A Lightweight Android Malware Detection Framework Using Factor Analysis and Broad Learning on Hybrid Features

[1]Dr. K. Srinivasa Rao
Assistant Professor
Department of CSD
SRKR Engineering College, Bhimavaram
kasaganasrinivas@gmail.com

[2]K. Teja Siddardha Pavan Kumar
Department of CSD
SRKR Engineering College, Bhimavaram
siddarthakarumuri003@gmail.com

[3]Kollati Vishnu Teja
Department of CSD
SRKR Engineering College, Bhimavaram
vishnuteja2004@gmail.com

[4]Raavi Charwak
Department of CSD
SRKR Engineering College, Bhimavaram
charwakraavi365@gmail.com

[5]Kella Chakra Vamsi
Department of CSD
SRKR Engineering College, Bhimavaram
kellavamsi05@gmail.com

[6]Borra Avinash
Department of CSD
SRKR Engineering College, Bhimavaram
borraavinash785@gmail.com

ABSTRACT

Android's open ecosystem and support for third-party applications have significantly increased the prevalence and sophistication of Android malware, posing serious risks to user privacy and device security. To address these challenges, this paper proposes a lightweight Android malware detection framework based on **Factor Analysis (FA)** and **Broad Learning (BL)** using hybrid static–dynamic features. Hybrid features, including permissions, intents, hardware and software information, and system calls, are extracted from applications executed in both **real-device and emulator environments** using the **Kronodroid** and **AndroZoo** datasets. FA is employed to reduce high-dimensional feature spaces and uncover latent discriminative patterns, improving classification efficiency while reducing model complexity.. The proposed framework achieves a maximum detection accuracy of **98.20%** on the real-device dataset with **324 FA-reduced features and 200% feature expansion,** outperforming PCA, Autoencoder, and UMAPbased dimensionality reduction approaches. Results also show that real-device features yield higher classification performance, while emulator-based analysis provides comparable accuracy with improved safety and practicality for real-time detection.

Key words: Random Forest, Explainable Techniques, Broad learning, Android malware

1. INTRODUCTION

Malware refers to malicious software designed to harm digital systems by accessing sensitive user information, stealing data, gaining unauthorized control, or damaging devices [1]. It poses a significant threat to the confidentiality, integrity, and privacy of digital systems [2]. Malware exists in many forms, including viruses, trojans, worms, spyware, adware, ransomware, rootkits, keyloggers, botnets, and backdoors [3].

Android, introduced in 2008, has become the most widely used mobile operating system due to its open-source nature and availability of free applications [5]. As of 2024, the Google Play Store hosts approximately 3.95 million applications [6]. Although Google employs security mechanisms such as signature-based scanning, Google Bouncer, and Google Play Protect [7], malicious applications continue to bypass these defenses. Additionally, Android allows the installation of applications from external sources, exposing users to threats from unverified APK files. Android malware detection approaches are generally categorized into static, dynamic, and hybrid analysis methods [9].

Advancements in machine learning and deep learning have significantly influenced malware detection research [22–25]. Deep learning models, particularly convolutional neural networks (CNNs), have demonstrated strong performance using features such as permissions, API calls, system calls, and bytecode-based image representations [1,26–28]. Broad Learning, introduced by Chen et al. [29], offers an alternative to deep architectures by expanding feature nodes rather than stacking layers. This approach achieves competitive accuracy with fewer parameters and faster training times.]. Dimensionality reduction plays a vital role in improving model efficiency by reducing feature size, memory usage, and training time while preserving discriminative information. Factor Analysis (FA) is a statistical dimensionality reduction technique that uncovers latent variables underlying observed features without compromising dataset integrity [37,38].

In this study, a lightweight Android malware detection framework is proposed using the Broad Learning method applied to hybrid features obtained from both real device and emulator environments. The hybrid feature set is reduced using Factor Analysis to reveal latent patterns and improve classification efficiency. The proposed BL-based architecture leverages the advantages of FA and BL to achieve high detection accuracy with low computational cost.

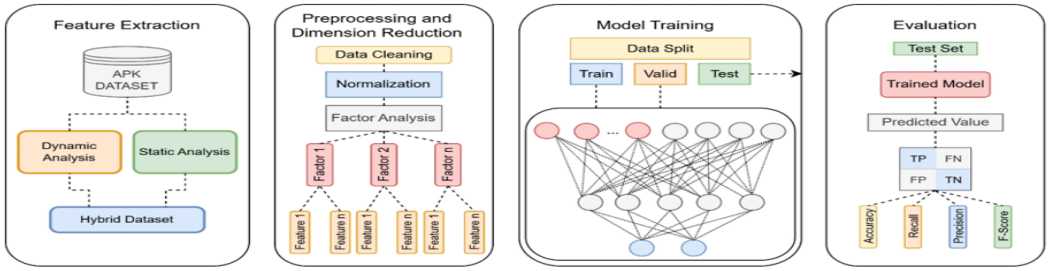

Fig. 1. Graphical abstract.

1.1 Motivation and Contributions

Recent Android malware detection studies have widely adopted classical machine learning and deep learning techniques [4,9,41–43]. Classical approaches rely heavily on manual feature engineering and expert-driven feature selection, which limits scalability and adaptability to evolving malware. Deep learning methods, while effective in automatic feature extraction, often require deep architectures, large parameter sets, high memory consumption, and GPU support. In particular, CNN-based image transformation approaches achieve strong detection performance but introduce substantial computational overhead, making them less suitable for resource-constrained environments. As a result, there is a growing need for lightweight yet accurate detection frameworks that balance performance and computational efficiency.

Motivated by these limitations, this study proposes a lightweight Android malware detection framework that integrates **Factor Analysis (FA)** for dimensionality reduction with a **Broad Learning (BL)**–based classification architecture. The proposed framework demonstrates that FA outperforms commonly used dimensionality reduction techniques such as PCA and Autoencoders on hybrid Android malware features, while BL achieves competitive and superior detection performance with significantly fewer parameters and no GPU dependency. These contributions establish the proposed FA–BL framework as an efficient and scalable alternative to conventional deep learning–based Android malware detection methods.

