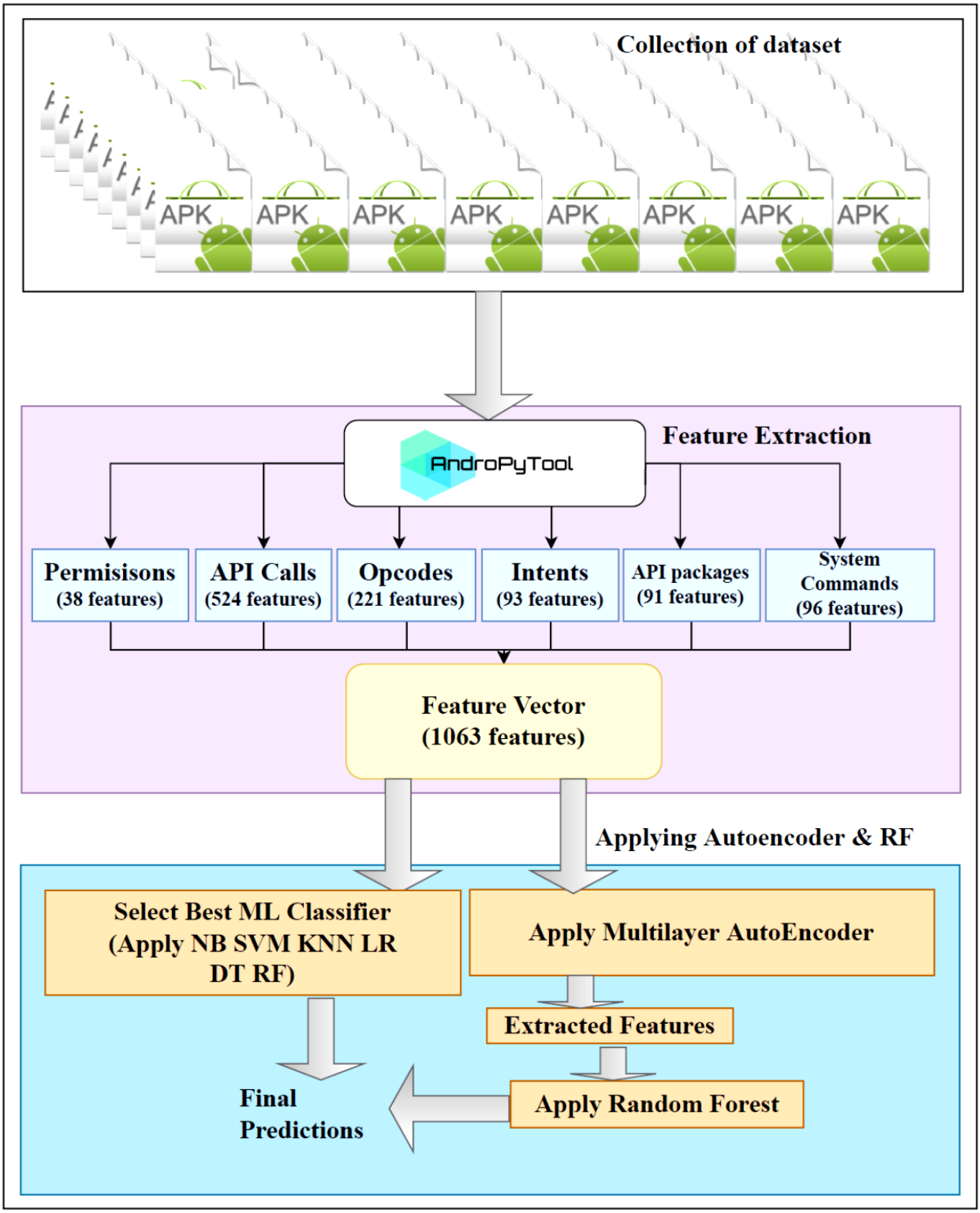

Fig. 1 Proposed Method for Android Malware Detection

5. Experimentation and Results

All the experiments are done with the Python language. The Spyder environment was used for applying machine learning algorithms.

