Cluster Computing
https://doi.org/10.1007/s10586-024-04475-7

Novel hybrid classifier based on fuzzy type-III decision maker and ensemble deep learning model and improved chaos game optimization

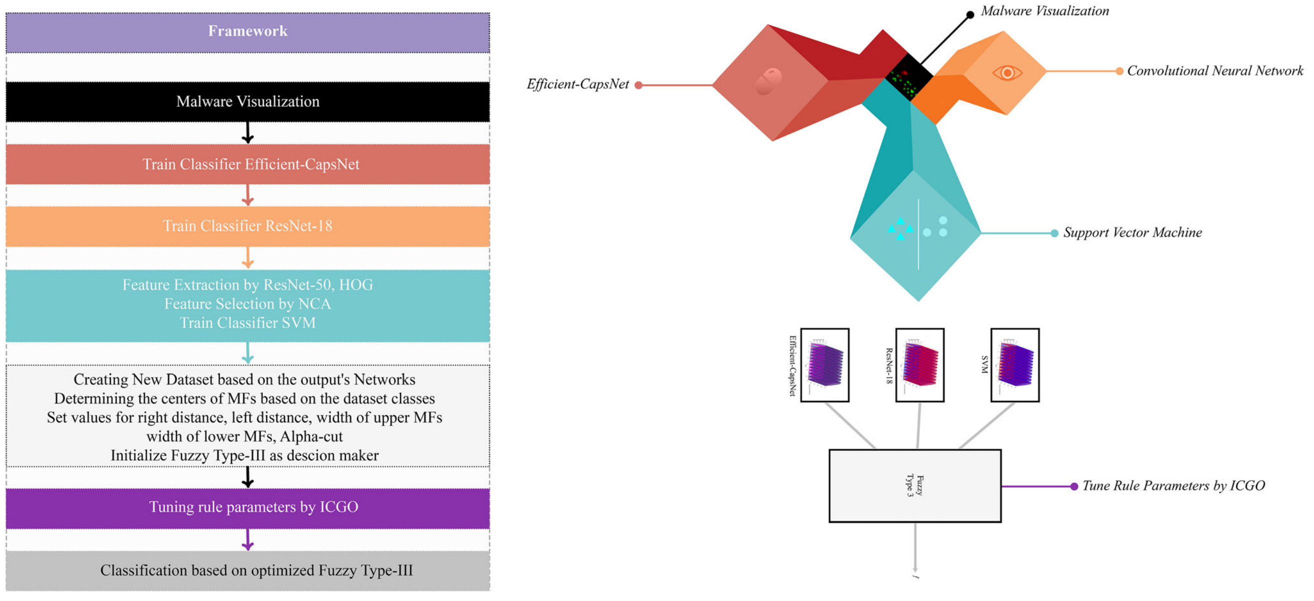
Nastaran Mehrabi Hashjin[1] · Mohammad Hussein Amiri[1] · Ardashir Mohammadzadeh[5] · Seyedali Mirjalili[2,3] · Nima Khodadadi[4]

Received: 18 December 2023 / Revised: 10 February 2024 / Accepted: 26 March 2024

Abstract

This paper presents a unique hybrid classifier that combines deep neural networks with a type-III fuzzy system for decision-making. The ensemble incorporates ResNet-18, Efficient Capsule neural network, ResNet-50, the Histogram of Oriented Gradients (HOG) for feature extraction, neighborhood component analysis (NCA) for feature selection, and Support Vector Machine (SVM) for classification. The innovative inputs fed into the type-III fuzzy system come from the outputs of the mentioned neural networks. The system's rule parameters are fine-tuned using the Improved Chaos Game Optimization algorithm (ICGO). The conventional CGO's simple random mutation is substituted with wavelet mutation to enhance the CGO algorithm while preserving non-parametricity and computational complexity. The ICGO was evaluated using 126 benchmark functions and 5 engineering problems, comparing its performance with well-known algorithms. It achieved the best results across all functions except for 2 benchmark functions. The introduced classifier is applied to seven malware datasets and consistently outperforms notable networks like AlexNet, ResNet-18, GoogleNet, and Efficient Capsule neural network in 35 separate runs, achieving over 96% accuracy. Additionally, the classifier's performance is tested on the MNIST and Fashion-MNIST in 10 separate runs. The results show that the new classifier excels in accuracy, precision, sensitivity, specificity, and F1-score compared to other recent classifiers. Based on the statistical analysis, it has been concluded that the ICGO and propose method exhibit significant superiority compared to the examined algorithms and methods. The source code for ICGO is available publicly at https://nimakhodadadi.com/algorithms-%2B-codes.

Graphical abstract


Framework
Malware Visualization
Train Classifier Efficient-CapsNet
Train Classifier ResNet-18
Feature Extraction by ResNet-50, HOG
Feature Selection by NCA
Train Classifier SVM
Creating New Dataset based on the output's Networks
Determining the centers of MFs based on the dataset classes
Set values for right distance, left distance, width of upper MFs
width of lower MFs, Alpha-cut
Initialize Fuzzy Type-III as descion maker
Tuning rule parameters by ICGO
Classification based on optimized Fuzzy Type-III
Malware Visualization
Efficient-CapsNet
Convolutional Neural Network
Support Vector Machine
Efficient-CapsNet
ResNet-18
SVM
Fuzzy Type 3
Tune Rule Parameters by ICGO

Extended author information available on the last page of the article

Published online: 05 May 2024

Keywords Type-III fuzzy system · Efficient-capsule network · Malware classification · Chaos game optimization algorithm · Residual neural network

Abbreviations

CNN	Convolutional neural network
BF	Benchmark function
UM	Unimodal
MM	Multimodal
ResNet	Residual neural network
HM	High-dimensional multimodal
TBT	Three bar truss
TCS	Tension/compression spring
WB	Welded beam
PV	Pressure vessel
CSI	Car side impact
F	Function
CEC	IEEE Congress on Evolutionary Computation
D	Dimension
C19	CEC2019
C17	CEC2017
LSHADE-EnSin	Linear population size reduction in success-history/adaptation of differential evolution with ensemble of sinusoidal functions
CGO	Chaos game optimization
ICGO	Improved chaos game optimization
TLBO	Teaching–learning-based optimization
IWO	Invasive weed optimization
GOA	Grasshopper optimization algorithm
AOA	Arithmetic optimization algorithm
PSO	Particle swarm optimization
SSA	Salp swarm algorithm
CD	Critical difference
GWO	Gray wolf optimization
SCA	Sine cosine algorithm
WOA	Whale optimization algorithm
Best	The best result
Worst	The worst result
Std.	Standard deviation
Mean	Average best result
EDP	Engineering design problem
RWA	Real world application

1 Introduction

The introduction of fuzzy systems by Lotfi A. Zadeh gave rise to a new research field in soft computing [1]. Although fuzzy logic was proposed for modeling uncertainties, ambiguity, and inaccuracy, one of the known shortcomings of type-I fuzzy systems is that while membership functions are used to express uncertainties, the outputs of these functions are crisp values. Lotfi A. Zadeh introduced type-II fuzzy systems in 1975 to address the problem [2]. Nonetheless, fewer studies were published on type-II fuzzy systems until late in the previous century [3]. Liang and Mendel first investigated type-II non-singleton fuzzy systems [4]. After Mendel's work on type-II fuzzy systems, these systems drew the interest of more researchers. The results obtained from the literature and the application of interval type-II fuzzy systems in practical situations with high noise levels, dynamic environments, and high complexity have demonstrated the superior performance of type-II over type-I fuzzy systems [5]. General type-II fuzzy systems can produce better results than interval type-II fuzzy systems due to their ability to handle higher uncertainty levels [6].

Type-III fuzzy systems, introduced in [7], are the evolved versions of the type-II fuzzy systems. They have been recently employed in various applications, such as robot control [8–11], time series prediction [12, 13], fault detection in gas turbine [14], and controller gain adjustment [15].The obtained results indicate that these systems can outperform type-I and type-II fuzzy systems by dealing with higher uncertainty levels.

Malware is among the most serious security threats on the Internet and leads to economic consequences for governments and businesses in addition to breaches of privacy [16]. Microsoft Windows is the most popular operating system, having a 74.79% share of the market [17]. This has resulted in the annual creation of a large number of malware programs for this operating system although it does not mean that other operating systems are safe from malware. According to reports by AVTEST, the number of malware programs created for Microsoft Windows was 793299151 in June 2023, 730142412 in June 2022, and 644297811 in June 2021, indicating an annual increase of 8.64% and 11.75%, respectively [18].

The act of identifying and subsequently categorizing malware is referred to as Malware Analysis. Static, Dynamic and both analyses [19] stand out as the predominant techniques employed in malware analysis. Static analysis generally involves parsing the executable code and extracting features and signatures that can aid in detection. Dynamic analysis involves executing the executable file in a controlled environment to observe its behavior. During dynamic malware analysis, the executable's interactions with the system are monitored, including system calls, executed instructions, and accessed registers. Dynamic analysis has been proven to be a highly efficient method of expediting the identification of malicious code, owing to the multifarious dynamic behaviors that malware exhibits during its execution [20]. Although dynamic techniques for analyzing malware are usually more accurate than entirely static methods, their primary limitation is that they can only detect malicious actions when those actions occur during the analysis process. This limitation has led to fewer studies incorporating dynamic features in their research [20].

Deep learning is now widely used for malware classification tasks. To use deep neural networks for malware detection, first a raw binary malware file is converted into an image, which can be grayscale or colored. As this process is repeated, a dataset is created that can be used to train the network. The network can then be utilized as a malware classifier. It's essential to emphasize the research gap regarding the classification of malware, particularly in the context of dynamic features, which have received comparatively less attention in previous studies. The article introduces a novel classifier that integrates a type-III fuzzy decision maker with an ensemble of deep neural networks and support vector machines to address this gap and offer a fresh perspective on malware classification. Additionally, it's worth highlighting that the paper pioneers the utilization of fuzzy type-III as a decision maker, aiming to better model uncertainty and enhance classification accuracy. This marks the first instance where fuzzy type-III has been employed in such a capacity, underscoring the innovative nature of the proposed approach.

The contributions of this article can be summarized as follows:

- Introducing a unique hybrid classifier that combines deep neural networks with a type-III fuzzy system.
- Implementing the type-III fuzzy system for decision-making, as opposed to traditional methods like voting.
- Showcasing an enhanced CGO optimization algorithm for training the type-III fuzzy system.
- To assess how well ICGO performs in addressing optimization challenges, it is subjected to testing across a spectrum of 126 BFs, encompassing different categories such as UM, MM, as well as the CEC 2019 and CEC 2017 sets. This evaluation spans dimensions ranging from 10 to 100, and it also includes five EDPs to examine how the problem dimensions influence the efficacy of the ICGO algorithm.
- Employing both balanced and imbalanced datasets and offering an in-depth analysis to provide a clearer insight into the performance of the proposed model.

The article is structured into five sections. Section 2 focuses on related work, while Sect. 3 covers material and methods. Section 4 presents simulation results and compares the performance of the different models. In Sect. 5, we will explain the proposed methodology. The Sect. 6 and final section provides conclusions based on the article's findings.

2 Related works

The field of operating system security extensively utilizes machine learning techniques for various applications, particularly for effectively classifying different malware families. In the following paragraphs, we will examine a selection of articles published in recent years that have employed machine learning techniques for the purposes of detecting and/or classifying malware.

Machine learning offers two main approaches for detecting and classifying malware: image-based and operational code (Opcode) families-based methods. The former involves analyzing visual representations of malware, while the latter leverages the sequence of operations, known as opcodes, within the program to identify malicious code. Parildi et al. [21] presented an alternative method for malware detection using assembly opcode sequences, utilizing natural language processing and deep learning techniques for deeper behavioral features, and achieving MCC scores of up to 0.95. In another similar study, Santos et al. [22] proposed a method that involved examining the frequency of opcode sequences and building a semi-supervised machine-learning classifier using a set of labeled and unlabeled malware and legitimate software instances. Empirical validation was performed to demonstrate that the labeling efforts required for this method were lower than those of supervised learning while still maintaining a high level of accuracy. Results indicate that by labeling only 50% of the software, more than 83% accuracy rates can be achieved.

The method of visualizing images is of great interest to many security researchers due to its ability to eliminate the need for feature engineering. The authors of [23] first created a feature vector by concatenating the extracted features from AlexNet and ResNet-152 and then used three fully connected layers and a Softmax function to classify malware. They evaluated the proposed method using Malimg, MBIG2015, and Malevis datasets. The classification accuracy for the proposed classifier was reported to be 97.78%, 94.88%, and 96.5% for the mentioned datasets, respectively. A lightweight deep neural network called IMCLNet has been introduced by [16] for classifying malware. To evaluate the proposed network, two datasets, Malimg and MBIG2015 (Microsoft BIG 2015), were used, and their classification accuracy was reported as 99.785% and 98.942%, respectively, by the proposed classifier.

A DNN-based malware classifier for Windows programs was proposed to address vulnerabilities in adversarial perturbation attacks [24]. A defensive mechanism uses a generative adversary network (GAN). The GAN-based adversarial samples achieve high-quality samples with medium cost, and the enhanced DNN achieves satisfactory accuracy with a 90.20% evasion ratio. GAN secures the DNN-based malware classifier with minimal performance degradation and minimizes evasion ratios when faced with powerful adversarial attacks. During [25], after feature extraction with VGG16, the features pass through two BiLSTM layers. Finally, the outputs generated by the BiLSTM layers and the features extracted by VGG16 are combined for malware classification. In order to mitigate the problem of imbalanced data, data augmentation techniques such as image shifting, vertical flipping, horizontal flipping, 45-degree clockwise rotation, and 45-degree counterclockwise rotation were employed in this article. A graph convolutional network malware classifier was developed to adapt to malware characteristics, achieving 98.32% accuracy and superior performance compared to existing methods [26].

A feature selection technique based on frequent Android permissions is explored to reduce computational effort [27]. The authors of [28] have introduced IMCFN, a classifier designed to identify various types of malware and enhance detection by implementing a deep learning architecture based on CNN. The method converts raw malware binaries into color images, using the fine-tuned CNN architecture to identify malware families. The IMCFN outperforms other CNN models, with an accuracy of 98.82% in the Malimg malware dataset and over 97.35% in the IoT-android mobile dataset. Hosseini et al. [29] demonstrated the effectiveness of Deep Neural Networks in malware classification, primarily using a combined convolutional neural network and RNNs. The proposed algorithm achieves maximum accuracy of 98.8% using fivefold cross-validation, surpassing CNN, Ensemble-learning, and SVM algorithms. Improvements are needed to enhance robustness and detect malware families for higher accuracy.

MAPAS is a malware detection system that uses Grad-CAM to analyze malicious applications' behaviors and API call graphs. Grad-CAM stands as a method that preserves the structure of complex models while providing insight into their decisions without sacrificing precision. It's lauded as a localization technique that delineates classes, providing visual insights for CNN-based networks sans the need for altering their architecture or undergoing re-training. MAPAS classifies applications 145.8% faster and uses ten times less memory than MaMaDroid, with higher accuracy (91.27%) for detecting unknown malware and any type of malware with high accuracy. This innovative approach offers a cost-effective solution for protecting users from emerging malware [30]. Reference [31] aims to propose a hybrid deep learning model called DeepVisDroid for detecting Android malware samples using image-based features. Four grayscale datasets were constructed, and a 1D-convolutional layers-based neural network model was trained using extracted local and global features. The model achieved classification accuracy of over 98% with efficient run-time overhead. Current deep CNN-based models require higher resources and heavy training operations, making them insufficient for IoT applications. Reference [32] proposes a lightweight CNN model for malware image classification, achieving 96.64% accuracy and suitable for resource-constrained applications.

In the subsequent section, we discuss various researches that have employed different machine learning techniques for the detection and categorization of malware. An example of this is the research carried out by Aurangzeb and his team [33]. In this article, a combination of five classifiers, namely Gradient Boosting, KNN, Random Forest, XGBoost, and Multilayer Perceptron, has been utilized to detect malware in software programs operating on the Android platform. The proposed classifier employs a classification method based on the voting mechanism among the mentioned classifiers. Authors of [34] present a hybrid approach for Android malware classification using fuzzy C-means clustering and LightGBM. Fuzzy clustering generates clusters of app permissions, while LightGBM classifies apps as malware or good ware after training, offering high learning efficiency and precise classification.

Reference [35] aimed to improve the detection performance of Android malware using machine learning-based malware detectors and investigate the impact of adversarial attacks on classifiers. The framework integrates static and dynamic features, with machine learning algorithms achieving 97.59% accuracy and random forest at 95.64%. When combined, deep neural network models achieve 99.28% accuracy and 99.59% accuracy. The paper also evaluates classifiers' robustness against evasion and poisoning attacks, revealing a significant drop in performance when simulating evasion attacks using static features. Dynamic features are also vulnerable to attack but exhibit more resilience than static features. A machine-learning approach was tested for detecting Android malware using a SVM classifier and Harris Hawks Optimization algorithm (HHO) [36].

Reference [37] proposed an Android malware detection technique using supervised learning to detect malware behavior. The supervised model achieves 97% accuracy in detecting malware, malicious API calls, and unusual app behavior. A simulated annealing algorithm and fuzzy logic were used in feature selection and neighbor generation stages to test ten feature sets, achieving 99.02% accuracy in feature selection with the KNN classifier [38].

Table 1 provides a summary of the selected related methods. The table indicates the use of k-fold cross-validation (CV), feature extraction (FE), classification algorithms (C), optimization techniques (Opt) for parameter tuning, introduction of new datasets (DS), reported accuracy (ACC), feature selection (FS), and feature processing (FP) in each work. This allows for a concise overview of the methodologies and contributions of previous studies relevant to the current research. Despite the critical role that these dynamic attributes play in identifying and analyzing malware, however, there has been a dearth of research focused on integrating them into the malware analysis process [39]. Moreover, a limited number of articles concerning the classification of malware have undertaken assessments of their classification techniques employing another dataset, including the renowned Fashion MNIST dataset and the MNIST dataset.

3 Material and methods

3.1 Dataset

This article uses the public dataset presented in [39] comprising seven distinct datasets. In order to generate this data, 65,536 malicious samples were extracted from the VirusShare repository and then filtered to yield 15,872 viable executable malware files. The Cuckoo Sandbox software by Linux Software was utilized to conduct dynamic analysis on these files safely. This software executed the malware in an isolated environment and logged the behaviors and functions, outputting 15,872 report.json files. The Cuckoo Sandbox system also integrates with the VirusTotal scan service to identify files containing viruses and other malware. The AVClass2 was then applied to automatically label the malware samples into categories based on their attributes and actions [40]. The outcome of this process was a dataset including 3749 real malware samples categorized into 11 distinct classes. The distribution of the Malware Family is reported in Table 2, and the visual representation of it can be observed in Fig. 1.

From the analysis report generated in the previous stage, we identified three dynamic features included in the report. Subsequently, we will elaborate on each of these features,

Table 1 The summary of selected related work is reported

#	Ref. no.	Year	Analysis	Method	Results and dataset	CV
1	Aslan et al.	2021	Static	FE: ResNet-152, AlexNet C: Fully connected layers and Softmax function	Microsoft BIG 2015, ACC = 94.88 Malimg, ACC = 97.78 Malevis, ACC = 96.5	✗
2	Mallik et al.	2022	Static	FE: VGG16 FP: BiLSTM C: Fully connected layers and Softmax function	Malimg, ACC = 99.56 Microsoft BIG 2015, ACC = 98.36	✗
3	Anupama et al.	2022	Static and dynamic	C: Random Forest, SVM, CART	Drebin, ACC = 100	✔
4	Alzubi et al.	2022	Static	C: SVM Opt: HHO	CICMalAnal2017, F-Measure: 93.20	✔
5	Seyfari and Meimandi	2023	Static and dynamic	C: SVM, KNN, DT FS: SA-Fuzzy	DREBIN, ACC = 99.02	✔
6	Taha and Malebary	2021	Static	C: Light Gradient Boosting PD: FCM	DS, ACC = 94.63	✔
7	Hosseini et al.	2021	Static and dynamic	C: CNN-LSTM	DREBIN, ACC = 98.8 DS, ACC = 90.48	✔
8	Vasan et al.	2020	Static	C: CNN	Malimg, ACC = 98.82 IoT-Android Mobile Dataset, ACC = 97.35	✗
9	Zou et al.	2022	Static	C: CNN	Microsoft BIG 2015, ACC = 97.976 Malimg, ACC = 99.143	✔

Table 2 The distribution of Malware Family

Class name	Number	Label
AgentTesla	628	0
Autolt	508	1
Bladabindi	215	2
Delf	125	3
Emotet	756	4
Guloader	234	5
Playtech	187	6
Qbot	143	7
Taskun	281	8
Trickbot	517	9
Ursnif	155	10

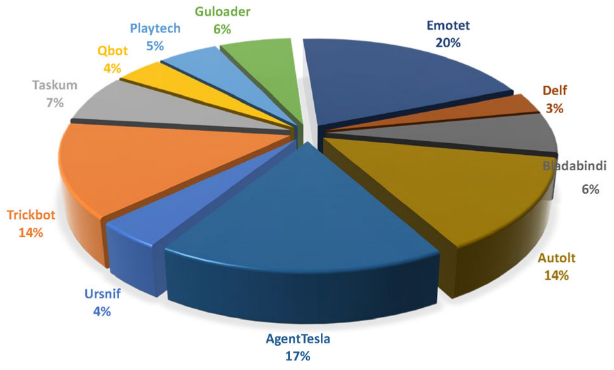


Fig. 1 The distribution of the malware family

namely, The Application Programming Interface (API) calls and their frequency, Loaded Dynamic Link Libraries (DLL) files, and Registry operations (REG), to provide a detailed explanation of how each feature can reveal insights into the behavior of the malware during its execution. API calls refer to the requests made by Malware to initiate an interaction with the operating system or other software components. By analyzing the type and frequency of these API calls, we can deduce the malware's behavior and intentions such as network access, file creation, and modification [41]. DLL files contain functions and data used by other programs. Using DLL files helps promote code modularization, code reuse, and efficient memory usage [42]. Malware often loads specific DLL files to utilize their functions and perform malicious activities. Identifying the DLL files loaded by the malware can expose the malware's dependencies and reveal its tactics. REG is a database that stores configurations and settings for the operating system, hardware, and software [43]. Malware frequently manipulates the registry to achieve persistence, disable security features, or modify system settings [44]. Examining the operations performed on the registry can provide insights into the malware's objectives and persistence mechanisms.

3.1.1 Malware visualization

In this work same as [39], the total number of API calls, DLL files loaded and malware registry keys present were 249, 730, and 5013 respectively. To improve the extraction of features and attain a more thorough understanding of malware properties, the following methodology was employed:

The aforementioned features were converted into three distinct vectors with varying dimensions (Γ_1, Γ_2, Γ_3), specifically 249, 730, and 5013. These vectors were subsequently transformed into 2D arrays, resulting in new vectors with dimensions 16×16, 28×28, and 71×71, analogous to an RGB channel image. Nonetheless, due to the differing dimensions of the three vectors, it was necessary to employ a technique known as bilinear interpolation to adjust their sizes for compatibility [39]. In particular, using $\Gamma_2{}'$ served as a reference point, with $\Gamma_1{}'$ being expanded to 28×28 and being reduced to $\Gamma_3{}'$ to 28×28. Finally, Malware visualization maps API calls, DLLs, and registry accesses to the red, green and blue (RGB) channels of an image, respectively, then combined into a single RGB image, with the intensity of each color representing the volume and specifics of the corresponding behavior by mapping these malware behaviors to color channels, visualization provides an intuitive representation of the runtime actions of a malware sample [39]. Figure 2 illustrates the complete process.

3.1.2 K-fold cross validation

To mitigate the impact of partitioning the dataset on the classifier's performance, we employed k-fold cross-validation in our study. The entire dataset is divided into k distinct subsets of similar sizes. Then, $k-1$ subsets are used as the training set to train a model, and the remaining subsets are used as the test set to evaluate the model. This process is repeated k times, and each subset is used exactly once as the test set. The most common values for k are 5 and 10. In this article, a fivefold cross-validation method, similar to the Fig. 3, is employed. This method significantly reduces bias (a model with high bias pays very little attention to the training data, resulting in an overly simplistic model and high errors in both training and test data) and variance (a model with high variance overfits the training data and fails to generalize to unseen data), as it ensures that each sample from the entire dataset has a chance to appear in both the training and test sets.