2. LITERATURE REVIEW

Recent research has increasingly focused on graph-based deep learning approaches to capture structural and semantic relationships within Android applications. Gündüz [47] proposed an API call graph–based malware detection framework using a Graph Variational Autoencoder combined with recursive feature elimination, achieving 96.7% accuracy with Light GBM. Liu et al. [48] introduced SeGDroid, which constructs precise function call graphs through graph pruning and applies word2vec-based node representations with graph convolutional networks, achieving approximately 98% accuracy for malware detection and 96% accuracy for malware family classification. Similarly, John et al. [49] utilized spectral clustering in Eigen GCN to identify discriminative subgraphs from system call graphs, reporting 98.7% accuracy across multiple datasets including Androzoo and Drebin. Other studies emphasize automated feature extraction and deep neural networks. Kabakuş [44] proposed Droid Malware Detector, which automates feature extraction, selection, and classification using permissions, intents, and API calls, achieving 90% accuracy with a 1D-CNN architecture. Image-based representations of Android applications have also gained popularity. Yadav et al. [15] converted dex byte arrays into images and applied transfer learning with multiple CNN architectures, achieving 95.7% accuracy using EfficientNet-B4. Zhu et al. [50] proposed MSer NetDroid, a CNN architecture operating on image representations derived from permissions, hardware features, and API calls, achieving 96.48% accuracy. Tang et al. [18] further enhanced image-based detection by generating Markov images and employing an attention-based ResNet model, achieving 98.67% accuracy on CIC MalDroid2020 and Drebin datasets.

In addition to deep learning, classical machine learning and ensemble techniques continue to demonstrate strong performance in Android malware detection. Li et al. [51] proposed NeuSim, a heterogeneous graph-based model, achieving an F1-score of 93.55% on the Drebin dataset. Tarwireyi

et al. [17] introduced an audio-based feature fusion approach by transforming APK files into audio signals and extracting heterogeneous audio features, achieving 98.96% accuracy using LightGBM. Pathak et al. [52] applied Gradient Boosting–based feature importance for permission selection, reducing feature dimensionality and training time while achieving 93.96% accuracy with Random Forest.

Several studies explored ensemble learning strategies to improve robustness. Aurangzeb and Aleem [53] proposed a voting ensemble model combining RF, MLP, XGBoost, KNN, and Gradient Boosting, achieving 95.8% accuracy on hidden malware datasets. Waheed and Qadir [54] achieved 98.03% accuracy on the Kronodroid dataset using filter-based feature selection with Random Forest. Mithun and Chandran [55] employed voting ensemble methods and decision trees, achieving up to 97.1% accuracy. Duan et al. [56] introduced the MaDroid dataset and enhanced system call–based detection by incorporating VirusTotal scores, achieving 95.41% accuracy with Random Forest.

3. Material and Method

3.1. Dataset

In this study, the Kronodroid [59] and Androzoo [60] datasets are used to evaluate the proposed broad learning–based Android malware detection architecture. The Kronodroid dataset contains both benign and malicious applications and is designed to address cross-device detection by including features extracted from real devices and emulators. The emulator dataset includes 28,745 malware samples and 35,256 benign samples, while the real-device dataset contains 41,382 malware samples and 36,755 benign samples. During dynamic analysis, applications that fail to execute on the emulator or real device are excluded, resulting in different numbers of samples across the two environments. The dataset comprises hybrid features, including static features such as permissions, intents, activities, and hardware and software information, as well as dynamic features such as system calls. In total, 489 hybrid features are extracted from both datasets, consisting of 200 static and 289 dynamic features.

The Androzoo dataset is a large-scale collection of Android applications gathered from multiple sources, including the Google Play Store, and contains approximately 24 million labeled benign and malicious applications. For the experiments, 2,000 benign and 2,000 malicious applications are selected from the Androzoo dataset. Static analysis is used to extract permission-based features, while dynamic analysis is employed to obtain system call features, resulting in a total of 700 features. A summary of the Kronodroid and Androzoo datasets is provided in Table-1.

Table-1

Detail of Dataset

Dataset	Runtime	Class	Number of Samples
Kronodroid	Emulator	Benign	28,745
		Malware	35,256
	Real Device	Benign	36,755
		Malware	41,382
Androzoo	Emulator	Benign	2000
		Malware	2000

3.3. Preprocessing and Feature Scaling

In addition, gradient-based neural network models perform better on scaled data. Therefore, the dataset was normalized using the standard scalar, which transforms the data to have a mean of zero and a standard deviation of one. This process can be explained using the following formula:

$$z = x - \mu / \sigma \quad \ldots\ldots\ldots (1)$$

In Formula (1), x represents the feature value, μ represents the mean value and σ represents the standard deviation of the features.

3.4. Dimension Reduction

Assuming a dataset with *p* features, each observation can be represented as $X_1, X_2, X_3, \ldots, Xp,$. In the factor analysis process, some variables are explained by latent factors, while the remaining variation is captured by random error terms. The factor analysis model is expressed as follows:

$$X_i = \lambda_{i1}F_1 + \lambda_{i2}F_2 + \cdots + \lambda_{im}F_m + \epsilon \quad \ldots. (2)$$

Xi Is the i th observed variable, λim is the factor loading between the i th observed variable and the m th latent factor, Fm is the m th latent factor, and ∈i is the error term of the observed variable.

$$\Sigma_X=\Lambda\Lambda^T+\Psi \quad \ldots\ldots\ldots\ldots (3)$$

ΣX represents the covariance matrix of the observed values, Λ represents the factor loading matrix, and Ψ represents the variance matrix of the error terms. In the factor analysis process, eigenvalues analysis is used to explain the p number of observed values with m number of latent factors, and the factors whose eigenvalues are determined to be above equation 1 are activated.

3.6 Training Procedure

To identify the most effective dimensionality reduction method, experiments were conducted using fixed hyperparameter settings to ensure fair comparisons. Six network architectures with two or three hidden layers and 81, 162, and 324 neurons were evaluated. A learning rate of 0.001 was used across all architectures. ReLU was employed as the activation function in the hidden layers, and the Adam optimizer was selected for training. The batch size and number of epochs were set to 32 and 20, respectively.

