ISSN: 2584-0495 Vol. 2, Issue 9, pp. 1203-1210

International Journal of Microsystems and IoT

ISSN: (Online) Journal homepage: https://www.ijmit.org

Android Malware Detection on Imbalanced Data Using Deep Learning

Enugala Rajitha, Anil Pinapati

Cite as: Rajitha, E., & Pinapati, A. (2024). Android Malware Detection on Imbalanced Data Using Deep Learning. International Journal of Microsystems and IoT, 2(9), 1203-1210. https://doi.org/10.5281/zenodo.14101737

Published online: 23 Sept 2024

DOI: https://doi.org/10.5281/zenodo.14101737

International Journal of Microsystems and IoT
Vol. 2, Issue 9, pp.1203-1210; DOI: https://doi.org/10.5281/zenodo.14101737

Android Malware Detection on Imbalanced Data Using Deep Learning

Enugala Rajitha, Anil Pinapati

Department of Computer Science and Engineering, National Institute of Technology Calicut, Kerala, India

ABSTRACT

This Android remains the most widely adopted operating system due to its open nature, enabling users to install applications from various origins. Malware causes severe problems for Android users and steals personal data by injecting malware into various applications. Many previous works used machine learning models to detect malware but suffer from data imbalance and high feature size. In this work, we perform three tasks: 1) Data pre-processing using the Synthetic Minority Oversampling Technique to Fix the Class Inequality Problem, and Standardization is used to normalize the features; 2) Principal Component Analysis is applied to the dataset to shrink the dimensionality of the feature vector; 3) Multilayer Perceptron is used for classification. We performed the proposed approach on Drebin, Malgenome, and Maldroid2020 and it achieved 98.27, 98.15, and 97.12 accuracy respectively. The results of the experiments demonstrate that the proposed strategy is more accurate than previous studies

KEYWORDS
Android, PCA, SMOTE, Multilayer Perceptron, Malware

1. INTRODUCTION

Android is a prevalent operating system worldwide [26]. As per the International Data Corporation (ICD) report, the sales of mobile devices increase by 351 million units annually [11]. The Android platform's open environment enables users to install applications from various sources effortlessly due to its versatile operating system (OS), which can function on various devices, including Internet of Things (IoT) devices, tablets, smartphones, and smartwatches. Android devices enjoy extensive usage. The Android OS is establishing itself as the leading mobile platform, mostly because of its open-source capabilities, scalability, and user-friendly [14]. Nowadays, most people store their details on social media, banking, and shopping apps. Attackers shift their focus to mobile apps to cause threats to Android mobiles [6, 18]. Malicious applications steal personal information like banking information, message lists, and contact numbers without user concern. To safeguard users from malware, many researchers have concentrated their efforts on dynamic analysis, static analysis, and hybrid analysis employing machine learning approaches [1, 2, 3, 4]. Android protection performs in three ways: 1. Signature-based 2.Dynamic analysis based 3.Static analysis based [29]. In signature-based methods, we create a malware signatures database [29]. It detects the malware if the malware signature matches the signatures stored in the database. It fails to detect unknown malware. Traditional antivirus software makes use of signature-based approaches. Signature-based techniques can be vulnerable to obfuscation via byte-code-level transformation attacks [27]. Extracting malware signatures based on known patterns requires substantial time and expertise due to the rising number of newly identified malware instances [15]. Moreover, this method can lead to false negatives when generating signatures for variations within established malware categories [32].

In the dynamic-based method, we examine the behavior of the application while it is operating in real-time or in an open sandbox, including memory utilization, system calls, information flow, and power consumption [4, 24].

Environmental setup is expensive, and extracting features takes longer. The features are extracted in static analysis by decompiling the Android Application Package (APK) file. Static feature extraction is easier than dynamic feature extraction because it requires no extra setup. It takes less time and is cost-effective [23]. Hybrid analysis combines dynamic and static information, allowing for very efficient Android malware detection [30].
Algorithms developed using machine learning (ML) are used to determine malware in the Android ecosystem. Zarni et al. [33] proposed a permission-based method using K means clustering. This method utilized information gain to select the best features. The dataset used in this approach contains few samples. It achieved 90% accuracy. Shubair et al. introduced an intelligent method for Android malware detection utilizing a neuro-fuzzy technique [31]. This approach uses information gain to extract features converted into feature patterns. Feature patterns give input to the neuro-fuzzy strategy. It achieved 70% accuracy, which is very low compared to previous techniques.

Ke Xu et al. presented ICCDetector, a method designed to find malware by leveraging ICC-related features [28]. It classifies five different malware categories. This method achieves 97.4% accuracy, but the limitation is that the dataset needs to be more balanced. Sometimes, a single classifier may not give better results. Xin et al. proposed the Mlifdect method. Mlifdect utilized a combination of data and a parallel machine-learning technique to detect malware [25]. It creates two feature sets in Mlifdect, using a feature selection algorithm from the set of eight static features. This technique

used the Dempster-Shafer information fusion technique and probability analysis-based technique. Deep learning represents a relatively recent and rapidly advancing domain within machine learning research, drawing substantial attention in artificial intelligence.

