Journal of Intelligent & Fuzzy Systems 44 (2023) 10477–10493
DOI:10.3233/JIFS-230009
IOS Press

An ensemble deep learning classifier stacked with fuzzy ARTMAP for malware detection

Mohammed Nasser Al-Andoli[a], Shing Chiang Tan[b,*], Kok Swee Sim[a], Pey Yun Goh[b] and Chee Peng Lim[c]
[a]*Faculty of Engineering and Technology, Multimedia University, Malaysia*
[b]*Faculty of Information Science and Technology, Multimedia University, Malaysia*
[c]*Institute for Intelligent Systems Research and Innovation, Deakin University, Australia*

Abstract. Malicious software, or malware, has posed serious and evolving security threats to Internet users. Many anti-malware software packages and tools have been developed to protect legitimate users from these threats. However, legacy anti-malware methods are confronted with millions of potential malicious programs. To combat these threats, intelligent anti-malware systems utilizing machine learning (ML) models are useful. However, most ML models have limitations in performance since the training depth is usually limited. The emergence of Deep Learning (DL) models allow more training possibilities and improvement in performance. DL models often use gradient descent optimization, i.e., the Back-Propagation (BP) algorithm; therefore, their training and optimization procedures suffer from local sub-optimal solutions. In addition, DL-based malware detection methods often entail single classifiers. Ensemble learning overcomes the shortcomings of individual techniques by consolidating their strengths to improve the performance. In this paper, we propose an ensemble DL classifier stacked with the Fuzzy ARTMAP (FAM) model for malware detection. The stacked ensemble method uses several heterogeneous deep neural networks as the base learners. During the training and optimization process, these base learners adopt a hybrid BP and Particle Swarm Optimization algorithm to combine both local and global optimization capabilities for identifying optimal features and improving the classification performance. FAM is selected as a meta-learner to effectively train and combine the outputs of the base learners and achieve robust and accurate classification. A series of empirical studies with different benchmark data sets is conducted. The results ascertain that the proposed ensemble method is effective and efficient, outperforming many other compared methods.

Keywords: Ensemble learning, fuzzy ARTMAP, deep learning, malware detection, particle swarm optimization, backpropagation algorithm

1. Introduction

The Internet has become an indispensable part of our daily activities as computers and digital technologies have become increasingly ubiquitous. Many of us rely on Internet services on a daily basis, e.g., e-banking, e-commerce, instant communication, education, and entertainment [1]. All these activities open up opportunities for cyber-crimes, leading to an increase in spending on digital system protection. In this respect, malware (malicious software) is often used to launch cyber-attacks on victims' computers. Various types of malware are available, e.g. Trojans, viruses, horses, worms, ransomware, and rootkits. They have received due attention from the communities, as they present a major threat to cybersecurity. In 2018, for example, Symantec recorded 246,002,762 new malware variants, and Kaspersky detected 5,321,142 malicious Android packages [2].

*Corresponding author. Shing Chiang Tan, Faculty of Information Science and Technology, Multimedia University, Malaysia. E-mail: sctan@mmu.edu.my.

There are more and more malware variants that attempt to subvert antivirus tools and bypass many malware detection systems. Indeed, cyber-attacks have resulted in trillions of dollars in damage to the global economy [1, 3].

To protect computer users, threats from malware attacks should be detected as soon as possible. Researchers have developed machine learning (ML)-based classification methods for detecting malware [4]. While ML models achieve impressive results, they may perform poorly in detecting complex malware variants [3, 5]. In addition, many ML models suffer from a limited training depth. The advent of deep learning (DL) models has paved the way for improving training capabilities and performance. DL models have been employed to overcome the limitations of existing malware detection methods [6]. They offer several major strengths over standard ML models, which include automatic generation of high quality features and ability to deal with large data sets [7]. In the literature, various DL models, such as the Recurrent Neural Network (RNN) and Graph Convolutional Network (GCN) [4, 8], and hybrid DL-ML method [9] have been devised for malware detection. Even though good results have been reported, there is still room for improvement. This is because DL models often use the gradient descent technique, i.e., the Back-Propagation (BP) algorithm [10], for network optimization, which suffers from local sub-optimal solutions [10, 11]. Moreover, DL-based malware detection methods often entail the use of single classifiers.

Ensemble methods can be coupled with ML/DL models to improve their classification performance [12–14]. The underlying principle is to combine multiple ML/DL models and leverage the advantages of each model for compensating the individual shortcomings, leading to an improved performance [12, 13]. As a generalization of the standard ensemble method, a stacked ensemble model operates at two levels; the first-level learners, i.e., base learners, and the final-level learner, i.e., a meta-learner. The base learners provide processed information for building the next-level learners. The meta-learner leverages a combination of algorithms to make a final prediction by integrating all predictions of the base learners [15, 16]. In the stacked ensemble model, the selection of a meta-learner is crucial as it makes the final decision. The stacked ensemble learning method has demonstrated its efficacy in various domains, e.g. Android malware detection [8], slope stability analysis [14], enhancement of water quality [16], and stock market categorization [17]. ML models are often used in ensemble methods as the base learners as well as meta-learner. Nonetheless, certain ML models suffer from the overfitting issue with respect to training samples, leading to a reduction in the overall performance [18]. In this respect, Fuzzy ARTMAP (FAM) is a supervised Adaptive Resonance Theory (ART) neural network that is useful for dealing with the overfitting problem [19, 20]. This study aims to overcome the limitations of traditional ML methods in detecting complex malware variants through the use of DL models and an ensemble framework. DL models suffer from a limitation in terms of their optimization process, i.e., local optima of the BP algorithm, while single DL-based classifiers are also susceptible to local sub-optimal solutions [10, 11]. To address these issues, we investigate the use of a stacked ensemble method with multiple DL models to enhance generalization and performance. Additionally, we leverage the incremental learning property of ART-based models to prevent overfitting by allowing the meta classifier (FAM) to adapt to new knowledge from incoming data streams while retaining previously learned knowledge. This reduces the chance of overfitting by capturing only the essential knowledge from new data while avoiding irrelevant information and noise [18]. Ensemble models also mitigate errors and biases as well as reduce variance by introducing diversity. The heterogenous DL models used in the ensemble method are trained and optimized with a hybrid PSO and BP algorithm. This further reduces the possibility of overfitting, while improving generalization. Indeed, the combination of DL and FAM in an ensemble framework offers an effective approach to malware detection, as demonstrated in the empirical studies.

In this paper, we propose an Ensemble Deep Learning model stacked with Fuzzy ARTMAP (EDL-FAM) for malware detection. The proposed EDL-FAM method combines the advantages of both DL models and FAM in addressing the overfitting problem. A group of heterogeneous Deep Neural Network (DNN) models is utilized as the base learners at the first level. Each DNN employs a hybrid BP and Particle Swarm Optimization (PSO) algorithm for tuning its parameters. The aim is to find the optimal solution and improve the capability in malware detection. PSO is capable of searching for global optimal solutions [21, 22], while BP is effective in finding accurate local solutions [23]. Therefore, a hybrid PSO-BP approach entails the advantages of both global and local search capabilities to optimize

the DNN parameters. To realise robust and accurate classification, FAM is deployed as a meta-learner to combine the outputs from the base learners. The contributions of this paper are summarized, as follows:

- An ensemble method based on the DNN models stacked with FAM is proposed for malware detection and classification;
- A two-level learning process comprising a group of base learners and a meta-learner is devised. Several heterogeneous DNN models are adopted as the first-level base learners. They are trained with a hybrid optimization method based on PSO and BP that combines both local and global search capabilities. FAM is used as a meta learner to effectively combine the output from the base learners.

The remaining part of this paper is organized as follows. Section 2 presents the related studies in the literature. The proposed ensemble method, i.e. EDL-FAM, is introduced in Section 3. Section 4 presents the experimental results, analysis and discussion, while Section 5 draws conclusions, along with suggestions for future research.

2. Literature review

Malware data require a careful analysis to identify malicious behaviours, e.g. malware structure, infection risk, and spread speed. Various techniques have been developed to detect malware, covering signatures, behaviours, heuristics, and model-checking aspects. Signature-based techniques generate a set of bits to identify malware structure and analyse suspicious files for identifying malware signatures [24]. Behaviour-based techniques monitor the characteristics of data samples and determine whether they are benign or malware [1]. Malware behaviours can be traced by detecting registry changes, system calls, files, Application Program Interface (API) calls, and computer networks. A system-centric behavioural model was used to explore the interaction of malware with system resources, aiming to dynamically identify malware [25]. In the heuristic approach, ML models were embedded with heuristic rules to handle malware detection tasks [26]. In [27], a dynamic heuristic method used a Naïve Bayes classifier to compute the API call frequencies and detect suspicious files. Model-checking methods leverage linear temporal logic to extract the representative features of malicious and benign samples and encode them as flow relations [2]. In [28], a model-checking method was developed. The method converted all executable programs and software into a pushdown system. Overall, these malware detection methods suffer from different limitations. The signature-based methods are unable to discover different malware variants [29]; the behaviour-based methods yield compromised performance when handling complex malware variants; the model-checking and heuristic-based methods produce low performances when detecting malware created by obfuscation and packaging mechanisms. Moreover, the existing methods are deemed ineffective in detecting new and emerging malware variants [3].