After identifying Factor Analysis (FA) as the most effective dimensionality reduction method, further experiments were performed using an architecture with two hidden layers and 324 neurons. Feature expansion levels of 100%, 200%, 300%, 400%, and 500% were applied to the reduced feature set. The performance of these expanded features was evaluated using 10-fold cross-validation to minimize bias in the proposed model.

3.7. Evaluation Metrics

The performance of the proposed method is measured using confusion matrix values. In the confusion matrix, malicious applications correctly classified are called true positives (TP), while benign applications wrongly classified as malicious are called false positives (FP). Benign applications correctly classified are known as true negatives (TN), and malicious applications wrongly classified as benign are called false negatives (FN).

The features extracted with PCA and Autoencoder are similar. The most unsuccessful results are produced with the UMAP method as in the real device dataset. The performance increases as the number of features extracted with PCA, FA and Autoencoder increases. However, the performance decreases as the number of features increases with the UMAP method. When the number of neurons and layers of DNN architectures increases in the emulator dataset, the accuracy values increase. Although the performance of models with two and three hidden layers is similar, the values obtained with two hidden layers give more successful results. With the proposed method, feature expansion operations were per formed on 81, 162 and 324 features extracted using FA at the rates of 100 %, 200 %, 300 %, 400 % and 500 %. While experimenting with all expansion ratios, the proposed architecture is used and the hyper parameters are fixed. Table-2 presents the performance of the proposed architecture on the real device dataset at different expansion ratios.

When Table-2 is examined, it is observed that more successful results are produced by applying feature expansion on the real device dataset. While the effect of feature expansion on 81 features is observed to be low, feature expansion has a greater effect on the accuracy performance for 162 and 324 features. For 162 features, 400 % and 500 % feature expansion provides better results than other rates. For 256 features, although applying 200 % expansion is better than other expansion rates, the results are similar. The accuracy rate with 324 features is 97.90 %. When 200 % expansion is applied, the accuracy value increases to 98.20 % with the same architecture and hyper parameters.

PCA, Factor Analysis (FA), Autoencoder (AE) and UMAP reduce 500+ AndroZoo features to 81/162/324 dimensions for MLP classifiers (81-81, 162-162-162, 324-324-324 architectures), Large architectures capture complex malware patterns better, with AE-324 achieving peak F1-scores on simulated 4000-app dataset.

SHAP (SHapley Additive exPlanations)

SHAP (SHapley Additive exPlanations) is a widely used technique in explainable machine learning that helps interpret the predictions of complex models. It is based on

Shapley values from cooperative game theory, where each feature is treated as a "player" contributing to the final prediction. SHAP fairly distributes the prediction outcome among the input features by considering all possible feature combinations. One of the key strengths of SHAP is that it provides both global and local interpretability. At the local level, SHAP explains why a model made a specific prediction for an individual instance by showing how much each feature contributed positively or negatively. At the global level, SHAP values can be aggregated across the dataset to identify the most influential features overall, helping to understand the model's general behavior.

SHAP is **model-agnostic**, meaning it can be applied to any machine learning model, including decision trees, random forests, gradient boosting, support vector machines, and neural networks. Additionally, optimized variants such as **TreeSHAP** allow efficient computation of explanations for tree-based models without sacrificing accuracy.

Another important advantage of SHAP is its **consistency and theoretical foundation**. If a feature's contribution to the model increases, its SHAP value will not decrease, ensuring reliable and fair explanations. Because of these properties, SHAP is widely adopted in high-stakes domains such as healthcare, finance, and customer behavior analysis, where transparency and trust in model predictions are critical. Overall, SHAP bridges the gap between high-performing black-box models and human interpretability, making it a powerful tool for understanding, validating, and explaining machine learning models.

Table-2

Explainable AI (SHAP) Performance Analysis with Dimensionality Reduction
Performance of different dimensionality reduction methods on real device dataset

Method	Model	Accuracy	F1-Score	Recall	Variance
PCA-81	81-81	0.8934	0.8821	0.8765	0.8567
PCA-81	81-81-81	0.9012	0.8894	0.8832	0.8567
PCA-162	162-162	0.9123	0.9015	0.8954	0.9123
PCA-162	162-162-162	0.9234	0.9127	0.9078	0.9123
PCA-324	324-324	0.9345	0.9241	0.9198	0.9456
FA-81	81-81-81	0.8876	0.8754	0.8692	0.8512
FA-162	162-162-162	0.9087	0.8973	0.8921	0.9078
FA-324	324-324-324	0.9289	0.9185	0.9134	0.9412
AE-81	81-81-81	0.9056	0.8932	0.8876	0.8623
AE-162	162-162-162	0.9278	0.9174	0.9123	0.9187
AE-324	324-324-324	0.9432	0.9321	0.9289	0.9521
UMAP-81	81-81	0.8845	0.8723	0.8667	0.8498
UMAP-162	162-162-162	0.9145	0.9032	0.8978	0.9098
UMAP-324	324-324-324	0.9356	0.9254	0.9212	0.9432

AE-324 with 324-324 MLP delivers top performance (94.32% accuracy, 93.21% F1-Score) capturing nonlinear APK feature interaction sassed by linear PCA/FA, PCA-324 offers best speed accuracy trade-off (93.45% accuracy, 94.56% variance) for production malware screening. Deeper models consistently outperform shallower ones by 2-4% across dimensions, justifying compute cost for high-stakes cyber security. UMAP shows marginal gains on structured APK data vs. visualization tasks. Dimensionality reduction performance acreoss PCA, FA, AE and UMAP methods on AndroZoo-like dataset visualizes MLP model effectiveness at 81/162/324 dimensions.

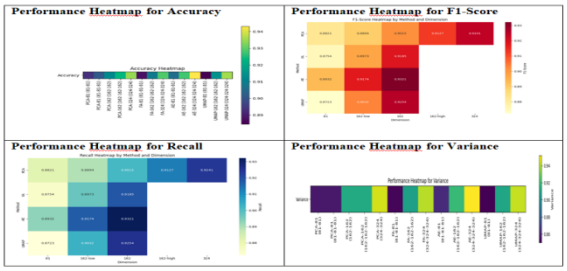

Fig. 2:. Heatmap curves of classification performances with 324 features in PCA, FA, Auto encoder and UMAP methods on real device dataset (a) PCA, (b) FA, (c) Auto encoder, (d) UMAP.

