Sridaran Rajagopal · Priti Sajja ·
Rohit Thanki · Ajay Kumar (Eds.)

Communications in Computer and Information Science 2821

Artificial Intelligence Based Smart and Secured Applications

4th International Conference, ASCIS 2025
Gujarat, India, September 11–13, 2025
Revised Selected Papers, Part III

Part 3

Springer

Leveraging Market Basket Analysis and Clustering for Personalized Retail Recommendations: A Case Study of a Local Supermarket 340
Rathimala Kannan and K. Asher Josh Immanuel

A Systematic Review of Sentiment Analysis for Applications Utilizing Opinion Mining 350
Vyanktesh Dnavantrai Raval and Vimal Parmar

Nonlinear Structure Detection in Cloud Workload Series: A Comparative Study of Testing Methods 374
Rohit Prajapati and Hiren Joshi

Enhancing BB84 Quantum Key Distribution: Analyzing the Impact of Noisy Channels and Future Advancements 386
Tejal Upadhyay, Shailee Kapadia, and Sonia Mittal

AI-Based Early Learning Assistant Using Speech Recognition and Concept Mapping 404
V. Prem Kumar and K. Rajakumari

Optimizing Query Performance in Relational Databases: A Comparative Study of Index Merge Strategies and Caching Techniques 418
Patel Nilen and Himanshu Maniar

Multi-class Land Cover Classification in Satellite Images Using U-Net and Attention Mechanisms 429
T. Lakshmi Prasanna and C. Rajabhushanam

Augmenting Performance Precision of Brain Signals through Corporeal Modes to Identify Bradykinesia Through Multilevel SVM and Entropy Computation 443
R. Syed Jamalullah, L. Mary Gladence, K. Mohammed Gaffoor, and K. Nirmala

Magnetic Resonance Images Detection of Alzheimer's Disease 454
Velumani Thiyagarajan, L. Jaya Singh Dhas, T. Deepika, S. Thilagavathi, and C. Sivaprakasam

Android Malware Detection (AMD) Using Stacked Deep Learning Ensemble Classifier Fusion (SDLECF) with Nature-Inspired Based Ensemble Feature Selection (NIEFS) 466
Anuja A. Rajan and R. Durga

Author Index 491

Android Malware Detection (AMD) Using Stacked Deep Learning Ensemble Classifier Fusion (SDLECF) with Nature-Inspired Based Ensemble Feature Selection (NIEFS)

Anuja A. Rajan[1] and R. Durga[2(✉)]

[1] Department of computer Science, Vels Institute of Science, technology and Advanced Studies, Chennai, Tamil Nadu, India

[2] Department of Advanced Computing & Analytics, Vels Institute of Science, technology and Advanced Studies, Chennai, Tamil Nadu, India

durga.scs@vistas.ac.in

Abstract. Android is the most rapidly expanding mobile computer platform, which has been targeted by a variety of malware. It can be effectively identified using Deep Learning (DL) techniques. Typical feature selection algorithms disregard feature correlation which has been solved by using wrapper-based feature selection models. Wrapper-based techniques take a lot of time to select feature subsets. In this paper, Nature-Inspired Based Ensemble Feature Selection (NIEFS), and Stacked Deep Learning Ensemble Classifier Fusion (SDLECF) classifier has been introduced for combination of DL methods with increased detection accuracy. NIEFS model is developed based on a variety of evolutionary computation approaches, such as the Cauchy Operator Squirrel Search Algorithm (COSSA), Lévy Flight Pigeon-Inspired Optimization (LEFPIO), and Fuzzy Membership Grasshopper Optimization Algorithm (FMGOA) for eliminating redundant or unnecessary features. The outputs of various approaches have been integrated using Mutual Information (MI). SDLECF is introduced by merging many models (Bidirectional Gated Recurrent Unit (Bi-GRU), Sparse Autoencoder based Deep Neural Network (SAE-DNN), Bidirectional Long Short-Term Memory (BDLSTM), and Mean Weight Deep Belief Network (MWDBN)) to attain highest malware detection performance. Bi-GRU can handle data sequences in both forward and backward direction. SAE-DNN includes of three components like an encoder, a decoder, and a classification. BDLSTM classifier is a category of Recurrent Neural Network (RNN) which works on both forward and backward directions. MWDBN includes of Multiple Restricted Boltzmann Machine (RBM) layers for classification. Finally, classifier performance was measured using MATrix LABoratory R2020a (MATLABR2020a) and the metrics like Precision (Pre), Recall (Rec), F-measure (FM), and Weighted F-measure (WFM).

Keywords: Android Malware Detection (AMD) · Nature-Inspired Based Ensemble Feature Selection (NIEFS) · Deep Learning (DL) · Stacked Deep Learning Ensemble Classifier Fusion (SDLECF) · Bidirectional Gated Recurrent Unit (Bi-GRU) · Sparse Autoencoder based Deep Neural Network (SAE-DNN) · and Mean Weight Deep Belief Network (MWDBN)

S. Rajagopal et al. (Eds.): ASCIS 2025, CCIS 2821, pp. 466–490, 2026.
https://doi.org/10.1007/978-3-032-17840-4_31

1 Introduction

In people's daily lives, mobile phones have grown in importance. Android has emerged as the top mobile operating system recently with a significantly bigger share of the global market [1]. Numerous android apps, including those for banking, gaming, leisure, education, and more, can include malware [2]. Android malware samples are more advanced than those that were previously identified in terms of avoiding antivirus detection through encryption and code obfuscation.

A common signature-based detection method is used to find malware [3]. Since the database only has a small number of signatures, signature-based detection techniques are unable to identify unknown malware. It showed a few issues, including the lowest accuracy and slow detection speed. Small variations in the signature may cause malware detection technologies to ignore malicious software. Machine Learning (ML) algorithms have been used by numerous researchers to overcome these issues in recent years [4].

ML is the process of discovering unusual events or observations that, due to their statistical differences from the majority of observations, may give rise to concerns. Depending on the analytical method, ML detection can be divided into three categories: dynamic, hybrid, and static. Analyzing static data such as the executable file and source code obtained without executing the application is the main goal of static analysis. While the Android application is operating, the dynamic analysis approach gathers behavior data and turns it into features. By analyzing the application and extracting both static and dynamic data, the hybrid analysis improves recognition accuracy by combining static and dynamic analysis. Furthermore, a few of the most advanced malicious programs are able detect when they are working in virtual environments and avoid detection. By depending more on human knowledge and individual decision-making, traditional machine learning algorithms demonstrate their inefficiency [5].

Theoretical and highly non-linear patterns are better learned by Deep Learning (DL) models, which aid in automatically learning features and capturing the significant features of complicated data, improving generalization on new data. DL techniques are usually more appropriate than traditional ML techniques for collecting semantic data within Android applications [6]. The most crucial features are chosen during classification to maximize detection accuracy and reduce misclassification error. Feature Selection (FS) is a procedure which assessing suitable feature sets to eliminate unnecessary features from the trained model.

Typically, there are three types of FS methods: 1) filter, 2) wrapper, and 3) embedding approaches [7]. With filter approaches, the selection is carried out step-by-step without the use of an induction algorithm using the general features of the training data. A predictor is optimized as part of the selection process in a wrapper-based approach. Consequently, a wrapper technique outperforms a filter method in terms of classification performance. Overlay techniques, which are distinguished by a deeper connection between feature selection and classifier, include embedded methods as a subset. One of the main problems with FS for the conventional methods is determining the ideal subset. An exhaustive generation of all possible subsets could be used to find the optimal subset. In recent years, metaheuristics have been demonstrated to be effective and successful [8]. Nature-Inspired (NI), metaheuristics are based on natural phenomena and make up most algorithms.

An effective way to increase the generalization ability of classification systems is through ensemble learning [9]. Ensemble learning combines the results of several classifiers to enhance detection is more accurate than single classifier [10, 11]. DL techniques are used to create a prediction model that integrates several models has been researched extensively [12–14]. Stacking, Adaboost, Random Forest (RF), and bagging are the most widely used ensemble techniques.

In this paper, Nature-Inspired Based Ensemble Feature Selection (NIEFS) and the Stacked Deep Learning Ensemble Classifier Fusion (SDLECF) is introduced for AMD. NIEFS model is introduced for discarding irrelevant or redundant features from dataset. SDLECF is introduced by combining the several models (Bi-GRU, SAE-DNN, BDLSTM, and MWDBN) which aim to improve detection performance. Finally, classifiers results was measured using MATLABR2020a and the metrics Precision (Pre), Recall (Rec), F-measure (FM), and Weighted F-measure (WFM). Proposed method considerably improves results when compare to other deep models.

2 Related Works

Lu et al., [15] suggested a hybrid DL model that integrates the Gate Recurrent Unit (GRU) and Deep Belief Network (DBN) for AMD. Static features, such as resource and semantic features are extracted by decompiling the Android Package Kit (APK) file. One-hot encoding is used to create a binary feature vector from the static features. Dynamic features are extracted by keeping attention on the relevant API calls while the APK. Static and dynamic features are familiar to DBN and GRU. Back Propagation Neural Network (BPNN) with the training data has been introduced from DBN and GRU. The most popular optimization technique for back propagation is gradient descent for parameter fine-tuning. According to experimental data, the AMD model based on hybrid DL algorithms has an increased detection effect on obfuscated malware and higher detection accuracy. VirusShare provided the non-obfuscated malware dataset, while PRAGuard provided the obfuscated malware dataset. Metrics like accuracy, precision, and recall are used to evaluate the results.