Yuan et al. introduced DroidDetector to detect malware using Deep Learning [27]. DroidDetector used hybrid features, achieving 96.76% accuracy, better than machine learning techniques. The dataset used in this technique is imbalanced and it reduces the total number of elements by applying a feature selection algorithm. Suleiman et al. introduced the DroidFusion technique for detecting malware in Android. DroidFusion uses multilevel architecture. In this approach, machine learning algorithms act as the primary classifier on a base level. In comparison, a set of ranking methods is utilized at a higher level to produce the result [20].

Mulhem et al. suggested a deep learning-based technique for spotting malware in Android. They used static analysis [7]. This method uses binary classification to detect whether it is malware or not, and it performs multilevel classification (five classes) to detect the type of malware. Jin et al. used multilevel data pruning techniques to extract significant patterns and used a Support Vector machine as a classifier [21]. This technique streamlined the feature set from 135 to 25, resulting in analysis times reduced by a factor of 4 to 32 compared to utilizing all available features. Taeguen et al. introduced a multimodal deep-learning technique [22]. In this approach, features are removed using methods for generating feature vectors, such as existing-based and similarity-based approaches. It uses a set of Deep Neural Networks (DNN) to get the final result. Xiaofei et al. stated a method using the Auto Encoder (AE) technique [8]. This technique extracts features using byte code and converts them to a grayscale image fed to the autoencoder. This technique used dimensionality reduction features. Esraa et al. introduced a method that leverages the co-existence of features [2]. This method used a frequent pattern growth algorithm to identify the top frequent pattern and used Random Forest (RF) as a classifier to detect malware.

Pakarat et al. introduced ensemble learning using different deep learning methods [9]. It uses a set of base classifiers, and the output generated from all hidden layers is given to the Meta classifier to get the final result. Iman et al. introduced Ransomware malware detection. This approach involves the Synthetic Minority Over-sampling Technique (SMOTE) oversampling method to resolve an issue of class imbalance and the SVM classifier for Ransomware detection [12]. Lwin et al. balanced the dataset using the SMOTE oversampling technique [13]. Yuan et al. proposed two sampling methods for the class imbalance problem. This approach uses Center point SMOTE and IO-SMOTE methods [5].

After the literature review, it was observed that the majority of prior research efforts concentrated on feature extraction and classification methods. Most of the datasets used in the above results are imbalanced. More balanced data is needed to give more accuracy. This work proposed a new approach by combining the SMOTE oversampling technique for class imbalance. Most of the existing works used the total dataset. It is vital to select important features from the dataset. This work makes use of towards reduction of dimensionality, principal component analysis (PCA), and deep learning techniques used for classification

The main works performed in the proposed method are as follows:

- We propose the SMOTE oversampling as a technique to overcome the class imbalanced problem.
- Standardization is the process of normalizing the features
- PCA for dimensionality reduction.
- The Multilayer Perceptron (MLP) for classification.
- We assess the effectiveness of the suggested approach using three different datasets: Drebin, Malgenome, and Maldroid2020, and it achieves an accuracy of 98.15%, 98.42%, and 97.12%, respectively.

In our evaluation, we compared the proposed method to earlier approaches, and the experimental findings unambiguously demonstrate that the proposed strategy reaches a noteworthy degree of accuracy.

2. METHODOLOGY

Figure 1 offers an illustration of the proposed method's architecture. This paper provides a malware detection method based on deep learning in datasets with unequal class distributions. The proposed work comprises three parts: Data pre-processing, Dimensionality reduction using PCA, and MLP for classification. Deep learning is the widely used classification technique at present. Many previous studies employed deep learning and machine learning techniques on imbalanced datasets with many features. Consequently, these methods often yielded lower accuracy and incurred longer classification times. The proposed work focuses on class imbalance problems, dimensionality reduction, and classification techniques.

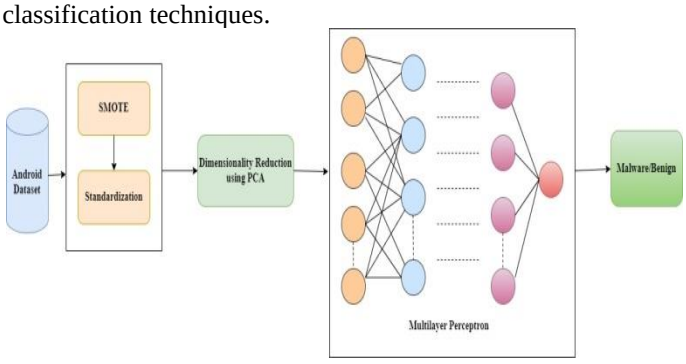


Figure 1. Design of the proposed approach

2.1 Data Pre-processing

Data preparation is vital in data mining, encompassing tasks such as data cleansing, transformation, and integration. These activities are crucial in readying the data for subsequent

analysis. The proposed work uses SMOTE oversampling and Standardization.

2.1.1 Synthetic Minority Over-sampling Technique

In cases where one class of data represents an underrepresented minority within the data sample, it uses over-sampling techniques to replicate instances of this class, thereby achieving a more balanced representation of positive results during training. Over-sampling becomes essential when the quantity of collected data is insufficient, one commonly used over-sampling technique is SMOTE. SMOTE creates synthetic samples simply by randomly picking features from examples in the minority class [5]. When a particular class of data represents the overrepresented majority in a dataset, under-sampling can be employed to balance it with the minority class. Under-sampling is a suitable approach when the quantity of collected data is large. Standard under-sampling methods, such as cluster centroids and Tomek links, focus on identifying potential overlaps in features within the dataset to reduce the volume of majority class data.