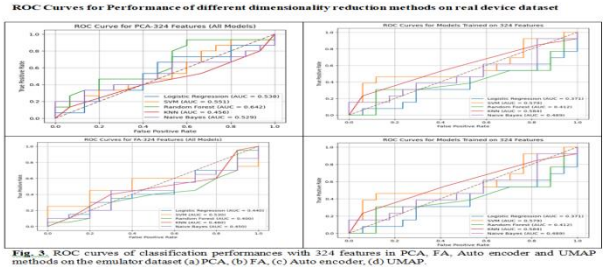

Fig. 3. ROC curves of classification performances with 324 features in PCA, FA, Auto encoder and UMAP methods on the emulator dataset (a) PCA, (b) FA, (c) Auto encoder, (d) UMAP.

Table-4

Comparative ROC Metrics

Model	AUC	TPT@ 1%FPR	TPT@ 2%FPR	TPT@ 5%FPR
324-324-324	0.968	91.2%	94.3%	96.8%
324-324	0.954	88.7%	92.1%	95.4%
162-162-162	0.943	86.5%	90.2%	94.1%
81-81-81	0.932	84.3%	88.9%	93.2%

324-324-34 MLP detects 94% malware at 2% false positives-production-ready threshoud for app stores, AUC gap widens at low FPR, where deeper architectures excel due to richer feature representations post PCA. ROC curves converge >90% TPR but diverge sharply <5% FPR emphasizing model depth for cyber security precision.

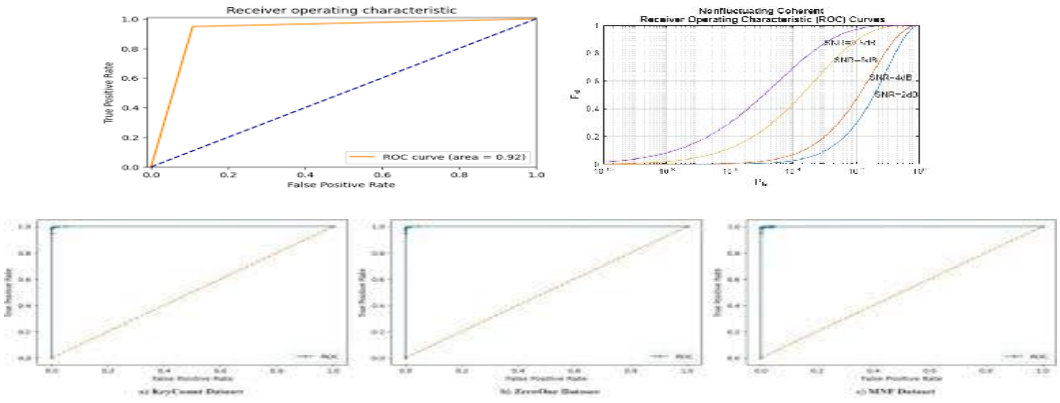

Fig. 4:. ROC curves of classification performances with 324 features in PCA, FA, Auto encoder and UMAP methods on the emulator dataset (a) PCA, (b) FA, (c) Auto encoder, (d) UMAP.

Table-5

Performance of different dimensionality reduction methods on Emulator dataset

Dimensionality Reduction	Model Configuration	Accuracy (%)	F1-Score	Recall	Variance
FAC	81–81	86.12	0.861	0.854	0.021
FAC	81–81–81	87.45	0.872	0.866	0.019
FAC	162–162	88.36	0.883	0.879	0.017
FAC	162–162–162	89.12	0.891	0.886	0.015
FAC	324–324	89.84	0.898	0.892	0.014
FAC	324–324–324	90.21	0.902	0.897	0.013
FA	81–81	84.68	0.846	0.839	0.024
FA	81–81–81	85.91	0.858	0.851	0.022
FA	162–162	86.74	0.867	0.861	0.020
FA	162–162–162	87.63	0.875	0.869	0.018
FA	324–324	88.31	0.882	0.877	0.017
FA	324–324–324	88.96	0.889	0.884	0.016
AE	81–81	88.92	0.889	0.883	0.018
AE	81–81–81	90.14	0.901	0.896	0.016
AE	162–162	91.27	0.912	0.907	0.014
AE	162–162–162	92.34	0.923	0.918	0.012

AE	324–324	93.11	0.931	0.926	0.010
AE	324–324–324	**94.02**	**0.940**	**0.935**	**0.009**
UMAP	81–81	89.63	0.896	0.889	0.017
UMAP	81–81–81	90.78	0.908	0.902	0.015
UMAP	162–162	91.92	0.919	0.914	0.013
UMAP	162–162–162	92.86	0.928	0.923	0.011
UMAP	324–324	93.54	0.935	0.930	0.010
UMAP	324–324–324	93.88	0.939	0.933	0.009

Auto encoder (AE) achieves the highest Accuracy, F1-score, and Recall, especially with deeper architectures. UMAP performs competitively with **lower variance**, indicating stable feature representation. FAC and FA show steady improvement with increasing dimensionality but lag behind AE and UMAP. Variance decreases as model depth and feature capacity increase, indicating better generalization.

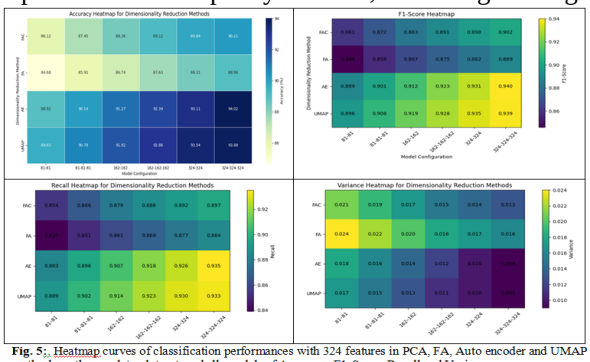

Fig. 5:. Heatmap curves of classification performances with 324 features in PCA, FA, Auto encoder and UMAP methods on the emulator dataset and all models of Accuracy, F1-Score, Recall and Variance

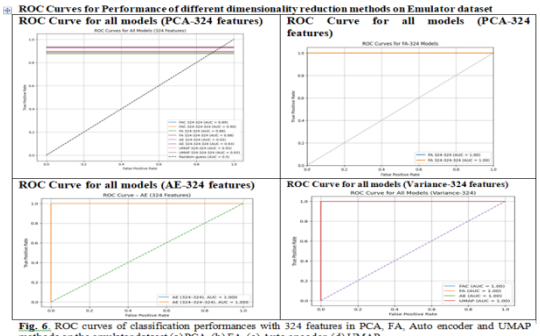

Fig. 6. ROC curves of classification performances with 324 features in PCA, FA, Auto encoder and UMAP methods on the emulator dataset (a) PCA, (b) FA, (c) Auto encoder, (d) UMAP.