While DL models have been employed in various domains, there is room to further develop an effective DL-based method for malware detection [3]. An DL framework for malware detection was proposed in [30] to analyse and identify static and dynamic malware. DL has also been integrated with image processing techniques to detect malware. In [4], DL was used to learn the representations of Android software and identify malware. In addition, GCN and RNN models have been utilized to automatically learn and identify semantic and sequential patterns. In [31], an automatic detection method based on Stacked Autoencoders (SAEs) and Deep Belief Networks (DBNs) was developed to detect unknown and new Android malware. In [32], a GCN model was devised to detect the relationships among system calls, leading to discovery of malicious Android data. In [33], an Android malware detection method was developed, along with GCN based on Jumping-Knowledge (JK) and its variants, namely GCN-JK, GraphSAGE-JK, and GIN-JK. In [34] a Generative Adversarial Network (GAN)-based Android malware classifier was used to create an API graph embedding solution.

Hybrid methods have been employed to improve malware detection performance. In [9], a DL model was combined with Random Forest (RF) to discover malignant data samples. A semi-supervised DL model was developed to detect obfuscated malware and identify its variants [35]. DL processing, feature engineering, and image transformation were used for such tasks. A Long Short-Term Memory (LSTM) and a CNN model were combined in [36] to detect malicious activities in Android software. It used Call-Graph as well as dynamic and static analyses for malware detection and classification. An exploratory analysis was carried out in [37] to identify important features for improving the performance of Drebin malware detectors. In [38], a fast Android malware

detection method using Dalvik-Opcode features and permission features was designed. The method first reduced features dimensionality to scale down time complexity of the method, and then performed malware detection with a CatBoost classifier.

Ensemble methods overcome the shortcomings of individual models and consolidate their strengths in problem solving. Several ensemble-based malware detection methods are available in the literature. In [13], a new Android malware identification method with an ensemble learning technique was developed for collaborative malware detection using a combination of intents and permissions. A meta-ensemble-based Android malware detection method was designed in [39], which applied static analyses for detecting malicious applications. An automated and configurable hybrid analysis method for malware detection was developed in [40]. Denoted Anatomist, it analysed Android application behaviours using an integration of static analysis and dynamic instrumentation. An ensemble method to malware detection by averaging the outputs of several classification methods was introduced in [40]. In [41], an extrinsic random-based ensemble method for Android malware detection was proposed. The model outputs were averaged to improve the malware detection performance. An Android malware prediction using ensemble ML models was devised in [42] to select the best performer in malware detection. In [43], an Android malware detection method with multi-faceted deep generative adversarial network model was proposed. Various ML models were assessed in [44] by applying an ensemble learning method for detecting Android malware. Overall, the abovementioned methods indicate that the ensemble approach is effective for undertaking malware detection tasks.

From the literature analysis, the existing DL-based malware detection methods often entail single classifiers. The reported DL methods typically adopt gradient optimization with BP, which can result in sub-optimal generalization. Moreover, most of the existing ML/DL-based ensemble malware detection methods are not equipped with an incremental learning capability. Stacked ensemble methods are also often based on standard ML and neural network models, which suffer from limitations on generalization and overfitting issues [18, 20]. To close this research gap, an ensemble DNN models with their parameters optimized using a hybrid PSO-BP algorithm is proposed in this paper. In addition, FAM is deployed as the meta learner to combine the outputs from the ensemble DNN models, aiming to improve performance generalization for malware detection.

3. Methodology

In this section, the proposed EDL-FAM method is explained in detail. Firstly, data pre-processing and cleansing is conducted. Then, several heterogeneous DNN models are formed as the base learners to perform the first-level data classification. The outputs are merged and sent to the meta-learner (FAM) to conduct the final-level data classification. The ensemble DNN models are trained using a hybrid PSO-BP algorithm. Figure 1 shows a schematic diagram of EDL-FAM. A summary of EDL-FAM is presented in Algorithm 1.

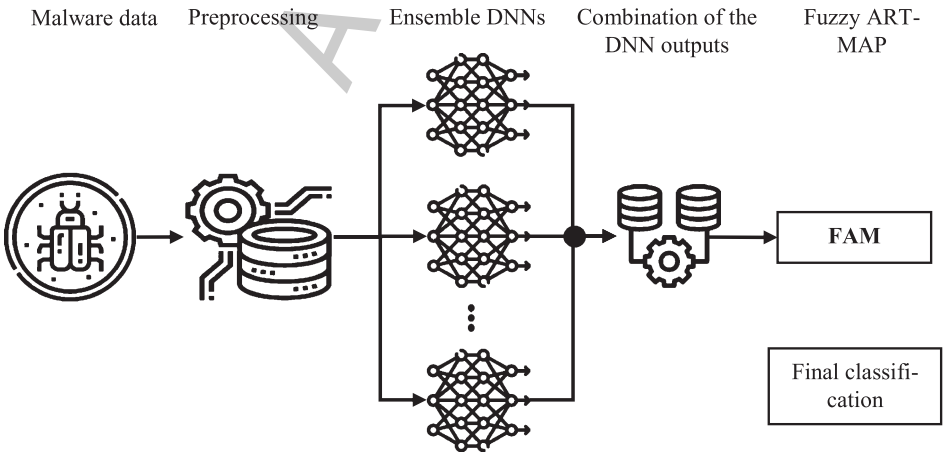

Fig. 1. A schematic diagram of EDL-FAM.

3.1. Pre-processing

During the pre-processing phase, the Python Numpy library is used for data pre-processing and cleansing. Missing and invalid data samples are firstly replaced by the associated average values. Next, the data samples are normalized with the z-score normalization [45], as defined in Equation 1.

$$z - score = \frac{(x - \mu)}{\sigma}, \tag{1}$$

where variables x, σ, μ are the original value, standard deviation, and mean value, respectively. The z-score is a widely used pre-processing technique in ML and data analysis. It is used to standardize the data sample values such that they have a mean of zero and a standard deviation of one. As such, data samples from populations with varying means and standard deviations can be standardized, simplifying comparisons among different variables [45].

The dimension reduction task is conducted to reduce the computational load in dealing with high dimensional data. It transforms a data sample from a high dimensional space into a low dimension one, in such a way that the new representation retains the meaningful features of the data sample. In this study, the procedure of averaging a set of neighbouring features and reducing them to single feature is conducted [46]. The number of adjacent features is governed by a hyper-parameter $F \in [1, a]$, where a represents the original features. The new feature dimension (F_{size}) is defined in Equation 2:

$$F_{size} = \frac{a}{F}. \tag{2}$$

After going through data normalization and dimensionality reduction, the original data set with a features is reduced to one with F features, where $F < a$.

3.2. Ensemble classification

3.2.1. DNN models (Base learners)

The stacked ensemble model performs two-level learning. The first-level classification involves a number of base learners while the second-level involves a meta-learner. The base learners comprise DNN models and the meta-learner model is FAM.

Once the data features are cleansed and prepared, they are sent to the base (DNN) learners. Each DNN model is executed independently. The input features are processed by each DNN model, and a prediction is produced at the final layer ($n * d$), where d and n indicate the target class and the number of samples, respectively. The DNN hidden layers between the input and output layers use functions $h = f(W_1 x + b_1)$, $h_1 = f_1(W_2 h_1 + b_2)$, $\ldots$ $o = f_1(W_{l+1} h_1 + b_{l+1})$, where x, h, o, l are the sample features, the hidden layer output, the final layer output, and the number of layers, respectively. The DNN layers are dense, and a Rectified Linear Unit (*ReLU*) activation function is used in the hidden layers. In the ensemble DNN models, two activation functions are employed in the last hidden layer, i.e., sigmoid and Softmax functions. Both are used separately.

The base DNN learners are trained and optimized with a hybrid metaheuristic (PSO) and gradient decent (BP) algorithm. PSO supports constructive collaboration among all particles to search for a global optimal solution. BP has the ability to produce a non-linear mapping through local search. PSO-BP yields solutions better than those from individual methods.

Specifically, BP employs the Weighted Cross Entropy (WCE) as a loss function to adjust the DNN trainable parameters, θ, in a minimization mode. The loss function (J_θ) is computed using Equation 3:

$$\begin{aligned} J_\theta = \frac{1}{n} \sum\nolimits_{i=1}^{n} [X_i \log(Y_i) + \\ (1 - X_i) \log(1 - X_i)], \end{aligned} \tag{3}$$

where n is the number of samples, X_i and Y_i are the original and predicted class labels. Parameters θ are updated in local search with BP using Equations 4 and 5:

$$\theta_\alpha^{ij} = \theta_\alpha^{ij} - \gamma \frac{\partial}{\partial \theta_\alpha^{ij}} J_\theta(X, Y), \tag{4}$$

$$\begin{aligned} &\frac{\partial}{\partial \theta_\alpha^{ij}} J_\theta\,(X, g(f(X))) = \\ &\sum_{i=1}^{N} \frac{\partial}{\partial \theta_\alpha^{ij}} J_\theta\,(X_i, g(f(X_i))) = \\ &\sum_{i=1}^{N} \frac{\partial}{\partial z_\alpha^{j}} J_\theta\,(X_i, g(f(X_i))) \frac{\partial}{\partial \theta_\alpha^{ij}} z_\alpha^{j} = \\ &\sum_{i=1}^{N} \delta_\alpha^{j} X_i^{T}, \end{aligned} \tag{5}$$

where $\gamma$is the learning rate, N is the number of neurons, and α refers to the activation values of the hidden layer, H, and the output layer, Y. The contribution of each sample in the optimization process leas to the total error $\delta_{\alpha}^{j} = \frac{\partial}{\partial z_{\alpha}^{j}} J_{\theta}(X_i, g(f(X_i)))$.