5.1. Collection of Dataset

Benign apps are gathered from the play store, the unb site [22] and the apkpure site. Malware apps are gathered from "virus share" sites. All the apps are applied with a tool, "andropytool", to extract static features. The dataset contains 2,511 malicious apks and 2,508 non-malware apks (Table 1). An equal number of malware and non-malware samples were used in the experiments to create a balance in the dataset.

5.2. Feature Extraction

A python tool named "Andropytool" is used for extracting static features. This utility extracts static features from Android apps. It includes several well-known Android app inspection utilities, including "Droid Box", "Flow Droid", "Strace", "AndroGuard", etc. The tool can be used via docker or individual installation. The authors used the docker approach for using an installing tool. After installation, a path containing a folder of android apps can be given for analysis. It produced apk features in json format, and after extracting json files, a python script was written to convert json files to csv files. The algorithm for extracting features from apps is given below.

Table 1. Dataset

Type	Number
Malware	2,511
Non-Malware	2,508
Total	5,019

Algorithm for Extraction of Static Features from Android Apps:

Input: A directory with android apks
Output: csv file
Step 1. Start
Step 2. Install andropytool using the docker approach
Step 3. For each mobile application
Step 3.1. Apply the andropytool command to retrieve json file with feature files
Step 3.1.1. for each feature file (json file)
extract Permissions, Opcodes, API calls, Strings, API packages, and system commands from json files and store them in a list of lists and convert list of lists into csv file
Step 4. Stop

A csv file is produced after applying the above algorithm. It consists of six columns, each for one feature out of seven. Each row represents one Android apk. In a single cell of a csv file, all features (for example, all permissions used by a particular app are stored in a single cell) are nested. These are separated by using a python script. "Permissions" is a binary feature in the six selected features. All the remaining features are frequency-oriented (for example, opcode represents the number of times a particular opcode is used in an app) features. So, all these features are separated into six datasets for further analysis. The dataset with all combined features is also generated as final_dataset. The number of features collected for seven datasets is shown in Table 2.

5.3. Applying ML Classifiers

After creating final_dataset, the authors applied several ML classifiers. The algorithms are also applied to individual datasets to analyze which feature is more helpful in distinguishing malware and benign app. The accuracies obtained after applying algorithms are shown in Table 3.
Although all ML classifiers produced accurate results, Random Forest performed comparatively better on all datasets. Random Forest achieved a maximum accuracy of 98.7% for malware detection.

Table 2. Information about datasets

Dataset Name	Number
permission_dataset	38
opcodes_dataset	221
intents_dataset	93
apicalls_dataset	523
api_packages_dataset	91
system_commands_dataset	96
final_dataset	1,062

With random forest, the apicall dataset achieved 98.7% accuracy. Next, api packages have more impact. Random Forest achieved 98.6% accuracy with api packages dataset. After these two types of features, opcodes and permissions can also classify malware and benign apks with 98.5% and 98% accuracy. Intents and system commands are little impactful, with 96.5% and 95.9% accuracy. From the table-2, it is observed that the random forest algorithm performed well for android malware detection.

5.4. Applying Autoencoder for Feature Reduction

In order to further improve the generalization accuracy, the authors investigated the impact of explicit feature extraction from the final dataset, as there are more than a thousand features in it. Multilayer autoencoder is used for extracting latent features from the elaborate set of features gathered in the final dataset.

After applying all ML classifiers, it was found that the random forest algorithm is the best for malware detection among the ML classifiers. The authors applied a multilayer autoencoder followed by random forest for malware and benign classification. The final dataset with 1062 features is processed with a multilayer autoencoder to get reduced features. The impact of various autoencoder topologies on

detection accuracy is investigated and accordingly finalized an optimal topology for better feature reduction. Table 4 shows the experiments on six autoencoder topologies for feature reduction. The topology-2 is found to be the best with the highest accuracy.