SMOTE, developed by Chawla in 2002, is an early technique applied across various domains, including but not limited to the medical and industrial sectors [16].

Algorithm:

- Step 1: The dataset was analyzed to identify the majority and minority classes, and the numerical disparity between them was recorded.
- Step 2: For each sample x, in the minority sample data, apply the K-Nearest Neighbor (KNN) algorithm to find its neighbors and the difference between sample x and its neighbors using Euclidean distance.
- Step 3: Multiply the distance with any random number between 0 and 1, create a synthesized feature with that value, and add it to the previous feature vector.
- Step 4: Repeat step 3 until the amount of samples in the majority class equals the amount of classes provided in the minority class.

The blue circles in Figure. 2 represent benign data, the pink circles represent malware data, and the green circle represents synthesized data. Figure 3 indicates the Drebin dataset class distribution before and after SMOTE.

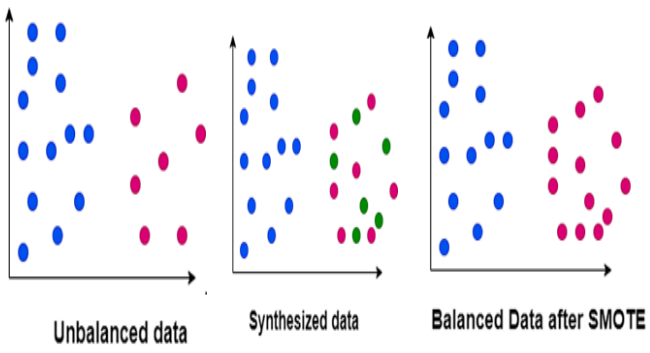


Figure 2. Dataset representation before and after the SMOTE oversampling technique

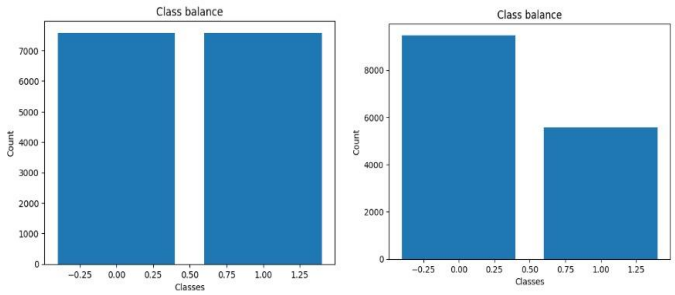


Figure 3. Drebin dataset before and after SMOTE

2.1.2 Standardization

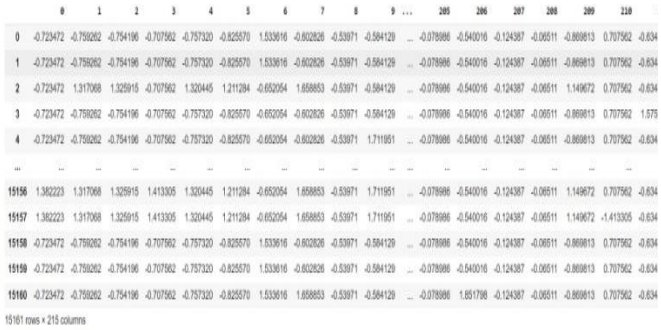
Standardization is scaling data to align with a standard normal distribution. It often has a zero mean and one deviation from the mean.

The equation of Standardization involves $X=((x-\mu))/\sigma$, where the feature is expressed by x. σ represents the standard deviation, while μ is the average of the value of the features. Figure. 4 and Figure. 5 symbolize the dataset before and after performing Standardization.

	transact	onServiceConnected	bindService	attachInterface	ServiceConnection	android.os.Binder	SEND_SMS	Ljava.lang.Class.getCanonicalName	Ljava.lang.Class.getMethods
3368	0	0	0	0	0	0	1	0	0
17288	0	0	0	0	0	0	1	0	0
10790	0	1	1	0	1	1	0	1	0
3755	0	0	0	0	0	0	0	0	0
9140	0	0	0	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...
11284	1	1	1	1	1	1	0	1	0
11964	1	1	1	1	1	1	0	1	0
5390	0	0	0	0	0	0	1	0	0
860	0	0	0	0	0	0	1	0	0
15795	0	0	0	0	0	0	1	1	0