The local optimization output from BP is M_{θ}, which is computed using Equations 3 to 5. Once local optimization with BP is performed, M_{θ} is re-used for PSO optimization. The PSO algorithm obtains M_{θ} and generates a pool of particles P_s. Particles P_s are distributed to several regions of the search space. Next, each particle P_i moves with velocity $V_{\theta i}$ over iterations t to update its position. The PSO functions are defined as follows (Equations 6 and 7):

$$V_{\theta i}^{(t+1)} = \lambda v_{\theta i}^{(t)} + c_1 r_1 \left[p_{\theta i}^{best(t)} - M_{\theta i}^{(t)} \right] + c_2 r_2 \left[g_{\theta}^{best(t)} - M_{\theta i}^{(t)} \right], i \in [1, P_s], \tag{6}$$

$$M_{\theta i}^{(t+1)} = M_{\theta i}^{(t)} + V_{\theta i}^{(t+1)}, i \in [1, P_s], \tag{7}$$

For each particle, P_i, the best local solution is updated with Equation 8.

$$P_{\theta i}^{best} = M_{\theta i} | f(M_{\theta i}) = \min_{j=1,2,3,\ldots,t+1} \left\{ f\left(M_{\theta i, j}\right) \right\}, \tag{8}$$

In Equations 6 to 8, r_1 and r_2 are two randomly chosen numbers in the interval [0, 1]; $M_{\theta i}$ is the output of a local replica; λ determines the P_s movement inertia; c_1 is the cognitive parameter and c_2 is the social parameter, while t, v, g^{best}, and P^{best} are the number of iterations, velocity, global best particle and local best particle, respectively. The best global particle is computed by averaging P^{best} or the best local particle with the highest rank. The global best particle is computed using Equation 9.

$$g_{\theta}^{best} = \min \left(\begin{bmatrix} P_{\theta i}^{best} | f(P_{\theta i}^{best}) = \min_{i=1,2,3,\ldots,P_s} \left\{ f(P_{\theta i}^{best}) \right\} \end{bmatrix}, \begin{bmatrix} \frac{1}{P_s} \sum_{i=1}^{P_s} P_{\theta i}^{best} \end{bmatrix} \right). \tag{9}$$

The fitness function is calculated with the average values of the loss function (i.e., WCE) from local BP optimization ($J_{\theta M} = (J_{\theta 1}, J_{\theta 2}, J_{\theta 3}, \ldots, J_{\theta m})$) in Equation 3, as indicated in Equation 10:

$$f(J_{\theta M}) = \frac{1}{P_s} \sum_{i=1}^{P_s} J_{\theta i} \tag{10}$$

where P_s refers to the number of particles. Once the fitness function is improved in a minimization mode, $P_{i\theta}^{best}$ and $g_{i\theta}^{best}$ are updated using Equations (6) to (9).

Optimization of DNN models is repeated until a high performance is achieved at the first-level classification. Then, each DNN base learner sends its output a to the merging step, in which all DNN outputs are combined into single vector. Finally, the merged output is sent to FAM, i.e. the meta-learner, to perform the final-level data classification, as explained in Section 3.2.2.

3.2.2. Fuzzy ARTMAP (Meta-learner)

FAM is a neuro-fuzzy model developed by Carpenter and Grossberg [19]. It combines the concepts of ART and fuzzy set theory to provide an incremental learning algorithm capable of addressing the stability-plasticity dilemma in pattern recognition.

FAM consists of two modules, namely ART_a and ART_b. Both modules are bridged with a map field [19] linking the generated clusters from the input domain (ART_a) and the output domain (ART_b). Each ART_a/ART_b consists of three layers: the normalization layer, input layer, and output layer, denoted by F_0, F_1, and F_2, respectively. Figure 2 shows the FAM structure.

In the F_0 layer, ART_a receives a d-dimensional input vector a and normalizes it through a complement-coding process to a 2d-dimensional input vector A, as in Equation (11):

$$A = (a, a^c) = (a_1, \ldots, a_M, 1 - a_1, \ldots, 1 - a_M) \tag{11}$$

where a is the original input vector and a^c is the complement of a. The complement-coding procedure also takes place in ART_b simultaneously, where B=(b, b^c).

Complement-coding preserves the norm of individual input vector and prevents category proliferation by concatenating an input with its complement-coded part. The complement-coded input vector A (as well as B) is forwarded to the input layer F_1 of each ART module. The F_2 layer is a prototype layer consisting of a set of prototype nodes that represents knowledge learned and coded based on training data.

In ART_a, the degree of similarity between A and each prototype node in the F_2 layer is computed using a choice function, as defined in Equation (12):

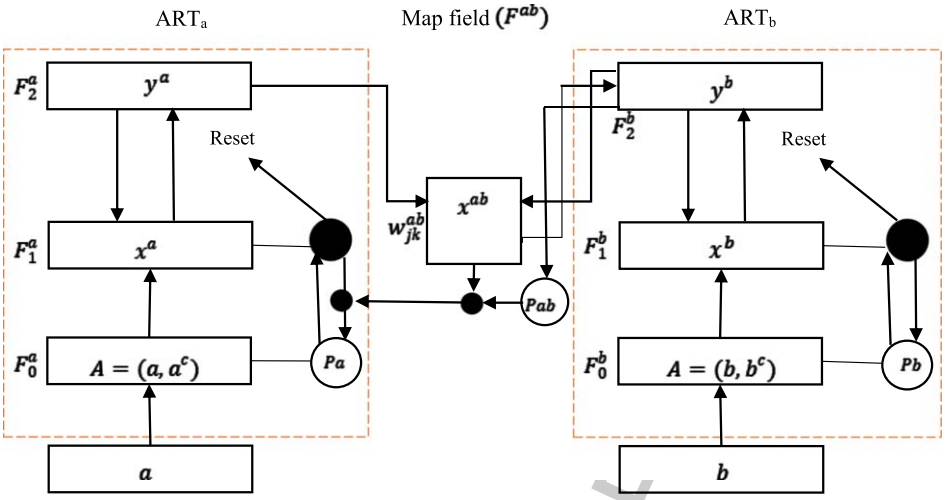

Fig. 2. The structure of Fuzzy ARTMAP.

$$T_j(A) = \left| \frac{A \wedge w_j^a}{\left| w_j^a \right| + \alpha_a} \right|, \tag{12}$$

where w_j^a is the weight of the j-th prototype node in the F_2 layer, and $a_a$is the choice parameter of ART_a. The norm $|\cdot|$ performs the summation of vector elements whereas the fuzzy AND operator $\wedge$ compares two vectors and returns a vector containing the element-wise minimum. In accordance with the winner-take-all principle, the node with the highest activation value from Equation 12 is selected as the winning node J, while all other nodes are de-activated.

Once a wining node is selected, a vigilance test is executed to check the degree of similarity between A and the weight vector of node J against a vigilance parameter, $\rho_a \in [0, 1]$, where:

$$\left| \frac{A \wedge w_J^a}{A} \right| \geq \rho_a, \tag{13}$$

If the vigilance test is not satisfied, node J is de-activated. A new search cycle to find another winning node is conducted. In other words, the search process continues until the chosen winning node J is able to satisfy Equation (13). The same search cycle and vigilance test take place in ART_b to confirm a winning node for the target class vector.

The winning node from ART_a sends a predicted class (encoded in w_J^{ab}) to ART_b through the map field. A map field vigilance test (Equation 14) takes place to check if the predicted class matches the target class (encoded in y^b).

$$\frac{\left| y^b \wedge w_J^{ab} \right|}{\left| y^b \right|} \geq \rho_{ab}, \tag{14}$$

where ρ_{ab} denotes the output of ART_b, and ρ_{ab} is normally set to a value close to 1 for classification problems. When the map field vigilance test is not satisfied, a mismatch occurs between the winning nodes from both ART modules, and a match-tracking procedure is triggered in ART_a to increase ρ_a using Equation (15):

$$\rho_a = \frac{\left| A \wedge w_J^a \right|}{|A|} + \delta, \tag{15}$$

where δ is a small positive value (e.g., 0.0001). Upon executing match tracking, the current winning node in ART_a is de-activated. A new node is created in F_2 if none of the existing F_2 nodes in ART_a is able to satisfy the ART_a vigilance test as well as map field vigilance test.

If the map field vigilance test is satisfied, a learning phase ensues. The weight vector of the winning node, w_J, in ART_a is updated using Equation (16):

$$w_J^{new} = \beta \left(A \wedge w_J^{old} \right) + (1 - \beta)\, w_J^{old}, \tag{16}$$

where $\beta \in [0, 1]$ is the learning rate.