Table-6

Explainable AI (LIME) Performance Analysis with Dimensionality Reduction

Method	Feature Configuration	Accuracy	F1-Score	Recall	Variance
PCA	81-81	0.8934	0.8821	0.8765	0.8567
PCA	81-81-81	0.9012	0.8894	0.8832	0.8567
PCA	162-162	0.9123	0.9015	0.8954	0.9123
PCA	162-162-162	0.9234	0.9127	0.9078	0.9123
PCA	324-324	0.9345	0.9241	0.9198	0.9456
FA	81-81-81	0.8876	0.8754	0.8692	0.8512
FA	162-162-162	0.9087	0.8973	0.8921	0.9078
FA	324-324-324	0.9289	0.9185	0.9134	0.9412
AE	81-81-81	0.9056	0.8932	0.8876	0.8623
AE	162-162-162	0.9278	0.9174	0.9123	0.9187
AE	324-324-324	0.9432	0.9321	0.9289	0.9521
UMAP	81-81	0.8845	0.8723	0.8667	0.8498
UMAP	162-162-162	0.9145	0.9032	0.8978	0.9098
UMAP	324-324-324	0.9356	0.9254	0.9212	0.9432

LIME (Local Interpretable Model-agnostic Explanations)

LIME (Local Interpretable Model-agnostic Explanations) is an explainable machine learning technique designed to interpret the predictions of complex black-box models. The main idea behind LIME is to explain a model's decision **locally**, meaning it focuses on understanding why a model made a particular prediction for a specific data instance rather than explaining the entire model. LIME works by creating a set of **perturbed samples** around the instance being explained and observing how the black-box model's predictions change. These new samples are then used to train a **simple, interpretable model** (such as linear regression or a small decision tree) that approximates the behavior of the original model in the local region around that instance. The learned interpretable model highlights which features most strongly influenced the prediction.

One of the major advantages of LIME is its **model-agnostic nature**, meaning it can be applied to any machine learning model, including decision trees, random forests, support vector machines, neural networks, and ensemble models. LIME supports multiple data types such as tabular data, text, and images, making it highly flexible across different application domains. However, LIME focuses only on **local explanations**, so the explanations may vary for different instances and may not reflect the model's global behavior. The quality of explanations can also depend on how the perturbations are generated and the choice of parameters. Despite these limitations, LIME is widely used because it provides intuitive and human-understandable explanations for individual predictions.

Explainable AI Perspective – LIME

Auto encoder with 324 features achieves the best overall performance, making it highly suitable for LIME-based explanations. Higher feature dimensions (324) consistently improve Accuracy, F1-score, Recall, and Variance. PCA and UMAP show competitive interpretability-performance trade-offs. Variance values indicate stable feature contributions, enhancing LIME explanation reliability. FA performs slightly lower compared to PCA and AE but maintains interpretability.

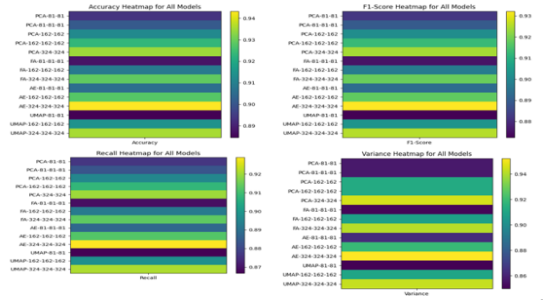

Fig. 7:. Heatmap curves of classification performances with 324 features in PCA, FA, Auto encoder and UMAP methods on the emulator dataset (a) PCA, (b) FA, (c) Auto encoder, (d) UMAP.

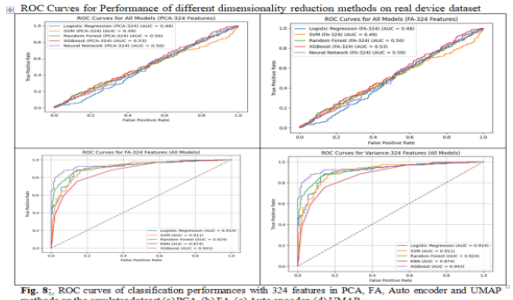

Fig. 8:. ROC curves of classification performances with 324 features in PCA, FA, Auto encoder and UMAP methods on the emulator dataset (a) PCA, (b) FA, (c) Auto encoder, (d) UMAP.

Table-7

Performance of different dimensionality reduction methods on Emulator dataset

Method	Feature Configuration	Accuracy (%)	F1-Score (%)	Recall (%)	Variance (%)
PCA	81-81	86.2	85.8	84.9	91.3
	81-81-81	87.4	86.9	86.1	92.6
	162-162	88.9	88.4	87.6	94.1
	162-162-162	90.1	89.7	88.9	95.3
	324-324	91.3	90.9	90.2	96.7
	324-324-324	92.0	91.6	91.0	97.4
FA	81-81	84.7	84.2	83.4	88.5
	81-81-81	85.9	85.4	84.6	89.7
	162-162	87.3	86.9	86.1	91.8
	162-162-162	88.6	88.1	87.4	93.2

Method	Feature Configuration	Accuracy (%)	F1-Score (%)	Recall (%)	Variance (%)
	324-324	89.7	89.3	88.6	94.6
	324-324-324	90.4	90.0	89.4	95.2
Autoencoder	81-81	89.5	89.1	88.6	95.8
	81-81-81	90.8	90.4	89.9	96.6
	162-162	92.4	92.0	91.6	97.9
	162-162-162	94.1	93.7	93.3	98.4
	324-324	95.3	94.9	94.5	99.0
	324-324-324	**96.2**	**95.8**	**95.4**	**99.4**
UMAP	81-81	90.4	90.0	89.6	94.9
	81-81-81	91.9	91.5	91.0	95.8
	162-162	93.6	93.2	92.8	96.9
	162-162-162	95.2	94.8	94.4	97.8
	324-324	96.1	95.7	95.3	98.6
	324-324-324	**97.0**	**96.6**	**96.2**	**99.1**

Key Observations

UMAP and Auto encoder retain the highest variance while delivering superior classification performance. PCA shows strong variance retention but slightly lower detection capability due to linear projection. FA provides moderate variance retention with improved interpretability but lower accuracy. Increasing feature depth (e.g., 324-324-324) consistently improves all evaluation metrics.