Ibrahim et al., [16] created a Gated Recurrent Unit (GRU) for AMD. DL model that uses the most helpful experimental aspects of Android applications as inputs is proposed via static analysis. GRU component using the fuzzy hash is compared with RNN the model can identify similarities in Android apps. The University of New Brunswick provides CICMalDroid 2020, which includes 17,341 present and sophisticated Android examples. Results are measured using four metrics like accuracy, precision, recall, and F1-score.

Al Ogaili et al., [17] created a hybrid mobile malware detection framework that makes use of Ant Colony Optimization (ACO) and Deep Neural Network (DNN). ACO model is introduced which dynamically lowers the dimensionality of features and prevent dimensionality. DNN is introduced to train the selected features and a self-optimizing feedback loop to iteratively enhance the selection process. CICMalDroid2020 dataset, the proposed approach obtains highest results in terms of metrics like precision, recall, F1-score, accuracy, and Matthews Correlation Coefficient (MCC).

Dong et al., [18] proposed Deep Convolutional Neural Network (D-CNN) for AMD. Firstly, Min-Max approach is introduced to normalize, the authorization features. Additionally, it is employed to guarantee the data's representativeness and uniformity. Secondly, dimensionality reduction and over-fitting prevention are achieved by the ACO approach. Thirdly, CNN and DNN are used to train high-level abstract representation to construct multiscale feature representation, which will improve the detection accuracy of the model and improve its protections versus new malicious attacks. The Drebin and Google Play Store databases are used for the experiments. More thorough and accurate malware detection becomes feasible by the hybrid structure of D-CNN, which demonstrates a more detailed considerate of the data structure and it achieves a greater accuracy.

Alhussen [19] developed a Long Short-Term Memory (LSTM) to detect malware on the Android operating system. First, the Drebin-215 dataset records a variety of features of Android apps, including permission requests, API calls, and other traits associated with the activity of the program to Android malware detection. Second, the class imbalance problem is addressed by the Synthetic Minority Over-sampling Technique (SMOTE), which generates synthetic samples by interpolating among instances of the minority class which currently exists. This increases the minority class representation without creating duplicate samples. Lastly, the goal of hyperparameter adjusting and strong data preprocessing methods is to improve the efficiency of LSTM and NN in detecting and preventing different types of malware using precision, recall, F1-score, and accuracy.

Xu et al., [20] proposed a Multilayer Perceptron (MLP) and LSTM for AMD. LSTM is applied to the semantic structure of Android bytecode to avoid missing the specifics of method-level bytecode semantics. Based on the discovery that the majority of malware is to be effectively identified using information from xml files, MLP is applied to the xml files to improve efficiency. There are 47,525 benign and 62,915 malicious implementations of DeepRefiner. According to the experimental findings, DeepRefiner performs noticeably better than anti-virus and StormDroid scanners. DeepRefiner is effective for detecting malicious apps that have been obfuscated including True Positive Rate (TPR), False Positive Rate (FPR), and Accuracy.

Wang et al., [21] presented a hybrid model based on Convolutional Neural Networks (CNN) and Deep Autoencoders (DAE) for AMD. Firstly, several CNNs and high-dimensional properties of Android applications (apps) are introduced to increase detection accuracy. Secondly, Relu is utilized to improve sparseness and "dropout" in the serial CNN architecture (CNN-S) to avoid over-fitting. To improve feature extraction capabilities, the full-connection layer is paired with the convolutional and pooling layers. DAE can quickly acquire adaptable patterns and it is utilized as a CNN pre-training technique. Experimentation, totally dataset consists of 13,000 malicious apps and 10,000 benign apps. True Positive Rate (TPR)/recall, False Positive Rate (FPR), Accuracy (ACC), Positive Predictive Value (PPV), and F-score are used to evaluate performance of the structures.

Mahdavifar et al., [22] presented a semi-supervised learning method based on the Pseudo-Label Stacked Auto-Encoder (PLSAE). This has been applied for both labeled and unlabeled cases. Feature vectors are created using a hybrid approach that combines static and dynamic analysis. CICMalDroid2020 comprises of 17,341 current instances with 5 different Android app types are used to perform the experimentation. Next, accuracy and efficiency of the results are compared with modern techniques. According to experimental data, the proposed model performs better than ML algorithms and other semi-supervised methods.

Mahindru et al., [23] proposed feature selection approach that aids in choosing appropriate features. Firstly, categorize their ability to identify malware, t-test and univariate logistic regression to the gathered feature dataset. Secondly, correlation analysis and multivariate linear regression stepwise forward selection are used to assess the accuracy of the chosen features. Thirdly, a NN using ML techniques and three ensemble approaches are employed to create malware detection models utilizing the generated features as input. Finally results show that the malware detection model by implementing the proposed approach with selected features produced higher results on F-measure and accuracy.

3 Methodology

In this paper, Nature-Inspired Based Ensemble Feature Selection (NIEFS) and the Stacked Deep Learning Ensemble Classifier Fusion (SDLECF) is introduced for feature selection and classification of AMD. NIEFS model, irrelevant and redundant features are removed to enhance the detection accuracy. SDLECF is introduced which aims to increase detection performance by integrating several models (Bi-GRU, SAE-DNN, BDLSTM, and MWDBN). SDLECF, base classifiers and second level classifiers are consequently developed on a set of examples using N-fold cross validation. Finally, the effectiveness of AMD techniques is assessed using Pre, Rec, FM, and WFM. Figure 1 depicts the overall flow of proposed system.

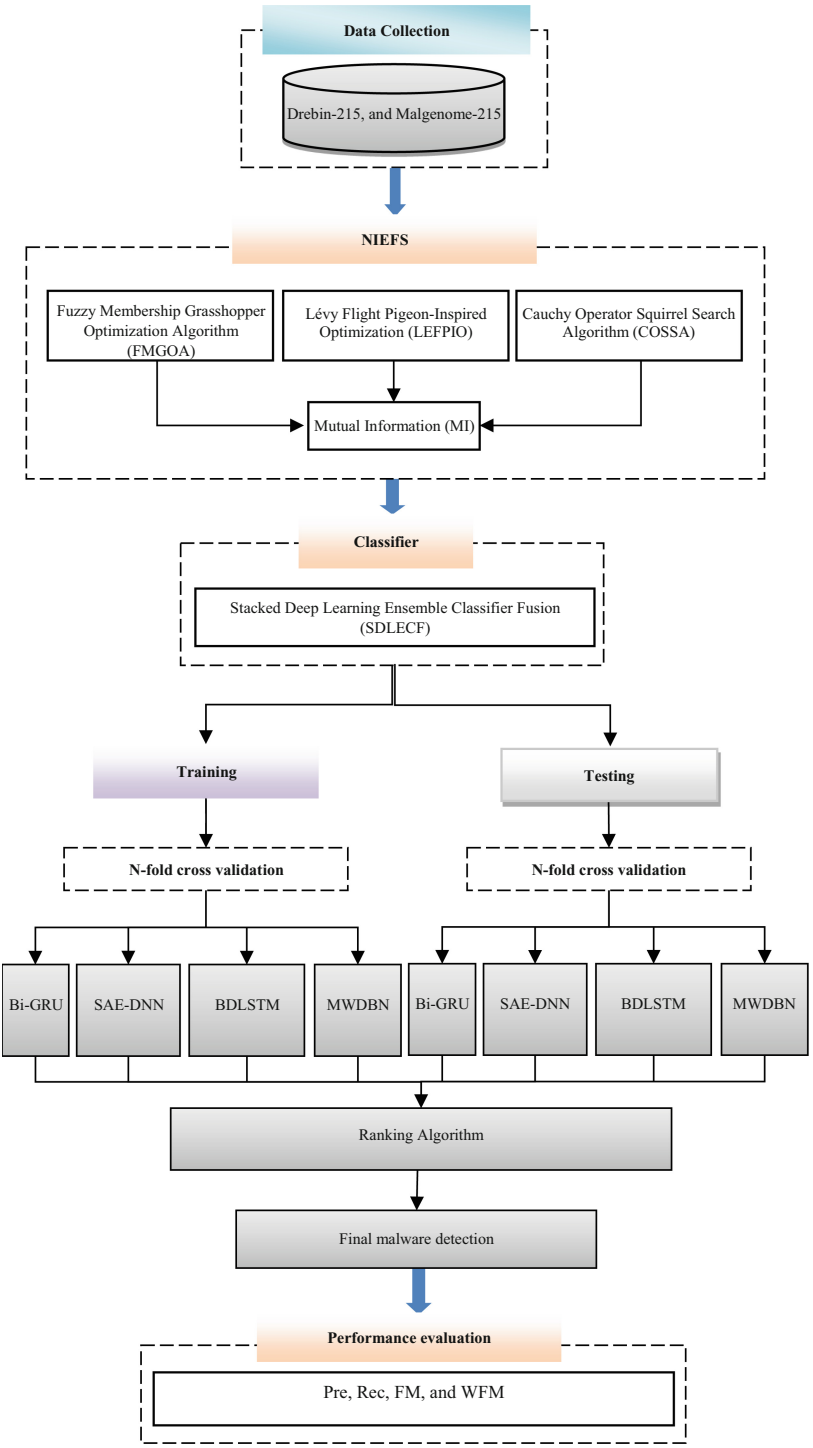


Fig. 1. PROPOSED FRAMEWORK

3.1 Dataset Collection

The AMD approaches were evaluated using two datasets drawn from three sets of example Android apps. The features of the dataset are explained in detail in Table 1 [24].