5.4.1. Applying Multilayer (Deep) Autoencoder

The proposed encoder architecture is shown in figure-2. An autoencoder comprises two sub-models, an encoder and a decoder. The encoder reduces the input while the decoder tries to reconstitute the input from the encoder's compact form. The encoder prototype is saved and used as a data preparation approach to extract features from raw data before training an ML classifier. The architecture of the encoder contains 3 hidden layers followed by a bottleneck layer. The first hidden layer has n*2 nodes, the second has (n*2)/2 nodes, and the third has (n*2)/4 units. There is a bottleneck layer after these three hidden layers. This layer has (n*2)/8 nodes (input/4). The architecture of the decoder is in reverse order of the encoder. It has three hidden layers with (n*2)/4, (n*2)/2, and n*2 units, respectively. The output layer has 1,062 inputs. It has an activation function to produce output. The "adam" optimizer is used for the autoencoder model. The loss function used is "mse". Later, the model is trained with 250 epochs to reproduce inputs. Later, the encoder model was saved.

5.4.2. Applying Random Forest with Reduced Features

The trained encoding model was used to represent input features in compact form. The final_dataset with all features is applied with a trained encoder model. The final_dataset has 1062 features. So, after applying the trained encoder, the 1062 features are reduced to 265 features. (Because the bottleneck layer has n/4 features, where n is the number of input features).

Table 3. Accuracies of ML Classifiers

Dataset	NB	KNN	LR	SVM	DT	RF
permission_dataset	94.6%	96.9%	97.2%	97.6%	97.1%	98%
opcodes_dataset	94.7%	98%	95.2%	96%	98.1%	98.5%
intents_dataset	78.7%	96.5%	89.5%	90%	96.2%	96.5%
apicalls_dataset	98.6%	98.5%	96.9%	96.8%	97.9%	98.7%
api_packages_dataset	95.7%	98.4%	95.2%	96%	97.8%	98.6%
system_commands_dataset	90.1%	94.7%	88.8%	92.1%	95%	95.9%
final_dataset (with the combination of all features)	96%	97.6%	98.6%	95.1%	98.3%	98.7%