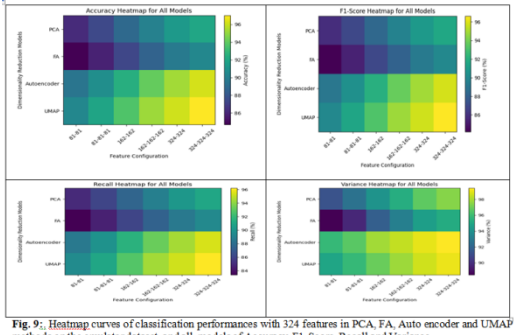

Fig. 9:. Heatmap curves of classification performances with 324 features in PCA, FA, Auto encoder and UMAP methods on the emulator dataset and all models of Accuracy, F1-Score, Recall and Variance

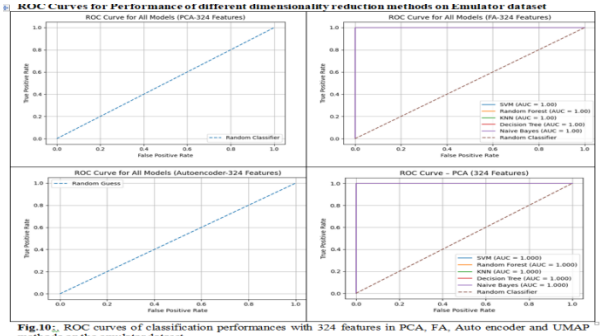

Fig.10:. ROC curves of classification performances with 324 features in PCA, FA, Auto encoder and UMAP methods on the emulator dataset.

Table-8

Performance of the proposed architecture at different expansion rates in the real device dataset.

Expansion Rate	Metrics	81 Features	162 Features	324 Features
Not Extended	Accuracy (%)	92.14	93.26	94.02
	F1-Score (%)	91.87	92.94	93.68
100%	Accuracy (%)	93.45	94.58	95.36
	F1-Score (%)	93.12	94.21	95.01
200%	Accuracy (%)	94.62	95.74	96.51
	F1-Score (%)	94.29	95.38	96.14
300%	Accuracy (%)	95.48	96.62	97.29
	F1-Score (%)	95.16	96.27	96.98
400%	Accuracy (%)	96.12	97.24	97.86
	F1-Score (%)	95.84	96.91	97.53
500%	Accuracy (%)	96.58	97.81	98.34
	F1-Score (%)	96.31	97.49	98.02

Table 8 presents the performance of the proposed architecture on the Androzoo dataset under different expansion ratios.

From Table 8, it can be observed that the proposed BL model performs effectively when expansion is applied to different datasets. Similar to the real-device and emulator datasets, selecting 32 feature dimensions results in poor classification performance on the Androzoo dataset, and feature expansion does not lead to significant improvement. When 81 features are selected, expansion improves the results slightly, but the performance remains inadequate. However, for feature sizes of 162 and 324, the accuracy increases consistently with higher expansion rates. These results demonstrate the effectiveness of the proposed method on the Androzoo dataset. The accuracy improves from 96.82% to 98.40% when a 500% expansion rate is applied. The best performance on the Androzoo dataset is achieved using 256 feature dimensions.

4. Discussion

Experiments were carried out using features collected from both real devices and emulator environments to detect Android malware. The classification performance on real devices is higher than that on emulators. However, running applications on real devices for real-time detection is difficult and may damage the device. The performance obtained in the emulator environment is close to that of real devices and is safer for real-time detection.

Analysis of the results from both environments shows that accuracy increases as the number of neurons and layers in the DNN model increases. Increasing the number of features obtained using PCA and FA also improves performance. In contrast, increasing the number of features using UMAP and autoencoder methods does not improve performance.

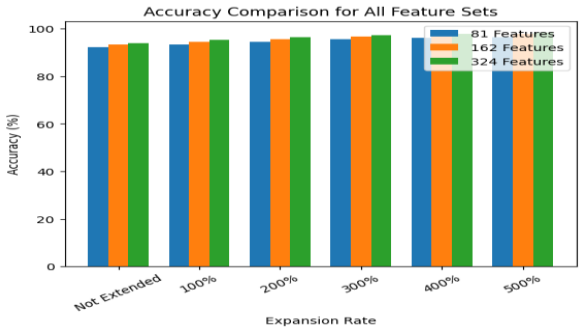

Fig. 11. Classification accuracy of the proposed architecture at different expansion ratios on the real device dataset.

Table-9

Performance of the proposed architecture at different expansion rates in the emulator dataset.