The FAM algorithm is shown in Algorithm 2.

FAM within EDL-FAM helps mitigate overfitting and improve generalization through the following two aspects:

Algorithm 1: EDL-FAM

Input: Data samples
Output: Classification of each sample as malware or benign

//Preprocessing data: Refer to Section 3.1
Handling missing and invalid values
Normalization with Equation 1
Reducing sample features with Equation 2
//Ensemble classification:
// a) DNN models (Base learners):
Refer to Section 3.2.1
for $i = 1$ to e
for $i = 1$ to t

Forall (p in P_s):
Generate DNN models
Optimize DNN models with PSO-BP
Produce output of DNN models
end for
end for
Generate prediction values for the entire data set
// b) Form the input for Meta-learner:
Out_*base* = Combine the outputs of all base learners
// c) Fuzzy ARTMAP (Meta-learner):
labels = FAM(Out_base) Refer to Section 3.2.2
Return malware and benign samples.

Algorithm 2: FAM

Input: Outputs from Base learners
Output: Classification labels

//Complement Coding
In F_0, ART_a receives a d-dimensional input vector a
Normalize a *and* Produce A with Equation (11)
//Category Choice (ART_a)
Input vector A is propagated to the F_2 layer
//foreach i ($i = 1$ to a) in ART_a, ***do***:
// foreach w_j ($j = 1$ ***to*** N) in F_2, ***do***:
Compute $T_j()$ with Equation (12)
Find the category node
Perform vigilance test with Equation (13)
if (Vigilance is met), ***then***:
Select winning category
else:
De-activate node j
Add a new node in F_2 if all existing nodes fail to pass the vigilance test
end //end //end
//Category Choice (ART_b)
Repeat category choice of ART_a (lines) for ART_b
//Map Field Prediction
Do map-filed prediction with Equation (14)
If (*the map-filed prediction is not satisfied*):
Perform match-tracking procedure with Equation (15)
else
//Map Field Learning
The weight vector of the winning node is updated with Equation (16).
Return classification labels

1. Incremental Learning: FAM utilizes the incremental learning principle to adjust its model structure, i.e. prototype nodes in the F_2 layer. Comparing with the batch learning scheme, the risk of overfitting in FAM is minimised with its incremental learning property. This is because the FAM structure is able to adapt to new incoming data streams based on a sample-by-sample basis. As such, instead of establishing a predefined number of hidden nodes as in standard feedforward neural networks, FAM creates new nodes whenever it is necessary in accordance with the vigilance setting to incorporate new information, therefore mitigating the risk of overfitting.
2. Ensemble Learning: As the meta-learner, the outputs from multiple DNN models are combined by FAM to reach a final prediction. This helps improves generation by focusing on the DNN outputs and avoiding potential outliers and noise in the original data samples [18]. Ensemble models are also effective in reducing errors, biases, and variance by incorporating diversity and reducing redundancy, leading to a parsimonious FAM model to avoid overfitting and achieve enhanced generalization and improved performance.

4. Experimental studies and results

A set of experiments is presented in this section to demonstrate and validate the performance of EDL-FAM. The obtained results are analysed and discussed comprehensively.

4.1. Data sets

The proposed EDL-FAM model is evaluated with four malware data sets, *viz.*,:

1. *CICMalDroid*2020[1]. This is a recent Android Malware data set, which consists of 9,803 malware samples and 1,795 benign samples.
2. *DikeDataset*[2]. It consists of 1,082 benign and 10,841 malicious Portable Executable (PE) and Object Linking and Embedding (OLE) files.
3. *Drebin*[3]. This is a common data set used to detect Android malicious software. It consists of 10,000 samples, comprising 4,500 benign

[1] https://www.unb.ca/cic/datasets/maldroid-2020.html
[2] https://github.com/iosifache/DikeDataset
[3] https://www.sec.cs.tu-bs.de/~danarp/drebin/

and 5,500 malwares from 179 different malware families.

4. *Network Traffic Android Malware (NTAM)*[4]. This data set consists of network layer features for malware detection applications. It consists of 7,845 samples, with 3,141 malware and 4,704 benign samples.

4.2. Experimental settings

Coded using the Python programming language, EDL-FAM is executed on a computer with an Intel Core-i7 processor, Ubuntu 20.04 operating system, and 8 GB RAM. For the base learners (i.e., DNNs), the number of iterations, batch size, and learning rate are set to 100, 256, and 0.001, respectively. Five DNN models are used to form an ensemble model for performance evaluation. For the PSO parameters, *c1, c2*, the inertia weight, the number of particles are set to 0.5, 0.3, 20, 0.9, respectively. Each particle's position is initialised randomly between 0 and 1. The hyper-parameter F in Equation 2 is set to 1 for NTAM, and DikeDataset, while that for CICMalDroid2020 and Drebin is set to 3 and 5, respectively. The parameter setting is chosen after several trails, which allows EDL-FAM to achieve the best results. The data set is divided into a training set and a test set with a ratio of 4 to 1. The experiment is repeated ten times, and the average results are computed.

For performance comparison, six commonly used ML models are used to form ensemble classifiers, namely Ensemble Decision Tree (EDT), Ensemble Random Forest (ERF), Ensemble Gradient Boosting (EGB), Ensemble Ada Boost (EAB), Ensemble Support Vector Machine (ESVM), and Ensemble K-Nearest Neighbors (EKNN). In addition, DT, RF, GB, AB, KNN, and SVM are combined to form a heterogeneous ML-based ensemble classifier (denoted as HML-EC). The ScikitLearn Python package is employed to implement these ML methods with the default parameter settings in the package. Furthermore, three state-of-the-art DL-based single classifiers are used for comparison, i.e., DBN-SAE-MD [31], CNN-MD [30], and CNN-LSTM-MD [36].

Since the Drebin data set is commonly used for malware detection, we conduct another evaluation to compare EDL-FAM with several DL and ML methods, which are based on single and ensemble classifiers. Specifically, seven state-of-the-art single classifiers published in 2020-2022 are employed, i.e., GCN-AMD [32], GCN-JK [33], GIN-JK [33], SAGE-JK [33], VGAEMalGAN [34], DeepDiveDrebin [37], and FMulAMD [38]. Nine state-of-the-art ensemble methods are also used, i.e., PIndroid [13], FSEC-MD [39], TA-AMD [40], ERE-AMD [41], EAMP-EML [42], MDGAN-MD [43], Stacking DT-SVM-LR [44], Blending DT-SVM-LR [44].

[4] https://www.kaggle.com/datasets/xwolf12/network-traffic-android-malware

4.3. Evaluation metrics

Accuracy, precision, recall, and F1-score metrics are adopted for performance evaluation and comparison. These metrics are computed based on the True Positive (*TP*), False Positive (*FP*), True Negative (*TN*), and False Negative (*FN*) rates, as in Equations (17) to (20):

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}. \tag{17}$$

$$Precision = \frac{TP}{TP + FP}. \tag{18}$$

$$Recall = \frac{TP}{TP + FN}. \tag{19}$$

$$F1 - score = \frac{Recall \times Precision}{Recall + Precision}. \tag{20}$$

4.4. Experimental assumptions

We make the following experimental assumptions pertaining to performance evaluation of EDL-FAM for malware detection: (1) The data sets used for evaluation, namely Drebin, CICMalDroid2020, and NTAM, are sufficiently diverse and representative of various malware instances; (2) It is adequate to use two target classes (benign and malicious) for performance evaluation in malware detection; (3) It is effective to use random over-sampling technique to tackle the imbalanced class distribution issue in the data sets for experimentation; (4) It is representative to replace missing/invalid data samples with the associated average values, and to normalize all data samples using z-score normalization; (5) it is sufficient to use accuracy, precision, recall, and F1-score for evaluation, as each indicator assesses a different aspect of classification performance.

4.5. Results and discussion

The results of EDL-FAM pertaining to the CICMalDroid, Dike, Drebin, and NTAM data sets are summarized in Tables 1 to 5, where the bold scores indicate the best performance.

4.5.1. Results of EDL-FAM

Table 1 summarizes the EDL-FAM results, along with those of FAM, DNN-Sigmoid, and DNN-Softmax models. EDL-FAM achieves the best accuracy rates of 97.41%, 98.82%, 99.1%, and 99.49% for the CICMalDroid, Dike, Drebin, NTAM data sets, respectively. Table 1 also shows that DNN-Sigmoid performs well, while DNN-Softmax performs the worst in terms of the performance indicators. FAM yields acceptable results, but is inferior to DNN-Sigmoid. Unlike DNN-Sigmoid, the poor performance of DNN-Softmax is because of its inefficiency in tackling binary classification problems [47], as per the four data sets used in this experimental study.