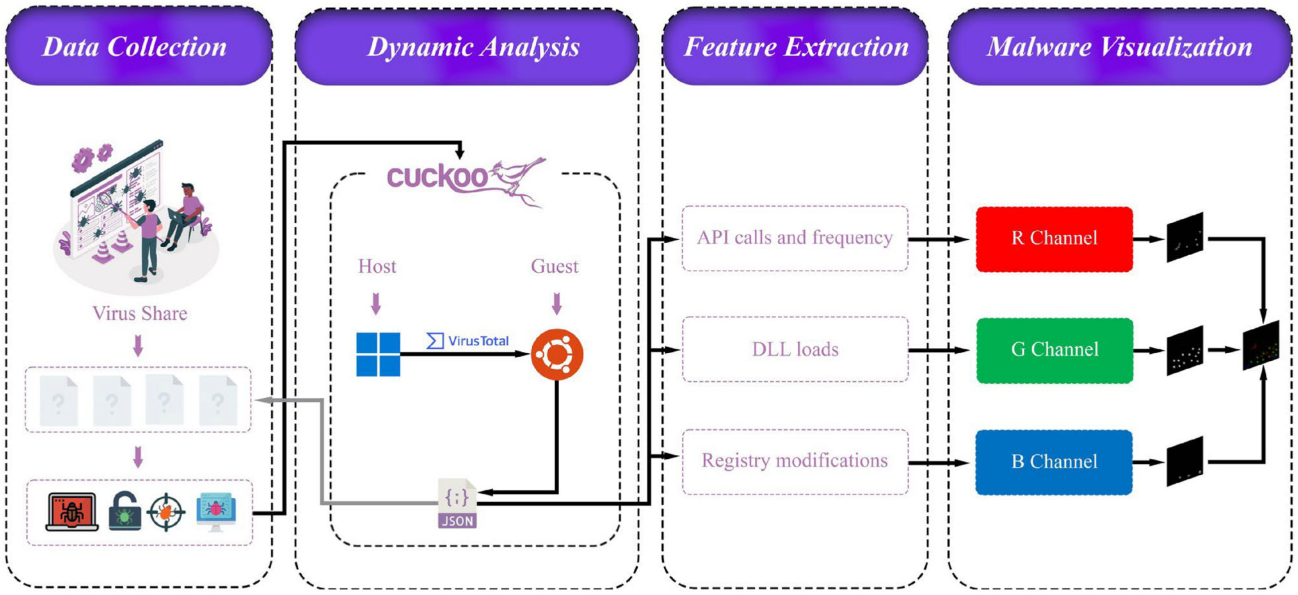


Fig. 2 Steps of generating malware dataset for classification

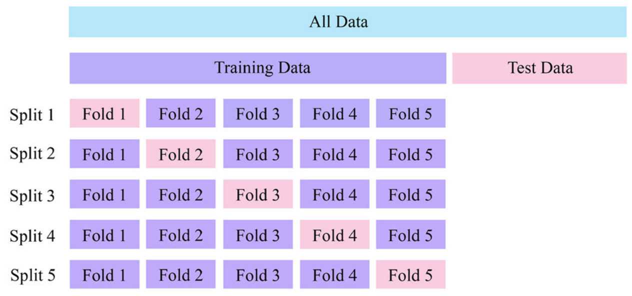


Fig. 3 Fivefold cross-validation

3.2 Efficient capsule network

In CNN, the use of a type of pooling operation called max pooling can result in some loss of information regarding object location and affine transformations. The authors of [45] solved two problems of CNN by replacing a vector capsule with a scalar CNN feature detector, a method known as routing. To resolve the second problem, they encoded the location information into the low-level capsule. In the first approach, the authors used a Capsule Network (CapsNet) for their problem because of the advantages discussed in this section, such as biomedical image segmentation [46], breaking CAPTCHA [47], text classification [48], classification of lung cancer [49], drowsiness detection [50], detecting fake news [51].

In the second approach, they improved and modified the structure of this neural network, as in RPI-CapsGAN [52], Dual-attentional spatial-aware CapsNet [53], Multi-Column CapsNet [54], Multi-Lane CapsNet [55], CapsNet with non-iterative cluster routing [56], Graph CapsNet [57], Adaptive CapsNet [58], Transformer CapsNet [59].

Although CapsNets have many advantages, they have a computational efficiency problem, which is why we used Efficient-CapsNet in our research. Efficient-CapsNet proposed in [60] by reducing parameters solves the traditional CapsNet's computational efficiency problem. The efficient-CapsNet structure is represented in Fig. 4. The Efficient-CapsNet consists of three main components: a convolutional layer, a primary capsule layer, and a self-attention mechanism for routing between capsules. To prepare for the creation of the capsule, the input (image) passes respectively through the convolution multi-layer, the ReLU

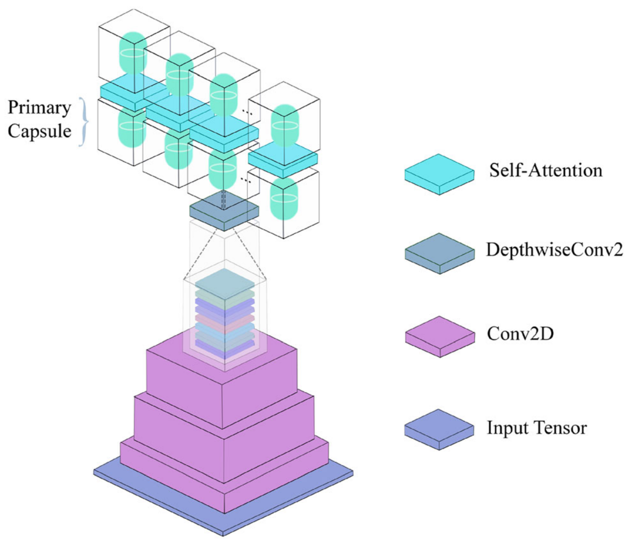


Fig. 4 Overall efficient-CapsNet structure

activation functions, and the bath normalization multilayer. In the Efficient-CapsNet, the main capsule layer passes high-dimensional features from a depth-separable convolution. A depth-separable convolution combined with a linear activation function creates a vector representation of the features. In this part, the output size is 128 1 $\times$ 1.

The primary capsule layer is crucial in decreasing parameters and enhancing the efficiency of this network. The final output of the preceding layer is reshaped according to user-defined parameters for the primary capsules, such as their number and dimensionality. The primary capsules then go through the squash function, and the resulting values are used to determine the digits capsules and classification through the fully connected capsules layer based on a self-attention routing.

Our attention in this part turns to the specific details of the Efficient-CapsNet architecture utilized in our research. During the forward pass of the network, 28 $\times$ 28 RGB images are fed into the input layer and then go through four convolution layers. The first convolution layer consists of 32 convolution kernels, each with a size of 5 $\times$ 5, a stride of 1, and valid padding, resulting in 32 feature maps of 24 $\times$ 24 dimensions. The second convolution layer has 64 kernels, each 3 $\times$ 3 in size, with a stride of 1 and valid padding, while the third convolution layer also has 64 kernels, with the same specifications. The second and third layers generate 62 feature maps of 22 $\times$ 22 and 20 $\times$ 20 dimensions, respectively. Lastly, the fourth convolution layer contains 128 convolution kernels of 3 $\times$ 3 size, a stride of 2, and valid padding, producing 128 feature maps with 9 $\times$ 9 dimensions. In order to train deep neural networks, Batch Normalization (BN) standardizes each mini-input batch to a layer. As a result, the learning process becomes more stable, and the number of training epochs necessary to create deep networks is greatly diminished. This transformation keeps the average output around zero, and the output's standard deviation is around one [61]. A layer with *dim* dimensional input $\left(\mathscr{x}^{(1)} \cdots \mathscr{x}^{(dim)}\right)$, normalize each dimension using Eq. (1):

$$\mathscr{x}^{(k)}_{\text{Normalize}} = \frac{\mathscr{x}^{k} - E\left[\mathscr{x}^{k}\right]}{\sqrt{Var\left[\mathscr{x}^{k}\right]}}. \tag{1}$$

Mazzia et al. [60] proposed a transformation reconstruction algorithm in order to prevent the features learned by each layer from being destroyed by normalization Eq. (1). Through Eqs. (1) and (2), it is possible to restore the learned distribution of features for each layer [61]:

$$\mathscr{y}^{(k)} = \mathscr{T}^{(k)} \mathscr{x}^{(k)}_{\text{Normalize}} + \mathscr{g}^{(k)}. \tag{2}$$

In Eq. (2) $\mathscr{x}^{(k)}_{\text{Normalize}}$ is the input after k-layer normalization, and $\mathscr{T}$ and $\mathscr{g}$ are learnable parameters. Due to the advantages of BN, in Efficient-CapsNet, a batch normalization layer is embedded between the mentioned four convolution layers. Depth-separable convolution is a 9 $\times$ 9 convolution kernel with 128 channels. The output shape of this convolutional layer is 128 1 $\times$ 1. We set the number and dimension, 16 and 8, respectively, for primary capsules. The multiplication of these values gives 128 (16 $\times$ 8). The construction of the primary capsules is now complete. The self-attention routing (Fig. 5) part is similar to the fully connected network. The upper capsule takes in a combination of all "prediction vectors" from the lower capsule, and the output capsules correspond to different classes of Malware. Where $U^{l}_{n^l,d^l}$ shown in Fig. 5, l represents lth capsule layer, n^l represent number of capsule layers and d^l represents the dimensions of each capsule layer. We set $U^{l}_{16,8}$ in this article. $U^{l+1}_{11,16}$, number capsule in $l+1$ capsule layer (digit capsules) is set equal to the number of classes (11 in this work). $W^{l}_{16,11,8,16}$ stands for a weight matrix with a dimension size of $(16, 11, 8, 16)$. $\widehat{U}^{l}_{16,8}$ refers to all predictions of the front capsule, while C^l represents the coupling coefficient matrix generated by the self-attention algorithm, as shown in Eqs. (3) and (4).

Where, n^l denotes the number of capsules present in layer l, while n^{l+1} indicates the number of capsules present in the next layer, which is $l+1$. d^l refers to the dimension of l-layer capsule, and $\sqrt{d^l}$ is utilized to maintain equilibrium between the coupling coefficient and logarithmic priority, ensuring a stable training process. The self-attention matrix is represented by A^l, and each capsule is associated with a self-attention matrix:

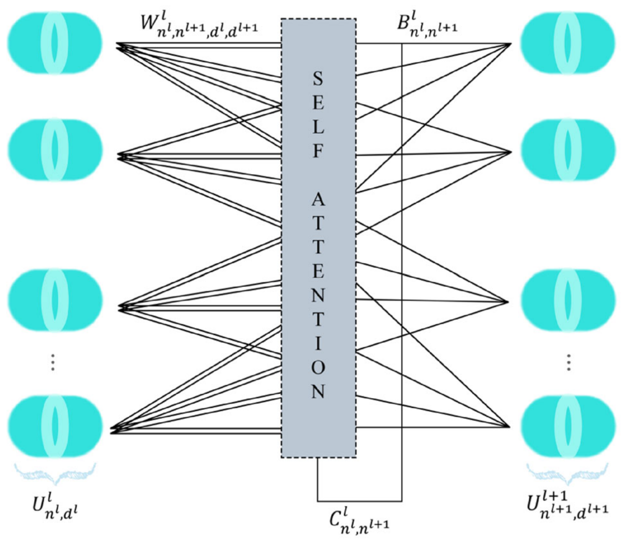


Fig. 5 Self-attention routing in an efficient-CapsNet

$$A^{l}_{(:,:,n^{l+1})} = \frac{\widehat{U}^{l}_{(:,n^{l+1},:)} \times \widehat{U}^{Tl}_{(:,n^{l+1},:)}}{\sqrt{d^{l}}} \tag{3}$$

$$C^{l}_{(:,n^{l+1})} = \frac{\exp\left(\sum_{n^l} A^{l}_{(:,n^l,n^{l+1})}\right)}{\sum_{n^{l+1}} \exp\left(\sum_{n^l} A^{l}_{(:,n^l,n^{l+1})}\right)}. \tag{4}$$

The B^l matrix holds all weight discrimination information and is predetermined. Calculating all capsules $S^{l+1}_{n^{l+1},n^l}$ in the $l+1$ layer can be accomplished using Eq. (5); this equation compresses the length of all capsule vectors in layer $l+1$ between zero and one using the squeeze function to obtain $U^{l+1}_{11,16}$.

There is a difference between the squeeze function of the network and that of CapsNet. A variant of the squeeze function is proposed to prevent the network from converging due to the vector length of zero, as in Eq. (6):

$$s^{l+1}_{n} = \widehat{U}^{Tl}_{(:,n^{l+1},:)} \times \left(C^{l}_{(:,n^{l+1})} + B^{l}_{(:,n^{l+1})}\right) \tag{5}$$

$$squash\left(s^{l}_{n}\right) = \left(1 - \frac{1}{e^{\|s^{l}_{n}\|}}\right) \frac{s^{l}_{n}}{\| s^{l}_{n} \|}. \tag{6}$$

In the end, the output of digit capsules layer provides the classification by using a final operation that reduces the shape 11×16 into 11×1. This represents a one-hot classification vector. The total loss function includes two losses, the first margin loss and the second reconstruction regularizer as Eq. (7) [62].

$$\mathcal{L}_{n^l} = T_{n^L} \max\left(0, m^{+} - \| u^{L}_{n} \|\right)^2 + \lambda(1 - T_{n^L}) \max\left(0, \| u^{L}_{n} \| - m^{-}\right)^2 \tag{7}$$

where, n represents the class. If class n exists, then T_n is equal to one. On the other hand, if n does not exist, then T_n is equal to zero. λ is a reference to a coefficient that is used for down weighting. During the initial learning phase, it is possible to prevent the shrinkage of all vectors if n is not present. Training options for the Efficient-CapsNet in our work are reported in Table 3. The performance curve of the Efficient-CapsNet for one of the API Dataset classification runs is illustrated in Fig. 6.

Table 3 Training options efficient-capsule

Training options efficient-capsule	
Training algorithm	Adam
Minimum batch size	25
Learning rate schedule	✔
m^+	0.9
m^-	0.1
λ	0.5

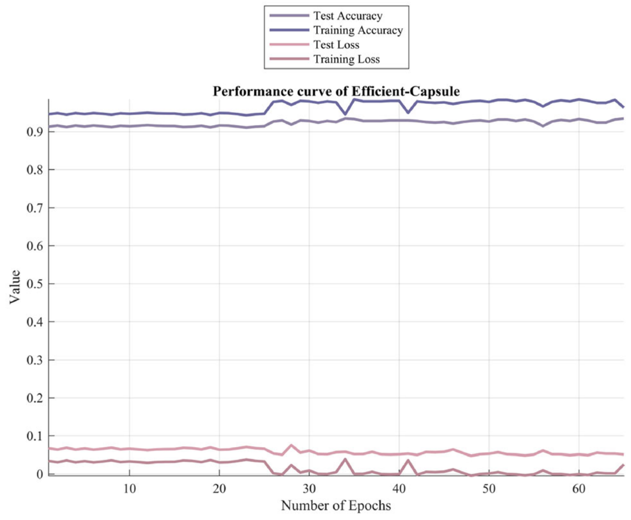


Fig. 6 Performance curve of efficient-CapNet

3.3 Residual network

The Residual Network (ResNet) was proposed by Microsoft researchers in 2015 to address the challenge of training very deep networks. One of the primary drivers for the development of ResNet was to mitigate the issue of vanishing gradient. ResNets use shortcut connections to bypass one or more layers (Fig. 7), which helps in solving this problem [63].

3.3.1 ResNet-18

The ResNet-18 architecture is a variant of the ResNet family that has been widely used in image recognition tasks. This network is a 72-layer architecture with 18 deep learning layers. Composed of 18 layers, ResNet-18 includes 17 convolutional layers that operate on the input

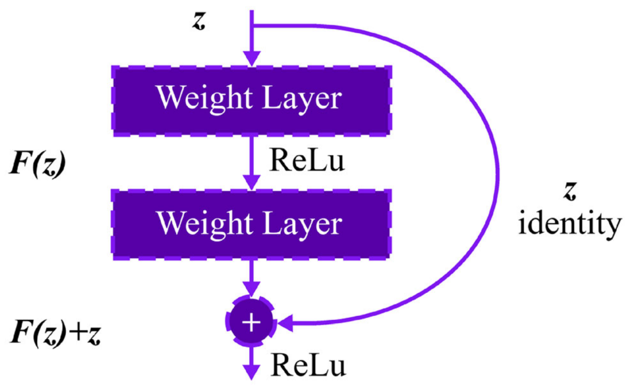


Fig. 7 Bypass layer in residual network

image and extract features at various levels of abstraction. In addition, ResNet-18 has a fully connected layer and a Softmax function for classification. The convolutional layers in ResNet-18 use a 3 × 3 kernel size, which has been shown to be effective in capturing important spatial features in images. In ResNet-18, the residual shortcut connections skip two convolutional layers. This design has been carefully optimized to balance the trade-off between model complexity and performance [63]. A visual representation of the ResNet-18 architecture can be seen in Fig. 8.

Table 4 shows color what mean in Figs. 8 and 10.

To use ResNet-18 (architecture in Table 5), we first must resize images dataset 28 × 28 RGB to 224 × 224 RGB. This model is trained with summation cross-entropy loss function and L_2 regularization. The cross-entropy loss function aims to minimize the distance between predicted and ground-truth probabilities. L_2 regularization reduce the overfitting possibility. It is defined as follows:

$$Loss = -\frac{1}{N}\left(\sum_{i=1}^{N} t_i log(p_i) + \frac{\lambda}{2}\sum \|w\|_2^2\right). \tag{8}$$

In Eq. (8) λ is the hyper-parameter scale of the regularization term to have a gentle influence on the $Loss$, $\|w\|_2^2$ is the L2-norm expression for the entire set of weights in the model, t_i is the truth label and p_i is the Softmax probability for the ith class.

To update parameters, it uses the Adam (Adaptive Moment Estimation) optimizer. This optimization technique employs an updating process for parameters akin to RMSProp. However, it distinguishes itself by incorporating a momentum term into Adam [64]. The update rule of the Adam optimizer is described as:

$$\theta_{\ell+1} = \theta_\ell - \frac{\eta m_l}{\sqrt{v_l} + \epsilon} \tag{9}$$

$$m_l = \frac{m_\ell}{1-\beta_1^\ell} \tag{10}$$

$$v_l = \frac{v_\ell}{1-\beta_2^\ell} \tag{11}$$

$$m_\ell = \beta_1 m_{\ell-1} + (1-\beta_1) \times \nabla E(\theta_\ell) \tag{12}$$

$$v_\ell = \beta_2 v_{\ell-1} + (1-\beta_2) \times (\nabla E(\theta_\ell))^2. \tag{13}$$

In this set of equations θ denotes the network's parameters, E is the loss function to optimize, m_ℓ and v_ℓ are exponential average of gradients along θ_ℓ and exponential average of squares of gradients along θ_ℓ and η is initial learning rate.

The hyperparameters β_1^ℓ and β_2^ℓ are gradient decay factor and squared gradient factor. Training options Resnet-18 in our proposed model is reported in Table 6. Additionally, Fig. 9 illustrates the training progress of ResNet-18 for one of the API Dataset classification runs.

3.3.2 ResNet-50

ResNet-50 is another variant of the ResNet family. This network is a 176-layer architecture with 50 deep learning layers. It consists of 48 convolutional layers, 1 max pooling layer, and 1 average pooling layer. The ResNet-50 consists of five convolutional layers as shown in Table 7. The Input image with 224 ×224 RGB passed through a convolutional layer (Conv1) with 64 filters, kernel size 7 × 7 and a stride of size 2 followed by a max pooling layer with stride size same previous layer. The second convolutional layer (Conv2_x) consists of three convolutional layers: (1) convolutional layer with 64 filters, kernel size 1 ×1, (2) convolutional layer with 64 filters, kernel size 3 ×3, and (3) convolutional layer with 256 filters, kernel size 1 ×1 and

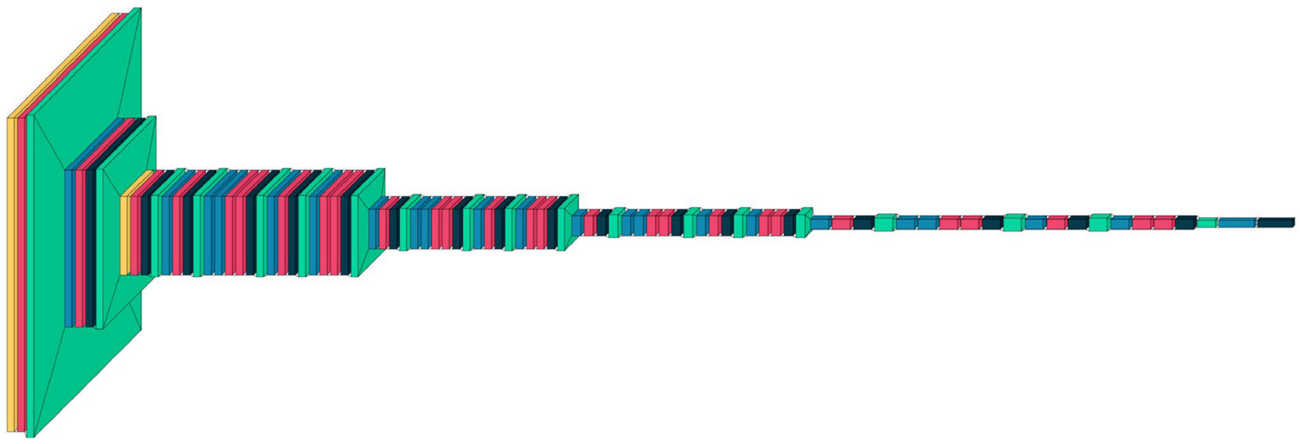


Fig. 8 ResNet-18 model architecture

Table 4 Legend of means color in Figs. 8 and 10

Name Layer	Shape
Input Layer	🟨
Batch Normalization	🟥
Zero Padding 2D	🟩
Conv2D	🟦
Activation	⬛
Max Pooling 2D	🟨
Add	🟥
Global Average Polling 2D	🟩
Dense	🟦

this layer repeated 3 times. This process is the same for the other three convolutional layers. Next, the features are passed through five convolutional layers and an average pooling layer. ResNet-50 is employed as the feature extractor, with the fully connected layer and Softmax function removed. Instead, the output from the average pooling layer is utilized [66]. A visual representation of the ResNet-50 architecture can be seen in Fig. 10.

3.4 Feature extraction

One common application of convolutional neural networks is feature extraction, where each convolutional layer extracts feature maps from an image. Features of an image, such as edge information, gradient information, are obtained after each convolutional layer [67]. In contrast to deep learning, machine learning require feature extraction, making it crucial to select a technique that not only extracts suitable features but also maintains a desirable computational cost. CNN or HOG approaches have been employed in most recent studies, such as [68–71]. In this article, ResNet-50 and HOG are employed as feature extractor.

3.4.1 Histogram of oriented gradients (HOG)

The HOG method is a highly prevalent technique for feature extraction. Its favorable computational efficiency and robustness properties have made it useful in diverse domains, including medical applications [72], facial recognition [73], and fault detection in wind turbines [74]. The implementation steps of the HOG feature extraction algorithm are as follows:

(1) To utilize this method, the images must have fixed dimensions and be in grayscale.

(2) Firstly, the images are divided into smaller sections called cells and then the horizontal gradient (G_x) and vertical gradient (G_y) are calculated for each cell. Equations (14–15)

$$G_x = I(x+1, y) - I(x-1, y) \tag{14}$$

$$G_y = I(x, y+1) - I(x, y-1). \tag{15}$$

Therefore, it is possible to calculate both the magnitude and orientation of the gradient by:

$$m(x, y) = \sqrt{(G_x)^2 + (G_y)^2} \tag{16}$$

$$\theta(x, y) = tan^{-1}\left(\frac{G_y}{Gx}\right). \tag{17}$$

In this method, a histogram is generated for each cell by binning the gradient orientations of pixels within that cell. To normalize the histograms, adjacent cells are combined into larger spatial regions known as blocks, and the information obtained is used to standardize all cells within the block. We define cell size at 24 $\times$24, block size 4 $\times$4 and number of bins is 5. Figure 11 shows random image of dataset with HOG descriptors. Finally, a feature vector with 720 columns is extracted for the datasets using the HOG feature extraction algorithm.