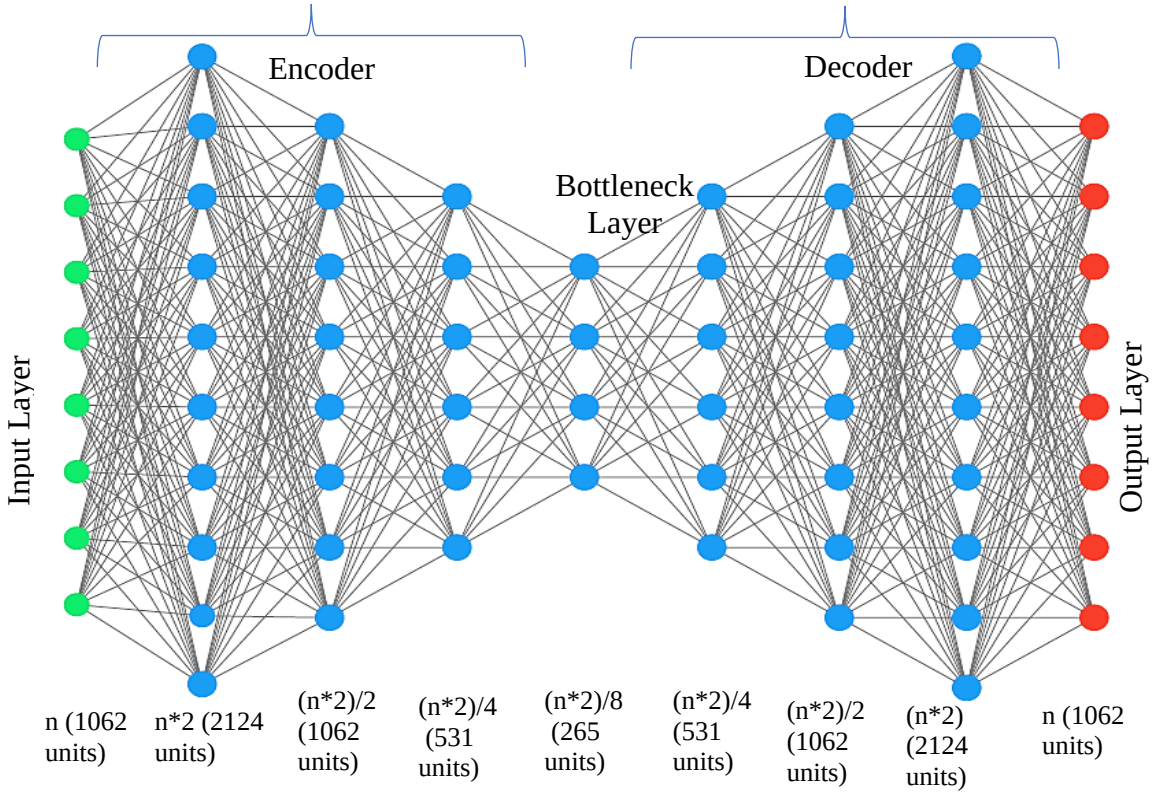

Fig. 2 Architecture of Multilayer Autoencoder

Table 4. Autoencoder Architecture with a specific Topology

S.no	Autoencoder Topology	Encoder structure (n- input units, here n=1062)					Accuracy obtained with Random Forest
1	Topology-1	n*2 (2124)	(n*2)/2 (1062)	(n*2)/4 (531)	(n*2)/8 (265)	(n*2)/16 (132) (Bottleneck layer)	98.9%
2	Topology -2	n*2 (2124)	(n*2)/2 (1062)	(n*2)/4 (531)	(n*2)/8 (265) (Bottleneck layer)	----------	99.1%
3	Topology -3	n*2 (2124)	(n*2)/2 (1062)	(n*2)/4 (531) (Bottleneck layer)	----------	----------	99%
4	Topology -4	n	n/2	n/4	n/8 (Bottleneck layer)	----------	98.8%
5	Topology -5	(n*3)	(n*3)/3	(n*3)/9	(n*3)/27 (Bottleneck layer)	----------	98.8%
6	Topology -6	n	n/2	n/3	n/4	n/5 (Bottleneck layer)	98.9%

In this paper, the authors used the random forest as an ML classifier because it has the best accuracy with the raw dataset (Table 3). The encoded input data is used for training with random forest. After applying random forest, the authors achieved 99.1% accuracy.

5.5. Comparison with Previous Work

The proposed model's performance was compared to prior studies. (Table 5 and figure-3). The authors in [6] applied deep learning ANN with multiple static features and achieved 98% accuracy. The authors in [14] applied ensemble learning algorithms and achieved 89.9% accuracy. In [26], hybrid features are used for malware detection and achieve an accuracy of 98%.

Table 5. Performance Evaluation

Model	Accuracy
Multimodal Deep Learning [6]	98%
Ensemble Learning [14]	89.9%
Hybrid Approach [26]	98%
Proposed Method	99.1%

In this paper, the authors extracted multiple features from android apps and proposed several classifiers. Among all classifiers, the random forest was given the best accuracy of 98.7%. Later the authors applied an autoencoder to encode input features in compressed form. Later, the random forest was applied to the compressed form of input obtained using an autoencoder and achieved an accuracy of 99.1%.

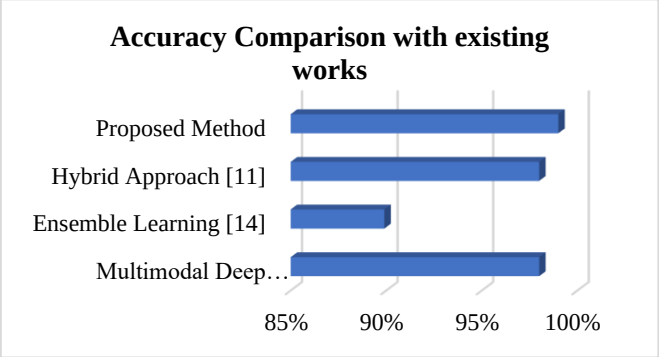

Fig. 3 Accuracy Comparison

6. Conclusion

In this paper, the authors proposed a multilayer autoencoder with a random forest classifier for android malware detection. Several android apk features are extracted using 'andropytool'. The extracted features are created as six datasets with permissions, opcodes, api calls, api packages, system commands, and intents. All these datasets are applied with ML classifiers and achieved 98.7% accuracy with random forest. It is also observed that api calls, api packages, and opcodes are beneficial for detecting malware apps among extracted features. A multilayer autoencoder is applied to the final_dataset with all features. The trained encoder was used to compress the input features, and the compressed dataset was applied with random forest and achieved 99.1% accuracy. The generalization performance of the proposed model was also good. The achieved accuracy values show that the proposed work outperforms traditional ML and DL classifiers for android malware detection.