4.5.2. Results of EDL-FAM and ML-based ensemble methods

Table 2 presents the results of EDL-FAM and seven ML-based ensemble methods pertaining to the (a) CICMalDroid, (b) Dike, (c) Drebin, and (d) NTAM data sets. In general, EDL-FAM returns better results than those from all compared ML-based ensemble methods, i.e., EDT, ERF, EGB, EAB, EKNN, ESVM, and HML-EC. HML-EC achieves the best performance among the ML-based ensemble methods. On the other hand, ERF has the lowest accuracy

Table 1
Malware detection performance of EDL-FAM, Fuzzy ARTMAP, DNN-Sigmoid, and DNN-Softmax performed on (a) CICMalDroid, (b) Dike, (c) Drebin, and (d) NTAM data sets

	(a) CICMalDroid				(b) Dike			
Method	Accuracy	Precision	Recall	F1-score	Accuracy	Precision	Recall	F1-score
Fuzzy ARTMAP	89.35	89.77	97.78	93.6	96.28	97.32	98.58	97.94
DNN-Sigmoid	94.74	95.42	98.46	96.91	97.12	98.24	98.59	98.42
DNN-Softmax	85.32	86.71	90.89	95.51	90.23	90.23	95.31	94.86
EDL-FAM	**97.41**	**97.96**	**98.97**	**98.46**	**98.82**	**99.07**	**99.63**	**99.35**
	(c) Drebin				(d) NTAM			
Method	Accuracy	Precision	Recall	F1-score	Accuracy	Precision	Recall	F1-score
Fuzzy ARTMAP	84.05	83.35	87.36	85.31	98.15	98.68	96.62	97.64
DNN-Sigmoid	96.1	97.49	95.09	96.28	99.33	99. 2	**99.36**	99.12
DNN-Softmax	82.0	81.8	84.9	83.3	87.95	85.96	84.27	85.11
EDL-FAM	**99.1**	**99.09**	**99.27**	**99.18**	**99.49**	**99.84**	98.87	**99.35**

Table 2
Comparison with seven ML-based ensemble methods on (a) CICMalDroid, (b) Dike, (c) Drebin, and (d) NTAM data sets

	(a) CICMalDroid				(b) Dike			
Method	Accuracy	Precision	Recall	F1-score	Accuracy	Precision	Recall	F1-score
EDT	93.0	95.2	96.6	95.9	95.4	95.6	95.4	95.6
ERF	89.0	94.9	92.0	93.4	96.4	96.7	96.4	96.7
EGB	90.8	93.3	96.1	94.7	95.2	95.2	95.2	95.2
EAB	93.6	93.6	99.4	96.4	96.5	97.3	96.5	97.3
EKNN	93.5	96.9	95.4	96.1	94.1	94.4	94.1	94.4
ESVM	93.2	93.6	98.9	96.1	95.9	96.3	95.9	96.3
HML-EC	94.3	95.6	97.8	96.7	97.5	97.9	97.5	97.9
EDL-FAM	**97.41**	**97.96**	**98.97**	**98.46**	**98.82**	**99.07**	**99.63**	**99.35**
	(c) Drebin				(d) NTAM			
Method	Accuracy	Precision	Recall	F1-score	Accuracy	Precision	Recall	F1-score
EDT	95.4	94.3	97.3	95.7	95.1	99.3	88.3	93.5
ERF	96.3	96.8	96.7	96.7	96.9	97.6	96.0	96.8
EGB	90.6	89.0	94.5	91.7	96.7	96.1	97.2	96.6
EAB	90.9	91.2	92.3	91.8	96.1	95.6	96.5	96.1
EKNN	88.4	96.6	80.8	88.4	95.0	95.6	94.3	94.5
ESVM	95.6	98.7	92.8	95.8	96.7	96.8	96.6	96.7
HML-EC	96.7	97.0	96.9	96.9	97.7	98.0	97.4	97.6
EDL-FAM	**99.1**	**99.09**	**99.27**	**99.18**	**99.49**	**99.84**	**98.87**	**99.35**

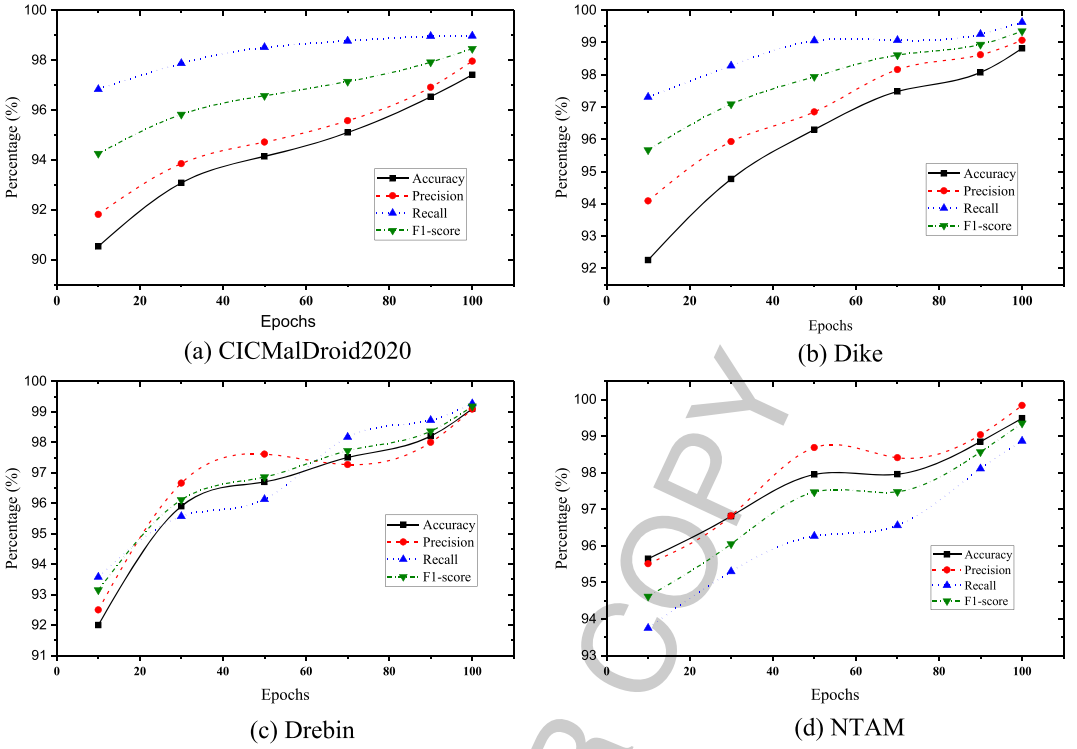

Fig. 3. The performance of EDL-FAM at different numbers of epochs of the base learner models.

rate with the CICMalDroid data set, while EKNN has the lowest accuracy rate with both Dike and Drebin data sets. Meanwhile, EAB and EKNN report the worst accuracy scores with the NTAM data set. In addition, HML-EC outperforms other ML-based ensemble classifiers (EDT, ERF, EGB, EAB, EKNN, and ESVM).

The impact of the base learners on the overall EDL-FAM performance is investigated, as shown in Fig. 3. The performance of EDL-FAM varies with different number of epochs of the base learners. Good training and optimization of the base learners lead to improved EDL-FAM performance. As an example, increasing the number of epochs of the base learners, i.e., DNN-Sigmoid and DNN-Softmax, can enhance EDL-FAM in terms of accuracy, precision, recall, and F1-score.

4.5.3. Results of EDL-FAM and DL methods

A comparison among EDL-FAM and five DL-based malware detection methods is conducted. The compared methods are DBN-SAE-MD [31], CNN-MD [30], and CNN-LSTM-MD [36], and two DNN models with Sigmoid and Softmax activation functions.

From Table 3(a), EDL-FAM performs well with the CICMalDroid data set, outperforming all compared methods (CNN-MD, DBN-SAE-MD, CNN-LSTM-MD, DNN-Sigmoid, and DNN-Softmax) in all four performance metrics. DNN-Sigmoid and CNN-LSTM achieve the best results among the compared methods. On the other hand, DNN-Softmax produces the lowest performance in accuracy, precision, and recall, while DBN-SAE-MD yields the lowest F1-score. For the Dike data set, the results in Table 3(b) indicate that EDL-FAM achieves the best performance with the highest rates pertaining to accuracy, precision, F1-score and recall. The results in Table 3(c) and (d) indicate that EDL-FAM outperforms all DL methods for the Drebin and NTAM data sets, respectively. CNN-LSTM and DNN-Sigmoid return the best results among the compared methods, which are close to those of EDL-FAM. DNN-Softmax yields the worst performance in all four performance indicators.