Table 1. Datasets for Experimentation

Datasets	Samples	Malware class	Benign class	Features
Drebin-215	15036	5560	9476	215
Malgenome-215	3799	1260	2539	215

3.2 Nature-Inspired Based Ensemble Feature Selection (NIEFS)

Feature–Class Mutual Information (FCMI) and Feature–Feature Mutual Information (FFMI), which mix several feature subsets, are the ensemble approach used by NIEFS to increase classifier accuracy. Whereas non-redundant features are chosen using FFMI, relevant features are chosen using FCMI. Without taking into account redundant or unnecessary features in terms of prediction dataset, the coming together process just chooses unique features from various feature subsets. Thus, some relevant and non-redundant features are chosen using the NIEFS approach. Eq. (1) shows the fitness function to assess solutions.

$$\text{fitness} = \alpha\gamma_R(D) + \beta\frac{|R|}{|C|} \tag{1}$$

where $\gamma_R(D)$ is denoted as the classification error rate, |R| is denoted as the cardinality of the chosen subset and |C| denoted as the features in the dataset, the dual parameters $\in$ 0,1 and $=1-$ is represented as the impact of subset length and detection accuracy [25].

FMGOA: FMGOA method is used to choose the best features from the AMD dataset, which mimics grasshoppers natural foraging and swarming behaviors via Eq. (2) [26].

$$GP_i = r_1 SI_i + r_2 GF_i + r_3 A_i \tag{2}$$

Three random vectors are $r_1 \in [0, 1]$, $r_2 \in [0, 1]$, and $r_3 \in [0, 1]$. For the purpose of feature selection, GP_i standsfor the i^{th} grasshopper position, SI_i for gravitational force, represented by GF_i, and wind advection, represented by A_i grasshopper social interaction. Membership functions that are triangular (TMF) is used to generate random values to GP_i. As a result, Eq. (3) gives an adequate testable relationship between intensification and diversity [26] as Eq. (3).

$$c = c_{max} - t\frac{c_{max} - c_{min}}{t_{max}} \tag{3}$$

where c_{max} and c_{min} is represented as the maximum and minimum values of c, respectively, and tis stands for the current iteration and t_{max} for the maximum number of iterations required to finish the FS process. The current position of a grasshopper is changed by taking into account its prior positions within the swarm, the global best location, and its current location [26, 27].

3.3 Lévy Flight Pigeon-Inspired Optimization (LEFPIO)

LEFPIO algorithm derives from the behavior of pigeons that needs both local optimum knowledge and global optimization. Using the behavior of pigeons in identifying the best features from the AMD dataset, PIO is compose d of two operators: the operators for the map, compass, and landmarks. For maximum malware detection in Android analysis, these operators can support both the local and the global best position.

(1). **Map and compass operator:** The feature position X_i and the velocity V_i of pigeon i in a D-dimension search space are defined by this operator [28].
(2). **Landmark operator:** Every generation, this operator reduces the pigeon population by half. The remaining pigeons fly straight to the greatest detection rate in the classifier to obtain the finest functionalities with AMD. Lévy flight is more successful than Brown motion while looking for an unknown and large-scale space since its variance grows faster. Lévy flying may be used to replicate pigeon search habits [29].

Step 1: Set up the settings for the LEFPIO method, such as the feature space dimension D, population size No_p, maximum iteration number Ic_{max}, and starting set of pigeons at random.

Step 2: Assess the pigeons' fitness function.

Step 3: Lévy flight operator is used to create novel pigeons.

Step 4: Elite operation is introduced individual with best selected features.

Step 5: If N_p creating pigeons, proceed to Step 6. If not, move on to Step 3.

Step 6: When the current number of iterations, N_c, approaches Ic_{max}, output the features. Else, go to Step 3.

Cauchy Operator Squirrel Search Algorithm (COSSA) Glide technique used by squirrels during their dynamic foraging is emulated by the squirrel search algorithm, which helps choose the best features from the AMD dataset [30]. Particularly well-suited for glide movement, flying squirrels are a kind of rodent that lives in trees and is mostly active at night. There is a belief that squirrel gliding is the most intricate and energy-efficient type of aerodynamic motion [30]. The optimal utilization of food resources is also achieved by its dynamic foraging behavior [30]. Four prerequisites must be met in order to use the COSSA,

1. There are n squirrels and n trees in a deciduous forest with individual squirrel per tree.
2. A hickory tree and N_{fs} ($1 < N_{fs} < n$) acorn trees comprise these n trees; the rest trees are usual trees.
3. The woodland is home to just three different kinds of trees. Acorn trees have a general food supply in the form of acorn nuts, regular trees have no food, and hickory trees have the greatest food source in the form of hickory nuts.
4. Using dynamic foraging behavior, each squirrel looks for the best feature with the highest detection rate on its own, and the food resources that are accessible are chosen depending on the detection rate.

Mutual Information (MI) MI is introduced with the NIEFS approach to select the optimal features from the subset. To extent uncertainty in X resulting from knowledge

of Y is known as I(X, Y) in information theory [31]. It is described as follows as Eq. (4),

$$I(X, Y) = \sum_{x,y} p(x, y) \log_2 \frac{p(x, y)}{p(x), p(y)} \tag{4}$$

where p(x, y) is a joint probability distribution function for X and Y, and the individual ratio distribution for X and Y are p(x) and p(y).

Feature-class relevance: It is described as the extent of FCMI within a feature(fe) and a class label C.

Feature-feature relevance: It is described as the extent of FFMI between any two features ($fe_i$and fe_j).

A heuristic based on empirical research is used to determine the value of K. First, a classifier is used to determine the classification accuracy of the highest-ranked feature in the proposed technique. The technique calculates classification accuracy for each subset of features. If a portion of K highly regarded features yields the highest accuracy in classification when contrasted with another subset with K + 1 extremely ranked features, then the subset of K extremely valued features as final feature subset.

3.4 Stacked Deep Learning Ensemble Classifier Fusion (SDLECF)

SDLECF is developed based on classifiers model that combines the predictions from multiple DL methods like Bidirectional Gated Recurrent Unit (Bi-GRU), SAE-DNN, BDLSTM, and Mean Weight Deep Belief Network (MWDBN) by feeding their outputs as input to a stacking meta-model which then makes the final detection. SDLECF classifiers of DL methods are trained at lowest level using N-fold cross-validation. Highest level, ranking techniques are used to choose the best pair to construct the final model.

3.4.1 Model Establishment

To construct the training set, multilevel ensemble classifier, and during the training phase, it is assessed on a test set. Later on in the testing process, it is assessed. Each DL classifier with their malware detection accuracy is tested at the lower level using an N-fold cross validation [32]. The features $f_1, f_2, \ldots, f_k$ have been used to get probabilities from the fundamental DL methods. After appending the first class designation as Eq. (5) [32],

$$\dot{V}(x) = \{f_1, f_2, \ldots, f_k, l\}, \forall k \in \{1 \text{ to } 4\}, , l \in \{0, 1\} \tag{5}$$

Level will use D_{base} and $\dot{V}(x), \forall x \in X$. During the model construction, Base DL methods can now achieve final malware detection by introducing majority voting. During the reclassification process, each scheme in S is assigned a Z-weight determined by $\dot{V}(x)$, $x \in X$ for respectively [32].

3.4.2 Base DL Classifiers

AMD has used the most basic DL classifiers for classification.

Bidirectional Gated Recurrent Unit (Bi-GRU) Classifier: In the Bi-GRU model, the update gate, d_{TS}, takes the position of the input and forget gates of the LSTM network. The model is assisted by the update gate in identifying the historical malware detection data that must be supplied with the upcoming malware detection data. Fig. 2 mentions the architecture of Bi-GRU model.

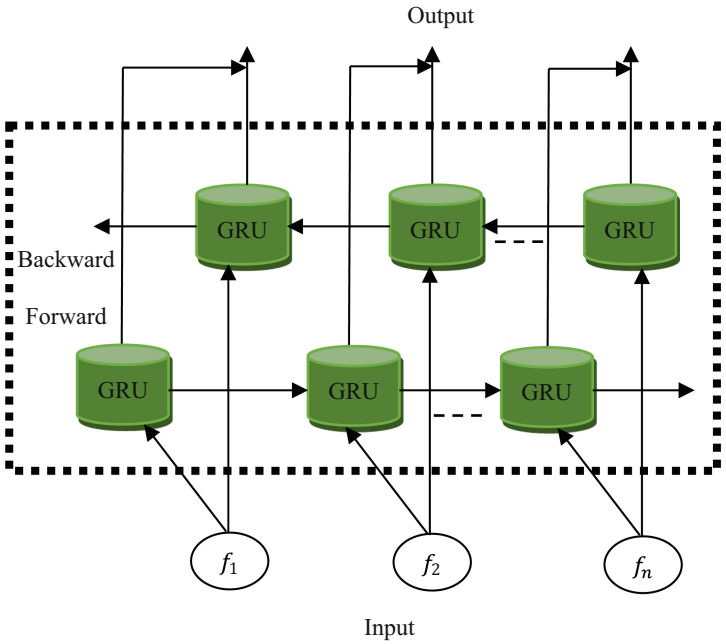


Fig. 2. Architecture of Bi-Gru Model

The vanishing-gradient issue is reduced by this procedure [33]. Eq. (6) provides a mathematical specification for the update gate d_{TS},

$$d_{TS} = \sigma\left(We_d \times \left[h_{TS-1}, f_{TS}\right] + b_d\right) \tag{6}$$

where the input matrix (chosen features) at time step (TS) is marked as f_{TS}, the sigmoid activation function is stated as σ, the weight matrix is represented as We_d, the bias matrix is indicated as b_d, and the hidden state at the earlier time step (TS – 1) is indicated as h_{TS-1}. The reset gate (p_{TS}) in the Bi-GRU model is used to regulate the historical time-series malware detection dataset, which is in charge of the network short-term memory when it is concealed. Eq. (7) expresses the p_{TS},

$$p_{\mathrm{TS}} = \sigma\left(We_p \times h_{TS-1}, f_{TS}\right] + b_p\Big) \tag{7}$$

where b_p and We_p is denoted as the matrix of bias and the weight in p_{TS}. Eq. (8) then specifies the hidden state candidate ($\tilde{h}_{TS}$),

$$\tilde{h}_{TS} = \tanh\left(We_h \times \left[h_{TS-1} \odot p_{TS}, f_{TS}\right] + b_h\right) \tag{8}$$

where $\odot$ stands for the dot multiplication operation, tanh is denoted as the tangent activation function, and b_h and We_h is denoted as the bias and weight matrices of the memory cell state. Eq. (9), which shows the linear interpolation of $\tilde{h}_{TS}$ and h_{TS-1} yields the output (h_{TS-1}),

$$\mathrm{h}_{TS} = (1 - \mathrm{d}_{TS}) \odot \mathrm{h}_{TS} - 1 + \mathrm{d}_{TS} \odot \tilde{\mathrm{h}}_{TS} \tag{9}$$