References

[1] M. Kumaran and W. Li, "Lightweight Malware Detection Based on Machine Learning Algorithms and the Android Manifest File," *IEEE MIT Undergraduate Research Technology Conference (URTC)*, pp. 1-3, 2016. Crossref, https://doi.org/10.1109/URTC.2016.8284090

[2] A. Fatima, R. Maurya, M. K. Dutta, R. Burget and J. Masek, "Android Malware Detection Using Genetic Algorithm-based Optimized Feature Selection and Machine Learning," *2019 42nd International Conference on Telecommunications and Signal Processing (TSP)*, pp. 220-223, 2019. Crossref, https://doi.org/10.1109/TSP.2019.8769039

[3] Sirisha.P, K. P. B., A. K. K. and A. T, "Detection of Permission Driven Malware in Android Using Deep Learning Techniques," *2019 3rd International Conference on Electronics, Communication and Aerospace Technology (ICECA)*, pp. 941-945, 2019. Crossref, https://doi.org/10.1109/ICECA.2019.8821811

[4] Dr.S.Masood Ahamed and Dr.V.N.Sharma, "Malware Detection using Optimized Random Forest Classifier within Mobile Devices," *SSRG International Journal of Computer Science and Engineering*, vol. 3, no. 5, pp. 90-99, 2016. Crossref, https://doi.org/10.14445/23488387/IJCSE-V3I5P118

[5] X. Su, D. Zhang, W. Li and K. Zhao, "A Deep Learning Approach to Android Malware Feature Learning and Detection," *2016 IEEE Trustcom/BigDataSE/ISPA*, pp. 244-251, 2016. Crossref, https://doi.org/10.1109/TrustCom.2016.0070

[6] T. Kim, B. Kang, M. Rho, S. Sezer and E. G. Im, "A Multimodal Deep Learning Method for Android Malware Detection Using Various Features," *in IEEE Transactions on Information Forensics and Security*, vol. 14, no. 3, pp. 773-788, 2019. Crossref, https://doi.org/10.1109/TIFS.2018.2866319

[7] A. Lakshmanarao and M.Shashi, "Android Malware Detection Using Convolutional Neural Networks," *In Data Engineering and Intelligent Computing Advances in Intelligent Systems and Computing*, vol. 1407, pp. 151-162, 2021. Crossref, https://doi.org/10.1007/978-981-16-0171-2_15

[8] Lakshmanarao. A and Shashi. M, "An Efficient Android Malware Detection Framework with Stacking Ensemble Model," *International Journal of Engineering Trends and Technology*, vol. 70, no. 4, pp. 294-302, 2022. Crossref, https://doi.org/10.14445/22315381/IJETT-V70I4P226

[9] Omar N. Elayan and Ahmad M. Mustafa, "Android Malware Detection Using Deep Learning," *Procedia Computer Science*, vol. 184, pp. 847-852, 2021. https://doi.org/10.1016/j.procs.2021.03.106

[10] Tianliang Lu, Yanhui Du, Li Ouyang, Qiuyu Chen and Xirui Wang, "Android Malware Detection Based on a Hybrid Deep Learning Model," *Security and Communication Networks*, vol. 2020, pp. 11, 2020. Crossref, https://doi.org/10.1155/2020/8863617

[11] Syeda Sara Samreen and Hakeem Aejaz Aslam, "Hyperspectral Image Classification using Deep Learning Techniques: A Review," *SSRG International Journal of Electronics and Communication Engineering*, vol. 9, no. 6, pp. 1-4, 2022. Crossref, https://doi.org/10.14445/23488549/IJECE-V9I6P101

[12] Gianni D'Angelo, Massimo Ficco and Francesco Palmieri, "Malware Detection in Mobile Environments Based on Autoencoders and API-Images," *Journal of Parallel and Distributed Computing*, vol. 137, pp. 26-33, 2020. Crossref, https://doi.org/10.1016/j.jpdc.2019.11.001