15161 rows × 215 columns

Figure 4. Dataset before Standardization



	0	1	2	3	4	5	6	7	8	9	...	205	206	207	208	209	210	
0	-0.723472	-0.759262	-0.754196	-0.707562	-0.757320	-0.825570	1.533616	-0.602826	-0.53971	-0.584129	...	-0.078986	-0.540016	-0.124387	-0.06511	-0.869813	0.707562	-0.634
1	-0.723472	-0.759262	-0.754196	-0.707562	-0.757320	-0.825570	1.533616	-0.602826	-0.53971	-0.584129	...	-0.078986	-0.540016	-0.124387	-0.06511	-0.869813	0.707562	-0.634
2	-0.723472	1.317068	1.325915	-0.707562	1.320445	1.211284	-0.652054	1.658853	-0.53971	-0.584129	...	-0.078986	-0.540016	-0.124387	-0.06511	1.149672	0.707562	-0.634
3	-0.723472	-0.759262	-0.754196	-0.707562	-0.757320	-0.825570	-0.652054	-0.602826	-0.53971	-0.584129	...	-0.078986	-0.540016	-0.124387	-0.06511	-0.869813	0.707562	1.575
4	-0.723472	-0.759262	-0.754196	-0.707562	-0.757320	-0.825570	-0.652054	-0.602826	-0.53971	1.711951	...	-0.078986	-0.540016	-0.124387	-0.06511	-0.869813	0.707562	-0.634
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
15156	1.382223	1.317068	1.325915	1.413305	1.320445	1.211284	-0.652054	1.658853	-0.53971	1.711951	...	-0.078986	-0.540016	-0.124387	-0.06511	1.149672	0.707562	-0.634
15157	1.382223	1.317068	1.325915	1.413305	1.320445	1.211284	-0.652054	1.658853	-0.53971	1.711951	...	-0.078986	-0.540016	-0.124387	-0.06511	1.149672	-1.413305	-0.634
15158	-0.723472	-0.759262	-0.754196	-0.707562	-0.757320	-0.825570	1.533616	-0.602826	-0.53971	-0.584129	...	-0.078986	-0.540016	-0.124387	-0.06511	-0.869813	0.707562	-0.634
15159	-0.723472	-0.759262	-0.754196	-0.707562	-0.757320	-0.825570	1.533616	-0.602826	-0.53971	-0.584129	...	-0.078986	-0.540016	-0.124387	-0.06511	-0.869813	0.707562	-0.634
15160	-0.723472	-0.759262	-0.754196	-0.707562	-0.757320	-0.825570	1.533616	1.658853	-0.53971	-0.584129	...	-0.078986	1.851798	-0.124387	-0.06511	-0.869813	0.707562	-0.634

15161 rows × 215 columns

Figure 5. Dataset after Standardization

2.2 Principal component analysis (PCA)

The values generated after Standardization are applied to PCA to create a new feature vector [19]. Figure 6 indicates the Scree plot of Drebin, Figure 7 indicates the Scree plot of Malgenome, and Figure 8 represents the Scree Plot of the Maldroid2020 datasets.

The PCA method is as follows:

i. Subtracting the mean from each dimension of the provided dataset results in a new dataset with a mean value of zero.

ii. Determine the covariance matrix for two distinct dimensions within the dataset.

iii. Obtain the Eigen vectors and Eigen values of the matrix derived in step ii.

iv. Construct a feature vector, transpose it, and multiply it by the initial dataset to store a new feature set in the new space.

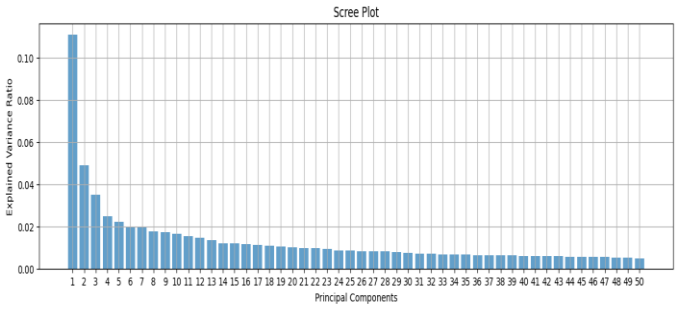


Figure 6. Drebin Principal Components

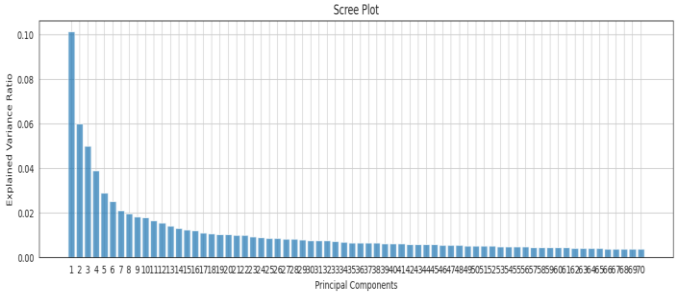


Figure 7. Malgenome Principal Components

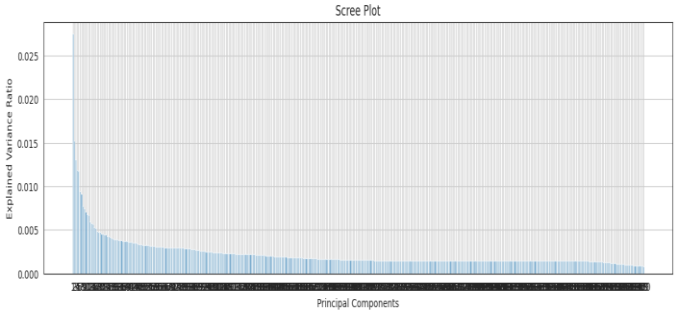


Figure 8. MalDroid2020 Principal Components

2.3 The deep learning model for classification

This work employs MLP and comes under a classification technique. An MLP consists of entirely linked dense layers that can effectively reshape input dimensions to the desired format. Multiple layers are a defining feature of this kind of neural network. It connects neurons so that their outputs can act as inputs for other neurons. A normal MLP comprises a single input layer, including one neuron for each input, and one output layer, having a single node per each output. It can additionally include any number of layers that are hidden, with a variable number of nodes within each of them. Figure 1 presents the key components of MLP.