The forward GRU in the Bi-GRU model retrieves data about previous malware from the historical malware data, while the backward GRU retrieves data about future malware from the same historical malware data [34]. Eq. (10), which specifies the Bi-GRU (O_{TS}) model,

$$O_{TS} = \mathrm{A}\left(\overrightarrow{h}_{TS}, \overleftarrow{h}_{TS}\right) \tag{10}$$

where A is denoted as the output of the forward and backward directions. Furthermore, $\overleftarrow{\mathrm{h}}_{\mathrm{TS}}$& $\overrightarrow{\mathrm{h}}_{\mathrm{TS}}$ is used to represent the hidden states of both forward and backward GRU.

Sparse Autoencoder Based Deep Neural Network (SAE-DNN) SAE-DNN model learns a shared encoding representation for malware detection with supervised classification and unsupervised reconstruction. The reconstruction error serves as the basis for malware detection.

SAE: An "encoder" and a "decoder" network make up an AE and AE is a symmetrical neural network. The "encoder" network decreases the dimensional of input malware dataset to a restricted set of features, and "decoder" network rebuilds the dataset from the selected features. Fig. 3 displays an AE with a symmetrical structure.

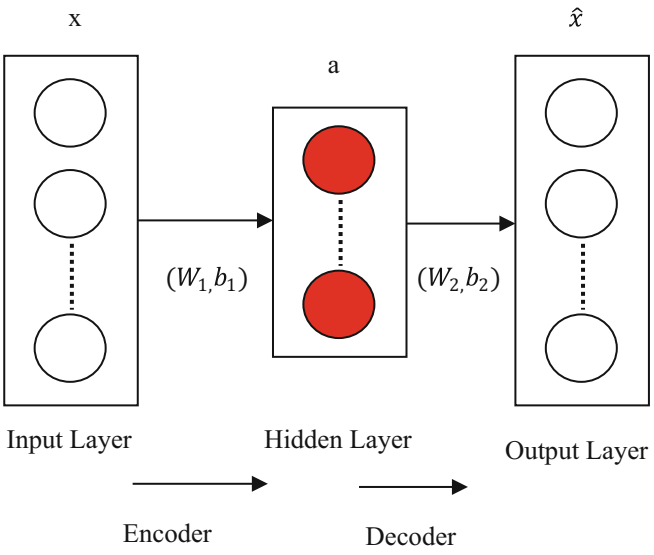


Fig. 3. Structure of Autoencoder (AE)

Eq. (11) describes the encoder, which converts the input D-dimensional malware detection dataset x into its hidden representation $a = a_1, a_2, \ldots, a_{D_1}$ from the input layer to the hidden layer,

$$a = \sigma(W_1 x + b_1) \tag{11}$$

where σ, W_1, b_1 is denoted as the encoder activation function, weight matrix, and bias vector, respectively. The input x is then reconstructed to $\hat{x}$ by the decoder from the hidden to the output, using the feature vector an in eq. (12),

$$\hat{x} = \sigma(W_2 a + b_2) \tag{12}$$

where W_2, b_2 are the bias vector and the decoder weight matrix. By including a sparse limitation for the network training, an SAE, a variation of the autoencoder, promotes the extraction of discriminative features [35]. SAE is used to reduce the cost function with input samples $x_{\,\mathrm{i}}$,$i = 1, \ldots, N$ by eq. (13),

$$F_{cost} = \frac{1}{N}\sum_{i=1}^{N} \|x_i - \hat{x}_i\|^2 + \frac{\lambda}{2}\|W\|^2 + \beta R_{sparse} \tag{13}$$

where the reconstruction error is represented by the $\|x_i - \hat{x}_i\|^2$, the L2 regularization term is represented by the $\|W\|^2$, SAE weight matrix is represented by W, the sparsity regularization by R_{sparse}, and the corresponding terms importance is controlled by the coefficients λ and β. It has been discovered that when hidden neurons are kept mostly hidden, they react to various patterns present in the data; this means that the characteristics that are recovered are discriminative. Taking into consideration all of the input malware samples x_i, $i = 1, \cdots, N$ in Eq. (14), let $\hat{p}_j$ be the average activation of the j^{th} hidden of the SAE hidden layer,

$$\hat{p}_j = \frac{1}{N}\sum\nolimits_{i=1}^{N} a_{i,j} \tag{14}$$

where the j^{th} element of the i^{th} hidden representation a_i, $j = 1, \cdots, D_1$ is denoted by $a_{i,j}$. To determine whether p_j is near a desired tiny sparsity percentage p in Eq. (15), the Kullback-Leibler (KL) divergence function to compute the sparsity regularization R_{sparse} in Eq. (13),

$$R_{sparse} = \sum\nolimits_{j=1}^{D_1} KL\left(p \middle\| \hat{p}_j\right) = \sum\nolimits_{j=1}^{D_1}\left[p log\frac{p}{\hat{p}_j} + (1-p)\log\frac{1-p}{1-\hat{p}_j}\right] \tag{15}$$

When all $\hat{p}_j$ are equal p, the KL function is zero; when they diverge, it rises.

Deep Neural Network (DNN): As seen in Fig. 4, DNN is composed of 3 steps: an encoder, a decoder, and a classification. The encoder, several layers of non-linear transformations is used to extract high-level representations from the dataset. Both the classifier and the decoder use the extracted representations as input. While the classification forecasts the input dataset label, the decoder searches to rebuild the input dataset.

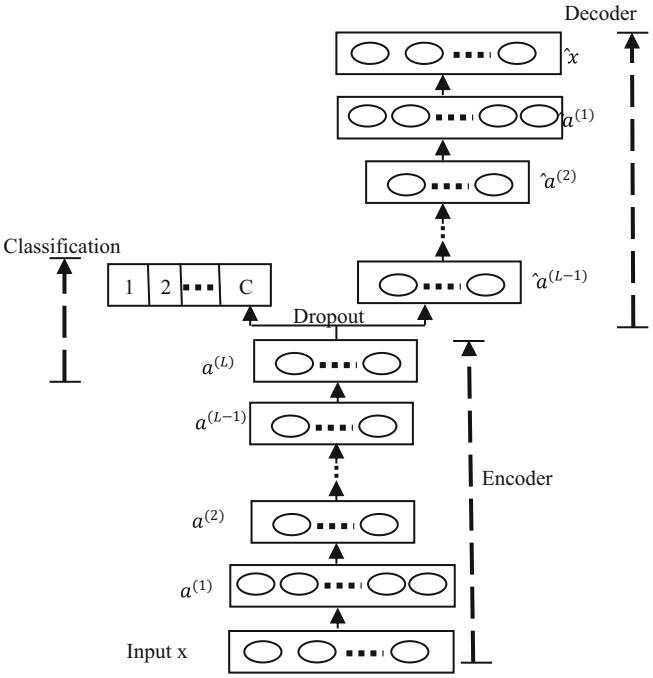


Fig. 4. Structure of Deep Neural Network (DNN)

More specifically, the DNN construction consists of following functions,

Encoder: The malware sample x is fed into the encoder, which uses L hidden layers, $L \in \mathbb{N}_+$, to extract the high-level representation of x. Let us consider that the hidden representations as $a^{(1)}$, ..., $a^{(L)}$ and hidden dimensions D_1, …, D_L are described as follows, (i) Sparse-over complete demonstration at the initial hidden layer, which has been demonstrated to be capable of extracting free basis functions for input datasets [36], (ii) Compressed representation with dimension D_l, $l = 1$ to L, and it must be smaller than D_{l-1}. Input layer and the total hidden layer make use of ReLU activation function. The L2 regularization is used to limit the encoder weights, and the λ is described in Eq. (13) to each hidden layer to perform feature extraction.

Decoder: The number of hidden layers and their dimensions are correctly balanced among the encoder and decoder. The decoder, $a^{(L)}$ is used to recover the input x. All hidden layers, ReLU as activation function, and the sigmoid activation function in output layer for reconstruction. The decoder weights are constrained by the L2 regularization.