3.5 Neighborhood component analysis (NCA)

Neighborhood component analysis is a well-known method in feature selection or dimensionality reduction. This method can be used in both classification and regression. The findings presented in the publications show that the NCA method has chosen more valuable features in reducing dimensions than methods such as PCA, GA, and Relief, leading to improved classification accuracy compared to the mentioned methods [75, 76]. The advantages of feature selection methods include reducing computational costs and, in some cases, improving classifier performance. In this article, 1720 features were extracted, with 1000 features obtained by ReNet-50 (Sect. 3.3.2) and the rest extracted by the HOG algorithm (Sect. 3.4.1). NCA

Table 5 ResNet-18 architecture summary [65]

Layer name	Output size	ResNet-18
Conv1	64 112 × 112	7 × 7, 64, stride = 2
Conv2_x	64 56 × 56	3 × 3 max pool, stride = 2
		$\begin{bmatrix}3\times3,64\\3\times3,64\end{bmatrix}\times2$
Conv3_x	128 28 × 28	$\begin{bmatrix}3\times3,128\\3\times3,128\end{bmatrix}\times2$
Conv4_x	256 14 × 14	$\begin{bmatrix}3\times3,256\\3\times3,256\end{bmatrix}\times2$
Conv5_x	512 7 × 7	$\begin{bmatrix}3\times3,512\\3\times3,512\end{bmatrix}\times2$
Average pool	512 1 × 1	7 × 7 average pool
Fully connected	11	512 × 11 fully connections
Softmax	11	

Table 6 Training options Resnet-18 in the proposed model

Training options Resnet-18	
Training algorithm	Adam
Minimum batch size	62
Maximum epoch	8
Squared gradient decay factor	0.9990
Gradient decay factor	0.9

reduces these features to 80 features with the highest feature weights. These 80 features are used as input for the SVM classifier, discussed in the next section. Figure 12 displays the 80 features selected by NCA from 1720 extracted features of the API dataset.

3.6 Support vector machine (SVM)

Support vector machine is one of the most well-known classification methods of machine learning. Since SVM is inherently a binary classifier, when applied to datasets with more than two classes, multiple binary classifiers are used in combination to form a multiclass classifier. In this work, we have employed the one-vs-one approach to utilize the SVM multiclass classifier. Due to the requirement of correctly classifying all samples without any errors in the hard margin SVM, the soft margin SVM is often employed where a certain degree of classification error is permitted. The soft margin SVM allows for the violation of this constraint, albeit to a limited extent. However, the soft margin must minimize the number of samples that violate this constraint while maximizing the margin [77]. This optimization problem is typically formulated according to the Eq. (18):

$$\min\left(\frac{1}{2}\|\mathcal{w}\|^2+\mathcal{C}\sum_{\mathcal{i}=1}^{\mathcal{M}}\xi_{\mathcal{i}}\right)$$
$$\textit{subject to}: \begin{cases}\mathcal{y}_{\mathcal{i}}\left(\mathcal{w}.\phi(\mathcal{x}_{\mathcal{i}})+\mathcal{b}\right)\geq 1-\xi_{\mathcal{i}} \\ \xi_{\mathcal{i}}\geq 0\end{cases},\mathcal{i}=1,\ldots,\mathcal{M} \tag{18}$$

where $\mathcal{C}$ is a penalty factor, $\xi_{\mathcal{i}}$ indicates the distance between the margin and $\mathcal{x}_{\mathcal{i}}$ is the feature vector. The set Eq. (18) dual to Eq. (19):

$$\textit{maximize } L(\mathcal{a})=\sum_{\mathcal{i}=1}^{\mathcal{M}}\mathcal{a}_{\mathcal{i}}-\frac{1}{2}\sum_{\mathcal{i},\mathcal{j}=0}^{\mathcal{M}}\mathcal{a}_{\mathcal{i}}\mathcal{a}_{\mathcal{j}}\mathcal{y}_{\mathcal{i}}\mathcal{y}_{\mathcal{j}}\mathcal{k}(\mathcal{x}_{\mathcal{i}},\mathcal{x}_{\mathcal{j}})$$
$$\textit{subject to}: \sum_{\mathcal{i}=1}^{\mathcal{M}}\mathcal{a}_{\mathcal{i}}\mathcal{y}_{\mathcal{i}}=0, 0\leq\mathcal{a}_{\mathcal{i}}\leq\mathcal{C}, \mathcal{i}=1,\ldots,\mathcal{M} \tag{19}$$

where $\mathcal{k}(\mathcal{x}_{\mathcal{i}},\mathcal{x}_{\mathcal{j}})$ denotes a kernel function, and commonly used kernel functions in SVM include the linear kernel, RBF kernel, and polynomial kernel. Our work utilizes the RBF kernel function (Eq. (20)):

$$\mathcal{k}\left(\mathcal{x}_{\mathcal{i}},\mathcal{x}_{\mathcal{j}}\right)=\exp\left(-\frac{\|\mathcal{x}_{\mathcal{i}}-\mathcal{x}_{\mathcal{j}}\|^2}{2\sigma^2}\right). \tag{20}$$

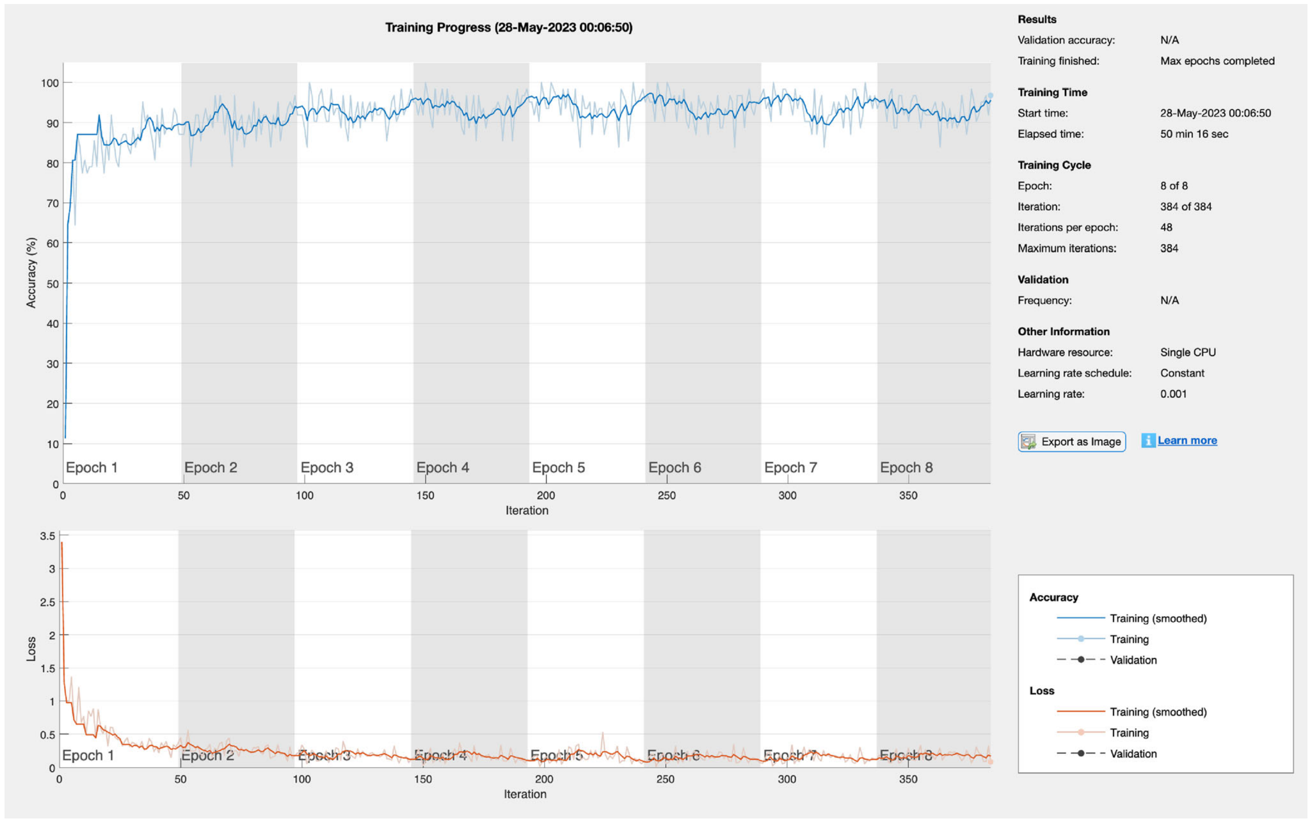
The decision function of the nonlinear SVM can be described by:

$$f(\mathcal{x})=sign\left(\sum_{\mathcal{i},\mathcal{j}=1}^{\mathcal{M}}\mathcal{a}_{\mathcal{i}}\mathcal{y}_{\mathcal{i}}\mathcal{k}\left(\mathcal{x}_{\mathcal{i}},\mathcal{x}_{\mathcal{j}}\right)+\mathcal{b}\right). \tag{21}$$

Note that the samples closest to the separating hyperplanes are those whose coefficients $\mathcal{a}_{\mathcal{i}}$ are nonzero.

3.7 Fuzzy type-III

The bases of fuzzy type-III is presented in [7]. The general structure of fuzzy type-III is illustrated in Fig. 13. Due to the ability of fuzzy type-III logic to model higher levels of uncertainty compared to fuzzy type-II, researchers have become interested in its use for practical applications. For example, fuzzy type-III logic has been utilized for flowmeter fault detection [14], aggregation neural networks for predication [12], forecasting [78] and control how improvement over fuzzy type-II because its membership functions have upper and lower uncertainties, unlike type-II. The type-III membership functions (MFs) offer multi



Results
Validation accuracy: N/A
Training finished: Max epochs completed

Training Time
Start time: 28-May-2023 00:06:50
Elapsed time: 50 min 16 sec

Training Cycle
Epoch: 8 of 8
Iteration: 384 of 384
Iterations per epoch: 48
Maximum iterations: 384

Validation
Frequency: N/A

Other Information
Hardware resource: Single CPU
Learning rate schedule: Constant
Learning rate: 0.001

Fig. 9 Training progress of ResNet-18 (API dataset)

Table 7 ResNet-50 architecture summary [65]

Layer name	Output size	ResNet-50
Conv1	112 × 112	7 × 7, 64, stride = 2
Conv2_x	56 × 56	3 × 3 max pool, stride = 2
		$\begin{bmatrix} 1\times1,64 \\ 3\times3,64 \\ 1\times1,256 \end{bmatrix}\times 3$
Conv3_x	28 × 28	$\begin{bmatrix} 1\times1,128 \\ 3\times3,128 \\ 1\times1,512 \end{bmatrix}\times 4$
Conv4_x	14 × 14	$\begin{bmatrix} 1\times1,256 \\ 3\times3,256 \\ 1\times1,1024 \end{bmatrix}\times 6$
Conv5_x	7 × 7	$\begin{bmatrix} 1\times1,512 \\ 3\times3,512 \\ 1\times1,2048 \end{bmatrix}\times 3$
Average pool	1 × 1	2D average pooling layer
Fully connected	1000	2048 × 1000 fully connections
Softmax	1000	

degree of freedom due to their uncertain secondary memberships and footprint of uncertainty (FOU). Type-III MFs can be formulated as Eq. (22):

$$\sum_{\mathcal{X}\in X}\sum_{\mathcal{P}\in\mathcal{J}_{\mathcal{X}}}\frac{\left[\frac{1}{\mathcal{S}_{\tilde{\mathcal{M}}}(\mathcal{X},\mathcal{P})}\right]}{(\mathcal{X},\mathcal{P})},\ \mathcal{J}_{\mathcal{X}}\subseteq[0,1]. \tag{22}$$

In Eq. (22) $\sum\sum$ represent s-norm and $\mathcal{S}_{\tilde{\mathcal{M}}}(\mathcal{X},\mathcal{P})$ is a type-II MF.

As shown in Fig. 13 inputs defined by $\mathcal{X}_i, i = 1,\ldots,n$ and for each input $\mathcal{Q}_i$ MFs is considered. The $\widetilde{\mathcal{M}}_{\imath}^{\jmath}$ denoted $\jmath$-th MF in i-th input. Subsequent to this, firing degree are computed for $\widetilde{\mathcal{M}}_{\imath}^{\jmath}$. Four primary memberships at each horizontal slice α_k (secondary membership) resulting two bounds for right side and two bounds for left side. $\overline{\mathcal{P}}_{\imath^{\jmath},r,\alpha_k}$ are upper bounds of right side, $\underline{\mathcal{P}}_{\imath^{\jmath},r,\alpha_k}$ is lower bounds of right side, $\overline{\mathcal{P}}_{\imath^{\jmath},l,\alpha_k}$ is upper bounds of left side and $\underline{\mathcal{P}}_{\imath^{\jmath},l,\alpha_k}$ is lower bounds of left side (Eqs. (23–26)). Figure 14 is illustrated MF of the type-III fuzzy system.

In Eqs. (23–26) $c_{\widetilde{\mathcal{M}}_{\imath}^{\jmath}}$, $\overline{\vartheta}_{\widetilde{\mathcal{M}}_{\imath}^{\jmath}}$ and $\underline{\vartheta}_{\widetilde{\mathcal{M}}_{\imath}^{\jmath}}$ is center of $\jmath$-th MF in $\imath$-th input and are the widths of left and right side $\jmath$-th MF in $\imath$-th input, respectively. α_k $k = 1,\ldots,K$ represent the value of the horizontal slice. Δ_r and Δ_l represent the width of the MFs. Considering the memberships of the UB and LB of $\widetilde{\mathcal{M}}_{e}^{\cdot}$,the rule firings formulated as Eqs. (27–30):

$$\overline{z}^{h}_{r,\alpha_k} = \overline{s}_{\widetilde{\mathcal{M}}_1^{h_1}|r,\alpha_k}(\mathcal{X}_1)\overline{s}_{\widetilde{\mathcal{M}}_2^{h_2}|r,\alpha_k}(\mathcal{X}_2)\ldots\overline{s}_{\widetilde{\mathcal{M}}_n^{h_n}|r,\alpha_k}(\mathcal{X}_n) \tag{27}$$

$$\underline{z}^{h}_{r,\alpha_k} = \underline{s}_{\widetilde{\mathcal{M}}_1^{h_1}|r,\alpha_k}(\mathcal{X}_1)\underline{s}_{\widetilde{\mathcal{M}}_2^{h_2}|r,\alpha_k}(\mathcal{X}_2)\ldots\underline{s}_{\widetilde{\mathcal{M}}_n^{h_n}|r,\alpha_k}(\mathcal{X}_n) \tag{28}$$

$$\overline{z}^{h}_{l,\alpha_k} = \overline{s}_{\widetilde{\mathcal{M}}_1^{h_1}|l,\alpha_k}(\mathcal{X}_1)\overline{s}_{\widetilde{\mathcal{M}}_2^{h_2}|l,\alpha_k}(\mathcal{X}_2)\ldots\overline{s}_{\widetilde{\mathcal{M}}_n^{h_n}|l,\alpha_k}(\mathcal{X}_n) \tag{29}$$

$$\underline{z}^{h}_{l,\alpha_k} = \underline{s}_{\widetilde{\mathcal{M}}_1^{h_1}|l,\alpha_k}(\mathcal{X}_1)\underline{s}_{\widetilde{\mathcal{M}}_2^{h_2}|l,\alpha_k}(\mathcal{X}_2)\ldots\underline{s}_{\widetilde{\mathcal{M}}_n^{h_n}|l,\alpha_k}(\mathcal{X}_n). \tag{30}$$

$$\overline{\mathcal{P}}_{e^{\cdot},r,\alpha_k} = \begin{cases} 1-\left(\dfrac{\left|\mathcal{X}_{e^{\cdot}} - c_{\widetilde{\mathcal{M}}_{e}^{\cdot}}\right|}{\underline{\vartheta}_{\widetilde{\mathcal{M}}_{e}^{\cdot}}}\right)^{\Delta_r(1-\alpha_k)+1}, & \text{if } c_{\widetilde{\mathcal{M}}_{e}^{\cdot}} - \underline{\vartheta}_{\widetilde{\mathcal{M}}_{e}^{\cdot}} < \mathcal{X}_{e^{\cdot}} \le c_{\widetilde{\mathcal{M}}_{e}^{\cdot}} \\ 1-\left(\dfrac{\left|\mathcal{X}_{e^{\cdot}} - c_{\widetilde{\mathcal{M}}_{e}^{\cdot}}\right|}{\overline{\vartheta}_{\widetilde{\mathcal{M}}_{e}^{\cdot}}}\right)^{\Delta_r(1-\alpha_k)+1}, & \text{if } c_{\widetilde{\mathcal{M}}_{e}^{\cdot}} < \mathcal{X}_{e^{\cdot}} \le c_{\widetilde{\mathcal{M}}_{e}^{\cdot}} + \overline{\vartheta}_{\widetilde{\mathcal{M}}_{e}^{\cdot}} \\ 0, & \text{if } \mathcal{X}_{e^{\cdot}} > c_{\widetilde{\mathcal{M}}_{e}^{\cdot}} + \overline{\vartheta}_{\widetilde{\mathcal{M}}_{e}^{\cdot}} or \mathcal{X}_{e^{\cdot}} \le c_{\widetilde{\mathcal{M}}_{e}^{\cdot}} - \underline{\vartheta}_{\widetilde{\mathcal{M}}_{e}^{\cdot}} \end{cases} \tag{23}$$

$$\underline{\mathcal{P}}_{e^{\cdot},r,\alpha_k} = \begin{cases} 1-\left(\dfrac{\left|\mathcal{X}_{e^{\cdot}} - c_{\widetilde{\mathcal{M}}_{e}^{\cdot}}\right|}{\underline{\vartheta}_{\widetilde{\mathcal{M}}_{e}^{\cdot}}}\right)^{\frac{1}{\Delta_r(1-\alpha_k)+1}}, & \text{if } c_{\widetilde{\mathcal{M}}_{e}^{\cdot}} - \underline{\vartheta}_{\widetilde{\mathcal{M}}_{e}^{\cdot}} < \mathcal{X}_{e^{\cdot}} \le c_{\widetilde{\mathcal{M}}_{e}^{\cdot}} \\ 1-\left(\dfrac{\left|\mathcal{X}_{e^{\cdot}} - c_{\widetilde{\mathcal{M}}_{e}^{\cdot}}\right|}{\overline{\vartheta}_{\widetilde{\mathcal{M}}_{e}^{\cdot}}}\right)^{\frac{1}{\Delta_r(1-\alpha_k)+1}}, & \text{if } c_{\widetilde{\mathcal{M}}_{e}^{\cdot}} < \mathcal{X}_{e^{\cdot}} \le c_{\widetilde{\mathcal{M}}_{e}^{\cdot}} + \overline{\vartheta}_{\widetilde{\mathcal{M}}_{e}^{\cdot}} \\ 0, & \text{if } \mathcal{X}_{e^{\cdot}} > c_{\widetilde{\mathcal{M}}_{e}^{\cdot}} + \overline{\vartheta}_{\widetilde{\mathcal{M}}_{e}^{\cdot}} or \mathcal{X}_{e^{\cdot}} \le c_{\widetilde{\mathcal{M}}_{e}^{\cdot}} - \underline{\vartheta}_{\widetilde{\mathcal{M}}_{e}^{\cdot}} \end{cases} \tag{24}$$

$$\overline{\mathcal{P}}_{e^{\cdot},l,\alpha_k} = \begin{cases} 1-\left(\dfrac{\left|\mathcal{X}_{e^{\cdot}} - c_{\widetilde{\mathcal{M}}_{e}^{\cdot}}\right|}{\underline{\vartheta}_{\widetilde{\mathcal{M}}_{e}^{\cdot}}}\right)^{\Delta_l(1-\alpha_k)+1}, & \text{if } c_{\widetilde{\mathcal{M}}_{e}^{\cdot}} - \underline{\vartheta}_{\widetilde{\mathcal{M}}_{e}^{\cdot}} < \mathcal{X}_{e^{\cdot}} \le c_{\widetilde{\mathcal{M}}_{e}^{\cdot}} \\ 1-\left(\dfrac{\left|\mathcal{X}_{e^{\cdot}} - c_{\widetilde{\mathcal{M}}_{e}^{\cdot}}\right|}{\overline{\vartheta}_{\widetilde{\mathcal{M}}_{e}^{\cdot}}}\right)^{\Delta_l(1-\alpha_k)+1}, & \text{if } c_{\widetilde{\mathcal{M}}_{e}^{\cdot}} < \mathcal{X}_{e^{\cdot}} \le c_{\widetilde{\mathcal{M}}_{e}^{\cdot}} + \overline{\vartheta}_{\widetilde{\mathcal{M}}_{e}^{\cdot}} \\ 0, & \text{if } \mathcal{X}_{e^{\cdot}} > c_{\widetilde{\mathcal{M}}_{e}^{\cdot}} + \overline{\vartheta}_{\widetilde{\mathcal{M}}_{e}^{\cdot}} or \mathcal{X}_{e^{\cdot}} \le c_{\widetilde{\mathcal{M}}_{e}^{\cdot}} - \underline{\vartheta}_{\widetilde{\mathcal{M}}_{e}^{\cdot}} \end{cases} \tag{25}$$

$$\underline{\mathcal{P}}_{e^{\cdot},l,\alpha_k} = \begin{cases} 1-\left(\dfrac{\left|\mathcal{X}_{e^{\cdot}} - c_{\widetilde{\mathcal{M}}_{e}^{\cdot}}\right|}{\underline{\vartheta}_{\widetilde{\mathcal{M}}_{e}^{\cdot}}}\right)^{\frac{1}{\Delta_l(1-\alpha_k)+1}}, & \text{if } c_{\widetilde{\mathcal{M}}_{e}^{\cdot}} - \underline{\vartheta}_{\widetilde{\mathcal{M}}_{e}^{\cdot}} < \mathcal{X}_{e^{\cdot}} \le c_{\widetilde{\mathcal{M}}_{e}^{\cdot}} \\ 1-\left(\dfrac{\left|\mathcal{X}_{e^{\cdot}} - c_{\widetilde{\mathcal{M}}_{e}^{\cdot}}\right|}{\overline{\vartheta}_{\widetilde{\mathcal{M}}_{e}^{\cdot}}}\right)^{\frac{1}{\Delta_l(1-\alpha_k)+1}}, & \text{if } c_{\widetilde{\mathcal{M}}_{e}^{\cdot}} < \mathcal{X}_{e^{\cdot}} \le c_{\widetilde{\mathcal{M}}_{e}^{\cdot}} + \overline{\vartheta}_{\widetilde{\mathcal{M}}_{e}^{\cdot}} \\ 0, & \text{if } \mathcal{X}_{e^{\cdot}} > c_{\widetilde{\mathcal{M}}_{e}^{\cdot}} + \overline{\vartheta}_{\widetilde{\mathcal{M}}_{e}^{\cdot}} or \mathcal{X}_{e^{\cdot}} \le c_{\widetilde{\mathcal{M}}_{e}^{\cdot}} - \underline{\vartheta}_{\widetilde{\mathcal{M}}_{e}^{\cdot}} \end{cases} \tag{26}$$