The proposed work uses three different datasets. The number of input layers depends on many features. This work used one input layer, five numbers of hidden layers, two dropout layers, and a single output layer. The hidden layer utilizes the rectified linear (ReLU) activation function, whereas The activation function known as the sigmoid is utilized by the output layer. It uses Adam optimizer for optimization to adjust weights during training. The output of MLP is probability value; It gives malware if the probability is more significant than 0.5; otherwise, it is benign. It uses the following formulas for the Relu and Sigmoid activation function.

Relu:

if the input is more significant than zero

return input

else

return 0;

Sigmoid:

The sigmoid function turns every real-valued number into a number that ranges from 0 to 1, making it appropriate for binary classification.

$$F(x) = \frac{1}{1+e^{-x}}$$

Here, e is the natural logarithm's base (about 2.71828), and the function's argument is x. The plot of the sigmoid function resembles an S curve, with the function being continuous and differential at any point within its area.

3. FINDINGS AND CONTRIBUTIONS

This study conducts a series of studies to figure out the effectiveness of the recommended model using static analysis. We compared the results to those obtained using other state-of-the-art methodologies. This work uses Drebin, Malgenome, and Maldroid2020 datasets. Table 1 describes the details of the above three datasets.

Table 1. Android Malware Dataset Description.

Dataset	Type	Features	Samples	Classes	Malware samples	Benign Samples
Drebin	Static	215	15036	2	4434	7590
Malgenome	Static	215	3799	2	994	2045
Maldroid2020	Static	1633	2081	2	792	872

1. The Drebin dataset encompasses 215 static features, which include Permissions and API calls that have been obtained from a dataset comprising 15,306 APK files [34, 35]. These features are derived within the manifest file and disassembled code of the APK file. The dataset is composed of 7,590 benign instances and 4,434 malware samples.
2. The Malgenome dataset contains 215 static features, including permissions extracted from 3799 APK files [36]. The dataset comprises 2045 benign instances and 994 malware samples.
3. CICMaldroid2020 contains 1633 static features, including intents, permissions, and services from 2081 APK files. It contains 872 benign and 792 malware samples [17, 10].

In the proposed work, we deleted the dataset's missing and duplicate values. We performed data preprocessing (SMOTE+Standardization), and then PCA was used to lower the dimensionality. MLP takes the new features extracted after PCA as input. MLP classifies the given sample as either malware or benign. In the case of an MLP, the total amount of the input layer's neurons varies depending on the size of the features. The proposed MLP comprises five layers that are hidden in addition to one layer for input, each comprising 50 neurons using the output layer using the Sigmoid activation function and the ReLU activation function. The proposed work utilizes the Adam optimizer to reduce the loss function during neural network training.

The rate at which learning occurs is set at 0.01, and we conclude the training procedure after 20 epochs. Dropout is introduced into the training process following the second and third hidden layers To avoid overfitting, given a rate of dropout of 0.3. The training dataset is divided into smaller batches to modify the model's values during training. The batch size is 32.

Figure 9 conveys the model summary of the Drebin dataset, Figure 10 conveys the model summary of the Malgenome dataset, and Figure 11 represents the model summary of the MalDroid2020 dataset. Figures 12, 13, and 14 demonstrate both the loss and accuracy curves for the datasets utilized in this work.

We performed different sets of experiments with and without using data preprocessing (SMOTE) and feature reduction (PCA) to examine the suggested model's effectiveness. Experiment 1 achieves the highest accuracy, which uses data preprocessing and feature reduction techniques in combination with MLP.

- Experiment 1 conducted on the proposed model (combining SMOTE+PCA+MLP) Drebin gives the highest accuracy compared to the remaining datasets. Table 2 shows the performance of all three datasets based on the proposed model.

- Experiment 2 was conducted by performing only PCA+MLP on the given four datasets. It observes that all the datasets achieve less accuracy compared to experiment 1 because these datasets need to be more balanced. Table 3 shows the performance of all three datasets.

- Experiment 3 was conducted by performing only SMOTE+MLP. The Malgenome dataset achieves the highest accuracy, but it is less than the proposed model because using PCA combines correlated features into new features, reduces feature space, and improves accuracy. Table 4 shows the performance of all three datasets.

- In the proposed model, we conduct experiment 4 by performing only MLP. The Malgenome dataset achieves the highest accuracy, but it is less compared to the suggested approach. Table 5 shows the performances of all three datasets.

Table 2. Performances of the proposed model.

Dataset	Principal Components	Accuracy	Precision	Recall	F1_score
Drebin	50	98.27	97.25	98.12	97.69
Malgenome	70	98.15	96.32	98.49	97.39
MalDroid2020	400	97.12	95.7	98.5	97.1

Table 3. Performances of the PCA+MLP model.

Dataset	Principal Components	Accuracy	Precision	Recall	F1_score
Drebin	50	97.27	95.69	97.05	96.36
Malgenome	70	96.31	97.98	91.35	94.55
MalDroid2020	400	96.16	94.73	97.53	96.11

Table 4. Performances of the SMOTE+MLP model.