Classification: To explain the C-class, the softmax layer is used with C neurons which characterize different situations. The output of the softmax layer, given the representation $a^{(L)}$, is a vector $[\hat{y}_1, \hat{y}_{2,\ldots,}, \hat{y}_C]$. Given each sample x in Eq. (16), the k^{th} unit of the output, $\hat{y}_k, k \in \{1\ to\ C\}$ is commonly thought of as a value related to the conditional probability

in the k^{th} condition,

$$\hat{y}_k = \frac{\exp(z_k)}{\sum_{j=1}^{C} \exp(z_j)} \tag{16}$$

where z_k is the previous to softmax activation function as k^{th} condition in Eq. (17) with $0 \leq \hat{y}_k \leq 1$, $\sum_{k=1}^{C} \hat{y}_k = 1$,

$$z_k = w_k . a^{(L)} + b_k \tag{17}$$

where the weights and biases of the k^{th} softmax layer neuron are denoted by w_k and b_k. On the hidden layer $a^{(L)}$, dropout regularization is used to avoid overfitting. During forward and backpropagation, dropout randomly sets a percentage p_{drop} of the hidden neurons to zero. Consequently, z_k is calculated using Eq. (18) with dropout,

$$z_k = w_k . \left(a^{(L)} \circ r \right) + b_k \tag{18}$$

where $r \in \mathbb{R}^{D_L}$ is a masking vector of Bernoulli probability, p_{drop} of being 0 and $\circ$ is the multiplication operator. Only the unmasked neurons are used for gradient backpropagation. Each label y_i in the training set (x_i, y_i), $i = 1, \ldots, N$ is converted into a one-hot C-dimensional vector $(y_{i,1}, y_{i,2}, \ldots y_{i,C})$, $i = 1, \ldots, N$ using Eq. (19), which is linked to the neurons of the softmax layer,

$$y_{i,k} = \begin{cases} 1, & y_i = k \\ 0, & else \end{cases} \tag{19}$$

Reducing the following cost function in Eq. (20) is the DNN training goal over the training set,

$$F_{cost} = \frac{\eta_1}{N} \sum_{i=1}^{N} \|x_i - \hat{x}_i\|^2 + \frac{\lambda}{2} \|W\|^2 + \beta \sum_{j=1}^{L} R_{sparse}^{(j)} - \frac{\eta_2}{N} \sum_{i=1}^{N} \sum_{k=1}^{C} y_{i,k} \log \hat{y}_{i,k} \tag{20}$$

where the reconstruction error is represented by the $\|x_i - \hat{x}_i\|^2$. The $\lambda, \beta, \eta_1, \eta_2$ are coefficients measuring the significance of the related terms, W is the weight matrix, $R_{sparse}^{(j)}$ is the sparsity regularization on the j^{th} hidden layer, and $\|W\|^2$ is the L2 regularization. A test sample x_{test} is recognized as unidentified in terms of novelty detection if the size of its reconstruction error exceeds a predetermined threshold δ,

$$e^{test} = \left\| x^{test} - \hat{x}^{test} \right\|^2 > \delta \tag{21}$$

AMD samples from previously identified C classes are linked to both malware and normal situations to build and train a DNN. The DNN initially determines if a test data sample falls into one of the previously recognized C classes. If so, an unidentified condition is provided; if not, the DNN classifies the test sample into one of C previously identified classifications. Fig. 5 displays the flow chart for the proposed approach utilizing the SAE-DNN.

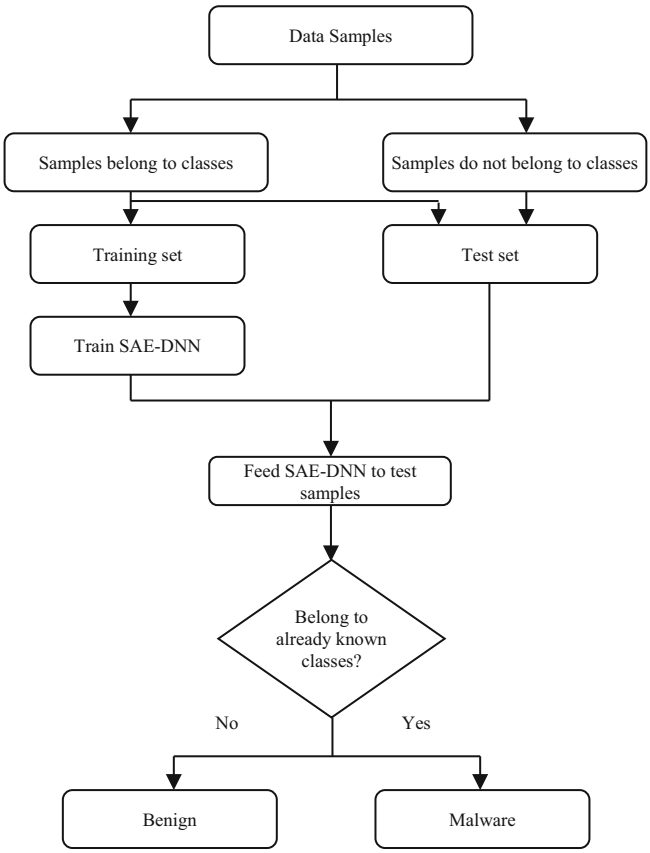


Fig. 5. Flowchart of SAE-DNN

Bidirectional Long Short-Term Memory (BDLSTM) Algorithm A variant of the Recurrent Neural Network (RNN) is called LSTM. The LSTM cell input is x_t at each iteration t, and its output is indicated by h_t. C_{t-1} denoted as the cell output state of the previous time step, whereas $\hat{C}_t$ and C_t is denoted as the current cell input and output states. Long-term dependences of the sequence AMD dataset can be modeled by LSTM to the structure of celled gates. Gates are used to regulate the LSTM cell states by permitting optional data flow. The i_t, f_t, o_t is denoted as the input gate, forget gate, and output gate, respectively [37]. Eqs. (22–25) are used to calculate the values of the cell input state and gates,

$$i_t = \sigma_g(W_i x_t + U_i h_{t-1} + b_i) \tag{22}$$

$$f_t = \sigma_g\left(W_f x_t + U_f h_{t-1} + b_j\right) \tag{23}$$

$$o_t = \sigma_g(W_o x_t + U_o h_{t-1} + b_o) \tag{24}$$

$$\hat{C}_t = \sigma_g(W_c x_1 + U_c h_{t-1} + b_c) \tag{25}$$

Weight matrices of input gate, the forget gate, the output gate, and the input cell state are represented by the symbols W_i, W_f, W_o, W_c. The weight matrices input, forget, output, and input cell state are represented by the symbols U_i, U_f, U_o, U_c. The associated bias vectors are indicated by the letters b_i, b_f, b_o, b_c.

The Bidirectional Recurrent Neural Network (BDRNN) concept is the basis for BDLSTM. Two RNNs, each of which is coupled to the same output layer, receive each training sequence both forwards and backwards in BDRNN. BDLSTM uses a forward LSTM and a backward LSTM layer to analyze sequence dataset. These two hidden layers are coupled to the output layer. Fig. 6 displays the BDLSTM standard topology. Eqs. (22–25) declare that the LSTM layer output h_t and cell output state C_t are determined using eqs. (26–27) at each iteration t,

$$C_t = f_t * C_{t-1} + \widetilde{C}_t * i_t \tag{26}$$

$$h_t = o_t * \tanh(C_t) \tag{27}$$

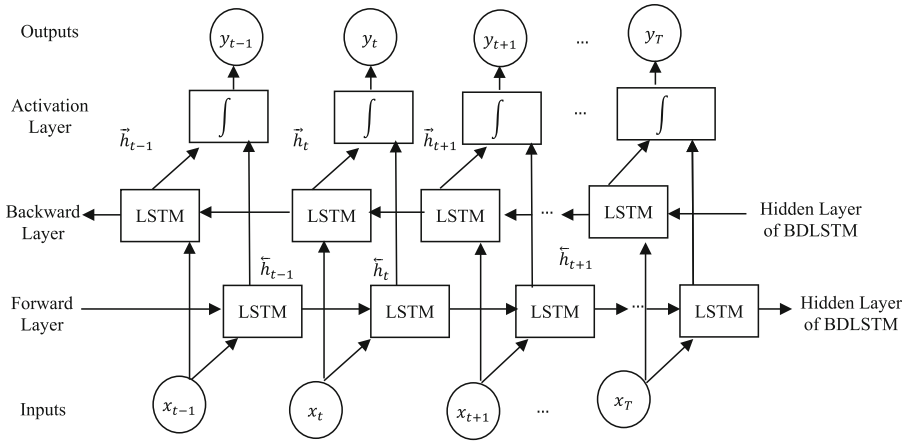


Fig. 6. Bidirectional Long Short-Term Memory (BDLSTM)

MWDBN MWDBN, a probabilistic productive network, is composed of several layers of Restricted Boltzmann Machines (RBM). Eq. (28), functions are associated with the following probability conditional functions, and they can produce cells i and j based on the joint probability distribution function,

$$\text{maximize}\{\theta\}\frac{1}{m}\sum_{l=1}^{m}\log\left(P\left(v^l\right)\right) \tag{28}$$

The training dataset dimensions are referred to by the word, where v stands for the visible layer. $\theta = \{b_i, a_j, mw_{ij}\}$, is the notation used, where mw_{ij} is the average weight

that connects the apparent cell with the hidden cell j. The symbols a_i and b_j represent the unit biases. The desired function, which includes a log-likelihood term, must has been increased via a gradient descent.

3.4.3 Ranking Algorithms

Ranked Aggregate of per class Performance based scheme (RAPC) calculates the cumulative early per-class rankings of primary classifier accuracy. This strategy prioritizes a clear classifier that performs well in both categories [38]. Ranked Aggregate of average accuracy and Class Differential based scheme (RACD) is a method for determining an overall ranking by combining the initial rankings of standard performance reliability with the performance variances between groups [38]. The accuracy levels for each malware and benign classifications are used while analyzing the data.