[13] X. Xing, X. Jin, H. Elahi, H. Jiang and G. Wang, "A Malware Detection Approach Using Autoencoder in Deep Learning," *IEEE Access*, vol. 10, pp. 25696-25706, 2022. Crossref, https://doi.org/10.1109/ACCESS.2022.3155695

[14] K.Aishwarya and C.Selvi, "Predicting Fraud Apps using Hybrid Learning Approach," *SSRG International Journal of Computer Science and Engineering*, vol. 5, no. 6, pp. 1-5, 2018. Crossref, https://doi.org/10.14445/23488387/IJCSE-V5I6P103

[15] Nektaria Potha, V. Kouliaridis and G. Kambourakis, "An Extrinsic Random-Based Ensemble Approach for Android Malware Detection," *Connection Science*, vol. 33, no. 4, pp. 1077-1093, 2021. Crossref, https://doi.org/10.1080/09540091.2020.1853056

[16] Parnika Bhat and Kamlesh Dutta, "A Multi-Tiered Feature Selection Model for Android Malware Detection Based on Feature Discrimination and Information Gain," *Journal of King Saud University - Computer and Information Sciences*, 2021. Crossref, https://doi.org/10.1016/j.jksuci.2021.11.004

[17] Immadi Murali Krishna, Pendem Durga Bhavani, Tiriveedhi M S Madhuvani and Vajja Poojitha, "An Effective Segmentation and modified Ada Boost CNN based classification model for Fabric Fault Detection system," *SSRG International Journal of Computer Science and Engineering*, vol. 7, no. 7, pp. 34-40, 2020. Crossref, https://doi.org/10.14445/23488387/IJCSE-V7I7P106

[18] Atika Gupta, Sudhanshu Maurya, Divya Kapil, Nidhi Mehra and Harendra Singh Negi, "Android Malware Detection using Machine Learning," *International Journal of Recent Technology and Engineering*, vol. 8, no. 2S12, 2019.

[19] Rana, M. S., Sung and A. H, "Malware Analysis on Android Using Supervised Machine Learning Techniques," *International Journal of Computer and Communication Engineering*, vol. 7, no. 4, pp. 178-188, 2018.

[20] Lakshmanarao A, and Shashi M, "Android Malware Detection with Deep Learning using RNN from Opcode Sequences," *International Journal of Interactive Mobile Technologies (iJIM)*, vol. 16, no. 1, pp. 145–157, 2022. Crossref, https://doi.org/10.3991/ijim.v16i01.26433

[21] Oyinloye Oghenerukevwe Elohor, Olatomide Awoyomi, "Modelling A Data Sniffing Malware Detector For Apks," *International Journal of Computer and Organization Trends*, vol. 9, no. 6, pp. 1-8, 2019. Crossref, https://doi.org/10.14445/22492593/IJCOT-V9I6P301

[22] [Online]. Available: unb.ca/cic/datasets/andmal2017.html

[23] M. Gohari, S. Hashemi and L. Abdi, "Android Malware Detection and Classification Based on Network Traffic Using Deep Learning," *2021* 7th International Conference on Web Research(ICWR), pp. 71-77, 2021. Crossref, https://doi.org/10.1109/ICWR51868.2021.9443025

[24] Hui-Juan Zhu, Tong-Hai Jiang, Bo Ma, Zhu-Hong You, Wei-Lei Shi and Li Cheng, "HEMD: A Highly Efficient Random Forest-Based Malware Detection Framework For Android," Neural Comput & Application, vol. 30, pp. 3353–3361, 2018. Crossref, https://doi.org/10.1007/s00521-017-2914-y

[25] Weiqing Huang, Erhang Hou, Liang Zheng and Weimiao Feng, "MixDroid: A Multi-Features and Multiclassifiers Bagging System for Android Malware Detection," *AIP Conference Proceedings*, vol. 1967, pp. 020015, 2018. Crossref, https://doi.org/10.1063/1.5038987

[26] Meghna Dhalaria and Ekta Gandotra, "A Hybrid Approach for Android Malware Detectionand Family Classification," *International Journal of Interactive Multimedia and Artificial Intelligence*, vol. 6, no. 6, 2020. Crossref, https://doi.org/10.9781/ijimai.2020.09.001