Fig. 10 ResNet-50 model architecture

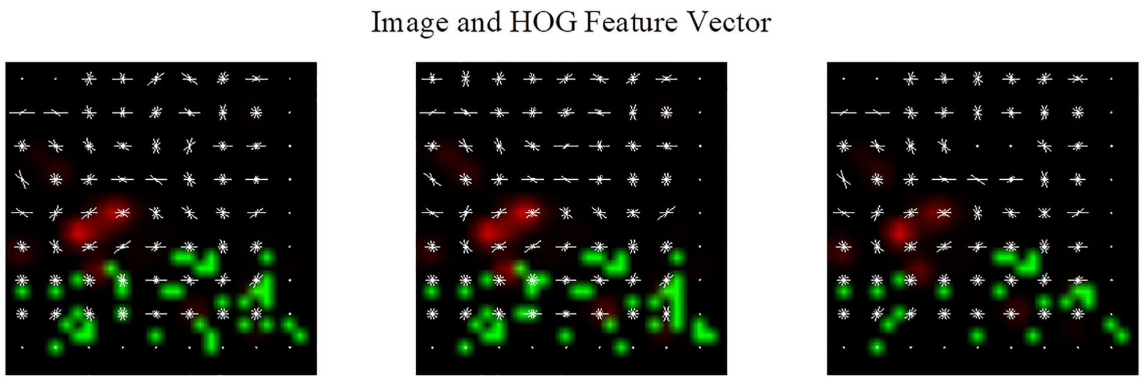


Fig. 11 Random image of dataset with HOG descriptors

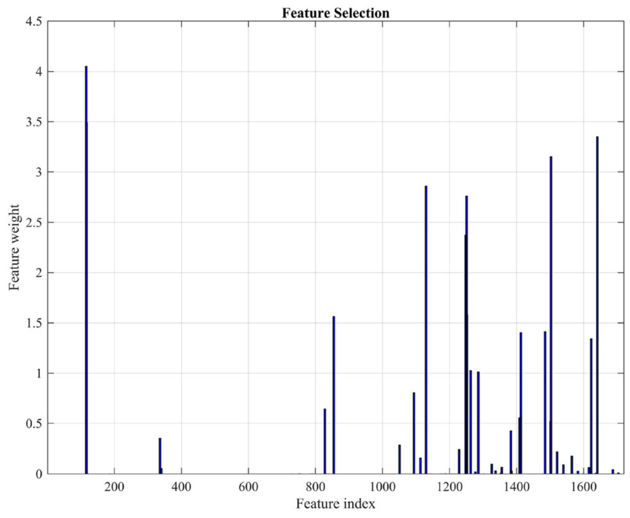


Fig. 12 8o features selected by NCA (API dataset)

In Eqs. (27–30) $k = 1, \ldots, K$ $\widetilde{\mathcal{M}}_{\ell}$ denote the h_{ℓ}-th MF for input x_{ℓ}. The h-th rule is given as:

$$Ifx_1 is \widetilde{\mathcal{M}}_1^{h_1}, \ldots, and \ldots x_n is \widetilde{\mathcal{M}}_n^{h_n}, Then y_h \in \left[\left[\underline{\Psi}_{\mathrm{l,h}}, \underline{\Psi}_{\mathrm{r,h}}\right]\left[\overline{\Psi}_{\mathrm{l,h}}, \overline{\Psi}_{\mathrm{r,h}}\right]\right]. \tag{31}$$

In Eq. (31) $\left[\underline{\Psi}_{\mathrm{l,h}}, \underline{\Psi}_{\mathrm{r,h}}\right]$ denote the lower bounds fuzzy number and $\left[\overline{\Psi}_{\mathrm{l,h}}, \overline{\Psi}_{\mathrm{r,h}}\right]$ are upper bounds fuzzy number. At last, output's fuzzy type-III is defined as Eq. (32):

$$y = \frac{\sum_{k=1}^{K}\left(\overline{\phi}_k + \underline{\phi}_k\right)\alpha_k}{\sum_{k=1}^{K}\alpha_k} \tag{32}$$

$\overline{\phi}_k$ and $\underline{\phi}_k$ defined as Eq. (32) and Eq. (33):

$$\overline{\phi}_k = \frac{\sum_{h=1}^{N}\left(\overline{z}_{r,\alpha_k}^{h}\overline{\Psi}_{\mathrm{r,k}}^{h} + \underline{z}_{l,\alpha_k}^{h}\underline{\Psi}_{\mathrm{l,k}}^{h}\right)}{\sum_{h=1}^{N}\overline{z}_{r,\alpha_k}^{h} + \underline{z}_{l,\alpha_k}^{h}} \tag{33}$$

$$\underline{\phi}_k = \frac{\sum_{h=1}^{N}\left(\underline{z}_{r,\alpha_k}^{h}\underline{\Psi}_{\mathrm{r,k}}^{h} + \overline{z}_{l,\alpha_k}^{h}\overline{\Psi}_{\mathrm{l,k}}^{h}\right)}{\sum_{h=1}^{N}\underline{z}_{r,\alpha_k}^{h} + \overline{z}_{l,\alpha_k}^{h}}. \tag{34}$$

The inputs to FIS are the outputs from three classifiers: Efficient-Capsule, ResNet-18, and SVM. The output of the FIS is the final decision regarding malware classification. In other words, FIS-III makes a choice based on its inputs to which class the data (image) belongs. This means FIS-III decides which output of classifiers is through to output, or when all classifiers are misclassified, it can be modified to achieve the best accuracy. For tuning the parameters of the FIS, meta-heuristic optimization algorithms are used instead of derivative-based methods. The meta-heuristic optimization algorithm utilized for tuning the rule parameters of the FIS-III ($\overline{\Psi}_{\mathrm{r,k}}^{h}$, $\underline{\Psi}_{\mathrm{r,k}}^{h}$,$\overline{\Psi}_{\mathrm{l,k}}^{h}$,$\underline{\Psi}_{\mathrm{l,k}}^{h}$) is ICGO (Sect. 3.8.1).

Since, the outputs of the classifiers can be integer numbers ranging from 0 to 10, the total number of rules considered, based on the 3 classifiers used, is the

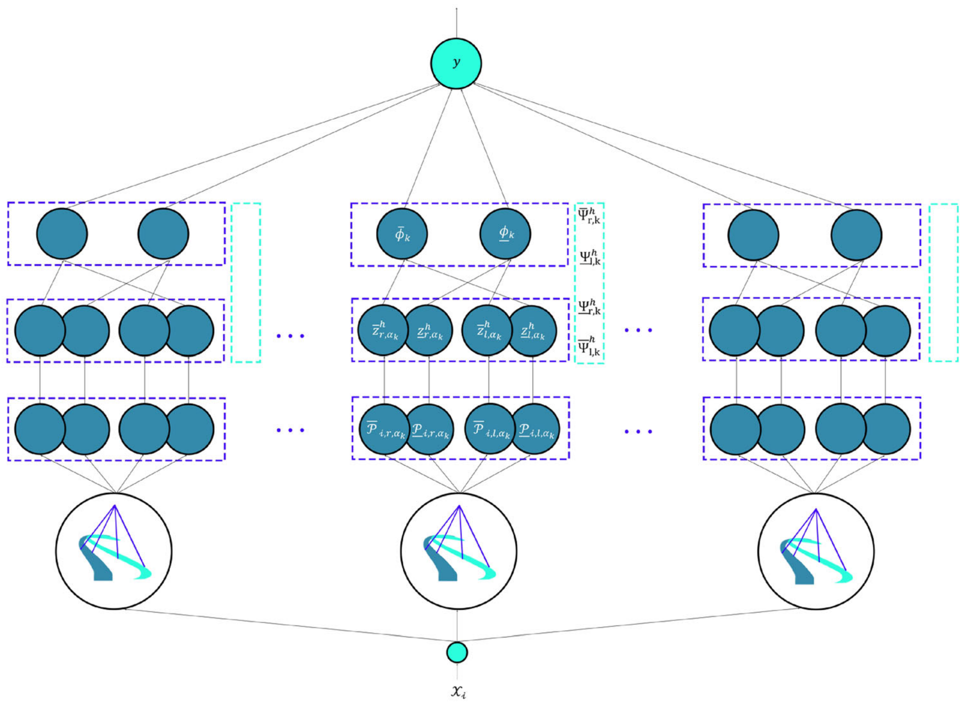


Fig. 13 The structure scheme of the fuzzy type-III

combination $\binom{11}{3}$ or 165 rules. Now, intelligently and by utilizing the associative memory property of fuzzy systems, these 165 rules can be reduced to just 43 rules. The intelligent selection of rules refers to first identifying the unique combinations of train data, and then choosing the output decisions to maximize accuracy. We present an example implementation to provide a simplified illustration of this intelligent rule selection approach. Suppose our fuzzy system gets an input of $[1, 9, 1]$. In train data, we have combination where the target is 1 for this input, and combination where the target is 9. We will look how many times each target occurs for this specific input pattern. If there are more cases where the target is 9 when the input is $[1, 9, 1]$, then we will set the fuzzy system output to be 9 for this rule. If the train data contains an equal number of target 1 and target 9 for input $[1, 9, 1]$, then the rule consequent will be randomly chosen as either 1 or 9 with equal probability. Given the described structure for the FIS-III with 43 rules, this result in $43 \times 4 = 172$ rule parameters. The rule parameters must be tuned by ICGO. The membership all three inputs are considered similar to each other and is reported in Table 8. The MFs of the FIS-III in Fig. 15 it has been shown.

Remark 1 It should be noted that the proposed model results in increased classification accuracy only if the neural networks used as base classifiers are not overfitted.

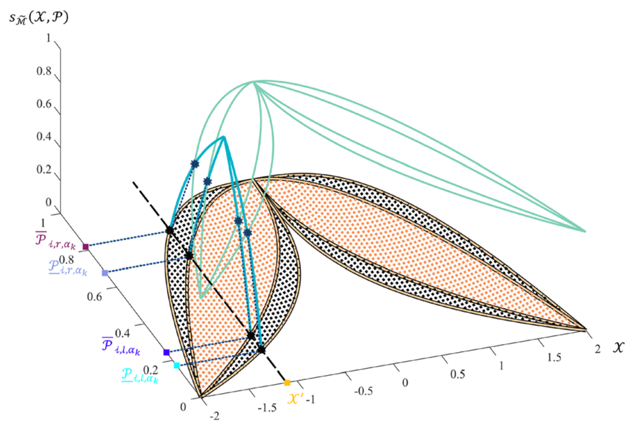


Fig. 14 Type-III MF

Remark 2 Another limitation of our proposed method is that as the number of classes increases, the number of membership functions also increases.

3.8 Optimization

Optimization is fundamental in the realm of machine learning. It revolves around the meticulous adjustment of model parameters to ensure peak performance. This process is particularly crucial because even slight parameter changes can significantly impact a model's overall accuracy and efficiency. Traditional optimization techniques might not always be the best fit in the context of fuzzy systems and neural networks, which are inherently complex and nonlinear. Meta-heuristic interpolation algorithms have emerged as a preferred choice in such scenarios. Unlike derivative-based methods, which rely on gradient information and can sometimes get stuck in local optima, meta-heuristic algorithms explore the solution space more broadly, making them more adept at finding global optima. This adaptability and flexibility make them especially effective for the intricate landscapes of fuzzy systems and neural networks. In intrusion detection and malware classification research, metaheuristic algorithms have also been utilized, such as [79–85].

3.8.1 Improved chaos game optimization (ICGO)

ICGO is an enhanced version of the Chaos Game Optimization (CGO) algorithm, with improvements made to the mutation phase of the traditional CGO method. Talatahari et al. [86] introduced this algorithm in 2020. The foundational mathematics of CGO is rooted in the self-resemblance properties of fractals in chaos theory and the fundamental principles of creating the Sierpinski triangle [87]. The Sierpinski triangle is crafted by segmenting each equilateral triangle into three smaller triangles, each having half the edge length of the original. When the count of initial fractal vertices rises to N, a Sierpinski triangle of N − 1 dimensions can be established, as illustrated in Fig. 16. The reason for choosing the CGO algorithm in this study is that its performance has been evaluated by 228 BFs. Additionally, 11 EDPs were assessed, and three comparative analyses were conducted to assess the outcomes of the CGO algorithm. It is rare to find an algorithm that has been evaluated with this number of tests, and the evaluation results indicate that the CGO algorithm is capable of solving various optimization problems. Therefore, there is hope that it can also demonstrate good performance for the problem addressed in this article. Both CGO and ICGO exhibit a computational complexity of $\mathcal{O}(\mathcal{N}m(1+4\mathcal{T}))$.

Firstly, an initialization procedure is configured by determining the initial positions of the solution candidates from the following equations:

$$\mathcal{X}=\begin{bmatrix}\mathcal{X}_1\\ \mathcal{X}_2\\ \vdots\\ \mathcal{X}_i\\ \vdots\\ \mathcal{X}_n\end{bmatrix}=\begin{bmatrix}x_1^1 & x_1^2 & \cdots & x_1^j & \cdots & x_1^d\\ x_2^1 & x_2^2 & \cdots & x_2^j & \cdots & x_2^d\\ \vdots & \vdots & \cdots & \vdots & \cdots & \vdots\\ x_i^1 & x_i^2 & \cdots & x_i^j & \cdots & x_i^d\\ x_n^1 & x_n^2 & \cdots & x_n^j & \cdots & x_n^d\end{bmatrix}, i = 1,2,\ldots,n, j = 1,2,\ldots,d \tag{35}$$

$$x_i^j(0) = x_{i,min}^j + .(x_{i,max}^j - x_{i,min}^j) \tag{36}$$

where d denotes the dimension of problem, n number of the solution candidates, random number in the range [0,1], $x_{i,min}^j$ and $x_{i,max}^j$ the lower and upper bounds, j specifies the decision variable, and i specifies the solution number. In this Algorithm 3 seeds and a dice are utilized for creating new seeds. Seeds can be calculated as follows:

$$Seed_i^1 = \mathcal{X}_i + \alpha_i \times (\beta_i \times \mathcal{GB} - \gamma_i \times \mathcal{MG}_i), i = 1,2,\ldots,n \tag{37}$$

$$Seed_i^2 = \mathcal{GB} + \alpha_i \times (\beta_i \times \mathcal{X}_i - \gamma_i \times \mathcal{MG}_i), i = 1,2,\ldots,n \tag{38}$$

$$Seed_i^3 = \mathcal{MG}_i + \alpha_i \times (\beta_i \times \mathcal{X}_i - \gamma_i \times \mathcal{GB}), i = 1,2,\ldots,n. \tag{39}$$

In Eqs. (37–39) $\mathcal{GB}$ is the global best, α_i is the movement limitation factor, β_i and γ_i represents vectors randomly created by numbers in range of [0, 1] and $\mathcal{MG}_i$ is the ith candidate's mean group ($\mathcal{X}_i$).

In conventional CGO algorithm, the fourth seed is considered a dice or mutation operator. The update of position for this seed is accomplished by making random modifications to the decision variables that are randomly selected. This seed formulated as Eq. (40):

$$Seed_i^4 = \mathcal{X}_i\left(x_i^k = x_i^k + \mathcal{R}\right), k = 1,2,\ldots,d, i = 1,2,\ldots,n. \tag{40}$$

In Eq. (40) $\mathcal{R}$ is a vector with a random number in range of [0, 1].

In this article for improve CGO algorithm replace simple mutation in convectional CGO algorithm to wavelet mutation. The $Seed_i^4$ in ICGO defined as Eq. (41):

$$Seed_i^4 = \begin{cases} GB + \sigma\left(\mathcal{X}_{i,max} - \mathcal{GB}\right), & if\ o < 0.5\\ GB + \sigma\left(\mathcal{GB} - \mathcal{X}_{i,min}\right), & if\ o \geq 0.5.\end{cases} \tag{41}$$

In Eq. (41), $\mathcal{X}_{i,max}$ and $\mathcal{X}_{i,min}$ are the lower and upper bounds, o is a random number and σ is defined as Eq. (42):

$$\sigma = \frac{1}{\sqrt{\alpha}}\psi\left(\frac{\varphi}{\alpha}\right). \tag{42}$$

In Eq. (42), $\psi\left(\frac{\varphi}{\alpha}\right) = e^{-\frac{\left(\frac{\varphi}{\alpha}\right)^2}{2}}.\cos\left(\frac{5\varphi}{\alpha}\right)$ is the Morlet wavelet function (Fig. 17) and $\alpha = s.\left(\frac{1}{s}\right)^{\left(1-\frac{\mathcal{T}}{\mathcal{T}_{max}}\right)}s$ is a random integer number, The current iteration is denoted by $\mathcal{T}$, and $\mathcal{T}_{max}$ represents the maximum number of iterations. The pseudocode is provided in Table 9.

The cost function used to adjust the rule parameters is described in Sect. 3.7 and is given by Eq. (47):

$$\text{Cost} = 1 - \sum_{\jmath=1}^{\ell} \frac{\mathcal{TP}_{\jmath} + \mathcal{TN}_{\jmath}}{\mathcal{TP}_{\jmath} + \mathcal{TN}_{\jmath} + \mathcal{FP}_{\jmath} + \mathcal{FN}_{\jmath}} \tag{47}$$

where, $\mathcal{TP}_{\jmath}$ is the number of samples correctly put in the $\jmath$th class, $\mathcal{FP}_{\jmath}$ is the number of samples incorrectly put in the ith class and $\mathcal{FN}_{\jmath}$ is the number of samples that belonged in the $\jmath$th class but were put in other classes. Moreover,ℓ is the number of the total labels. In all separate runs in order to adjust rule parameters FIS-III the population size is 30 and the maximum iteration is 200 of repetitions considered. Figure 18 depicts the average convergence and 0.5 standard division of the ICGO optimization algorithm across multiple runs (5 separate runs per dataset) for adjusting the rule parameters of the fuzzy system.

3.8.2 Evolution of CEC 2019 BFs, CEC 2017 BFs, and EDPs

We evaluate the effectiveness of ICGO in addressing challenges posed by CEC 2019 BFs, CEC 2017 BFs, and five EDPs scenarios in 30 independent runs and 1000 iterations. Our study juxtaposes the performance of ICGO against twelve established metaheuristic algorithms CGO, LSHADE-EnSin, AOA, SSA, SCA, TLBO, GOA, PSO, WOA to gauge its efficacy in achieving optimal outcomes. The modification of control parameters is described according to the specifications provided in Table 10. The ICGO is applied to resolve these issues, with a total of 30,000 evaluations conducted. The population size for both CGO and ICGO is kept constant at 15 members. The TLBO's population size is maintained at a constant of 30 members. The population size of other algorithms is maintained at a constant of 60 members.

3.8.2.1 CEC 2017 BFs The ability of the ICGO to tackle optimization challenges has been questioned when tested on the latest functions from the CEC 2017 test suite. This suite comprises thirty BFs categorized into four groups: three UMBFs (F1 to F3), seven MMBFs (F4 to F10), ten hybrid functions (F11 to F20), and ten composition functions (F21 to F30). We exclude the use of the F2 test function from the CEC 2017 set due to its unpredictable behavior, a decision shared by other researchers in their respective papers. Comprehensive details and information regarding these BFs can be found in [88]. The experiments are conducted across various dimensions of BFs, specifically 10, 30, 50, and 100. Results from optimizing BFs within the CEC 2017 set using the ICGO and competitor algorithms are presented in Table B.1–B.4. Furthermore, boxplots and convergence curves of algorithms illustrating the performance of the ICGO and competitor algorithms across the CEC 2017 BFs for different dimensionalities are depicted in Fig. B.1–B.8. The ICGO algorithm obtained the best solution among all algorithms for all 116 CEC 2017 BFs.

Table 8 Specifications of type-III fuzzy system

Specifications FIS-III	
Centers of MFs	$c_{\tilde{\mathcal{M}}_{1,2,3}^{\jmath}=\jmath-1, \jmath=1,\ldots,11}$
Width of upper MFs	$\Delta_r = 3$
Width of lower MFs	$\Delta_l = 2$
Right distance	$\overline{\vartheta}_{\tilde{\mathcal{M}}_{1,2,3}^{\jmath}} = 1, \jmath = 1, \ldots, 11$
Left distance	$\vartheta_{-\tilde{\mathcal{M}}_{1,2,3}^{\jmath}} = 1, \jmath = 1, \ldots, 11$
Alpha-cut	$\alpha_k = 0.5$

3.8.2.2 CEC 2019 BFs The CEC 2019 BFs comprise ten intricate functions outlined in [89]. F1 and F10 functions, part of the CEC 2019 test suite, are tailored for single-objective real parameter optimization, targeting the discovery of globally optimal solutions. These functions serve as valuable tools for evaluating the efficacy of algorithms in conducting comprehensive searches for optimal solutions. ICGO algorithm achieved the best results in functions F1–F3, F5, and F7–F10 compared to the competing algorithms. In function F6, ICGO outperformed in all criteria except the Best criterion (see Table B.5). Boxplots and convergence curves of algorithms illustrating the performance of the ICGO and competitor algorithms across the CEC 2017 BFs are depicted in Fig. B.9 and Fig. B.10.

3.8.2.3 EDPs The statistical results obtained through different methodologies are presented in Tables B.6. Furthermore, Fig. B.11 and Fig. B.12 display boxplots and convergence diagrams illustrating the algorithms' performance. PV Design (Fig. D.1), TCS Design (Fig. D.2), WB Design (Fig. D.3), TBT Design (Fig. D.4), CSI Design (Fig. D.5) [90] The ICGO achieved the lowest values compared to other algorithms.

3.8.3 Optimization algorithms' statistical evaluation

To meticulously assess the effectiveness of the ICGO, we perform an extensive statistical analysis, comparing it against the examined algorithms. The Wilcoxon nonparametric signed-rank test examines whether there's a substantial contrast between pairs of data (Table C.1). It evaluates the magnitudes of differences (disregarding their direction) by assigning ranks and computing a statistic based on these ranks. This figure aids in discerning whether distinctions are probably attributable to random variation

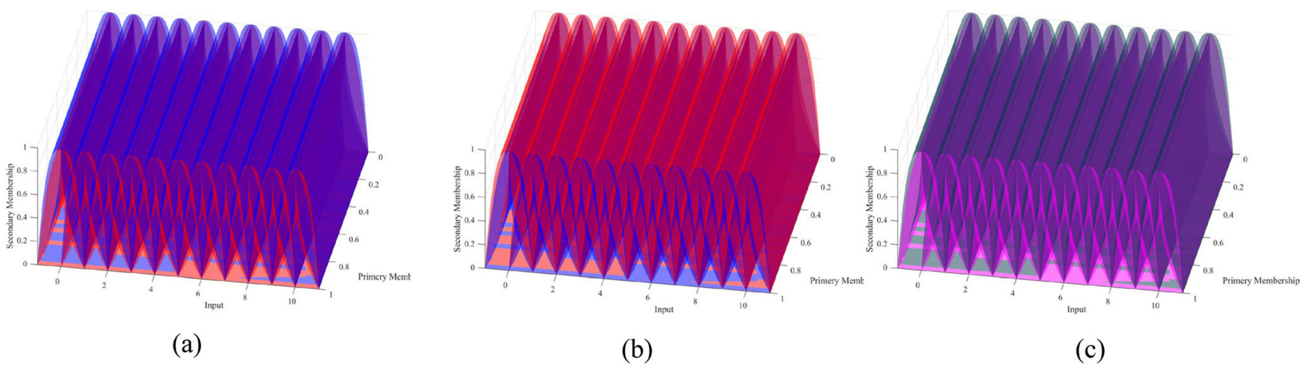


Fig. 15 MFs for three inputs FIS-III mentioned **a** SVM, **b** ResNet-18, **c** efficient-CapsNet

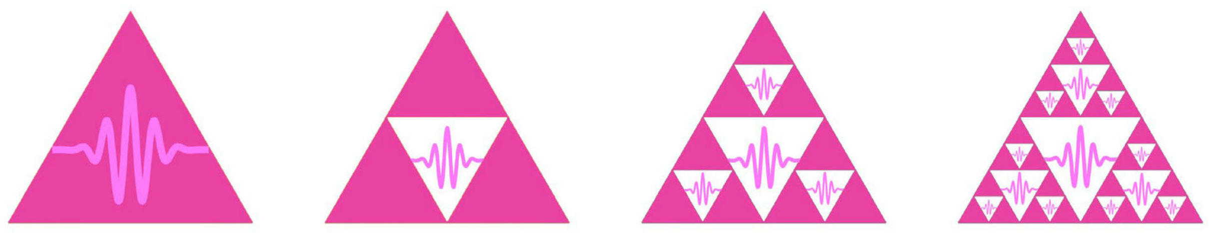


Fig. 16 Sierpinski triangle and self-similarity in ICGO

or if they carry significance. A low p-value indicates a substantial disparity between the paired data, while a high p-value suggests uncertainty regarding the existence of a noteworthy difference.

The Friedman test is indeed a non-parametric statistical test used to determine if there are statistically significant differences among multiple related groups (Table C.2). This research divided the BFs into five distinct groups to ensure the test's reliability. The first, second, third, and fourth groups included CEC 2017 BFs in different dimensions, respectively (Tables B.1–B.4), while the fifth group is formed by CEC 2019 BFs illustrated in Table B.5 [91].