4 Results and Analysis

In this section, performance comparison of methods are measured among datasets like Malgenome-215 (https://figshare.com/articles/dataset/Android_malware_dataset_for_machine_learning_1/5854590) and Drebin 215 (https://figshare.com/articles/dataset/Android_malware_dataset_for_machine_learning_2/5854653 are used to test the effectiveness of classifiers in this section. MATLABR2020a is used to implement and assess AMD methods. All methods were tested using a Windows 7 Enterprise 64-bit machine with 32GB of RAM and a 3.10GHz Intel Xeon CPU.

4.1 Evaluation Metrics

In this section the following evaluation metrics has been used to assess the performance of AMD detection methods,

Precision: Precision indicates the percentage of accurate positive predictions obtained from the model by eq. (29),

$$\text{Precision (Pre)} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{29}$$

Recall: Recall is denoted as the percentage of properly detected malware applications compared with all malicious apps. It has been described by eq. (30),

$$\text{Recall (Rec)} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{30}$$

where True Positive (TP) is denoted as the amount for malware which had been correctly recognized as malware by model, True Negative (TN) is denoted as the amount of accurately predicted benign as benign, False Positive (FP) is denoted as the amount of benign that were wrongly expected as malware, False Negative (FN) is denoted as the fraction of malware in the set that were incorrectly categorized as benign.

F-Measure (FM): FM is denoted as the harmonic mean of Pre and Rec, it is represented by eq. (31),

$$\mathrm{FM} = \frac{2.\mathrm{Pre}.\mathrm{Rec}}{\mathrm{Pre} + \mathrm{Rec}} \tag{31}$$

WFM is defined as the weighted total of FM and the number of examples in each class. F_m and F_b are are the FM for the M and B classes, whereas N_m and N_b are are the number of occurrences in each M and B class, Eq. (32) has been used to WFM,

$$WFM = \frac{F_m.N_m + F_b.N_b}{N_m + N_b} \tag{32}$$

These measures were employed to assess the performance of AMD methods against two benchmark datasets (Malgenome 215, and Drebin 215).

4.2 Results Comparison

Evaluation results of Malgenome 215 among detecting methods are discussed in Table 2.

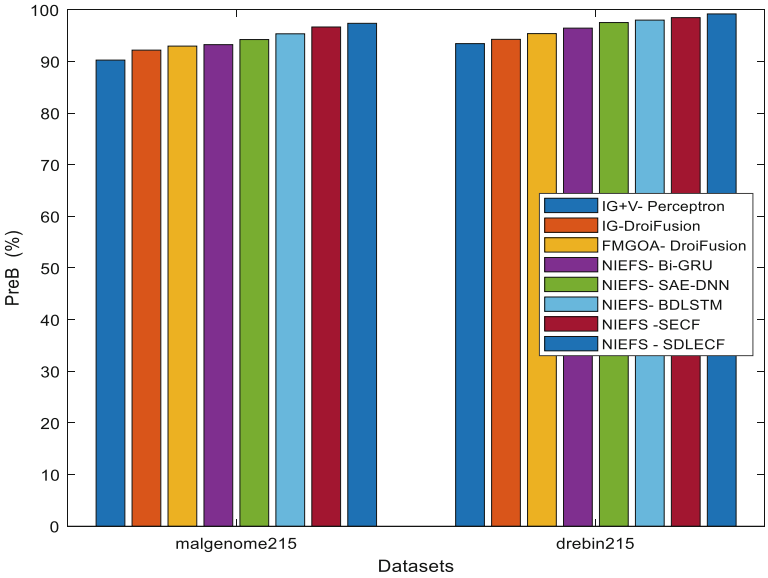
Table 2. Evaluation Metrics Comparison of Detection Classifiers (Malgenome 215 Dataset)

METRICS	IG + V-Perceptron	IG-DroiFusion	FMGOA-DroiFusion	NIEFS-Bi-GRU	NIEFS-SAE-DNN	NIEFS-BDLSTM	NIEFS -SECF	NIEFS -SDLECF
PreB(%)	89.19	91.72	92.14	92.89	93.47	94.32	95.19	96.78
RecB(%)	91.37	92.68	93.45	93.92	94.48	95.24	96.25	97.02
PreM(%)	90.26	92.19	92.97	93.43	94.20	95.35	96.67	97.38
RecM(%)	90.85	91.58	92.15	92.93	93.71	95.30	96.33	97.45
FM(%)	90.41	92.05	92.67	93.29	93.96	95.05	96.11	97.16
WFM (%)	91.57	92.88	93.45	94.32	94.96	95.70	96.70	97.77

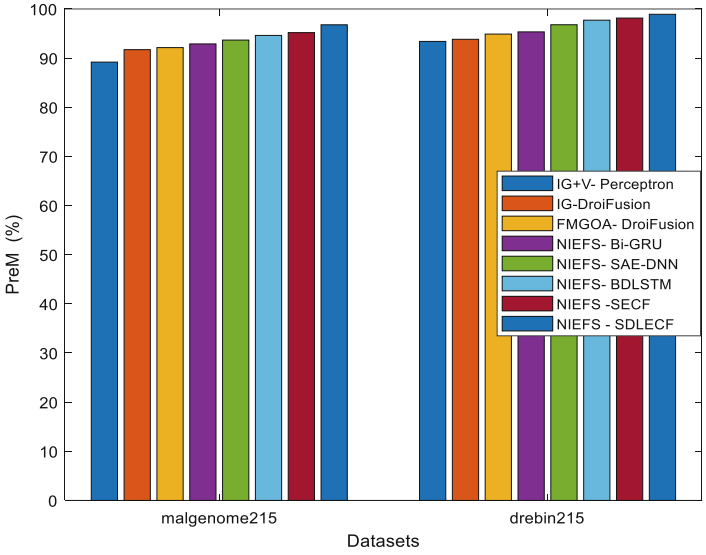
Evaluation results of Drebin 215 among detection techniques are discussed in Table 3.

Table 3. Evaluation Metrics Comparison of Detection Classifiers (Drebin 215 Dataset)

METRICS	IG + V-Perceptron	IG-DroiFusion	FMGOA-DroiFusion	NIEFS-Bi-GRU	NIEFS-SAE-DNN	NIEFS-BDLSTM	NIEFS -SECF	NIEFS -SDLECF
PreB(%)	93.40	93.83	94.89	95.61	96.40	97.25	98.15	98.92
RecB(%)	94.17	94.55	95.17	95.98	96.66	97.43	98.66	99.09
PreM(%)	93.45	94.28	95.39	95.45	96.73	97.18	98.48	99.20
RecM(%)	93.26	94.12	94.68	95.72	96.55	97.69	98.63	99.41
FM (%)	93.57	94.19	95.03	95.69	96.58	97.38	98.47	99.15
WFM (%)	93.88	94.74	95.41	96.10	96.96	97.57	98.68	99.24

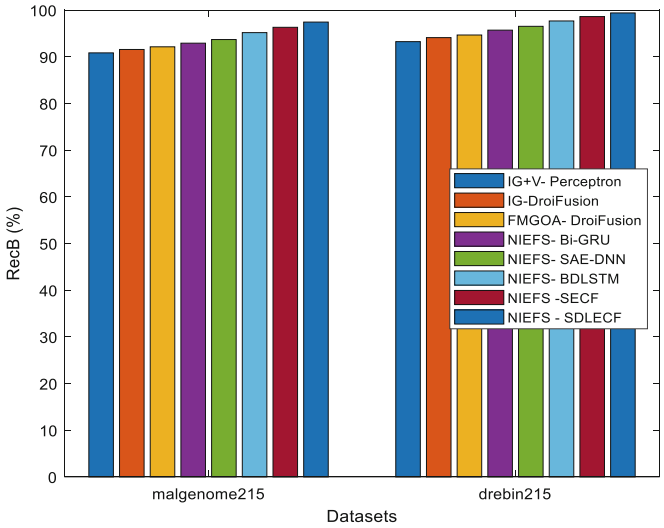


(a). PRECISION ANALYSIS OF BENIGN CLASS

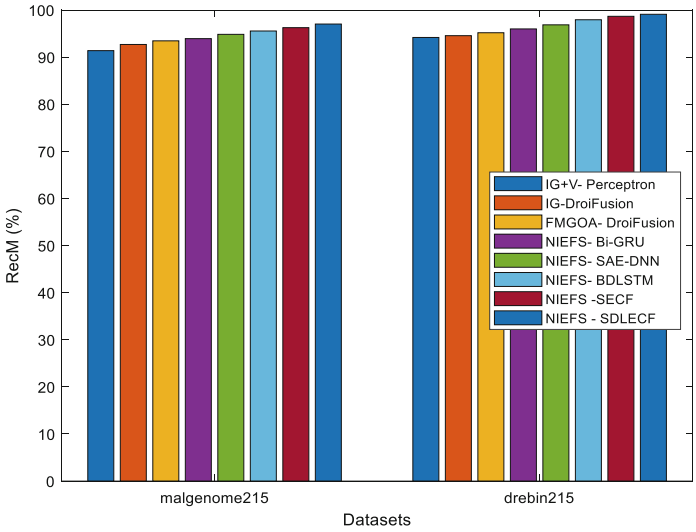


(b). PRECISION ANALYSIS OF MALWARE CLASS

Fig. 7. Precision Results Comparison of Detection Methods



(a). RECALL ANALYSIS OF BENIGN CLASS



(b). RECALL ANALYSIS OF MALWARE CLASS

Fig. 8. Recall Results Comparison of Detection Methods

Figure 7 shows a precision study of malware detection techniques using the malgenome 215 and drebin215 datasets. Fig. 7(a) displays the precision analysis of detection techniques for the benign class, while Fig. 7(b) does the same for the malware class. The outcomes are measured between the various selection methods like IG, FMGOA and NIEFS with detection methods. Malgenome 215 dataset, maximum