A set of confusion matrices has been generated to further illustrate the EDL-FAM performance. Figure 4 shows the confusion matrices of EDL-FAM for the CICMalDroid, Dike, Drebin, and NTAM data sets. EDL-FAM has high TP (malware) and TN

Table 3
Comparison of EDL-FAM with five DL methods on (a) CICMalDroid, (b) Dike, (c) Drebin, and (d) NTAM data sets

Method	(a) CICMalDroid				(b) Dike			
	Accuracy	Precision	Recall	F1-score	Accuracy	Precision	Recall	F1-score
CNN-MD	95.2	96.5	97.8	97.1	97.6	98.7	97.6	98.7
DBN-SAE-MD	85.0	85.0	96.64	92.0	95.0	93.1	95.0	93.1
CNN-LSTM-MD	94.4	96.2	97.8	97.0	97.7	98.8	97.7	98.8
DNN-Sigmoid	94.74	95.42	98.46	96.91	97.12	98.24	98.59	98.42
DNN-Softmax	85.32	86.71	90.89	95.51	90.23	90.23	95.31	94.86
EDL-FAM	**97.41**	**97.96**	**98.97**	**98.46**	**98.82**	**99.07**	**99.63**	**99.35**
Method	(c) Drebin				(d) NTAM			
	Accuracy	Precision	Recall	F1-score	Accuracy	Precision	Recall	F1-score
CNN-MD	95.0	95.4	93.5	97.5	98.1	97.8	98.4	98.0
DBN-SAE-MD	94.7	95.1	93.7	96.6	95.5	96.8	94.4	95.6
CNN-LSTM-MD	96.5	96.4	97.9	97.7	98.7	98.5	98.8	98.7
DNN-Sigmoid	96.1	97.49	95.09	96.28	99.33	99. 2	**99.36**	99.12
DNN-Softmax	82.0	81.8	84.9	83.3	87.95	85.96	84.27	85.11
EDL-FAM	**99.1**	**99.09**	**99.27**	**99.18**	**99.49**	**99.84**	**98.87**	**99.35**

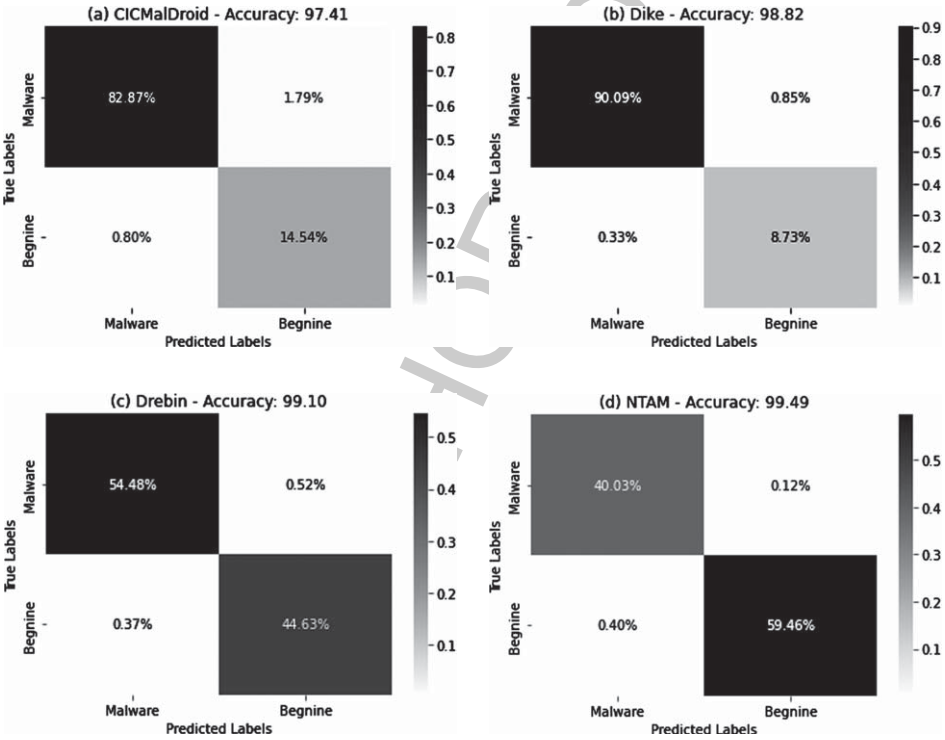

Fig. 4. The confusion matrices of EDL-FAM on the CICMalDroid, Dike, Drebin, and NTAM data sets.

(benign) predictions, and low FP and FN predictions. The high percentage of TP and TN demonstrates a high effectiveness of EDL-FAM in detecting malicious instances. Moreover, the low percentages of FP and FN indicate that over-fitting is not an issue in EDL-FAM. This observation is further supported by the high precision, recall, and F1-score results in Tables 1–3, ascertaining the efficacy of EDL-FAM in malware detection and classification.

4.5.4. Statistical analysis

An analysis to evaluate the results from the statistical perspective is conducted. Two nonparametric statistical hypothesis tests have been used, namely the Friedman test and the Nemenyi post-hoc tests. The Friedman test calculates the average ranked performance in terms of all four performance indicators at a significance level of α=0.05 (i.e., 95% confidence level) with respect to those from ML-based ensemble

Table 4
Friedman test results

Methods	Accuracy	Precision	Recall	F1-Score
ML-based ensemble methods	0.007454	0.013143	0.019677	0.002626
DL methods	0.003327	0.003755	0.003299	0.005075

methods comprising EDT, ERF, EGB, EAB, EKNN, ESVM, and HML-EC methods (Table 2) and the DL methods comprising DBN-SAE-MD [31], CNN-DM [30], CNN-LSTM-MD [36], DNN-Sigmoid, and DNN-Softmax (Table 3). Table 4 presents the *p-values* of the Friedman test, indicating a significant difference in performance between EDL-FAM and ML-based ensemble models as well as DL models at α=0.05.

We apply the Nemenyi post-hoc test to further ascertain the performance difference, as shown in Figs. 5 and 6. It is clear that EDL-FAM achieves the highest performance rank in all four performance indicators with respect to ML-based ensemble models and DL models. HML-EC has the highest rank among the ML-based ensemble methods, which is close to EDL-FAM in performance. This is because it uses heterogeneous ML-based ensemble models. Meanwhile, CNN-LSTM stands at the highest rank among the DL models, but it is still inferior to EDL-FAM.

Table 4 and Figs. 5 and 6 indicate that the there is a significant difference in performance between EDL-FAM and compared methods. The effectiveness of EDL-FAM is due to several reasons. Firstly, EDL-FAM employs ensemble methods to combine the strengths of multiple individual models. Secondly, EDL-FAM uses effective DL models as its base learners and FAM as a useful incremental learning meta-learner in its structure. Thirdly, EDL-FAM adopts a hybrid approach combining metaheuristic optimization with PSO and gradient optimization

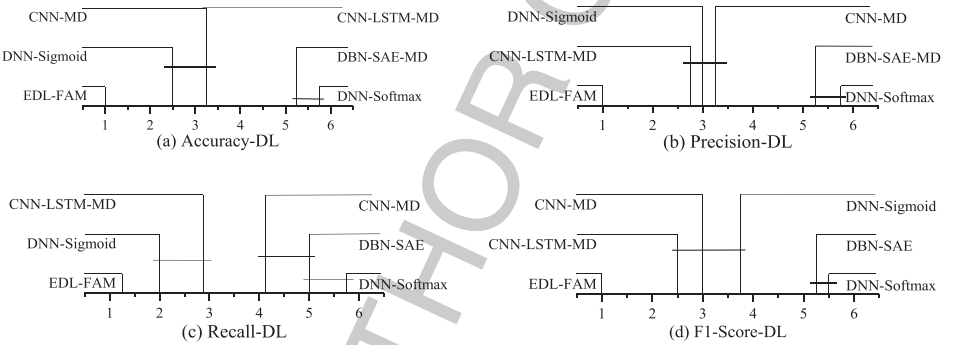

Fig. 5. Average ranking between EDL-FAM and DL-based methods

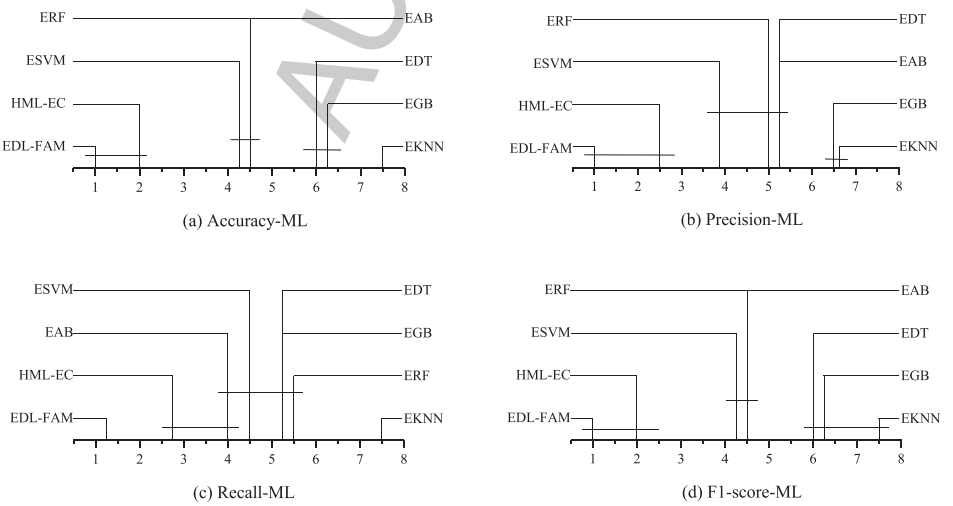

Fig. 6. Average ranking between EDL-FAM and the EML methods.