A post-hoc Nemenyi test was utilized to delve deeper into the distinctions among the algorithms. If the null hypothesis is rejected, a post-hoc test can be conducted. The Nemenyi test is employed when conducting pairwise comparisons among all algorithms. The performance disparity between two classifiers is deemed significant if their respective average ranks exhibit a difference equal to or exceeding the *CD* (Eq. (48)) [91].

$$CD = q_\alpha \sqrt{\frac{k(k+1)}{6N}} \tag{48}$$

N represents the number of BFs in each group, *k* represents the number of algorithms under comparison and in each group, we selected the top 10 algorithms for comparison. At a significance level of $\alpha = 0.05$, the critical value for 10 algorithms, the associated *CD* for each group has been specified in Fig. C.1 and $q_\alpha = 3.164$. To identify distinctions among the ten algorithms, the *CD* derived from the Nemenyi test was employed. The *CD* diagrams depicted in Fig. C.1 offer straightforward and intuitive visualizations of the outcomes from a Nemenyi post-hoc test. This test is specifically designed to assess the statistical significance of differences in average ranks among a collection of ten algorithms, each evaluated on a set of five groups.

Following the revelation of notable variations in performance among various algorithms, it becomes imperative to identify which algorithms exhibit significantly different

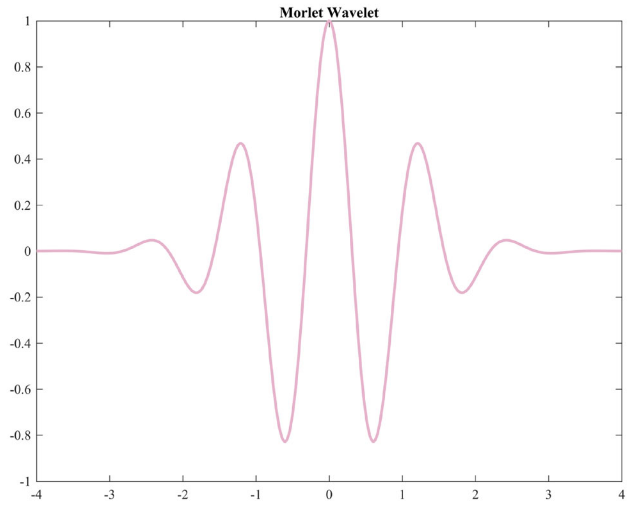


Fig. 17 Morlet wavelet

Table 9 Pseudocode improved chaos game optimization

```
Algorithm: Improved Chaos Game Optimization
Create random values for initial positions ($x_i^j$) of eligible seeds ($\mathcal{X}_i$)
Evaluate fitness value for each eligible seed
Find $\mathcal{GB}$, So far found best eligible seed
while (t<Maximum number of iterations)
        for i=1:number of initial eligible seed
        Find $\mathcal{MG}_i$
        Create temporary triangles with $\mathcal{X}_i$, $\mathcal{GB}$, $\mathcal{MG}_i$
        Calculate the $\alpha_i$, $\beta_i$, and $\gamma_i$ values
        Create new seed by Eqs. (43-46)
                If new seeds violate boundary conditions
                Control the position constraints for new seeds and amend it
                end if
        Evaluate the fitness values for new seeds
                If new seeds have better fitness values than the worst initial eligible
                seeds
                end if
        Update $\mathcal{GB}$ if a better solution is found
                end for
                t=t+1
end while
Return $\mathcal{GB}$
```

$$Seed_i^1 = \mathcal{X}_i + \alpha_i \times (\beta_i \times \mathcal{GB} - \gamma_i \times \mathcal{MG}_i), i = 1,2,\ldots,n \quad (43)$$

$$Seed_i^2 = \mathcal{GB} + \alpha_i \times (\beta_i \times \mathcal{X}_i - \gamma_i \times \mathcal{MG}_i), i = 1,2,\ldots,n \quad (44)$$

$$Seed_i^3 = \mathcal{MG}_i + \alpha_i \times (\beta_i \times \mathcal{X}_i - \gamma_i \times \mathcal{GB}), i = 1,2,\ldots,n \quad (45)$$

$$Seed_i^4 = \begin{cases} \mathcal{GB} + \sigma(\mathcal{X}_{i,max} - \mathcal{GB}), & if\ \sigma < 0.5 \\ \mathcal{GB} + \sigma(\mathcal{GB} - \mathcal{X}_{i,min}), & if\ \sigma \geq 0.5 \end{cases} \quad (46)$$

performances compared to ICGO. ICGO is regarded as control algorithm in this context. Figure C.1 displays the average ranking of each method across five groups, with significance levels of 0.05 in 30 distinct runs. ICGO demonstrates significant superiority over algorithms whose average ranking exceeds the threshold line indicated in the figure.

A post-hoc analysis determines that if the disparity in mean Friedman values between the two algorithms falls below the *CD* threshold, there is no notable distinction between them; conversely, if it surpasses the *CD* value, a significant difference between the algorithms exists. In Table C.3, a comparison has been conducted between 10 algorithms and ICGO across all five BF groups. Algorithms that are not significantly different from the ICGO algorithm are highlighted with a red mark. Conversely, algorithms that are deemed significantly different from the ICGO algorithm are highlighted with a green mark in this table. In accordance with Table C.3, none of the examined algorithms in this article can serve as a substitute for algorithm ICGO. This observation underscores the necessity of the existence of algorithm ICGO, which can potentially address limitations not covered by other algorithms.

4 Proposed method

The implementation steps of our proposed model are as follows:

(1) The Efficient-CapsNet was trained on a training set of RGB dimensions 28 × 28, according to the current k-fold cross-validation partitioning.
(2) The ResNet-18 was trained on the training set of resized images with RGB dimensions of 224 × 224, according to the current k-fold cross-validation partitioning.
(3) Features are extracted from the resized images of the training set using the ResNet-50 network and the HOG algorithm, according to the current k-fold cross-validation partitioning.

(4) The NCA algorithm reduces the 1720 feature vectors obtained from step 3 to 80 features.
(5) The 80 feature vectors obtained from the previous step are used to train the SVM classifier.
(6) To train the type-III fuzzy system as an appropriate decision maker, the results obtained from the three classifiers are considered as a new dataset for the type-III fuzzy system. Its special cases are separated, and the rest are removed, as explained in Sect. 3.7. Finally, we have a dataset of 43 samples with labels that provide the highest classification accuracy (for the original dataset under investigation).
(7) The fuzzy system is trained using the dataset created in step 6, and its rule parameters are fine-tuned by the ICGO algorithm (Sect. 3.8.1).
(8) The proposed model's performance is now evaluated using the test set.

Figure 19 presents a concise and graphical overview of the steps involved in implementing the proposed model.

Malware's impact on infiltrating diverse operating systems underscores the urgency of creating a precise classifier for effective categorization. This article introduces a novel hybrid classifier that combines type-III fuzzy decision-making with a deep neural network ensemble. An enhanced version of the CGO optimization algorithm is utilized to efficiently adjust fuzzy system parameters, boosting accuracy. The proposed classifier consistently achieves accuracy rates higher than 96% in datasets related to malware classification. Furthermore, comparisons with recently introduced classifiers on MNIST and Fashion-MNIST datasets showcased the supremacy of the proposed classifier in multiple performance metrics, highlighting its superiority over well-known networks. While a minor rise in parameters when compared to ResNet-18 could be viewed as a minor weakness of the proposed method, it's important to note that the proposed approach significantly outperforms ResNet-18 and even networks with higher parameter counts like AlexNet in terms of accuracy. In this context, this aspect might be seen as a strength of the proposed method.

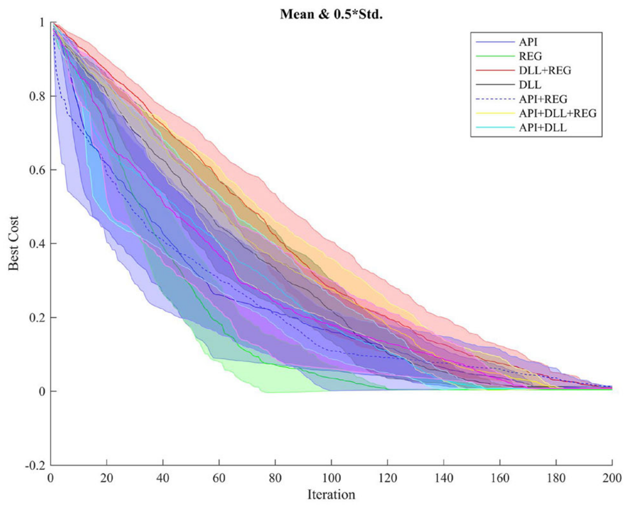


Fig. 18 The mean and 0.5 standard deviation were calculated for seven datasets to optimize rule parameters across five folds

5 Results and discussion

In this section, the results have been thoroughly examined to familiarize other researchers with the strengths of our proposed classifier and facilitate easy comparison with other classifiers. The proposed method was evaluated using publicly available 11-class datasets from Ref. [39]. As noted in Sect. 3.1, the mentioned dataset comprises seven datasets. We compared the models proposed in [39] and other classifiers employed in this paper with our proposed method to evaluate its performance. Further, we utilized fivefold cross-validation (Sect. 3.1.2) in all classifiers used in this study to reduce the impact of data grouping (test and train) on performance. Lastly, a total of 750 samples, representing the test data, and 2999 samples, for the train data, are used in the simulations throughout the dataset, which are classified through fivefold cross-validation. In the course of our analysis, the macro average metrics was employed. However, only the basic formula of the metric is mentioned herein for brevity and to minimize the complexity of the presented equations. Metrics such as accuracy (Eq. 48), precision (Eq. 51), sensitivity (Eq. 50), specificity (Eq. 49), F1-score (Eq. 52). Also, to avoid prolixity, only the confusion matrix of the most optimal result obtained from the proposed model was depicted in the Sect. 5, whereas other results were detailed in Tables E (1–7), Tables F (1–7) and Figs. E (1–14). Since deep learning and metaheuristic optimization methods do not generally provide mathematical proof to guarantee performance, various models were carefully analyzed in the simulation section during separate runs. The advantages or disadvantages of the methods can only be discussed through their statistical analysis and multiple runs [92–94]:

$$\text{Accuracy} = \frac{\mathcal{TP}+\mathcal{TN}}{\mathcal{TP}+\mathcal{TN}+\mathcal{FP}+\mathcal{FN}} \tag{49}$$

$$\text{Sensitivity} = \frac{\mathcal{TN}}{\mathcal{TN}+\mathcal{FP}} \tag{50}$$

$$\text{Specificity} = \frac{\mathcal{TP}}{\mathcal{TP}+\mathcal{FN}} \tag{51}$$

$$\text{Precision} = \frac{\mathcal{TP}}{\mathcal{TP}+\mathcal{FP}} \tag{52}$$

Table 10 Control parameters of algorithms

Algorithm	Parameter	Value
GWO	Convergence parameter (a)	Linear reduction from 2 to 0
SCA	A	2
WOA	Convergence parameter (a)	Linear reduction 2 to 0
	Parameter r	A random vector between 0 and 1
	Parameter l	A random vector between -1 and 1
PSO	Velocity limit	10% of dimension range
	Cognitive and social constant	$(C_1, C_2) = (2, 2)$
	Topology	Fully connected
	Inertia weight	Linear reduction from 0.9 to 0.1
GOA	l	1.5
	f	0.5
SSA	Initial speed (v_0)	0
LSHADE-EpSin	Pbest	0.1
	Arc rate	2
TLBO	Teaching factor (T_F)	round$(1 + \text{rand})$
	Rand	A random number between 0 and 1
CGO	Factor of pheromone concentration (α)	1
	Factor of visibility (β)	5
	Pheromone evaporation coefficient (ρ)	0.5
	Pheromone intensity (Q)	10
AOA	a	5
	μ	0.5
IWO	Minimum number of seeds (S_{min})	0
	Maximum number of seeds (S_{max})	5
	Initial value of standard deviation	1
	Final value of standard deviation	0.001
	Variance reduction exponent	2

$$\text{F1-score} = 2 \times \frac{Precision \times Sensitivity}{Precision + Sensitivity}. \tag{53}$$

5.1 API dataset

The malware made frequent API calls to the operating system for network access and file creation, represented. The R channel in the malware visualization is used to represent the API dataset, which causes the main characteristic of this dataset to be the presence of the R channel and the absence of the other two channels (G and B).

Figure E (1) and Fig. 20 demonstrate the confusion matrix of the best results in fivefold of Efficient-CapsNet, SVM, ResNet-18 methods our proposed method for test and train. Average accuracy: 0.9829, 0.9504, 0.9411, 0.9902, average sensitivity: 0.9810, 0.9351, 0.9231, 0.9920, average specificity: 0.9891, 0.9853, 0.9844, 0.9965, average precision: 0.9822, 0.9524, 0.9908, 0.9402, and average F1-score: 0.9819, 0.9380, 0.9269 and 0.9911 for the train data classification of the API dataset belong to the classifiers SVM, Efficient-CapsNet, ResNet-18, and the proposed classifier, respectively. Table (F.1) reports the results comparison of the classifiers for the train data of the API dataset. Throughout this study, we have used the abbreviation N/A to indicate Not Available information to maintain clarity and precision in our reporting.

Average accuracy: 0.9520, 0.9190, 0.9036, 0.9713 and N/A, average sensitivity: 0.9544, 0.9013, 0.9035, 0.9668, average specificity: 0.9870, 0.9821,0.8863, 0.9950 and N/A, average precision: 0.9618, 0.9237, 0.9107, 0.9770 and 0.93038, and average F1-score: 0.9530, 0.9056, 0.8891, 0.9688 and 0.90948 for the test data classification of the API dataset belong to the classifiers SVM, Efficient-CapsNet, ResNet-18, the proposed classifier, and DACN proposed classifier in Article [39], respectively. As can be observed, the most optimal average of various metrics belongs to our proposed classifier. The variance of the testing classification accuracy for our proposed method is only 0.0018, whereas it stands at 0.0062 for classifier

DACN which is more than three times our proposed method, doubling the value of this performance improvement. Table 11 and Table (E.1) depict the performance comparison of the classifiers for the test data of the API dataset.

To better depict the robustness of the proposed classifier, the box plot of the models was also depicted in this article (Fig. E(2)). In this chart, the test data accuracy of the proposed classifier, and the models analyzed in this article and the DACN were illustrated in various folds. It is noteworthy that not only is the accuracy dispersion obtained through variant folds classifier substantially less than that of the other classifiers, but also the accuracy of the proposed classifier in various folds is more than other classifiers.

5.2 DLL dataset

The malware loaded DLLs that contained encryption functions, represented by G channel in the malware visualization. The main characteristic of these images is the presence of the G channel and the absence of the R and B channels.

Figure E(3) and Fig. 21 demonstrate the confusion matrix of the best results in fivefold of Efficient-CapsNet, SVM, ResNet-18 methods and our proposed method for test and train.

Average accuracy: 0.9753, 0.9727, 0.9556, 0.9854, average sensitivity: 0.9766, 0.9681, 0.9495, 0.9867, average separation: 0.9893, 0.9874, 0.9858, 0.9971, average precision: 0.9782, 0.9710, 0.9512, 0.9881, and average F1-score: 0.9759, 0.9701, 0.9522, 0.9860 for the train data classification of the DLL dataset belong to the classifiers SVM, Efficient-CapsNet, ResNet-18, and the proposed classifier, respectively. Table (F.2) reports the results of the classifiers for the train data of the DLL dataset.

Average accuracy: 0.9479, 0.9601, 0.9426, 0.9634 and N/A, average sensitivity: 0.9349, 0.9598, 0.9427, 0.9524 and N/A, average specificity: 0.9867, 0.9863, 0.9953 and N/A, average precision: 0.9544, 0.9600, 0.9431, 0.9714 and 0.96212, and average F1-score: 0.9403, 0.9597, 0.9424, 0.9572 and 0.96504 for the test data classification of the DLL dataset belong to the classifiers SVM, Efficient-CapsNet, ResNet-18, the proposed classifier, and DACN, respectively. (Table 11 and Table (E.2)) Our proposed classifier also achieved the most optimal performance in this dataset as per different metrics. With a slight difference of 0.96% in the average classification accuracy compared to classifier DACN, the proposed classifier ranks 1st. We must take into consideration that our proposed classifier, with a 0.0014 standard deviation of average classification accuracy, and classifier DACN, with a 0.0090 standard deviation of average classification accuracy, were ranked 1st and 2nd, respectively. This shows that our proposed classification has achieved a more proper classification in different folds.

The proposed classifier for the DLL dataset achieved the highest accuracy compared to the models evaluated in this article and DACN. The box plot of the classification accuracy for different models evaluated and the DACN model for the DLL dataset is plotted in Fig. E(4). As can be seen, the proposed classifier in this case also achieved the highest classification accuracy and the lowest amount of dispersion around the best classification among all the evaluated models, and thus is placed in the first position.

5.3 REG dataset

The malware manipulated the registry to achieve persistence, represented by the B channel in the malware visualization.

Figure E(5) and Fig. 22 demonstrate the confusion matrix of the best results in fivefold of Efficient-CapsNet,

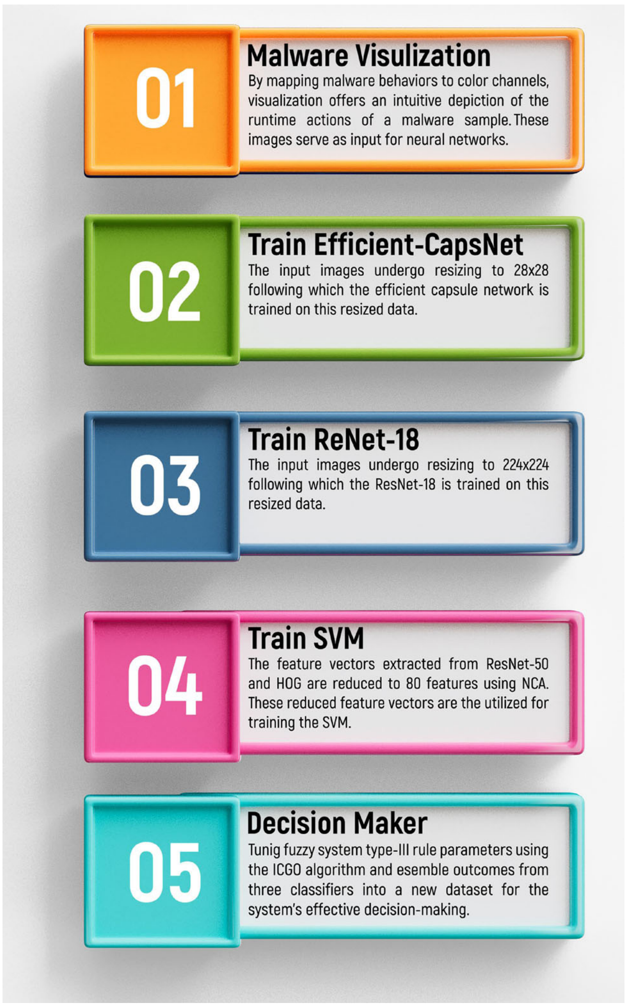

01
Malware Visulization
By mapping malware behaviors to color channels, visualization offers an intuitive depiction of the runtime actions of a malware sample. These images serve as input for neural networks.
02
Train Efficient-CapsNet
The input images undergo resizing to 28x28 following which the efficient capsule network is trained on this resized data.
03
Train ReNet-18
The input images undergo resizing to 224x224 following which the ResNet-18 is trained on this resized data.
04
Train SVM
The feature vectors extracted from ResNet-50 and HOG are reduced to 80 features using NCA. These reduced feature vectors are the utilized for training the SVM.
05
Decision Maker
Tunig fuzzy system type-III rule parameters using the ICGO algorithm and esemble outcomes from three classifiers into a new dataset for the system's effective decision-making.

Fig. 19 The proposed model's steps

SVM, ResNet-18 methods and our proposed method for test and train. Average accuracy: 0.9808, 0.9352, 0.9267, 0.9914, average sensitivity: 0.9763, 0.9222, 0.9039, 0.9908, average specificity: 0.9878, 0.9864, 0.9845, 0.9899, average precision: 0.9800, 0.9561, 0.9353, 0.9956, and average F1-score: 0.9784, 0.9221, 0.9897and 0.9911 for the train data classification of the REG dataset belong to the classifiers SVM, Efficient-CapsNet, ResNet-18, and the proposed classifier, respectively. Table (F.3) report the results of the classifiers for the train data of the REG dataset.

Average accuracy: 0.9335, 0.9260, 0.9239, 0.9451 and N/A, average sensitivity: 0.9057, 0.9162, 0.9101, 0.9389 and N/A, average specificity: 0.9839, 0.9854, 0.9848, 0.9939 and N/A, average precision: 0.9406, 0.9457, 0.9379, 0.9626 and 0.83703, and average F1-score: 0.9102, 0.9158, 0.9047, 0.9397 and 0.72883 for the test data classification of the DLL dataset belong to the classifiers SVM, Efficient-CapsNet, ResNet-18, the proposed classifier, and DACN, respectively (Table 11 and Table (E.3)).

Compared to DACN, our proposed classifier has improved by roughly 15% regarding the average classification accuracy and 28% regarding the average F1-score. Compared to DACN, our proposed classification has improved the standard deviation of the average classification accuracy by over 92%. (The standard deviation of our proposed classifier: 0.0020, and the standard deviation of classifier DACN: 0.0258) Compared to the other models analyzed in this article, classifier DACN showed the poorest performance for the test data of the REG dataset. Aside from exhibiting the weakest classification accuracy, this classifier had the highest classification accuracy dispersion in various folds. With insignificant dispersion and the highest classification accuracy in various folds, our proposed classifier had the best result among the other models (Fig. E(6)).

5.4 API + DLL dataset

The malware visualization showed integrated R channel and G channel representing API calls and loaded DLLs. The R channel indicated network and file activities. The G channel revealed encryption functions from external libraries.

Figure E(7) and Fig. 23 demonstrate the confusion matrix of the best results in fivefold of Efficient-CapsNet, SVM, ResNet-18 methods and our proposed method for test and train.

Average accuracy: 0.9854, 0.9818, 0.9608, 0.9937, average sensitivity: 0.9819, 0.9826, 0.9478, 0.9911, average specificity: 0.9892, 0.9894, 0.9850, 0.9974, average precision: 0.9842, 0.9816, 0.9514, 0.9924, and average F1-score: 0.9836, 0.9821, 0.9524 and 0.9923 for the train data classification of the API + DLL dataset belong to the classifiers SVM, Efficient-CapsNet, ResNet-18, and the proposed classifier, respectively. Table (F.4) reports the results of the classifiers for the train data of the API + DLL dataset.

Average accuracy: 0.9675, 0.9691, 0.9604, 0.9782 and N/A, average sensitivity: 0.9615, 0.9664, 0.9398, 0.9667 and N/A, average specificity: 0.9872, 0.9874, 0.9838, 0.9955 and N/A, average precision: 0.9658, 0.9635, 0.9410, 0.9752 and 0.96612, and average F1-score: 0.9643, 0.9675, 0.9472, 0.9720 and 0.96572 for the test data classification of the APP + DLL dataset belong to the classifiers SVM, Efficient-CapsNet, ResNet-18, the proposed classifier, and DACN, respectively (Table 11 and Table (E.4)).

The proposed classifier has performed better than other evaluated models in terms of all the classification metrics. In the test data of the API + DLL dataset, our proposed classifier had the highest classification accuracy and the lowest classifier accuracy dispersion in various folds. ResNet-18 showed the weakest performance among the analyzed models (Fig. E(8)). Our proposed classification had significantly better performance, considering the mentioned points, compared to the proposed classifier DACN.

5.5 API + REG dataset

The malware visualization displayed a combination of R channel and B channel for API calls and registry manipulations. The R channel pointed to network and file behaviors. The B channel showed registry changes for persistence.

Figure E(9) and Fig. 24 demonstrate the confusion matrix of the best results in fivefold of Efficient-CapsNet, SVM, ResNet-18 methods and our proposed method for test and train.