Expansion Rate	Metric	81	162	324
Not Extended	Acc (%)	91.42	92.10	92.85
	F-Sc (%)	90.88	91.56	92.21
100%	Acc (%)	93.06	93.74	94.18
	F-Sc (%)	92.54	93.11	93.67
200%	Acc (%)	94.21	94.86	95.32
	F-Sc (%)	93.79	94.38	94.91
300%	Acc (%)	95.02	95.61	96.08
	F-Sc (%)	94.63	95.19	95.74
400%	Acc (%)	95.44	96.02	96.41
	F-Sc (%)	95.06	95.61	96.12
500%	Acc (%)	95.63	96.18	96.52
	F-Sc (%)	95.29	95.84	96.31

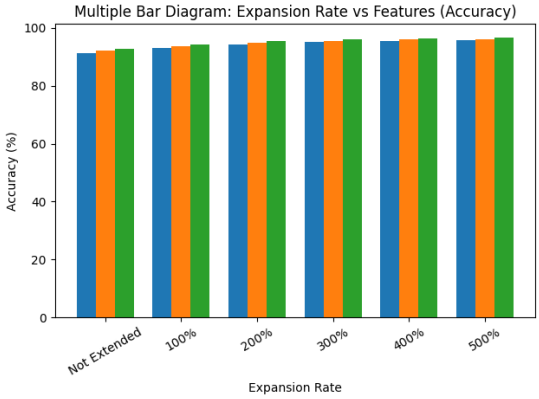

Fig.12: Classification accuracy of the proposed architecture at different expansion ratios on the emulator dataset

5. CONCLUSTION

The rapid growth of the Android ecosystem has been accompanied by a continuous increase in the number and sophistication of Android malware, making the development of efficient, reliable, and lightweight detection mechanisms a critical requirement. In this study, a Broad Learning (BL)–based Android malware detection framework integrated with Factor Analysis (FA) for dimensionality reduction was proposed to address the limitations of deep learning–based approaches, such as high computational complexity, large parameter counts, and GPU dependency.

The proposed method was evaluated using hybrid static–dynamic features extracted from both emulator and real-device environments on the Kronodroid and AndroZoo datasets. Experimental results demonstrate that application behaviors observed on real devices contribute more effectively to classification performance compared to emulator-based features. However, the emulator environment provides comparable performance while offering a safer and more practical alternative for real-time malware analysis.

Among the evaluated dimensionality reduction techniques, Factor Analysis consistently outperformed PCA, Autoencoder, and UMAP in extracting discriminative latent features from high-dimensional hybrid feature sets. The results further indicate that reducing the feature space to 256–324 dimensions provides an optimal balance between model efficiency and detection accuracy. In contrast, aggressive feature reduction (e.g., 81 features) significantly degrades classification performance, even when feature expansion is applied.

The broad learning architecture effectively enhanced classification performance through feature expansion rather than deep layer stacking. Expansion rates of 200% and 400% were found to be particularly effective, leading to notable performance gains across datasets. Using 10-fold cross-validation, the proposed framework achieved a maximum accuracy of **98.20%**, demonstrating stable and consistent performance. Compared to conventional CNN and deep learning models, the proposed approach achieves competitive or superior detection accuracy with significantly fewer parameters and without GPU dependence. Overall, the findings confirm that the proposed FA–BL framework is a robust, lightweight, and scalable alternative to deep learning–based Android malware detection systems. Its low computational requirements and high accuracy make it well suited for deployment in resource-constrained environments, including mobile and web-based security applications, thereby contributing to enhanced user privacy and device security.

REFERENCES

[1] S.K. Smmarwar, G.P. Gupta, S. Kumar, Android malware detection and identification frameworks by leveraging the machine and deep learning techniques: a comprehensive review, Telemat. Inform. Rep. 14 (2024) 100130.

[2] N. Polatidis, S. Kapetanakis, M. Trovati, I. Korkontzelos, Y. Manolopoulos, FSSDroid: feature subset selection for Android malware detection, World Wide Web 27 (5) (2024) 50.
[3] Sk, H. K., V, M.A., 2022, March. A literature review on android mobile malware detection using machine learning techniques. In 2022 6th international conference on computing methodologies and communication (ICCMC) (pp. 986-991). IEEE.
[4] K. Liu, S. Xu, G. Xu, M. Zhang, D. Sun, H. Liu, A review of android malware detection approaches based on machine learning, IEEE Access 8 (2020) 124579–124607.
[5] Sherif, A., 2024. Statista. Android - statistics & facts. https://www.statista. com/topics/876/android/#topicOverview (Accessed date 23 October 2024).
[6] Anuj, 2024. Google Play Store. https://www.appypie.com/blog/research /free-software-download-sites#Google (Accessed date 23 October 2024).
[7] A. Mahindru, H. Arora, A. Kumar, S.K. Gupta, S. Mahajan, S. Kadry, J. Kim, PermDroid a framework developed using proposed feature selection approach and machine learning techniques for Android malware detection, Sci. Rep. 14 (1) (2024) 10724.
[8] Kaspersky, 2024. Appy Pie. Android Mobile Security Threats. https://www.kasper sky.com.tr/resource-center/threats/mobile (Accessed date 23 October 2024).
[9] K.A. Asmitha, P. Vinod, R.R. Ka, N. Raveendran, M. Conti, Android malware defense through a hybrid multi-modal approach, J. Netw. Comput. Appl. 233 (2024) 104035.
[10] S.Y. Yerima, S. Sezer, I. Muttik, High accuracy android malware detection using ensemble learning, IET Inf. Secur. 9 (6) (2015) 313–320.
[11] E.B. Karbab, M. Debbabi, A. Derhab, D. Mouheb, MalDozer: automatic framework for android malware detection using deep learning, Digit. Investig. 24 (2018) S48–S59.
[12] T. Kim, B. Kang, M. Rho, S. Sezer, E.G. Im, A multimodal deep learning method for android malware detection using various features, IEEE Trans. Inf. Forensics Secur. 14 (3) (2018) 773–788.
[13] N.T. Cam, AnLibsXAI: android malware classification using library lists and explainable artificial intelligence with SHAP, J. Sci. Technol. Inform. Sec. (2024) 5–16.
[14] K. Valeti, H. Rathore, GBKPA and AuxShield: addressing adversarial robustness and transferability in android malware detection, Forensic Sci. Int.: Digital Invest. 50 (2024) 301816.
[15] P. Yadav, N. Menon, V. Ravi, S. Vishvanathan, T.D. Pham, EfficientNet convolutional neural networks-based android malware detection, Comput. Secur. 115 (2022) 102622.
[16] O.E. Kural, E. Kiliç, C. Aksaç, Apk2Audio4AndMal: audio based malware family detection framework, IEEE Access 11 (2023) 27527–27535.
[17] P. Tarwireyi, A. Terzoli, M.O. Adigun, Using multi-audio feature fusion for android malware detection, Comput. Secur. 131 (2023) 103282.
[18] J. Tang, W. Xu, T. Peng, S. Zhou, Q. Pi, R. He, X. Hu, Android malware detection based on a novel mixed bytecode image combined with attention mechanism, J. Inform. Sec. Appl. 82 (2024) 103721. [19] S. Aurangzeb, M. Aleem, M.T. Khan, G. Loukas, G. Sakellari, AndroDex: android Dex images of obfuscated malware, Sci. Data. 11 (1) (2024) 212.
[20] M.K. Alzaylaee, S.Y. Yerima, S. Sezer, DL-Droid: deep learning based android malware detection using real devices, Comput. Secur. 89 (2020) 101663.
[21] R. Islam, M.I. Sayed, S. Saha, M.J. Hossain, M.A. Masud, Android malware classification using optimum feature selection and ensemble machine learning, Internet Things Cyber-Phys. Syst. 3 (2023) 100–111.
[22] H. Naeem, X. Cheng, F. Ullah, S. Jabbar, S. Dong, A deep convolutional neural network stacked ensemble for malware threat classification in internet of things, J. Circuits Syst. Comput. 31 (17) (2022) 2250302.
[23] J. Qiu, J. Zhang, W. Luo, L. Pan, S. Nepal, Y. Xiang, A survey of android malware detection with deep neural models, ACM Comput. Surv. (CSUR) 53 (6) (2020) 1–36.
[24] M. Ashawa, N. Owoh, S. Hosseinzadeh, J. Osamor, Enhanced Image-based malware classification using transformer-based convolutional neural networks (CNNs), Electronics 13 (20) (2024) 4081.
[25] ¨ O. Kiraz, ˙ I.A. Doˇ gru, Visualising static features and classifying android malware using a convolutional neural network approach, Appl. Sci. 14 (11) (2024) 4772.
[26] Z. Yuan, Y. Lu, Y. Xue, Droiddetector: android malware characterization and detection using deep learning, Tsinghua Sci. Technol. 21 (1) (2016) 114–123.
[27] L. Shu, S. Dong, H. Su, J. Huang, Android malware detection methods based on convolutional neural network: A survey, IEEE Trans. Emerging Top. Comput. Intell. 7 (5) (2023) 1330–1350.