precision findings of 96.78% is achieved by SDLECF model while the other methods such as IG + V-Perceptron, IG-DroiFusion, FMGOA-DroiFusion, NIEFS-Bi-GRU, NIEFS-SAE-DNN, NIEFS-BDLSTM, and NIEFS-SECF gives the precision results of 89.19%, 91.72%, 92.14%, 92.89%, 93.47%, 94.32%, and 95.19% for benign class (See Table 2).Drebin215 dataset, precision score of 98.92% by SDLECF model whereas other techniques such as IG + V-Perceptron, IG-DroiFusion, FMGOA-DroiFusion, NIEFS-Bi-GRU, NIEFS-SAE-DNN, NIEFS-BDLSTM, and NIEFS-SECF gives the precision results of 93.40%, 93.83%, 94.89%, 95.61%, 96.40%, 97.25%, and 98.15% for benign class (See Table 3).Fig. 7(b) malgenome 215 dataset, SDLECF model gives increased precision results of 97.38% whereas other techniques such as IG + V-Perceptron, IG-DroiFusion, FMGOA-DroiFusion, NIEFS-Bi-GRU, NIEFS-SAE-DNN, NIEFS-BDLSTM, and NIEFS-SECF gives the precision results of 90.26%, 92.19%, 92.97%, 93.43%, 94.20%, 95.35%, and 96.67% for malware class (See Table 2). Drebin215 dataset, 99.20% is the maximum precision achieved by the SDLECF model; other techniques such as IG + V-Perceptron, IG-DroiFusion, FMGOA-DroiFusion, NIEFS-Bi-GRU, NIEFS-SAE-DNN, NIEFS-BDLSTM, and NIEFS-SECF gives the precision results of 93.45%, 94.28%, 95.39%, 95.45%, 96.73%, 97.18%, and 98.48% for malware class (See Table 3).

Figure 8 illustrates recall study of malware detection algorithms to malgenome 215 and drebin215 dataset. Figure 8(a) displays the recall analysis of detection techniques for the benign class, while Fig. 8(b) provides the same information for the malware class. Malgenome 215 dataset, SDLECF has the greatest recall outcomes of 97.02%, whereas other approaches such as IG + V-Perceptron, IG-DroiFusion, FMGOA-DroiFusion, NIEFS-Bi-GRU, NIEFS-SAE-DNN, NIEFS-BDLSTM, and NIEFS-SECF gives the recall results of 91.37%, 92.68%, 93.45%, 93.92%, 94.48%, 95.24%, and 96.25% for benign class (See Table 2). Drebin215 dataset, SDLECF has the greatest recall outcomes of 99.09%, whereas other approaches like as IG + V-Perceptron, IG-DroiFusion, FMGOA-DroiFusion, NIEFS-Bi-GRU, NIEFS-SAE-DNN, NIEFS-BDLSTM, and NIEFS-SECF gives the recall results of 94.17%, 94.55%, 95.17%, 95.98%, 96.66%, 97.43%, and 98.66% for benign class (See Table 3). Fig. 8(b) malgenome 215 dataset, SDLECF has the greatest recall outcomes of 97.45%, whereas other approaches like as IG + V-Perceptron, IG-DroiFusion, FMGOA-DroiFusion, NIEFS-Bi-GRU, NIEFS-SAE-DNN, NIEFS-BDLSTM, and NIEFS-SECF gives the recall results of 90.85%, 91.58%, 92.15%, 92.93%, 93.71%, 95.30%, and 96.33% for malware class (See Table 2).Drebin215 dataset, SDLECF model offers the greatest recall outcomes of 99.41%; other techniques, such as IG + V-Perceptron, IG-DroiFusion, FMGOA-DroiFusion, NIEFS-Bi-GRU, NIEFS-SAE-DNN, NIEFS-BDLSTM, and NIEFS-SECF gives the recall results of 93.26%, 94.12%, 94.68%, 95.72%, 97.69%, and 98.63% for malware class (See Table 3).

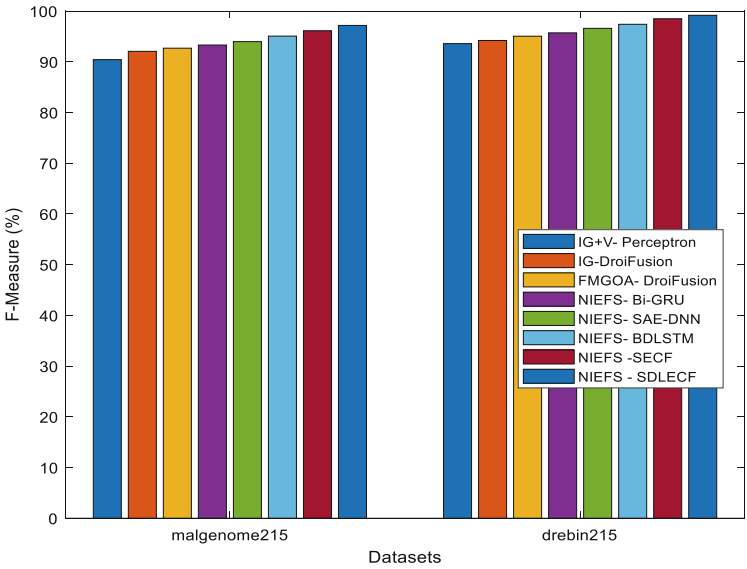


Fig. 9. FM Results Comparison of Detection Methods

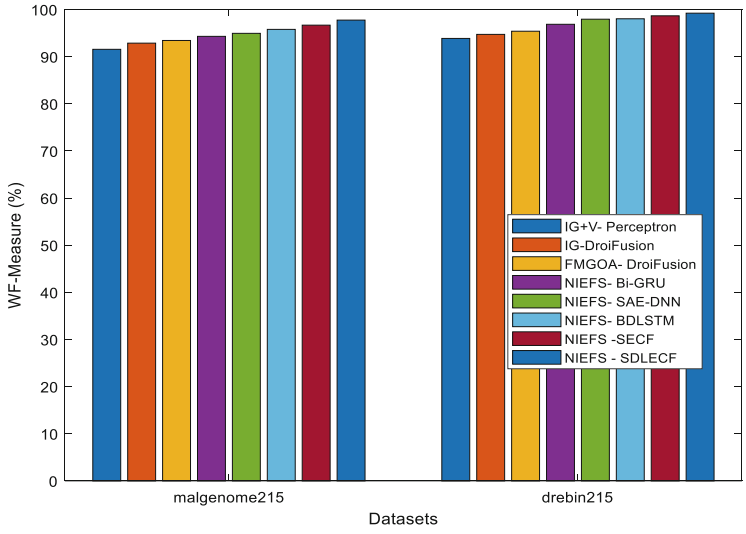


Fig. 10. WFM Results Comparison of Detection Methods

Malware detection methods to malgenome 215 and drebin215 dataset of FM are illustrated in Fig. 9. Malgenome 215 dataset, SDLECF has the greatest f-measure outcomes of 97.16%, whereas other approaches such as IG + V-Perceptron, IG-DroiFusion, FMGOA-DroiFusion, NIEFS-Bi-GRU, NIEFS-SAE-DNN, NIEFS-BDLSTM, and NIEFS-SECF

gives the f-measure results of 90.41%, 92.05%, 92.67%, 93.29%, 93.96%, 95.05%, and 96.11% (See Table 2). Drebin215 dataset, SDLECF model provides the greatest f-measure outcomes of 99.15%; other approaches such as IG + V-Perceptron, IG-DroiFusion, FMGOA-DroiFusion, NIEFS-Bi-GRU, NIEFS-SAE-DNN, NIEFS-BDLSTM, and NIEFS-SECF gives the f-measure results of 93.57%, 94.19%, 95.03%, 95.69%, 96.58%, 97.38%, and 98.47% (See Table 3). A Malware detection method to malgenome 215 and drebin215 dataset of WFM is illustrated in Fig. 10. Malgenome 215 dataset, With the greatest WFM scores of 97.77% for the SDLECF model, other techniques including IG + V-Perceptron, IG-DroiFusion, FMGOA-DroiFusion, NIEFS-Bi-GRU, NIEFS-SAE-DNN, NIEFS-BDLSTM, and NIEFS-SECF gives the WFM results of 91.57%, 92.88%, 93.45%, 94.32%, 94.96%, 95.70%, and 96.70% (See Table 2). Drebin215 dataset, WFM values for the SDLECF model are the highest at 99.24%, other techniques like IG + V-Perceptron, IG-DroiFusion, FMGOA-DroiFusion, NIEFS-Bi-GRU, NIEFS-SAE-DNN, NIEFS-BDLSTM, and NIEFS-SECF gives the WFM results of 93.88%, 94.74%, 95.41%, 96.10%, 96.96%, 97.57%, and 98.68% (See Table 3).