Average accuracy: 0.9814, 0.9691, 0.9603, 0.9788, average sensitivity: 0.9773, 0.9549, 0.9495, 0.9739, average specificity: 0.9880, 0.9884, 0.9880, 0.9960 average precision: 0.9806, 0.9775, 0.9679, 0.9870, and average F1-score: 9791, 0.9625, 0.9555, 0.9738 for the train data classification of the API + REG dataset belongs to the classifiers SVM, Efficient-CapsNet, ResNet-18, and the proposed classifier, respectively. Table (F.5) reports the results of the classifiers for the train data of the API + REG dataset.

Average accuracy: 0.9013, 0.8954, 0.8889, 0.9148 and N/A, average sensitivity: 0.9073, 0.9082, 0.9002, 0.9208 and N/A, average specificity: 0.9851, 0.9861, 0.9862, 0.9939 and N/A, average precision: 0.9528, 0.9574, 0.9514, 0.9676, and 0.95466, and average F1-score: 0.9026, 0.8933, 0.8829, 0.9161 and 0.93815 for the test

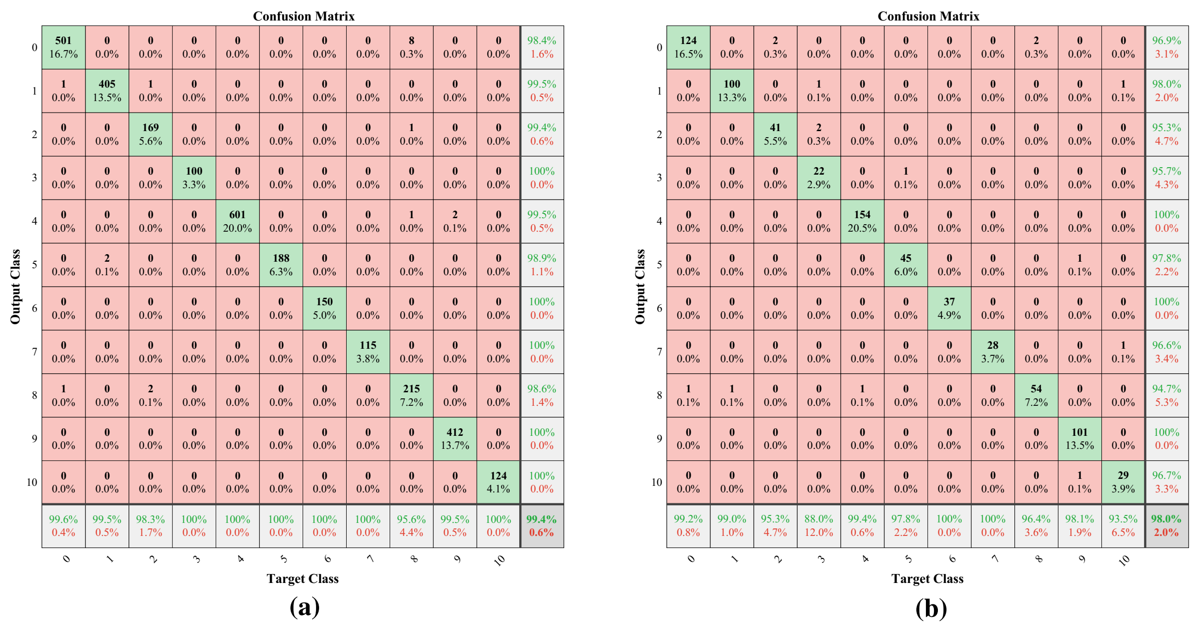


Fig. 20 Confusion matrix of the best result in fivefold of proposed model, **a** train, **b** test

data classification of the API + REG dataset belongs to the classifiers SVM, Efficient-CapsNet, ResNet-18, the proposed classifier, and DACN, respectively (Tables 11 and Table (E.5)).

In terms of average F1-score, classifier DACN ranks 1st and our proposed classifier ranks 2nd. However, in terms of average classification accuracy, our proposed method ranks 1st. The standard deviation of the average classification accuracy of our proposed classifier stands at 0.0019 while this number is 0.0092 for classifier DACN, indicating the proposed class is more robust. DACN classifier has the highest dispersion of classification accuracy in different folds for API + REG dataset classification. The lowest dispersion of classification accuracy and the highest classification accuracy in different folds belong to our proposed classifier (Fig. E(10)). The results show the efficiency of our proposed classifier for different datasets.

5.6 DLL + REG dataset

The malware visualization had integrated G channel and B channel representing loaded DLLs and registry activities. The G channel highlighted encryption functions. The B channel indicated registry modifications for persistence.

Figure E(11) and Fig. 25 demonstrate the confusion matrix of the best results in fivefold of Efficient-CapsNet, SVM, ResNet-18 methods and our proposed method for test and train.

Average accuracy: 0.9750, 0.9793, 0.9653, 0.9893, average sensitivity: 0.9708, 0.9785, 0.9610, 0.9854, average specificity: 0.9877, 0.9778, 0.9660, 0.9967, average precision: 0.9734, 0.9865, 0.9831, 0.9896, and average F1-score: 0.9726, 0.9771, 0.9597 and 0.9872 for the train data classification of the DLL + REG dataset belongs to the classifiers SVM, Efficient-CapsNet, ResNet-18, and the proposed classifier, respectively. Table (F.6) reports the result of the classifiers for the train data of the DLL + REG dataset.

Average accuracy: 0.9668, 0.9687, 0.9350, 0.9820 and N/A, average sensitivity: 0.9570, 0.9565, 0.9386, 0.9773 and N/A, average specificity: 0.9869, 0.9877, 0.9858, 0.9964 and N/A, average precision:0.9654, 0.9733, 0.9509, 0.9868 and 0.97306 and average F1-score: 0.9611, 0.9622, 0.9339, 0.9791 and 0.97036 for the test data classification of the DLL + REG dataset belongs to the classifiers SVM, Efficient-CapsNet, ResNet-18, the proposed classifier, and DACN, respectively (Tables 11 and Table (E.6)).

Considering various metrics, our proposed classification performed better compared to other classifiers examined. The order of the ranking of classifiers based on the average classification accuracy: our proposed method, Efficient-CapsNet, DACN, SVM, and RestNet-18. The order of the ranking of the classifiers according to the standard deviation of the average classification accuracy: our proposed method, ResNet-18, Efficient-CapsNet, SVM, and DACN. Similar to the API + DLL dataset, in the DLL + REG dataset, RestNet-18 accounts for the high classification accuracy dispersion, as well as the lowest classification accuracy in various folds. The highest classification accuracy in various folds and the lowest classification accuracy dispersion, such as other analyzed datasets so far, belong to our proposed classification (Fig. F(12)).

Table 11 The performance comparison proposed model using the fivefold CV procedure (test)

Feature: API						
Proposed method	Precision	0.9706	0.9712	0.9708	0.9695	0.97427
	Sensitivity	0.9661	0.9668	0.9663	0.965	0.9698
	Specificity	0.9943	0.9949	0.9945	0.9932	0.99798
	Accuracy	0.9763	0.977	0.9765	0.9752	0.98
	F-1 score	0.9681	0.9688	0.9683	0.967	0.97181
Feature: DLL						
Proposed method	Precision	0.9635	0.9624	0.9616	0.964	0.96534
	Sensitivity	0.9525	0.9514	0.9507	0.953	0.95439
	Specificity	0.9954	0.9943	0.9936	0.9959	0.99728
	Accuracy	0.9715	0.9704	0.9696	0.972	0.97333
	F-1 score	0.9573	0.9562	0.9555	0.9578	0.95915
Feature: REG						
Proposed method	Precision	0.9437	0.9443	0.9431	0.94783	0.9465
	Sensitivity	0.9375	0.9381	0.9369	0.94162	0.9403
	Specificity	0.9925	0.9931	0.9919	0.99659	0.9953
	Accuracy	0.9612	0.9618	0.9606	0.96533	0.964
	F-1 score	0.9383	0.939	0.9378	0.94248	0.9411
Feature: API + DLL						
Proposed method	Precision	0.9772	0.9771	0.9788	0.98031	0.9777
	Sensitivity	0.9657	0.9656	0.9673	0.96875	0.9661
	Specificity	0.9945	0.9944	0.9961	0.99762	0.995
	Accuracy	0.9742	0.9741	0.9758	0.97733	0.9747
	F-1 score	0.971	0.9709	0.9726	0.97407	0.9714
Feature: API + REG						
Proposed method	Precision	0.9133	0.9139	0.91794	0.9152	0.9139
	Sensitivity	0.9193	0.9199	0.92394	0.9212	0.9199
	Specificity	0.9923	0.9929	0.99697	0.9942	0.9929
	Accuracy	0.966	0.9666	0.97067	0.9679	0.9666
	F-1 score	0.9145	0.9151	0.91919	0.9165	0.9151
Feature: DLL + REG						
Proposed method	Precision	0.9824	0.98452	0.9822	0.9802	0.9809
	Sensitivity	0.9777	0.97978	0.9774	0.9754	0.9762
	Specificity	0.9968	0.99894	0.9966	0.9946	0.9953
	Accuracy	0.9872	0.98933	0.987	0.985	0.9857
	F-1 score	0.9794	0.98156	0.9792	0.9772	0.9779
Feature: API + DLL + REG						
Proposed method	Precision	0.9928	0.9927	0.9952	0.99631	0.9948
	Sensitivity	0.9871	0.9869	0.9894	0.99057	0.989
	Specificity	0.9958	0.9956	0.9981	0.99928	0.9977
	Accuracy	0.9898	0.9897	0.9922	0.99333	0.9918
	F-1 score	0.9898	0.9896	0.9921	0.99327	0.9917

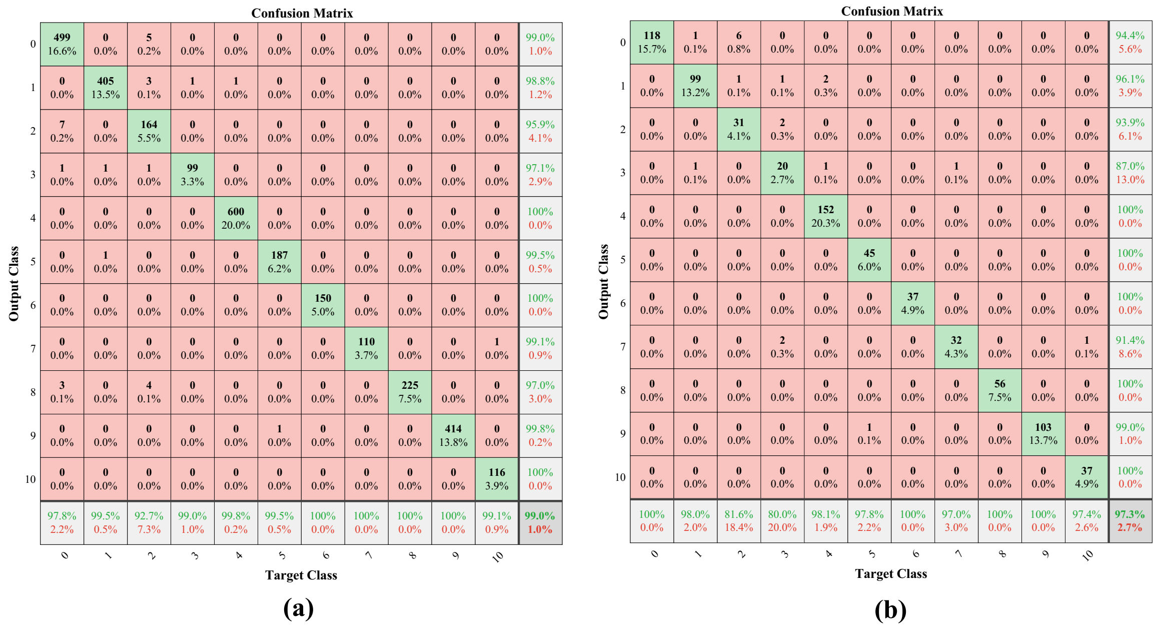


Fig. 21 Confusion matrix of the best result in fivefold of proposed model, **a** train, **b** test

5.7 API + DLL + REG dataset

The malware visualization showed a combination of R channel, G channel, and B channel representing API calls, loaded DLLs, and registry manipulations respectively. The R channel indicated network and file activities. The G channel pointed to encryption functions. And the B channel revealed registry changes for persistence. Together the RGB visualization provided an integrated profile of how the malware leveraged the operating system, external libraries, and registry to operate.

Figure E(13) and Fig. 26 demonstrate the confusion matrix of the best results in fivefold of Efficient-CapsNet, SVM, ResNet-18 methods and our proposed method for test and train.

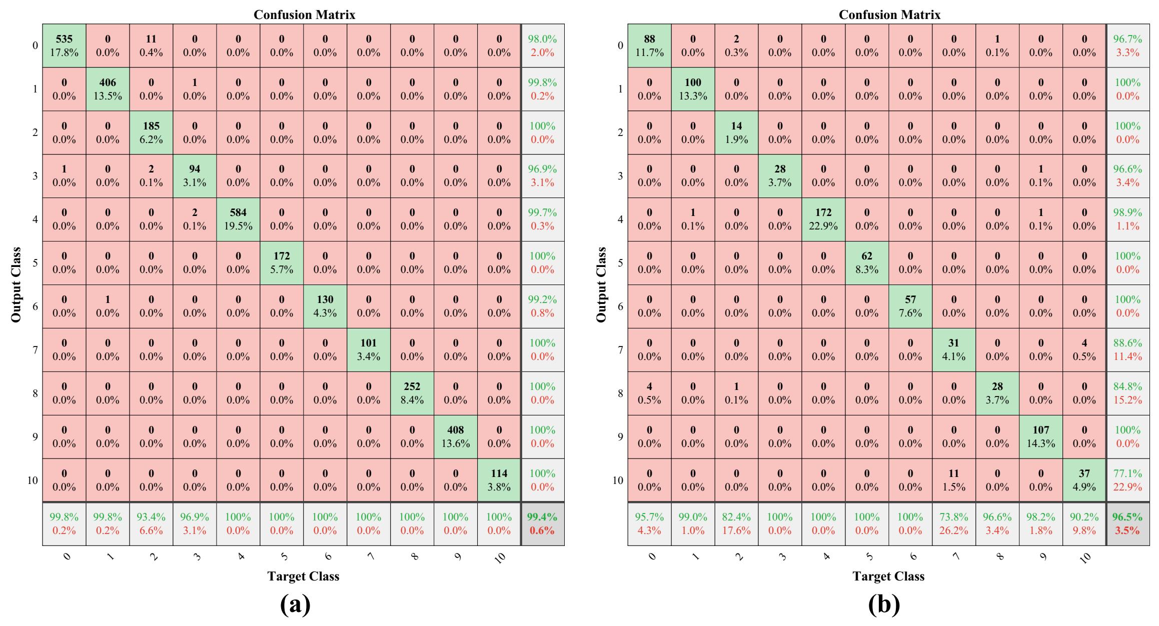


Fig. 22 Confusion matrix of the best result in fivefold of proposed model, **a** train, **b** test

Average accuracy: 0.9800, 0.9822, 0.9604, 0.9937, average sensitivity: 0.9800, 0.9799, 0.9517, 0.9892, average specificity: 0.9884, 0.9892, 0.9859, 0.9975, average precision: 0.9800, 0.9824, 0.9548, 0.9928, and average F1-score: 0.9800, 0.9809, 0.9546 and 0.9913 for the train data classification of the APP + DLL + REG dataset belongs to the classifiers SVM, Efficient-CapsNet, ResNet-18, and the proposed classifier, respectively. Table (F.7) reports the results of the classifiers for the train data of the APP + DLL + REG dataset.

Average accuracy: 0.9650, 0.9784, 0.9488, 0.9944, N/A, N/A, N/A, N/A and N/A, average sensitivity: 0.9459, 0.9704, 0.9334, 0.9886 N/A, N/A, N/A, N/A and N/A, average specificity: 0.9857, 0.9884, 0.9845, 0.9973, N/A, N/A, N/A, N/A and N/A, average precision: 0.9559, 0.9753, 0.9914, 0.9414, 0.97519, 0.94319, 0.62440, 0.96079 and 0.97199 and average F1-score: 0.9545, 0.9740, 0.9383, 0.9913, 0.97396, 0.93445, 0.42215, 0.95351 and 0.97032 for the test data classification of the APP + DLL + REG dataset belong to the classifiers SVM, Efficient-CapsNet, ResNet-18, the proposed classifier, and DACN, respectively (Table 11 and Table (E.7)).

Similar to the API + DLL and DLL + REG datasets, in the test data of the APP + DLL + REG dataset, ResNet-18 accounts for the lowest classification accuracy and highest classification accuracy dispersion (Fig. E(14)).

5.8 Main result

Considering the seven analyzed datasets, the efficiency of our proposed classifier for various datasets can be confirmed. Further, given the assessment of this class with 35 separate runs, it is possible to discuss the improvement of classification accuracy and the robustness of the proposed classifier in finding the proper classification with more confidence. Hence, in addition to the k-fold cross validation, the impact of data grouping significantly decreases due to the number of runs. The significance of this matter lies in the fact that neural networks and optimization algorithms often do not evaluate their performance through a mathematical proof and mostly compare them in terms of statistics, especially average and variance in numerous runs. This paper analyzed all the aforementioned, and the proposed classifier obtained the most optimal result.

Figure 27 demonstrates the best classification accuracy for different models analyzed in this article and DACN. It can easily be observed that all the datasets of our proposed classifier had the most optimal performance in terms of classification accuracy. Given the obtained results and their comparison with various models in this article, it can be claimed that the proposed classifier has properly and significantly improved compared to the classifiers from which it feeds. Further, it has performed better than highly used classifiers such as AlexNet, ResNet-18, and GoogleNet.

Nonetheless, increased classification accuracy has not led to changes in the order of computational complexity (despite the primary objective of this paper being solely the improvement of classification accuracy). Figure 28 compares the learning parameters of the three networks of AlexNet, ResNet-18, and our proposed classification. Compared to ResNet-18, the number of the parameters added by our proposed classifier was only 142,956, which can be ignored considering the performance improvement and the number of ResNet-18 parameters (11.6 million). The trainable parameters in AlexNet reach 60.9 million, which substantially exceeds our proposed network. Although designing a low-parameter network was not one of our objectives, it can be stated that our proposed network has not caused exponential growth of parameters compared to the other networks. In the conclusion and future work (Sect. 6), we will discuss ideas regarding the design of this network so that it would be significantly lightweight for other researchers to use if found useful.

In addition to the seven mentioned datasets, the performance improvement of the proposed classifier was also evaluated on two well-known datasets, MNIST [95] and Fashion-MNIST [96] (Table (A)). The reason for choosing these two datasets was to thoroughly investigate the proposed classifier's performance from various perspectives by evaluating it on two large balanced datasets and one unbalanced dataset. The MNIST and Fashion-MNIST datasets are 10-class datasets; consequently, the output of classifiers are integer numbers between 0 to 9. The specifications of the FIS-III are similar to those reported in Table 8, with the difference that the parameters associated with the eleventh membership function have been removed by eliminating it. The total number of rules for these two datasets is 120; however, as described in Sect. 3.7, this number decreases to 28 and 32, respectively.

Table 12 presents a comparison between the proposed model in this article and recent models for detecting and classifying malware. The results indicate that the proposed classification method achieves high performance in identifying and categorizing malware when all three extracted dynamic features are utilized (API + DLL + REG) and it is ranked among the top-performing classifiers for malware classification. Even when the proposed method uses only one dynamic feature, such as REG, it still performs well and achieves a classification accuracy above 0.96 in all conditions.

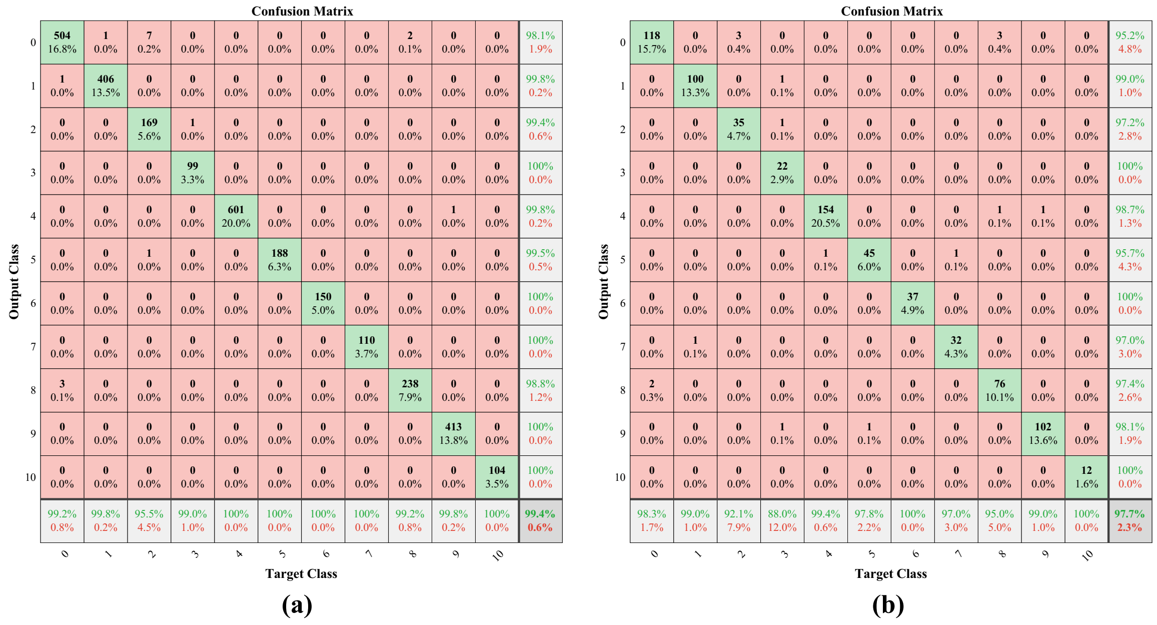


Fig. 23 Confusion matrix of the best result in fivefold of proposed model, **a** train, **b** test

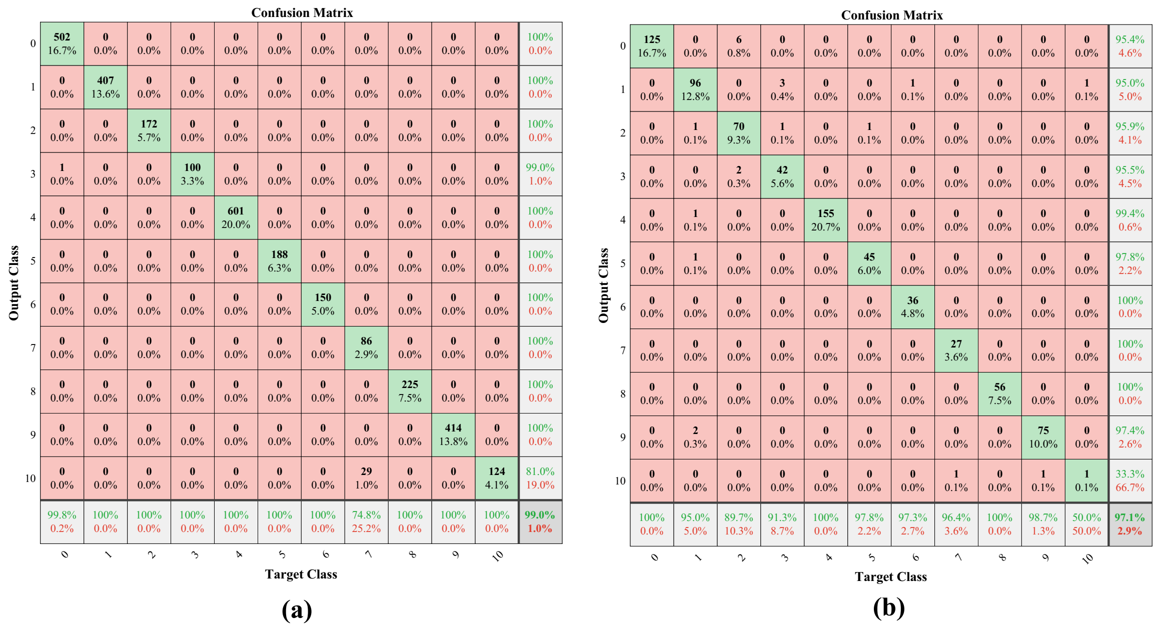


Fig. 24 Confusion matrix of the best result in fivefold of proposed model, **a** train, **b** test