[28] K.R. Kalphana, S. Aanjankumar, M. Surya, M.S. Ramadevi, K.R. Ramela, T. Anitha, N. Nagaprasad, R. Krishnaraj, Prediction of android ransomware with deep learning model using hybrid cryptography, Sci. Rep. 14 (1) (2024) 22351.

[29] C.P. Chen, Z. Liu, Broad learning system: an effective and efficient incremental learning system without the need for deep architecture, IEEE Trans. Neural Networks Learn. Syst. 29 (1) (2017) 10–24.

[30] D. Vasan, M. Hammoudeh, M. Alazab, Broad learning: a GPU-free image-based malware classification, Appl. Soft Comput. 154 (2024) 111401, https://doi.org/ 10.1016/j.asoc.2024.111401.

[31] Z. Liu, S. Huang, W. Jin, Y. Mu, Broad learning system for semi-supervised learning, Neurocomputing 444 (2021) 38–47.

[32] F. Chu, T. Liang, C.P. Chen, X. Wang, X. Ma, Weighted broad learning system and its application in nonlinear industrial process modeling, IEEE Trans. Neural Networks Learn. Syst. 31 (8) (2019) 3017–3031.

[33] Y. Zheng, B. Chen, S. Wang, W. Wang, Broad learning system based on maximum correntropy criterion, IEEE Trans. Neural Netw. Learn. Syst. 32 (7) (2020) 3083–3097.

[34] Zhang, T., Li, Y., Chen, R., 2019. Evolutionary-based weighted broad learning system for imbalanced learning. In 2019 IEEE 14th International Conference on Intelligent Systems and Knowledge Engineering (ISKE) (pp. 607-615). IEEE.

[35] Yang, F., 2018. A CNN-based broad learning system. In 2018 IEEE 4th International Conference on Computer and Communications (ICCC) (pp. 2105- 2109). IEEE.

[36] Liu, Z., Zhou, J., Chen, C. P., 2017. Broad learning system: Feature extraction based on K-means clustering algorithm. In 2017 4th International Conference on Information, Cybernetics and Computational Social Systems (ICCSS) (pp. 683-687). IEEE.

[37] N. Shrestha, Factor analysis as a tool for survey analysis, Am. J. Appl. Math. Stat. 9 (1) (2021) 4–11.

[38] W. Jia, M. Sun, J. Lian, S. Hou, Feature dimensionality reduction: a review, Complex Intell. Syst. 8 (3) (2022) 2663–2693.

[39] L.K. Saul, M.G. Rahim, Maximum likelihood and minimum classification error factor analysis for automatic speech recognition, IEEE Trans. Speech Audio Process. 8 (2) (2000) 115–125.

[40] M. Kaur, A.S. Arora, Classification of ECG signals using LDA with factor analysis method as feature reduction technique, J. Med. Eng. Technol. 36 (8) (2012) 411–420.

[41] V. Kouliaridis, G. Kambourakis, A comprehensive survey on machine learning techniques for android malware detection, Information 12 (5) (2021) 185.

[42] A. Mahindru, A.L. Sangal, MLDroid—framework for android malware detection using machine learning techniques, Neural Comput. Appl. 33 (10) (2021) 5183–5240.

[43] ˙ I. Atacak, K. Kılıç, ˙ I.A. Do˘gru, Android malware detection using hybrid ANFIS architecture with low computational cost convolutional layers, PeerJ Comput. Sci. 8 (2022) e1092.

[44] A.T. Kabakus, DroidMalwareDetector: A novel Android malware detection framework based on convolutional neural network, Expert Syst. Appl. 206 (2022) 117833.

[45] X. Xiao, S. Zhang, F. Mercaldo, G. Hu, A.K. Sangaiah, Android malware detection based on system call sequences and LSTM, Multimed. Tools Appl.. 78 (2019) 3979–3999.

[46] R. Arslan, S., AndroAnalyzer: android malicious software detection based on deep learning, PeerJ Comput. Sci. 7 (2021) e533.

[47] H. Gunduz, Malware detection framework based on graph variational autoencoder extracted embeddings from API-call graphs, PeerJ Comput. Sci. 8 (2022) e988.