Dataset	Feature size	Accuracy	Precision	Recall	F1_score
Drebin	215	97.27	94.8	98.03	96.4
Malgenome	215	97.76	96.6	96.9	96.8

MalDroid2020	1633	96.64	93.9	99.5	96.65

Table 5. Performances of the MLP model.

Dataset	Feature size	Accuracy	Precision	Recall	F1_score
Drebin	215	97	94	98	96
Malgenome	215	97.76	94.9	98.8	96.85
MalDroid2020	1633	95.9	93.45	98.5	95.2

Layer (type)	Output Shape	Param #
dense_7 (Dense)	(None, 50)	2550
dense_8 (Dense)	(None, 50)	2550
dense_9 (Dense)	(None, 50)	2550
dropout_2 (Dropout)	(None, 50)	0
dense_10 (Dense)	(None, 50)	2550
dropout_3 (Dropout)	(None, 50)	0
dense_11 (Dense)	(None, 50)	2550
dense_12 (Dense)	(None, 50)	2550
dense_13 (Dense)	(None, 1)	51

Figure 9. Model Summary for Drebin Dataset

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 50)	3550
dense_1 (Dense)	(None, 50)	2550
dense_2 (Dense)	(None, 50)	2550
dropout (Dropout)	(None, 50)	0
dense_3 (Dense)	(None, 50)	2550
dropout_1 (Dropout)	(None, 50)	0
dense_4 (Dense)	(None, 50)	2550
dense_5 (Dense)	(None, 50)	2550
dense_6 (Dense)	(None, 1)	51

Figure 10. Model Summary for Malgenome Dataset

Layer (type)	Output Shape	Param #
dense_14 (Dense)	(None, 50)	20050
dense_15 (Dense)	(None, 50)	2550
dense_16 (Dense)	(None, 50)	2550
dropout_4 (Dropout)	(None, 50)	0
dense_17 (Dense)	(None, 50)	2550
dropout_5 (Dropout)	(None, 50)	0
dense_18 (Dense)	(None, 50)	2550
dense_19 (Dense)	(None, 50)	2550
dense_20 (Dense)	(None, 1)	51

Figure 11. Model Summary for MalDroid2020 dataset

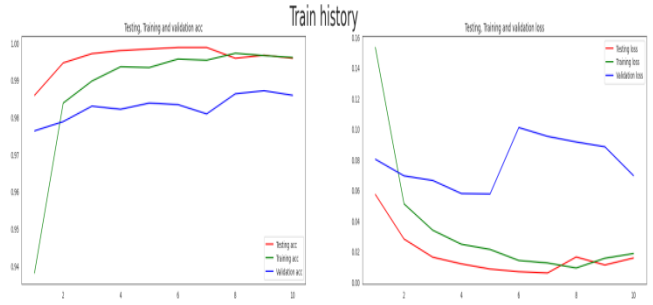


Figure 12. Model Training, Testing, and Validation accuracy and loss curves of Drebin

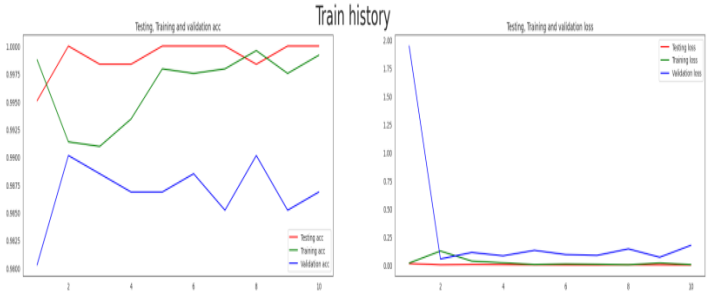


Figure 13. Training, Testing, and Validation accuracy and loss curves of Malgenome

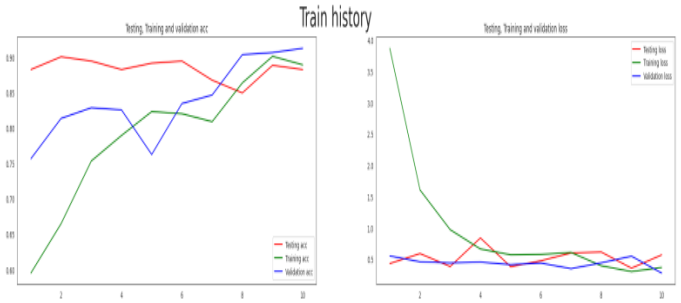


Figure 14. Training, Testing, and Validation accuracy and loss curves of MalDroid2020

Table 6
Comparison of the proposed approach with the Existing approach

S.No	Reference	Dataset	classifier	Accuracy
1	[2]	Drebin	RF	95%
2	[2]	Malgenome	RF	97%
3	[2]	Maldroid2020	RF	97%
4	[3]	Drebin	RL & RF	95.6%
5	[8]	Own dataset	AE	96%
6	[9]	CIC-2019	DNN	94.16%
7	[9]	CCCS-CIC	DNN	97.72%
8	[21]	Own dataset	SVM	93.62%
9	[22]	Malgenome and a few own samples	Deep Learning	98%
10	Proposed approach	Drebin	MLP	98.27%
11	Proposed approach	Malgenome	MLP	98.15%
12	Proposed	Maldroid2020	MLP	97.12%

	approach			

Table 6 demonstrates the comparative study table of the proposed approach with existing approaches from standard journals like IEEE Access, Springer Elsevier, etc. From Table 6, it is observed that the proposed approach achieves the best result.