5.8.1 Classifiers' statistical evaluation

In this section, the Friedman test and the Nemenyi post hoc test are employed to evaluate if the overall variances among accuracy and F1-score measurements are statistically meaningful. The Friedman test assesses algorithms individually for each dataset, ranking them based on their performance (Table 13). Subsequently, the Nemenyi test compares the average ranks of the algorithms. Then, it computes based on the χ_F^2 distribution with $k-1$ degrees of freedom (where k represents the number of algorithms). To identify a statistically notable variance in performance, a post hoc examination is necessary to pinpoint the algorithms responsible for these variances. The Nemenyi test is extensively employed in this scenario. This examination indicates notable discrepancies in their performance when the mean ranks of the two algorithms deviate from certain critical margins. Initially, we compute the mean rank for each algorithm examined in our trials, setting $k = 5$ and $n = 7$, as there are a total of 5 methods and 7 datasets

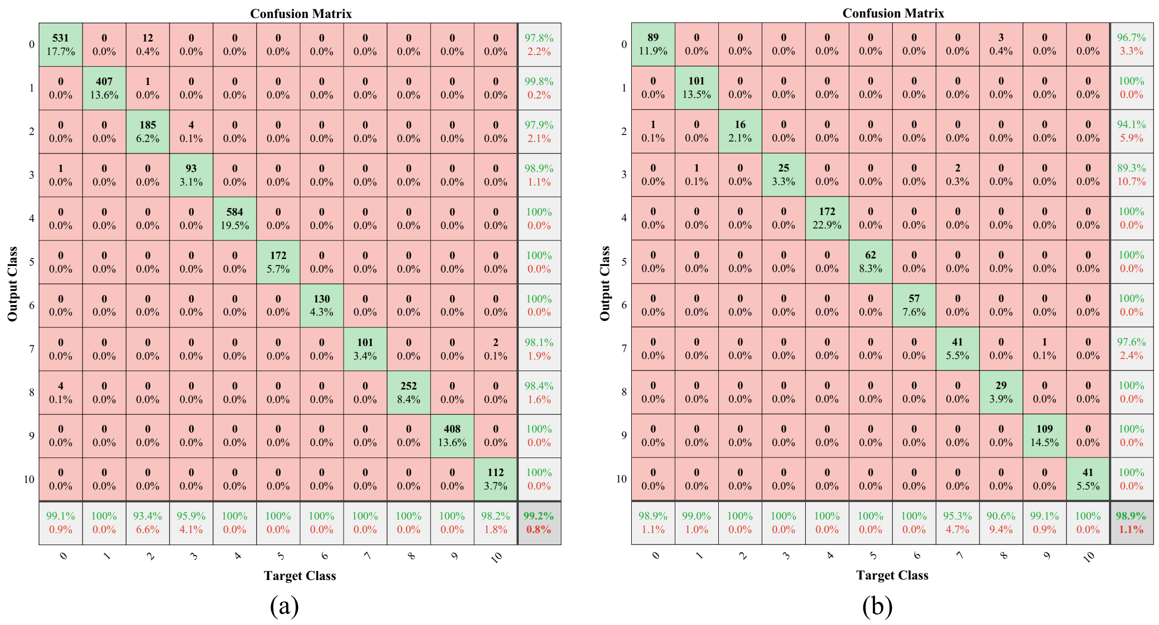


Fig. 25 Confusion matrix of the best result in fivefold of proposed model, **a** train, **b** test

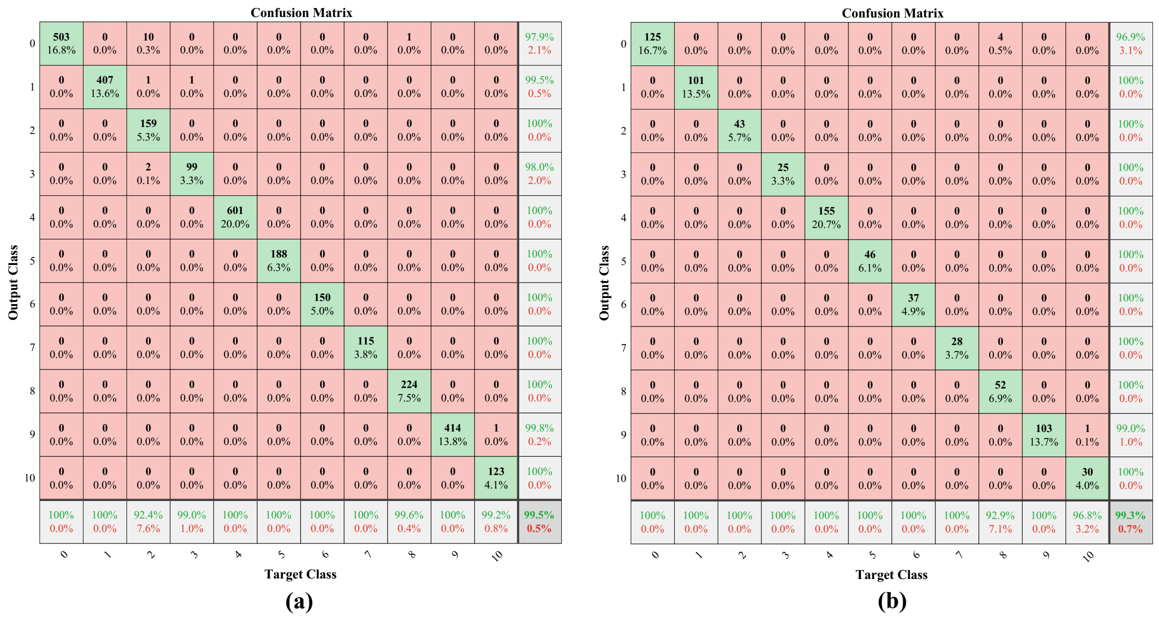
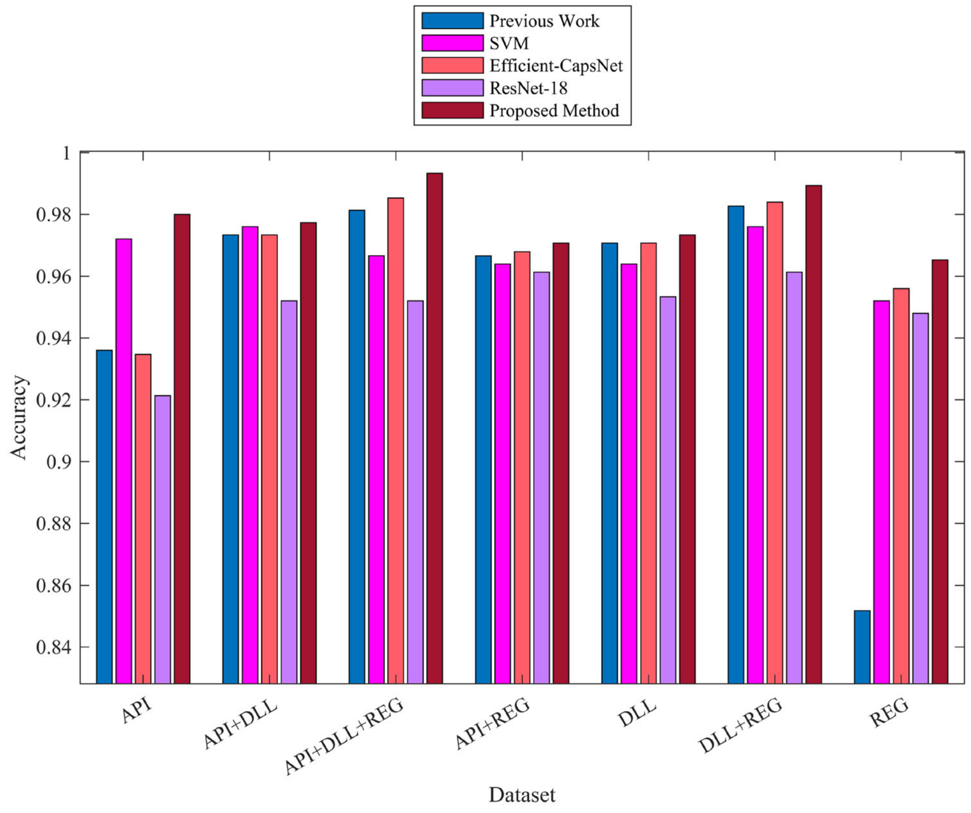


Fig. 26 Confusion matrix of the best result in fivefold of proposed model, **a** train, **b** test

involved. If the Friedman test statistic yields accuracy and F1-score values of $\chi_F^2 = 21.6$ and $\chi_F^2 = 16.8$ respectively, with $4(k-1)$ degrees of freedom, $\alpha = 0.05$ and the critical value for the Friedman test, given for $k = 5$ and $n = 7$, is 9.49 at a significance level of, we can infer that the accuracy and F1-score values of the evaluated methods exhibit significant differences ($21.6 > 9.49$, $16.8 > 9.49$ respectively). Since the null hypothesis has been disproved, we can move forward with a post hoc examination. The Nemenyi test becomes applicable when all classifiers are assessed against one another [104]. The critical value in our experiments with $k = 5$ and $\alpha = 0.05$ is $CD_\alpha = 2.3056$, $q_\alpha = 2.728$. As a result, the accuracy and F1-score of the proposed method is significantly different from SVM, Efficient-CapsNet, ResNet-18 and DACN. Figures 29 and 30 depicts the statistical comparison of the approaches examined in our experiments using the Nemenyi test.

Fig. 27 The best classification precision for different models (previous work = DACN)



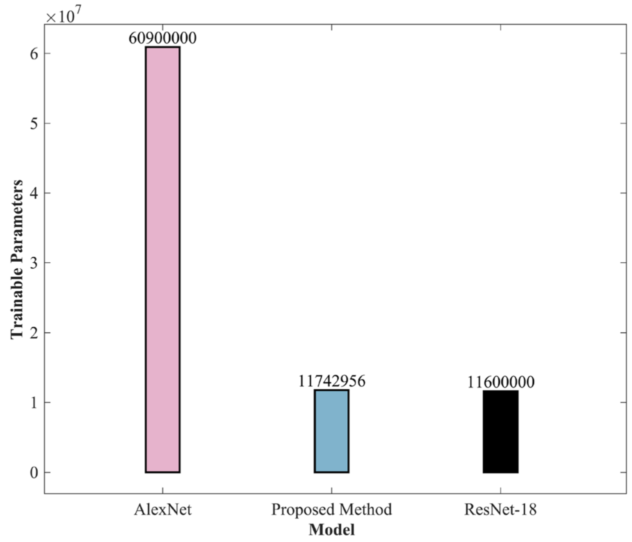


Fig. 28 Comparing the number of train parameters

6 Conclusion and future work

Scholarly evidence indicates that the utilization of artificial intelligence (AI) techniques has resulted in a significant enhancement of productivity across various domains. As smart devices continue to proliferate and become more deeply integrated into daily routines, the significance of safeguarding these devices against potential security threats becomes increasingly paramount. Malware constitutes a critical factor in the infiltration of diverse operating systems. Consequently, the development of a classifier that is capable of accurately categorizing malware becomes. For this reason, this article proposes a novel hybrid classifier based on type-III fuzzy decision-making and an ensemble of deep neural networks. To increase the speed of adjusting fuzzy system parameters and achieve higher accuracy, an improved version of CGO optimization algorithm was introduced and it was evaluated on 126 BFs and 5 EDPs, where it achieved first place in 124 BFs and all of EDPs. To evaluate the efficiency of the proposed classifier, we tested it on nine datasets. In 7 datasets related to malware classification, the classification accuracy was always over 0.96 and showed an improvement of 0.24–13.3% compared to other classifiers. Additionally, the statistical analysis results indicate that both ICGO and the proposed classifier exhibit superior performance compared to their competitors.

In addition, the performance of our proposed classifier was compared with six classifiers introduced in recent years for the well-known MNIST and Fashion-MNIST datasets. The proposed classifier ranked first in accuracy, specificity, sensitivity precision, and F1-score for these two datasets. The 45 separate runs of this classifier on nine

Table 12 Comparing existing models for malware detection and classification

Ref. no.	Model	Classification type	Accuracy	CV type
	Proposed method	Multi-class	0.98	Fivefold
		Multi-class	0.973	Fivefold
		Multi-class	0.965	Fivefold
		Multi-class	0.977	Fivefold
		Multi-class	0.971	Fivefold
		Multi-class	0.989	Fivefold
		Multi-class	0.99333	Fivefold
[97]	Inception V3	Multi-class	0.9876	Hold out
[98]	AI-HydRa	Binary	0.851	Tenfold
[99]	MoBIPCR	Binary	0.991	Hold out
[100]	EII-MBS	Multi-class	0.9940	Tenfold
		Multi-class	0.9929	Tenfold
[101]	GDroid	Binary	0.9899	Hold out
		Multi-class	0.9698	Hold out
[102]	MaliCage	Binary	0.982	Hold out
		Multi-class	0.978	Hold out
[103]	DTMIC	Multi-class	0.9892	Hold out
[41]	DMalNet	Binary	0.9843	Fourfold
		Multi-class	0.9142	Fourfold

datasets show that it is a suitable classifier for various applications and has significantly higher accuracy than well-known networks such as Efficient capsule, AlexNet, ResNet-18, and GoogleNet. This subject becomes important when we consider that the number of parameters in our proposed classifier is less than classifiers such as AlexNet, ResNet-18 and SIRE CNN [105].

Suggestions for future work include utilizing multi-objective optimization to optimize the parameters of a fuzzy system by considering a cost function that incorporates classification accuracy and F1-score. Another avenue for exploration involves utilizing a non-singleton Type-III fuzzy system as a decision-maker and employing posterior

Table 13 The ordinal rank of the superior outcomes achieved by five classifiers when evaluated against each other

Models	Metric	Feature							Mean rank
		API	DLL	REG	API + DLL	API + REG	DLL + REG	API + DLL + REG	
SVM	Accuracy	2	4	3	3	4	4	4	3.4286
	F-1 score	2	5	3	4	3	4	4	3.5714
Efficient-CapsNet	Accuracy	4	3	2	4	2	2	2	2.7143
	F-1 score	4	2	2	2	4	3	2	2.7143
ResNet-18	Accuracy	5	5	4	5	5	5	5	4.8571
	F-1 score	5	4	4	5	5	5	5	4.7143
DACN	Accuracy	3	2	5	2	3	3	3	3.0000
	F-1 score	3	1	5	3	1	2	3	2.5714
Proposed method	Accuracy	1	1	1	1	1	1	1	1.0000
	F-1 score	1	3	1	1	2	1	1	1.4286

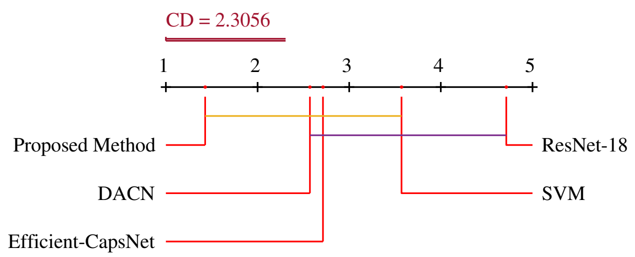


Fig. 29 The crucial charts for F1-score metric the outcomes derived from the Nemenyi post hoc examination, alongside the mean rank scores obtained from the Friedman test

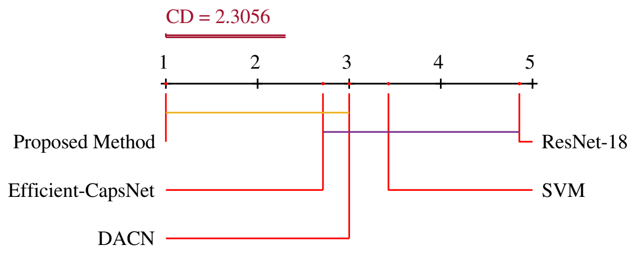


Fig. 30 The crucial charts for Accuracy metric the outcomes derived from the Nemenyi post hoc examination, alongside the mean rank scores obtained from the Friedman test

probabilities of classifiers instead of predicted classes by the classifiers.

Supplementary Information The online version contains supplementary material available at https://doi.org/10.1007/s10586-024-04475-7.

Acknowledgements This article is dedicated to all the kind mothers of the world, especially Shamsi, Shahnaz and Fattane, who are always the best supporters of their children, and we would like to thank the honorable reviewers who improved this article.

Author contributions N. M.H: Methodology, Writing – original draft and Formal analysis, M.H.A: Methodology, Writing – original draft and Formal analysis. A.M: Investigation and Validation, S.M: Supervision, Writing–Review and Editing, N.K: Writing–review & editing, Visualization and Validation.

Declarations

Competing interests The authors declare no competing interests.

References

1. Zedeh, L.A.: Knowledge representation in fuzzy logic. IEEE Trans. Knowl. Data Eng. **1**(1), 89–100 (1989). https://doi.org/10.1109/69.43406
2. Zadeh, L.A.: The concept of a linguistic variable and its application to approximate reasoning—I. Inf. Sci. (N. Y.) **8**(3), 199–249 (1975). https://doi.org/10.1016/0020-0255(75)90036-5
3. Mendel, J.M.: Advances in type-2 fuzzy sets and systems. Inf. Sci. (N. Y.) **177**(1), 84–110 (2007). https://doi.org/10.1016/j.ins.2006.05.003
4. Liang, Q., Mendel, J.M.: Interval type-2 fuzzy logic systems: theory and design. IEEE Trans. Fuzzy Syst. **8**(5), 535–550 (2000). https://doi.org/10.1109/91.873577
5. Sahab, N., Hagras, H., Sahab, N., Hagras, H.: Adaptive non-singleton type-2 fuzzy logic systems: a way forward for handling numerical uncertainties in real world applications. Int. J. Comput. Commun. Control (2011). https://doi.org/10.15837/ijccc.2011.3.2133
6. Wu, H., Mendel, J.M.: Uncertainty bounds and their use in the design of interval type-2 fuzzy logic systems. IEEE Trans. Fuzzy Syst. **10**(5), 622–639 (2002). https://doi.org/10.1109/TFUZZ.2002.803496
7. Mohammadzadeh, A., Sabzalian, M.H., Zhang, W.: An interval type-3 fuzzy system and a new online fractional-order learning algorithm: theory and practice. IEEE Trans. Fuzzy Syst. **28**(9), 1940–1950 (2020). https://doi.org/10.1109/TFUZZ.2019.2928509
8. Hua, G., Wang, F., Zhang, J., Alattas, K.A., Mohammadzadeh, A., The Vu, M.: A new type-3 fuzzy predictive approach for mobile robots. Mathematics (2022). https://doi.org/10.3390/math10173186
9. Alkabaa, A.S., Taylan, O., Balubaid, M., Zhang, C., Mohammadzadeh, A.: A practical type-3 fuzzy control for mobile robots: predictive and Boltzmann-based learning. Complex Intell. Syst. (2023). https://doi.org/10.1007/s40747-023-01086-4
10. Elhaki, O., Shojaei, K., Mohammadzadeh, A., Rathinasamy, S.: Robust amplitude-limited interval type-3 neuro-fuzzy controller for robot manipulators with prescribed performance by output feedback. Neural Comput. Appl. **35**(12), 9115–9130 (2023). https://doi.org/10.1007/s00521-022-08174-5
11. Amador-Angulo, L., Castillo, O., Melin, P., Castro, J.R.: Interval type-3 fuzzy adaptation of the bee colony optimization algorithm for optimal fuzzy control of an autonomous mobile robot. Micromachines (Basel) (2022). https://doi.org/10.3390/mi13091490
12. Castillo, O., Castro, J.R., Pulido, M., Melin, P.: Interval type-3 fuzzy aggregators for ensembles of neural networks in COVID-19 time series prediction. Eng. Appl. Artif. Intell. **114**, 105110 (2022). https://doi.org/10.1016/j.engappai.2022.105110
13. Melin, P., Sánchez, D., Castro, J.R., Castillo, O.: Design of type-3 fuzzy systems and ensemble neural networks for COVID-19 time series prediction using a firefly algorithm. Axioms (2022). https://doi.org/10.3390/axioms11080410
14. Wang, J., et al.: Non-singleton type-3 fuzzy approach for flowmeter fault detection: experimental study in a gas industry. Sensors (2021). https://doi.org/10.3390/s21217419
15. Peraza, C., Castillo, O., Melin, P., Castro, J.R., Yoon, J.H., Geem, Z.W.: A type-3 fuzzy parameter adjustment in harmony search for the parameterization of fuzzy controllers. Int. J. Fuzzy Syst. (2023). https://doi.org/10.1007/s40815-023-01499-w