Table 5
Performance comparison with various single classifiers on the Drebin data set

Method	Accuracy	Precision	Recall	F1-score
GCN-AMD [32]	92.30	91.50	92.30	93.30
SAGE-JK [33]	95.24	95.35	95.24	95.0
GCN-JK [33]	96.88	97.01	96.88	97.0
GIN-JK [33]	95.03	95.07	95.03	95.0
VGAEMalGAN [34]	97.68	95.27	91.08	93.13
DeepDiveDrebin [37]	96.8	97.8	89.4	93.4
FMulAMD [38]	96.00	97.00	96.00	95.00
EDL-FAM	**99.1**	**99.09**	**99.27**	**99.18**

Table 6
Performance comparison with various ensemble methods on the Drebin data set

Method	Accuracy	Precision	Recall	F1-score
PIndroid [13]	98.4	97.5	97.5	97.5
EAMP-EML [42]	**99.3**	99.0	99.0	99.0
MDGAN-MD [43]	96.2	95.1	94.6	94.7
Stacking DT-SVM-LR [44]	98.0	97.0	98.0	97.0
Blending DT-SVM-LR [44]	97.7	96.0	98.0	97.0
FSEC-MD [39]	97.6	–	–	–
TA-AMD [40]	98.2	–	–	–
ERE-AMD [41]	99.1	–	–	–
EDL-FAM	99.1	**99.09**	**99.27**	**99.18**

with BP. This allows EDL-FAM to search for the best solution and achieve the best performance.

4.5.5. Comparison with recent related studies

To further validate the effectiveness of EDL-FAM, we conduct two additional studies using the Drebin data set. The first is a comparison between EDL-FAM and seven state-of-the-art single classifiers, i.e., GCN-AMD [32], GCN-JK [33],GIN-JK [33], SAGE-JK [33], VGAEMalGAN [34], DeepDiveDrebin [37], and FMulAMD, as in Table 5. The second study is a comparison between EDL-FAM and eight state-of-the-art ensemble classifiers, i.e., PIndroid [13], FSEC-MD [39], TA-AMD [40], ERE-AMD [41], EAMP-EML [42], MDGAN-MD [43], Stacking DT-SVM-LR [44], and Blending DT-SVM-LR [44]), as in Table 6. The results in Tables 5 and 6 clearly indicate that EDL-FAM outperforms all compared methods (single and ensemble classifiers), producing the best scores in four performance metrics, except accuracy in the comparison with ensemble methods.

4.5.6. Run time analysis

Figure 7 shows the computation time of EDL-FAM, seven ML-based ensemble methods, and four DL-based methods. ESVM consumes the longest computational time, especially with the Drebin data set, as it has a large number of input features. However, it works well with an acceptable runtime for data sets with smaller numbers of features, such as Diki and NTAM. ERF consumes the shortest runtime. Other EML methods require shorter computational durations than those of DL methods, including EDL-FAM. EDL-FAM is faster than other DL methods in all four data sets, except for DNN-Sigmoid that yields the shortest runtime among the DL methods.

While the data sets used for performance comparison (i.e., CICMalDroid, Dike, Drebin, and NTAM) are well-known benchmark malware detection and classification problems, they have several limitations. The data samples are limited to numerical information for representing malware behaviours. Other forms of data modality such as text and images are not included. Besides, the target outputs are focused on benign and malicious categories only. Mapping input data samples to multiple target outputs, such as malware families and types, is important to have more accurate detection outcomes. Additionally, while the data set size is acceptable, it is not sufficient for big data analysis, which is increasingly common in today's digital environment. Despite these limitations, the results of the empirical study are good indication of the usefulness of EDL-FAM in tackling malware classification problems, and additional investigations can be conducted to further improve its robustness and performance.

5. Conclusion

In this paper, we have devised an effective and efficient ensemble method, denoted as EDL-FAM, for malware detection. Specifically, EDL-FAM is a stacked ensemble learning model. It assembles multiple DNNs as the base learners and utilizes FAM as a meta learner for malware classification. A hybrid PSO-BP algorithm has also been formulated to combine both local and global optimization capabilities for identifying optimal features and improving the classification performance. After pre-processing phase, the data samples with newly formed features are used for classification into either benign or malware categories. Based on a series of experimental studies with four benchmark malware data sets, EDL-FAM are able to outperform many well-known ensemble ML methods and state-of-art DL malware detection methods.

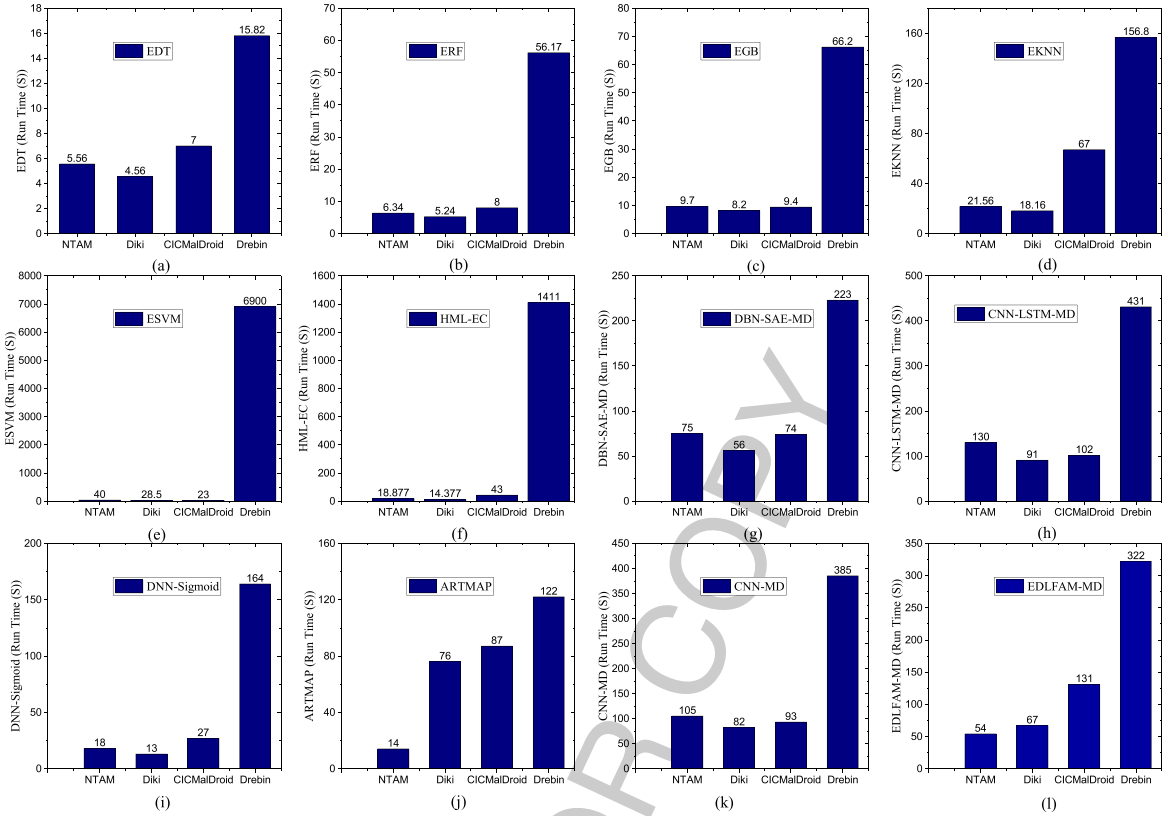

Fig. 7. Computation time of (a) EDT, (b) ERF, (c) EGB, (d) EKNN, (e) ESVM, (f) HML-EC, (g) DBN-SAE-MD, (h) CNN-LSTM-MD, (i) DNN-Sigmoid, (j) Fuzzy ARTMAP, (k) CNN-MD, and (l) EDL-FAM methods that performed on NTAM, Diki, CICMMallDroid, and Drebin data sets.

Malware detection is a rapidly evolving area, and different types of malware continue to appear quickly. In this study, while we have devised a useful ensemble method for malware detection, we have only focused on binary classification (benign and malicious) with numerical malware data samples in a spatial setting. As such, further research to address these issues would be useful. Correspondingly, a new DL architecture with different malware data modalities, e.g. text and image information, can be investigated. In addition, the use of temporal DL models for malware detection in the context of multimodal data analysis is important, particularly in situations where the data samples consisting of numerical and visual information can change over time. On the other hand, instead of classifying malware into benign and malicious categories only, further studies can focus on expanding the current work to tackle different malware families and types, and systematically evaluate its effectiveness for multi-class malware detection in real-world environments.

Acknowledgment

This work was supported by the Fundamental Research Grant Scheme (FRGS) from the Ministry of Higher Education and Multimedia University, Malaysia (Project ID: FRGS/1/2019/ICT02/MMU/02/2) and the MMU Grant (Project ID: MMUI/220154).

References

[1] Ö.A. Aslan and R. Samet, A comprehensive review on malware detection approaches, *IEEE Access* **8** (2020), 6249–6271.

[2] D. Li and Q. Li, Adversarial deep ensemble: Evasion attacks and defenses for malware detection, *IEEE Transactions on Information Forensics and Security* **15** (2020), 3886–3900.

[3] Ö. Aslan and A.A. Yilmaz, A new malware classification framework based on deep learning algorithms, *IEEE Access*, 2021.

[4] X. Pei, L. Yu and S. Tian, AMalNet: A deep learning framework based on graph convolutional networks

for malware detection, *Computers & Security* **93** (2020), 101792.