3.1 Evaluation Metrics:

To evaluate how well the proposed model predicts outcomes, We used typical evaluation criteria like precision, accuracy, recall, and the score of F1. The definitions of these assessment metrics are as follows:

Accuracy: The ratio of accurately predicted samples to the total number of samples is known as accuracy.

$$\text{Accuracy} = \frac{\text{Tp} + \text{Tn}}{\text{Tp} + \text{Fp} + \text{Tn} + \text{Fn}}$$

Precision: The percentage of accurately anticipated positive samples among all samples projected to be positive is known as precision.

$$\text{Precission} = \frac{\text{Tp}}{\text{Tp} + \text{Fp}}$$

Recall: The percentage of accurately predicted positive samples among all samples that are in the positive class is called recall, or sensitivity.

$$\text{Recall} = \frac{\text{Tp}}{\text{Tp} + \text{Fn}}$$

F1-Score: The score of F1 is a robust metric for calculating the harmonic average of precision and recall.

$$\text{F1\_score} = \frac{2 * \text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}}$$

Experimental setup: The tests were carried out utilizing Google Colaboratory (Colab), an online interactive computing environment hosted on the Google Cloud platform. We utilized the Pro Plus version of the Colab notebook for these experiments, which offers extended runtimes and more sessions compared to other versions.

4. CONCLUSION

A deep learning-based approach is presented in the suggested article that can identify Android malware. It first uses SMOTE oversampling to balance the dataset. Standardization is applied to convert all the features into the same range of values so that every feature has equal weight while performing classification. It involves PCA for dimensionality reduction. It applies MLP for final classification. It uses five layers that are hidden and one layer of output in an MLP. Activation functions known as ReLU and sigmoid are used by the hidden layer and output layer, respectively.

. This model uses Adam optimizer for optimization. The efficiency of the suggested technique was examined using four different datasets: Drebin, Malgenome, and Maldroid2020 achieving accuracies of 98.27%, 98.15%, and 97.12% respectively.

The classification result proves the proposed method is more accurate than earlier efforts. In the future, we will perform multilevel classification using multimodal deep neural networks to detect different types of malware.

REFERENCES

1. Dhalaria, Meghna, and Ekta Gandotra. (2024) MalDetect: A classifier fusion approach for detection of android malware." Expert Systems with Applications 235, 121155.
2. Odat, Esraa, and Qussai M. Yaseen. (2023). A novel machine learning approach for android malware detection based on the co-existence of features." IEEE Access 11 15471-15484.
3. Wu, Yinwei, et al. (2023). DroidRL: Feature selection for Android malware detection with reinforcement learning. Computers & Security 128 103126.
4. Bhat, Parnika, Sunny Behal, and Kamlesh Dutta. (2023). A system call-based android malware detection approach with homogeneous & heterogeneous ensemble machine learning. Computers & Security 130, 103277.
5. Bao, Yuan, and Sibo Yang. (2023). Two novel SMOTE methods for solving imbalanced classification problems. IEEE Access 11, 5816
6. O. Abendan. (2011). Fake Apps Affect Android OS Users. Accessed: Oct. 30, 2022. [Online]. Available: https://www.trendmicro.com/vinfo/us/threatencyclopedia/web-attack/72/fake-apps-affect-android-os- users.
7. İbrahim, Mülhem, Bayan Issa, and Muhammed Basheer Jasser. (2022). A method for automatic android malware detection based on static analysis and deep learning. IEEE Access 117334-117352.
8. Xing, Xiaofei, et al.(2022). A malware detection approach using autoencoder in deep learning. IEEE Access 10, 25696-25706.
9. Musikawan, Pakarat, et al (2022). An enhanced deep learning neural network for the detection and identification of android malware. IEEE Internet of Things Journal 10, 8560-8577.
10. Mahdavifar, Samaneh, Dima Alhadidi, and Ali A. Ghorbani 10 (2022):. "Effective and efficient hybrid android malware classification using pseudo-label stacked auto-encoder." Journal of network and systems management 30(1) 22.
11. H. Menear. (2021). IDC Predicts Used Smartphone Market Will Grow 11.2% by 2024. Accessed: Oct. 30, 2022. [Online]. Available: https://mobile-magazine.com/mobile-operators/idc-predicts-used-smartphone market-will-grow-112-2024?page=1.
12. Almomani, Iman, et al. 9 (2021). Android ransomware detection based on a hybrid evolutionary approach in the context of highly imbalanced data. IEEE Access 57674-57691.
13. Shar, Lwin Khin, Ta Nguyen Binh Duong, and David Lo. (2021). Empirical Evaluation of Minority Oversampling Techniques in the Context of Android Malware Detection." 2021 28th Asia-Pacific Software Engineering Conference (APSEC). IEEE.
14. Wang, Zhiqiang, Qian Liu, and Yaping Chi. (2020). Review of Android malware detection based on deep learning. IEEE Access 8, 181102-181126.
15. Namanya, Anitta Patience, et al. (2020). Similarity hash based scoring of portable executable files for efficient malware detection in IoT." Future Generation Computer Systems 110, 824-832.
16. Xu, Zhaozhao, et al. (2020). A hybrid sampling algorithm combining M-SMOTE and ENN based on Random forest for medical imbalanced data." Journal of Biomedical Informatics 107, 103465.
17. Mahdavifar, Samaneh, et al. 2020. Dynamic android malware category classification using semi-supervised deep learning. 2020 IEEE Intl Conf on Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Cloud and Big Data Computing, Intl Conf on Cyber Science and Technology Congress (DASC/PiCom/CBDCom/CyberSciTech). IEEE,
18. C. D. Vijayanand and K. S. Arunlal, . (2019). Impact of malware in modern society,'' J. Sci. Res. Develop., 2(1), 593–600
19. Roopa, H., and T. Asha.(2019). A linear model based on principal