5 Conclusion

In this paper, proposed a novel Stacked Deep Learning Ensemble Classifier Fusion (SDLECF) framework for AMD. SDLECF framework is based on algorithms are Bi-GRU, SAE-DNN, BDLSTM, and MWDBN that allow higher-level fusion using a stacked generalization. Nature-Inspired Based Ensemble Feature Selection (NIEFS) is based on LEFPIO, COSSA, and FMGOA to generate optimal feature subset from the dataset. Lévy flying, pigeon behaviors are simulated based on PIO. COSSA, tent chaotic map is introduced which replicates the dynamic foraging activity of squirrels. FMGOA, imitates the life behaviour of grasshopper swarms with Triangular Membership Function (TMF). Mutual Information (MI) is used to combine an output of multiple different approaches to create the final optimum subset of features. Bi-GRU, SAE-DNN, BDLSTM, and MWDBN classifiers are combined to create the SDLECF system. In the Bi-GRU model, the update gate is replaced alternated of the input and forget gates in the Long Short-Term Memory (LSTM) network. In a bi-GRU model, the forward GRU obtains information about previous malware from the historical malware data, while the backward GRU obtains information regarding future malware from identical historical malware data. SAE-DNN is introduced to discover a mutual encoding for both supervised classification and unsupervised reconstruction. Bidirectional Recurrent Neural Network (BDRNN) is the concept from which BDLSTM. For malware detection, each training sample proceeds both forward and backward to two RNNs in BDRNN, which are both connected to the same output layer. Ranking Algorithm is used to combine the results of individual methods. SDLECF and existing classifiers are simulated using two separate datasets (Malgenome-215 and Drebin 215) in terms of Pre, Rec, FM, and WFM. Future work, online learning, and Reinforcement Learning (RL) approaches have been introduced for AMD. Instead of using ensemble learning model, parallel ensemble learning methods has been introduced for handling high dimensional and multidimensional dataset to handle both dynamic and hybrid features.

References

1. Tam, K., Feizollah, A., Anuar, N.B., Salleh, R., Cavallaro, L.: The evolution of android malware and android analysis techniques. ACM Comput. Surv. (CSUR). **49**(4), 1–41 (2017)
2. Liu, S., Lin, G., Han, Q.L., Wen, S., Zhang, J., Xiang, Y.: DeepBalance: deep-learning and fuzzy oversampling for vulnerability detection. IEEE Trans. Fuzzy Syst. **28**(7), 1329–1343 (2019)
3. Yu, B., Fang, Y., Yang, Q., Tang, Y., Liu, L.: A survey of malware behavior description and analysis. Front. Inf. Technol. Electron. Eng. **19**, 583–603 (2018)
4. Li, J., Sun, L., Yan, Q., Li, Z., Srisa-An, W., Ye, H.: Significant permission identification for machine-learning-based android malware detection. IEEE Trans. Industr. Inform. **14**(7), 3216–3225 (2018)
5. Sugunan, K., Gireesh Kumar, T., Dhanya, K.A.: Static and dynamic analysis for android malware detection. In: Advances in Big Data and Cloud Computing, pp. 147–155 (2018)
6. Qiu, J., Zhang, J., Luo, W., Pan, L., Nepal, S., Xiang, Y.: A survey of android malware detection with deep neural models. ACM Computing Surveys (CSUR). **53**(6), 1–36 (2020)
7. Dhal, P., Azad, C.: A comprehensive survey on feature selection in the various fields of machine learning. Appl. Intell. **52**(4), 4543–4581 (2022)
8. Akinola, O.O., Ezugwu, A.E., Agushaka, J.O., Zitar, R.A., Abualigah, L.: Multiclass feature selection with metaheuristic optimization algorithms: a review. Neural Comput. & Applic. **34**(22), 19751–19790 (2022)
9. Zhou, Z.H., Zhou, Z.H.: Ensemble Learning, pp. 181–210. Springer Singapore (2021)
10. Fan, Z., Wu, F., Tang, Y.: A hierarchy-based machine learning model for happiness prediction. Appl. Intell. **53**(6), 7108–7117 (2023)
11. Dong, X., Yu, Z., Cao, W., Shi, Y., Ma, Q.: A survey on ensemble learning. Front. Comp. Sci. **14**, 241–258 (2020)
12. Rokach, L.: Ensemble Learning: Pattern Classification Using Ensemble Methods, pp. 1–22 (2019)
13. Mahajan, P., Uddin, S., Hajati, F., Moni, M.A.: Ensemble learning for disease prediction: a review. In Healthcare. **11**(12), 1–21 (2023)
14. Zhu, H.J., Wang, L.M., Zhong, S., Li, Y., Sheng, V.S.: A hybrid deep network framework for android malware detection. IEEE Trans. Knowl. Data Eng. **34**(12), 5558–5570 (2021)
15. Lu, T., Du, Y., Ouyang, L., Chen, Q., Wang, X.: Android malware detection based on a hybrid deep learning model. Secur. Commun. Networks. **6**(1), 1–11 (2020)
16. İbrahim, M., Issa, B., Jasser, M.B.: A method for automatic android malware detection based on static analysis and deep learning. IEEE Access. **10**, 117334–117352 (2022)
17. Al Ogaili, R.R.N., et al.: AntDroidNet cybersecurity model: a hybrid integration of ant Colony optimization and deep neural networks for android malware detection. Mesopotamian J. Cyber Secur. **5**(1), 104–120 (2025)
18. Dong, S., Shu, L., Nie, S.: Android malware detection method based on CNN and DNN Bybrid mechanism. IEEE Trans. Industr. Inform. **20**(5), 7744–7753 (2024)
19. Alhussen, A.: Advanced android malware detection through deep learning optimization. Eng. Technol. Appl. Sci. Res. **14**(3), 14552–14557 (2024)
20. Xu, K., Li, Y., Deng, R.H., Chen, K.: Deeprefiner: multi-layer android malware detection system applying deep neural networks. In: 2018 IEEE European Symposium on Security and Privacy (EuroS&P), pp. 473–487 (2018)
21. Wang, W., Zhao, M., Wang, J.: Effective android malware detection with a hybrid model based on deep autoencoder and convolutional neural network. J. Ambient. Intell. Humaniz. Comput. **10**, 3035–3043 (2019)

22. Mahdavifar, S., Alhadidi, D., Ghorbani, A.A.: Effective and efficient hybrid android malware classification using pseudo-label stacked auto-encoder. J. Netw. Syst. Manag. **30**(1), 1–34 (2022)
23. Mahindru, A., et al.: PermDroid a framework developed using proposed feature selection approach and machine learning techniques for android malware detection. Sci. Rep. **14**(1), 1–38 (2024)
24. Masum, M., Shahriar, H.: Droid-NNet: deep learning neural network for android malware detection. In: 2019 IEEE International Conference on Big Data (Big Data), pp. 5789–5793 (2019)
25. Mafarja, M., Mirjalili, S.: Whale optimization approaches for wrapper feature selection. Appl. Soft Comput. **62**, 441–453 (2018)
26. Saremi, S., Mirjalili, S., Lewis, A.: Grasshopper optimisation algorithm: theory and application. Adv. Eng. Softw. **105**, 30–47 (2017)
27. Elmi, Z., Efe, M.Ö.: Multi-objective grasshopper optimization algorithm for robot path planning in static environments. In: 2018 IEEE International Conference on Industrial Technology (ICIT), pp. 244–249 (2018)
28. Dou, R., Duan, H.: Lévy flight based pigeon-inspired optimization for control parameters optimization in automatic carrier landing system. Aerosp. Sci. Technol. **61**, 11–20 (2017)
29. Ghafarzadeh, H., Bouyer, A.: An efficient hybrid clustering method using an artificial bee colony algorithm and mantegna Lévy distribution. Int. J. Artif. Intell. Tools. **25**(02), 1–17 (2016)
30. Zhang, X., Zhao, K., Wang, L., Wang, Y., Niu, Y.: An improved squirrel search algorithm with reproductive behavior. IEEE Access. **8**, 101118–101132 (2020)
31. Hoque, N., Singh, M., Bhattacharyya, D.K.: EFS-MI: an ensemble feature selection method for classification: An ensemble feature selection method. Complex Intell. Syst. **4**, 105–118 (2018)
32. Yerima, S.Y., Sezer, S.: Droidfusion: a novel multilevel classifier fusion approach for android malware detection. IEEE Trans. Cybern. **49**(2), 453–466 (2018)
33. Li, X., Ma, X., Xiao, F., Xiao, C., Wang, F., Zhang, S.: Time-series production forecasting method based on the integration of bidirectional gated recurrent unit (bi-GRU) network and sparrow search algorithm (SSA). J. Pet. Sci. Eng. **208**, 1–13 (2022)
34. Zhu, Z., Dai, W., Hu, Y., Li, J.: Speech emotion recognition model based on bi-GRU and focal loss. Pattern Recogn. Lett. **140**, 358–365 (2020)
35. Long, J., Sun, Z., Li, C., Hong, Y., Bai, Y., Zhang, S.: A novel sparse echo autoencoder network for data-driven fault diagnosis of delta 3-D printers. IEEE Trans. Instrum. Meas. **69**(3), 683–692 (2019)
36. Yang, Z., Gjorgjevikj, D., Long, J., Zi, Y., Zhang, S., Li, C.: Sparse autoencoder-based multi-head deep neural networks for machinery fault diagnostics with detection of novelties. Chin. J. Mech. Eng. **34**(1), 1–12 (2021)
37. Xu, C., Xie, L., Xiao, X.: A bidirectional LSTM approach with word embeddings for sentence boundary detection. J. Signal Process. Syst. **90**(7), 1063–1075 (2018)
38. Alzaylaee, M.K., Yerima, S.Y., Sezer, S.: DL-droid: deep learning based android malware detection using real devices. Comput. Secur. **89**, 1–11 (2020)