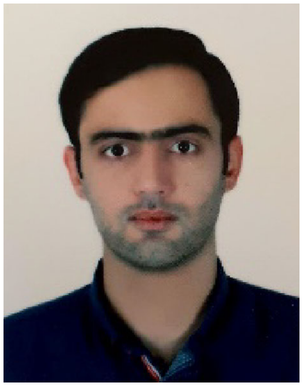
16. Zou, B., Cao, C., Tao, F., Wang, L.: IMCLNet: a lightweight deep neural network for image-based malware classification. J. Inf. Secur. Appl. **70**, 103313 (2022). https://doi.org/10.1016/j.jisa.2022.103313
17. Desktop Operating System Market Share Worldwide | Statcounter Global Stats. [Online]. https://gs.statcounter.com/os-market-share/desktop/worldwide/#monthly-202204-202204-bar. Accessed 14 July 2023
18. AV-ATLAS—Malware and PUA. [Online]. Available: https://portal.av-atlas.org/malware. Accessed 14 July 2023
19. Gibert, D., Mateu, C., Planes, J.: The rise of machine learning for detection and classification of malware: research developments, trends and challenges. J. Netw. Comput. Appl. (2020). https://doi.org/10.1016/j.jnca.2019.102526
20. Wong, M.Y., Lie, D.: IntelliDroid: a targeted input generator for the dynamic analysis of android malware. Internet Soc. (2017). https://doi.org/10.14722/ndss.2016.23118
21. Parildi, E.S., Hatzinakos, D., Lawryshyn, Y.: Deep learning-aided runtime opcode-based Windows malware detection. Neural Comput. Appl. **33**(18), 11963–11983 (2021). https://doi.org/10.1007/s00521-021-05861-7
22. Santos, I., Sanz, B., Laorden, C., Brezo, F., Bringas, P.G.: Opcode-sequence-based semi-supervised unknown malware detection. In: Herrero, Á., Corchado, E. (eds.) Computational intelligence in security for information systems, pp. 50–57. Springer, Berlin (2011)
23. Aslan, Ö., Yilmaz, A.A.: A new malware classification framework based on deep learning algorithms. IEEE Access **9**, 87936–87951 (2021). https://doi.org/10.1109/ACCESS.2021.3089586
24. Zhang, Y., Li, H., Zheng, Y., Yao, S., Jiang, J.: Enhanced DNNs for malware classification with GAN-based adversarial training. J. Comput. Virol. Hacking Tech. **17**(2), 153–163 (2021). https://doi.org/10.1007/s11416-021-00378-y
25. Mallik, A., Khetarpal, A., Kumar, S.: ConRec: malware classification using convolutional recurrence. J. Comput. Virol. Hacking Tech. **18**(4), 297–313 (2022). https://doi.org/10.1007/s11416-022-00416-3
26. Li, S., Zhou, Q., Zhou, R., Lv, Q.: Intelligent malware detection based on graph convolutional network. J. Supercomput. **78**(3), 4182–4198 (2022). https://doi.org/10.1007/s11227-021-04020-y
27. D'Angelo, G., Palmieri, F., Robustelli, A.: A federated approach to Android malware classification through Perm-Maps. Clust. Comput. **25**(4), 2487–2500 (2022). https://doi.org/10.1007/s10586-021-03490-2
28. Vasan, D., Alazab, M., Wassan, S., Naeem, H., Safaei, B., Zheng, Q.: IMCFN: image-based malware classification using fine-tuned convolutional neural network architecture. Comput. Netw. **171**, 107138 (2020). https://doi.org/10.1016/j.comnet.2020.107138
29. Hosseini, S., Nezhad, A.E., Seilani, H.: Android malware classification using convolutional neural network and LSTM. J. Comput. Virol. Hacking Tech. **17**(4), 307–318 (2021). https://doi.org/10.1007/s11416-021-00385-z
30. Kim, J., Ban, Y., Ko, E., Cho, H., Yi, J.H.: MAPAS: a practical deep learning-based android malware detection system. Int. J. Inf. Secur. **21**(4), 725–738 (2022). https://doi.org/10.1007/s10207-022-00579-6
31. Bakour, K., Ünver, H.M.: DeepVisDroid: android malware detection by hybridizing image-based features with deep learning techniques. Neural Comput. Appl. **33**(18), 11499–11516 (2021). https://doi.org/10.1007/s00521-021-05816-y
32. Hota, A., Panja, S., Nag, A.: Lightweight CNN-based malware image classification for resource-constrained applications. Innov. Syst. Softw. Eng. (2022). https://doi.org/10.1007/s11334-022-00461-7
33. Aurangzeb, S., Aleem, M.: Evaluation and classification of obfuscated Android malware through deep learning using ensemble voting mechanism. Sci. Rep. **13**(1), 3093 (2023). https://doi.org/10.1038/s41598-023-30028-w
34. Taha, A.A., Malebary, S.J.: Hybrid classification of Android malware based on fuzzy clustering and the gradient boosting machine. Neural Comput. Appl. **33**(12), 6721–6732 (2021). https://doi.org/10.1007/s00521-020-05450-0
35. Anupama, M.L., et al.: Detection and robustness evaluation of android malware classifiers. J. Comput. Virol. Hacking Tech. **18**(3), 147–170 (2022). https://doi.org/10.1007/s11416-021-00390-2
36. Alzubi, O.A., Alzubi, J.A., Al-Zoubi, A.M., Hassonah, M.A., Kose, U.: An efficient malware detection approach with feature weighting based on Harris Hawks optimization. Clust. Comput. **25**(4), 2369–2387 (2022). https://doi.org/10.1007/s10586-021-03459-1
37. Roy, S., Bhanja, S., Das, A.: AndyWar: an intelligent android malware detection using machine learning. Innov. Syst. Softw. Eng. (2023). https://doi.org/10.1007/s11334-023-00530-5
38. Seyfari, Y., Meimandi, A.: A new approach to android malware detection using fuzzy logic-based simulated annealing and feature selection. Multimed. Tools Appl. (2023). https://doi.org/10.1007/s11042-023-16035-z
39. Zou, B., Cao, C., Wang, L., Tao, F.: DACN: malware classification based on dynamic analysis and capsule networks. In: Communications in Computer and Information Science, pp. 3–13. Springer Science and Business Media Deutschland GmbH, Berlin (2022). https://doi.org/10.1007/978-981-19-0523-0_1
40. Sebastián, S., Caballero, J.: AVclass2: massive malware tag extraction from AV labels. In: Annual Computer Security Applications Conference, in ACSAC '20, pp. 42–53. Association for Computing Machinery, New York (2020). https://doi.org/10.1145/3427228.3427261
41. Li, C., et al.: DMalNet: dynamic malware analysis based on API feature engineering and graph learning. Comput. Secur. **122**, 102872 (2022). https://doi.org/10.1016/j.cose.2022.102872
42. Fernández-Álvarez, P., Rodríguez, R.J.: Module extraction and DLL hijacking detection via single or multiple memory dumps. Forensic Sci. Int.: Digit. Investig. **44**, 301505 (2023). https://doi.org/10.1016/j.fsidi.2023.301505
43. Gittins, Z., Soltys, M.: Malware persistence mechanisms. Procedia Comput. Sci. **176**, 88–97 (2020). https://doi.org/10.1016/j.procs.2020.08.010
44. Carvey, H.: Chapter 5—Registry analysis. In: Carvey, H. (ed.) Windows Forensic Analysis Toolkit, 4th edn., pp. 119–167. Syngress, Boston (2014). https://doi.org/10.1016/B978-0-12-417157-2.00005-9
45. Sabour, S., Frosst, N., Hinton, G.E.: Dynamic routing between capsules, Oct. 2017, [Online]. http://arxiv.org/abs/1710.09829
46. LaLonde, R., Xu, Z., Irmakci, I., Jain, S., Bagci, U.: Capsules for biomedical image segmentation. Med. Image Anal. **68**, 101889 (2021). https://doi.org/10.1016/j.media.2020.101889
47. Mocanu, I.G., Yang, Z., Belle, V.: Breaking CAPTCHA with capsule networks. Neural Netw. **154**, 246–254 (2022). https://doi.org/10.1016/j.neunet.2022.06.041
48. Kim, J., Jang, S., Park, E., Choi, S.: Text classification using capsules. Neurocomputing **376**, 214–221 (2020). https://doi.org/10.1016/j.neucom.2019.10.033
49. Bushara, A.R., VinodKumar, R.S., Kumar, S.S.: An ensemble method for the detection and classification of lung cancer using Computed Tomography images utilizing a capsule network with Visual Geometry Group. Biomed. Signal Process. Control **85**, 104930 (2023). https://doi.org/10.1016/j.bspc.2023.104930

50. Guarda, L., Tapia, J.E., Droguett, E.L., Ramos, M.: A novel Capsule Neural Network based model for drowsiness detection using electroencephalography signals. Expert Syst. Appl. **201**, 116977 (2022). https://doi.org/10.1016/j.eswa.2022.116977
51. Goldani, M.H., Momtazi, S., Safabakhsh, R.: Detecting fake news with capsule neural networks. Appl. Soft Comput. **101**, 106991 (2021). https://doi.org/10.1016/j.asoc.2020.106991
52. Wang, Y., et al.: RPI-CapsuleGAN: predicting RNA-protein interactions through an interpretable generative adversarial capsule network. Pattern Recognit. **141**, 109626 (2023). https://doi.org/10.1016/j.patcog.2023.109626
53. Ma, J., Li, J., Du, B., Wu, J., Wan, J., Xiao, Y.: Robust face alignment by dual-attentional spatial-aware capsule networks. Pattern Recognit. **122**, 108297 (2022). https://doi.org/10.1016/j.patcog.2021.108297
54. Mandal, B., Sarkhel, R., Ghosh, S., Das, N., Nasipuri, M.: Two-phase dynamic routing for micro and macro-level equivariance in multi-column capsule networks. Pattern Recognit. **109**, 107595 (2021). https://doi.org/10.1016/j.patcog.2020.107595
55. Sridhar, S., Sanagavarapu, S.: Multi-lane capsule network architecture for detection of COVID-19. In: 2021 2nd International Conference on Intelligent Engineering and Management (ICIEM), pp. 385–390 (2021). https://doi.org/10.1109/ICIEM51511.2021.9445363
56. Zhao, Z., Cheng, S.: Capsule networks with non-iterative cluster routing. Neural Netw. **143**, 690–697 (2021). https://doi.org/10.1016/j.neunet.2021.07.032
57. Zuo, X., Yuan, H., Yang, B., Wang, H., Wang, Y.: Exploring graph capsual network and graphormer for graph classification. Inf. Sci. (N Y) **640**, 119045 (2023). https://doi.org/10.1016/j.ins.2023.119045
58. Tao, J., Zhang, X., Luo, X., Wang, Y., Song, C., Sun, Y.: Adaptive capsule network. Comput. Vis. Image Underst. **218**, 103405 (2022). https://doi.org/10.1016/j.cviu.2022.103405
59. Wei, Y., Liu, Y., Li, C., Cheng, J., Song, R., Chen, X.: TC-Net: a transformer capsule network for EEG-based emotion recognition. Comput. Biol. Med. **152**, 106463 (2023). https://doi.org/10.1016/j.compbiomed.2022.106463
60. Mazzia, V., Salvetti, F., Chiaberge, M.: Efficient-CapsNet: capsule network with self-attention routing. Sci. Rep. (2021). https://doi.org/10.1038/s41598-021-93977-0
61. Raitoharju, J.: Chapter 3—Convolutional neural networks. In: Iosifidis, A., Tefas, A. (eds.) Deep learning for robot perception and cognition, pp. 35–69. Academic Press, Cambridge (2022). https://doi.org/10.1016/B978-0-32-385787-1.00008-7
62. Maitre, J., Bouchard, K., Gaboury, S.: Data filtering and deep learning for enhanced human activity recognition from UWB radars. J. Ambient. Intell. Humaniz. Comput. **14**(6), 7845–7856 (2023). https://doi.org/10.1007/s12652-023-04596-8
63. Kibriya, H., Amin, R.: A residual network-based framework for COVID-19 detection from CXR images. Neural Comput. Appl. **35**(11), 8505–8516 (2023). https://doi.org/10.1007/s00521-022-08127-y
64. Kingma, D.P., Ba, J.: Adam: a method for stochastic optimization. CoRR (2014). https://api.semanticscholar.org/CorpusID:662810
65. Wu, H., Xin, M., Fang, W., Hu, H.M., Hu, Z.: Multi-level feature network with multi-loss for person re-identification. IEEE Access **7**, 91052–91062 (2019). https://doi.org/10.1109/ACCESS.2019.2927052
66. Victor Ikechukwu, A., Murali, S., Deepu, R., Shivamurthy, R.C.: ResNet-50 vs VGG-19 vs training from scratch: a comparative analysis of the segmentation and classification of Pneumonia from chest X-ray images. Glob. Transit. Proc. **2**(2), 375–381 (2021). https://doi.org/10.1016/j.gltp.2021.08.027
67. Panigrahi, A., Subasi, A.: Chapter 3—Magnetic resonance imagining-based automated brain tumor detection using deep learning techniques. In: Subasi, A. (ed.) Applications of Artificial Intelligence in Medical Imaging, pp. 75–107. Academic Press, Cambridge (2023). https://doi.org/10.1016/B978-0-443-18450-5.00012-8
68. Sun, Z., Caetano, E., Pereira, S., Moutinho, C.: Employing histogram of oriented gradient to enhance concrete crack detection performance with classification algorithm and Bayesian optimization. Eng. Fail. Anal. **150**, 107351 (2023). https://doi.org/10.1016/j.engfailanal.2023.107351
69. Wang, B., Kang, Y., Huo, D., Chen, D., Song, W., Zhang, F.: Depression signal correlation identification from different EEG channels based on CNN feature extraction. Psychiatry Res. Neuroimaging **328**, 111582 (2023). https://doi.org/10.1016/j.pscychresns.2022.111582
70. Dou, T., Zhang, G., Cui, W.: Efficient quantum feature extraction for CNN-based learning. J. Frankl. Inst. **360**(11), 7438–7456 (2023). https://doi.org/10.1016/j.jfranklin.2023.06.003
71. Abbaskhah, A., Sedighi, H., Marvi, H.: Infant cry classification by MFCC feature extraction with MLP and CNN structures. Biomed. Signal Process. Control **86**, 105261 (2023). https://doi.org/10.1016/j.bspc.2023.105261
72. Bhattarai, B., Subedi, R., Gaire, R.R., Vazquez, E., Stoyanov, D.: Histogram of Oriented Gradients meet deep learning: a novel multi-task deep network for 2D surgical image semantic segmentation. Med. Image Anal. **85**, 102747 (2023). https://doi.org/10.1016/j.media.2023.102747
73. Déniz, O., Bueno, G., Salido, J., De la Torre, F.: Face recognition using histograms of oriented gradients. Pattern Recognit. Lett. **32**(12), 1598–1603 (2011). https://doi.org/10.1016/j.patrec.2011.01.004
74. Xiao, C., Liu, Z., Zhang, T., Zhang, L.: On fault prediction for wind turbine pitch system using radar chart and support vector machine approach. Energies (Basel) (2019). https://doi.org/10.3390/en12142693
75. Malan, N.S., Sharma, S.: Feature selection using regularized neighbourhood component analysis to enhance the classification performance of motor imagery signals. Comput. Biol. Med. **107**, 118–126 (2019). https://doi.org/10.1016/j.compbiomed.2019.02.009
76. Jin, M., Deng, W.: Predication of different stages of Alzheimer's disease using neighborhood component analysis and ensemble decision tree. J. Neurosci. Methods **302**, 35–41 (2018). https://doi.org/10.1016/j.jneumeth.2018.02.014
77. Lei, Y.: 3—Individual intelligent method-based fault diagnosis. In: Lei, Y. (ed.) Intelligent fault diagnosis and remaining useful life prediction of rotating machinery, pp. 67–174. Butterworth-Heinemann, Oxford (2017). https://doi.org/10.1016/B978-0-12-811534-3.00003-2
78. Castillo, O., Castro, J.R., Melin, P.: Interval type-3 fuzzy aggregation of neural networks for multiple time series prediction: the case of financial forecasting. Axioms (2022). https://doi.org/10.3390/axioms11060251
79. Ghanbarzadeh, R., Hosseinalipour, A., Ghaffari, A.: A novel network intrusion detection method based on metaheuristic optimisation algorithms. J. Ambient. Intell. Humaniz. Comput. **14**(6), 7575–7592 (2023). https://doi.org/10.1007/s12652-023-04571-3
80. Bu, S.-J., Cho, S.-B.: Malware classification with disentangled representation learning of evolutionary triplet network. Neurocomputing **552**, 126534 (2023). https://doi.org/10.1016/j.neucom.2023.126534
81. Vaiyapuri, T., et al.: Metaheuristics with federated learning enabled intrusion detection system in Internet of Things

environment. Expert. Syst. **40**(5), e13138 (2023). https://doi.org/10.1111/exsy.13138
82. Mora, A.M., Arenas, M.G., Romero-Horno, A., Camacho-Páez, J., Castillo, P.A.: Optimizing an IDS (intrusion detection system) by Means of advanced metaheuristics. In: Rojas, I., Joya, G., Catala, A. (eds.) Advances in Computational Intelligence, pp. 55–67. Springer Nature Switzerland, Cham (2023)
83. Bacanin, N., et al.: Addressing feature selection and extreme learning machine tuning by diversity-oriented social network search: an application for phishing websites detection. Complex Intell. Syst. **9**(6), 7269–7304 (2023). https://doi.org/10.1007/s40747-023-01118-z
84. Savanović, N., et al.: Intrusion detection in healthcare 4.0 Internet of Things systems via metaheuristics optimized machine learning. Sustainability (2023). https://doi.org/10.3390/su151612563
85. Jovanovic, L., et al.: Improving phishing website detection using a hybrid two-level framework for feature selection and XGBoost tuning. J. Web Eng. **22**(03), 543–574 (2023). https://doi.org/10.13052/jwe1540-9589.2237
86. Talatahari, S., Azizi, M.: Chaos Game Optimization: a novel metaheuristic algorithm. Artif. Intell. Rev. **54**(2), 917–1004 (2021). https://doi.org/10.1007/S10462-020-09867-W/METRICS
87. Azizi, M., Aickelin, U., Khorshidi, H.A., Shishehgarkhaneh, M.B.: Shape and size optimization of truss structures by Chaos game optimization considering frequency constraints. J. Adv. Res. **41**, 89–100 (2022). https://doi.org/10.1016/j.jare.2022.01.002
88. Kaveh, A., Dadras, A.: A novel meta-heuristic optimization algorithm: thermal exchange optimization. Adv. Eng. Softw. **110**, 69–84 (2017). https://doi.org/10.1016/j.advengsoft.2017.03.014
89. Price, K.V., Awad, N.H., Ali, M.Z., Suganthan, P.N.: The 100-Digit Challenge: Problem Definitions and Evaluation Criteria for the 100-Digit Challenge Special Session and Competition on Single Objective Numerical Optimization, vol. 1, pp. 1–21. Nanyang Technological University, Singapore (2018)
90. Talatahari, S., Azizi, M.: Chaos Game Optimization: a novel metaheuristic algorithm. Artif. Intell. Rev. **54**(2), 917–1004 (2021). https://doi.org/10.1007/s10462-020-09867-w
91. Wu, L., Wu, J., Wang, T.: Enhancing grasshopper optimization algorithm (GOA) with levy flight for engineering applications. Sci. Rep. **13**(1), 124 (2023). https://doi.org/10.1038/s41598-022-27144-4
92. Al-Dhief, F.T., Latiff, N.M.A., Baki, M.M., Malik, N.N.N.A., Sabri, N., Albadr, M.A.A.: Voice pathology detection using support vector machine based on different number of voice signals. In: 2021 26th IEEE Asia-Pacific Conference on Communications (APCC), pp. 1–6 (2021). https://doi.org/10.1109/APCC49754.2021.9609830
93. Al-Dhief, F.T., et al.: Voice pathology detection using machine learning technique. In: 2020 IEEE 5th International Symposium on Telecommunication Technologies (ISTT), pp. 99–104 (2020). https://doi.org/10.1109/ISTT50966.2020.9279346
94. Al-Dhief, F.T., et al.: Voice pathology detection and classification by adopting online sequential extreme learning machine. IEEE Access **9**, 77293–77306 (2021). https://doi.org/10.1109/ACCESS.2021.3082565
95. LeCun, Y., Cortes, C.: The MNIST Database of Handwritten Digits (2005). https://api.semanticscholar.org/CorpusID:60282629
96. Xiao, H., Rasul, K., Vollgraf, R.: Fashion-MNIST: A Novel Image Dataset for Benchmarking Machine Learning Algorithms (2017). https://doi.org/10.48550/arXiv.1708.07747
97. Ahmed, M., Afreen, N., Ahmed, M., Sameer, M., Ahamed, J.: An inception V3 approach for malware classification using machine learning and transfer learning. Int. J. Intell. Netw. **4**, 11–18 (2023). https://doi.org/10.1016/j.ijin.2022.11.005
98. Yoo, S., Kim, S., Kim, S., Kang, B.B.: AI-HydRa: advanced hybrid approach using random forest and deep learning for malware classification. Inf. Sci. (N. Y.) **546**, 420–435 (2021). https://doi.org/10.1016/j.ins.2020.08.082
99. Liu, C., Lu, J., Feng, W., Du, E., Di, L., Song, Z.: MobiPCR: efficient, accurate, and strict ML-based mobile malware detection. Future Gener. Comput. Syst. **144**, 140–150 (2023). https://doi.org/10.1016/j.future.2023.02.014
100. Hao, J., Luo, S., Pan, L.: EII-MBS: malware family classification via enhanced adversarial instruction behavior semantic learning. Comput. Secur. **122**, 102905 (2022). https://doi.org/10.1016/j.cose.2022.102905
101. Gao, H., Cheng, S., Zhang, W.: GDroid: Android malware detection and classification with graph convolutional network. Comput. Secur. **106**, 102264 (2021). https://doi.org/10.1016/j.cose.2021.102264
102. Gao, X., Hu, C., Shan, C., Han, W.: MaliCage: a packed malware family classification framework based on DNN and GAN. J. Inf. Secur. Appl. **68**, 103267 (2022). https://doi.org/10.1016/j.jisa.2022.103267
103. Kumar, S., Janet, B.: DTMIC: deep transfer learning for malware image classification. J. Inf. Secur. Appl. **64**, 103063 (2022). https://doi.org/10.1016/j.jisa.2021.103063
104. Erkan, U.: A precise and stable machine learning algorithm: eigenvalue classification (EigenClass). Neural Comput. Appl. **33**(10), 5381–5392 (2021). https://doi.org/10.1007/s00521-020-05343-2
105. Avola, D., Cinque, L., Fagioli, A., Foresti, G.L.: SIRe-Networks: convolutional neural networks architectural extension for information preservation via skip/residual connections and interlaced auto-encoders. Neural Netw. **153**, 386–398 (2022). https://doi.org/10.1016/j.neunet.2022.06.030

Nastaran Mehrabi Hashjin is a researcher who recently completed her M.Sc. in Control Engineering at Shahid Beheshti University. With a passion for advancing technology, she has devoted her research to exploring the intersection of machine learning, artificial intelligence, mathematical optimization, systems, and control. She is committed to solving complex problems using cutting-edge technology. Her current work focuses on engineering optimization and designing neural networks to solve various problems in industry and medicine. With her expertise in control engineering and her innovative approach to research, she is poised to contribute to the engineering and technology field.

Mohammad Hussein Amiri received his B.Sc. in July 2021 from Ayatollah Borujerdi University in electrical engineering, Borujerd, Iran; His M.Sc. in January 2024 from Shahid Beheshti University, Tehran, Iran, in control engineering. He reviews several journals like Applied Soft Computing, Expert Systems with Application, Alexandria Engineering Journal, Knowledge Based Systems, and Soft Commuting Journal. His research interests include Neural Networks, Machine learning, Deep learning, Fault isolation, and Fuzzy Systems.



Ardashir Mohammadzadeh received his B.Sc. in July 2011 from Sahand University of Technology, Tabriz, Iran; His M.Sc. in September 2013 from K.N Toosi University of Technology, Tehran, Iran; and his Ph.D. in November 2016 from the University of Tabriz, Tabriz, Iran. Since December 2017, he has worked as an assistant/associate professor at the University of Bonab, Bonab, Iran as an assistant/associate professor. He joined the Shenyang University of Technology as a professor in 2022. He has published many papers in most reputed journals. He is an academic editor and reviewer for several journals. His research interests include control theory, fuzzy logic systems, machine learning, neural networks, intelligent control, electric vehicles, power systems control, chaotic systems, and medical systems.

Seyedali Mirjalili is currently a Professor and the Founding Director of the Centre for Artificial Intelligence Research and Optimization, Torrens University Australia. He is internationally recognized for his advances in optimization and swarm intelligence, including the first set of algorithms from a synthetic intelligence standpoint-a radical departure from how natural systems are typically understood a systematic design framework to reliably benchmark, evaluate, and propose computationally cheap robust optimization algorithms. He has published over 400 publications with over 105,000 citations and an H-index of 109. As the most cited researcher in robust optimization, he has been on the list of 1% highly-cited researchers and named as one of the most influential researchers in the world by Web of Science for three consecutive years, since 2019. In 2020, he was ranked 21st across all disciplines and 4th in artificial intelligence and image processing in the Stanford University's list of World's Top Scientists. In 2021, The Australian newspaper named him as the top researcher in Australia in three fields: artificial intelligence, evolutionary computation, and fuzzy systems. His research interests include optimization, swarm intelligence, evolutionary algorithms, and machine learning. He is an Associate Editor of several AI journals, including Neurocomputing, Applied Soft Computing, Advances in Engineering Software, Computers in Biology and Medicine, Healthcare Analytics, Applied Intelligence, and IEEE ACCESS.

Nima Khodadadi is currently a research assistant, and he is studying for his 2nd Ph.D. at the University of Miami (UM), Miami, USA. He has published over 70 publications with over 2500 citations. He graduated with a bachelor's degree in civil engineering and a master's degree in structural, and civil engineering from the University of Tabriz (One of the top ten universities in Iran). He is currently studying for his 2nd Ph.D. at the University of Miami (UM). He got his 1st Ph.D. at the Iran University of Science and Technology (IUST). He worked on the subject of steel structures, particularly in the experimental and numerical investigation of Steel braced frames with Dr. Siamak Talatahari. In addition, he has been actively involved in engineering optimization, especially Evolutionary algorithms. His main research interests lie in Artificial Intelligence, Machine Learning, FRP, Concrete, and Metaheuristic algorithms. In addition, he has been actively involved in engineering optimization, especially evolutionary algorithms. He has been engaged in research in single and multi-objective engineering optimization, especially in solving large-scale and practical structural design problems. Recently, he has been working on artificial Intelligence techniques and applications for FRP-Integrated concrete structures.

Authors and Affiliations

Nastaran Mehrabi Hashjin[1] · Mohammad Hussein Amiri[1] · Ardashir Mohammadzadeh[5] · Seyedali Mirjalili[2,3] · Nima Khodadadi[4]

✉ Nima Khodadadi
nima.khodadadi@miami.edu

Nastaran Mehrabi Hashjin
na.mehrabi@mail.sbu.ac.ir

Mohammad Hussein Amiri
mohamm.amiri@mail.sbu.ac.ir

Ardashir Mohammadzadeh
a.mzadeh@sut.edu.cn

Seyedali Mirjalili
ali.mirjalili@gmail.com

1 Faculty of Electrical Engineering, Shahid Beheshti University, Tehran, Iran

2 University Research and Innovation Center (EKIK), Ubuda University, Budapest, Hungary

3 Centre for Artificial Intelligence Research and Optimisation, Torrens University Australia, Brisbane, Australia

4 Department of Civil and Architectural Engineering, University of Miami, Coral Gables, FL, USA

5 Department of Computational and Data Science, Astana IT University, Astana, Kazakhstan