[5] G. Lin, S. Wen, Q.-L. Han, J. Zhang and Y. Xiang, Software vulnerability detection using deep neural networks: a survey, *Proceedings of the IEEE* **108**(10) (2020), 1825–1848.

[6] M.N. Al-Andoli, S.C. Tan, K.S. Sim, C.P. Lim and P.Y. Goh, Parallel Deep Learning with a hybrid BP-PSO framework for feature extraction and malware classification, *Applied Soft Computing,* pp. 109756, 2022.

[7] Y. LeCun, Y. Bengio and G. Hinton, Deep learning, *Nature* **521**(7553) (2015), 436–444.

[8] J. Qiu, J. Zhang, W. Luo, L. Pan, S. Nepal and Y. Xiang, A survey of android malware detection with deep neural models, *ACM Computing Surveys (CSUR)* **53**(6) (2020), 1–36.

[9] S. Yoo, S. Kim, S. Kim and B.B. Kang, AI-HydRa: Advanced hybrid approach using random forest and deep learning for malware classification, *Information Sciences* **546** (2021), 420–435.

[10] M.N. Al-Andoli, S.C. Tan, W.P. Cheah and S.Y. Tan, A Review on Community Detection in Large Complex Networks from Conventional to Deep Learning Methods: A Call for the Use of Parallel Meta-Heuristic Algorithms, *IEEE Access* **9** (2021), 96501–96527.

[11] B. Akay, D. Karaboga and R. Akay, A comprehensive survey on optimizing deep learning models by metaheuristics, *Artificial Intelligence Review*, pp. 1–66, 2021.

[12] R. Damaševičius, A. Venčkauskas, J. Toldinas and Š. Grigaliūnas, Ensemble-based classification using neural networks and machine learning models for windows pe malware detection, *Electronics* **10**(4) (2021), 485.

[13] F. Idrees, M. Rajarajan, M. Conti, T.M. Chen and Y. Rahulamathavan, PIndroid: A novel Android malware detection system using ensemble learning methods, *Computers & Security* **68** (2017), 36–46.

[14] N. Kardani, A. Zhou, M. Nazem and S.-L. Shen, Improved prediction of slope stability using a hybrid stacking ensemble method based on finite element analysis and field data, *Journal of Rock Mechanics and Geotechnical Engineering* **13**(1) (2021), 188–201.

[15] D.H. Wolpert, Stacked generalization, *Neural networks* **5**(2) (1992), 241–259.

[16] T. Yan, A. Zhou and S.-L. Shen, Prediction of long-term water quality using machine learning enhanced by Bayesian optimisation, *Environmental Pollution* **318** (2023), 120870.

[17] Y. Wang et al., Stacking-based ensemble learning of self-media data for marketing intention detection, *Future Internet* **11**(7) (2019), 155.

[18] M. Jin, Z. Xu, R. Li and D. Wu, Fuzzy ARTMAP ensemble based decision making and application, *Mathematical Problems in Engineering* **2013**, 2013.

[19] G.A. Carpenter, S. Grossberg, N. Markuzon, J.H. Reynolds and D.B. Rosen, Fuzzy ARTMAP: A neural network architecture for incremental supervised learning of analog multidimensional maps, *IEEE Transactions on neural networks* **3**(5) (1992), 698–713.

[20] Z. Xu, Y. Li, Z. Wang and J. Xuan, A selective fuzzy ARTMAP ensemble and its application to the fault diagnosis of rolling element bearing, *Neurocomputing* **182** (2016), 25–35.

[21] P. Feng, L. Xiao-Ting, Z. Qian, L. Wei-Xing and G. Qi, Analysis of standard particle swarm optimization algorithm based on Markov chain, *Acta Automatica Sinica* **39**(4) (2013), 381–389.

[22] E.T. Mohamad, D.J. Armaghani, E. Momeni, A.H. Yazdavar and M. Ebrahimi, Rock strength estimation: a PSO-based BP approach, *Neural Computing and Applications* **30**(5) (2018), 1635–1646.

[23] U. Bhattacharya and S.K. Parui, Self-adaptive learning rates in backpropagation algorithm improve its function approximation performance, in *Proceedings of ICNN'95 - International Conference on Neural Networks*, 27 Nov.-1 Dec. **5** (1995), 2784–2788, doi: 10.1109/ICNN.1995.488172.

[24] M.F. Zolkipli and A. Jantan, A framework for malware detection using combination technique and signature generation, in *2010 Second International Conference on Computer Research and Development*, 2010: IEEE, pp. 196–199.

[25] A. Lanzi, D. Balzarotti, C. Kruegel, M. Christodorescu and E. Kirda, Accessminer: using system-centric models for malware protection, in *Proceedings of the 17th ACM conference on Computer and communications security*, 2010, pp. 399–412.

[26] K. Alzarooni, Malware variant detection, UCL (University College London), 2012.

[27] E.M. Alkhateeb and M. Stamp, A dynamic heuristic method for detecting packed malware using naive bayes, in *2019 International Conference on Electrical and Computing Technologies and Applications (ICECTA)*, 2019: IEEE, pp. 1–6.

[28] F. Song and T. Touili, Pushdown model checking for malware detection, *International Journal on Software Tools for Technology Transfer* **16**(2) (2014), 147–173.

[29] J. Scott, Signature based malware detection is dead, *Institute for Critical Infrastructure Technology*, 2017.

[30] R. Vinayakumar, M. Alazab, K. Soman, P. Poornachandran and S. Venkatraman, Robust intelligent malware detection using deep learning, *IEEE Access* 7 (2019), 46717–46738.

[31] S. Hou, A. Saas, L. Chen, Y. Ye and T. Bourlai, Deep neural networks for automatic android malware detection, in *Proceedings of the 2017 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining 2017*, (2017), 803–810.

[32] T.S. John, T. Thomas and S. Emmanuel, Graph convolutional networks for android malware detection with system call graphs, in *2020 Third ISEA Conference on Security and Privacy (ISEA-ISAP)*, 2020: IEEE, pp. 162–170.

[33] W. Weng Lo, S. Layeghy, M. Sarhan, M. Gallagher and M. Portmann, Graph Neural Network-based Android Malware Classification with Jumping Knowledge, *arXiv e-prints,* p. arXiv: 2201.07537, 2022.

[34] R. Yumlembam, B. Issac, S.M. Jacob and L. Yang, IoT-based Android Malware Detection Using Graph Neural Network With Adversarial Defense, *IEEE Internet of Things Journal*, 2022.

[35] A. Darem, J. Abawajy, A. Makkar, A. Alhashmi and S. Alanazi, Visualization and deep-learning-based malware variant detection using OpCode-level features, *Future Generation Computer Systems* **125** (2021), 314–323.

[36] S. Hosseini, A.E. Nezhad and H. Seilani, Android malware classification using convolutional neural network and LSTM, *Journal of Computer Virology and Hacking Techniques,* pp. 1–12, 2021.

[37] N. Daoudi, K. Allix, T.F. Bissyandé and J. Klein, A deep dive inside drebin: An explorative analysis beyond android malware detection scores, *ACM Transactions on Privacy and Security* **25**(2) (2022), 1–28.

[38] H. Bai, N. Xie, X. Di and Q. Ye, Famd: A fast multifeature android malware detection framework, design, and implementation, *IEEE Access* **8** (2020), 194729–194740.

[39] L.D. Coronado-De-Alba, A. Rodríguez-Mota and P.J. Escamilla-Ambrosio, Feature selection and ensemble of classifiers for Android malware detection, in *2016 8th IEEE Latin-American Conference on Communications (LATINCOM)*, 2016: IEEE, pp. 1–6.

[40] V. Kouliaridis, G. Kambourakis, D. Geneiatakis and N. Potha, Two anatomists are better than one—dual-level android malware detection, *Symmetry* **12**(7) (2020), 1128.

[41] N. Potha, V. Kouliaridis and G. Kambourakis, An extrinsic random-based ensemble approach for android malware detection, *Connection Science* **33**(4) (2021), 1077–1093.

[42] N. Al Sarah, F.Y. Rifat, M.S. Hossain and H.S. Narman, An efficient android malware prediction using Ensemble machine learning algorithms, *Procedia Computer Science* **191** (2021), 184–191.

[43] F. Mazaed Alotaibi, A Multifaceted Deep Generative Adversarial Networks Model for Mobile Malware Detection, *Applied Sciences* **12**(19) (2022), 9403.

[44] M.S. Rana and A.H. Sung, Evaluation of advanced ensemble learning techniques for Android malware detection, *Vietnam Journal of Computer Science* **7**(02) (2020), 145–159.

[45] D.G. Zill, *Advanced engineering mathematics*. Jones & Bartlett Publishers, 2020.

[46] N. SpolaôR, E.A. Cherman, M.C. Monard and H.D. Lee, A comparison of multi-label feature selection methods using the problem transformation approach, *Electronic Notes in Theoretical Computer Science* **292** (2013), 135–151.

[47] A.A. Lydia and F.S. Francis, Multi-label classification using deep convolutional neural network, in *2020 International Conference on Innovative Trends in Information Technology (ICITIIT)*, 2020: IEEE, pp. 1–6.