component analysis for disease prediction. IEEE Access 7, 105314-105318.
20. Yerima, Suleiman Y., and Sakir Sezer.(2018). Droidfusion: A novel multilevel classifier fusion approach for android malware detection." IEEE transactions on cybernetics 49.2, 453-466.
21. Li, Jin, et al. (2018). Significant permission identification for machine-learning-based android malware detection." IEEE Transactions on Industrial Informatics 14.7 , 3216-3225.
22. Kim, TaeGuen, et al. (2018). A multimodal deep learning method for android malware detection using various features." IEEE Transactions on Information Forensics and Security 14.3, 773-788.
23. Tao, Guanhong, et al. (2017). MalPat: Mining patterns of malicious and benign Android apps via permission-related APIs. IEEE Transactions on Reliability 67.1, 355-369.
24. Xue, Yinxing, et al. (2017). Auditing anti-malware tools by evolving android malware and dynamic loading technique." IEEE Transactions on Information Forensics and Security 12.7, 1529-1544.
25. Wang, Xin, et al. "Mlifdect: android malware detection based on parallel machine learning and information fusion." Security and Communication Networks 2017.1 (2017): 6451260.
26. Saracino, Andrea, et al. (2016). Madam: Effective and efficient behavior-based android malware detection and prevention." IEEE Transactions on Dependable and Secure Computing 15.1, 83-97.
27. Yuan, Zhenlong, Yongqiang Lu, and Yibo Xue. (2016). Droiddetector: android malware characterization and detection using deep learning. Tsinghua Science and Technology 21.1, 114-123.
28. Xu, Ke, Yingjiu Li, and Robert H. Deng. (2016). Icc detector: Icc-based malware detection on android. IEEE Transactions on Information Forensics and Security 11.6 1252-1264.
29. Abdulla, Shubair, and Altyeb Altaher. (2016). Intelligent approach for android malware detection. KSII Transactions on Internet and Information Systems (TIIS) 9.8, 2964-2983.
30. Lindorfer, Martina, Matthias Neugschwandtner, and Christian Platzer. (2016). Marvin: Efficient and comprehensive mobile app classification through static and dynamic analysis. 2015 IEEE 39th annual computer software and applications conference. Vol. 2. IEEE..
31. Abdulla, Shubair, and Altyeb Altaher. (2015). .Intelligent approach for android malware detection. KSII Transactions on Internet and Information Systems (TIIS) 9.8, 2964-2983.
32. Faruki, Parvez, et al. (2014). Android security: a survey of issues, malware penetration, and defenses. IEEE communications surveys & tutorials 17.2 998-1022.3.
33. Zarni Aung, Win Zaw. (2013). Permission-based android malware detection. International Journal of Scientific & Technology Research 2.3, 228-234.
34. Arp, Daniel, et al. (2013). Drebin: efficient and explainable detection ofandroid malware in your pocket." Georg-August Institute of Computer Science, Technical Report.
35. Michael, Spreitzenbarth, et al. (2013). Mobilesandbox: Looking deeper into android applications." Proceedings of the 28th International ACM Symposium on Applied Computing (SAC).
36. Zhou, Yajin, and Xuxian Jiang. (2012). Dissecting android malware: Characterization and evolution. 2012 IEEE symposium on security and privacy. IEEE.

AUTHORS

Enugala Rajitha received her B.Tech degree from Kakatiya University, Telangana, India in 2013 and M.Tech degree in Computer Science and Engineering from SR Engineering College, Telanagana, India in 2016. She is currently pursuing PhD at the Department of Computer Science and Engineering, National Institute of Technology Calicut, Kerala, India. Her areas of interest are Malware detection, Machine learning and Deep Learning.

Corresponding author E-mail:
rajitha_p210051cs@nitc.ac.in

Anil Pinapati received his B.Tech degree from Vignan's Engineering college, Andhra Pradesh, India in 2005 and M.Tech degree in Computer Science and Engineering from Vignan's Engineering college, Andhra Pradesh, India in 2010 and PhD degree in Computer Science and Engineering from National Institute of Technology Warangal, India in 2018. His areas of interest are Cryptography, Data Hiding and Information Security, Elliptic Curve Cryptography and Machine Learning, Malware Detection.

E-mail: anilpinapati@nitc.ac.in