

Enhancing Image-Based Malware Classification: A Feature Fusion Approach With Dwarf Mongoose Optimization Algorithm

Kajal Jaisinghani¹, Dr. Santosh Singh²

¹Research Scholar, UDIT, University of Mumbai, Mumbai, India

²PhD Guide, UDIT, University of Mumbai, Mumbai, India

Abstract

The rapid advancement in networks and communication technologies has significantly increased the complexity of systems, resulting in a surge of sophisticated malware designed to evade detection. Traditional methods of manually examining malware are time-consuming and often ineffective. In response to these challenges, this research introduces an automated image-based malware classification technique using an Optimal Deep Learning-Based Feature Fusion Approach (AIBMC-ODLFFA). The primary goal of the AIBMC-ODLFFA technique is to accurately identify and classify malware by utilizing a feature fusion method that integrates two deep learning models: Capsule Network (CapsNet) and SimAM-Efficient Net. These models extract comprehensive features from malware images, which are then classified using a Deep Belief Network (DBN). The performance of the DBN is further enhanced through hyperparameter tuning using the Dwarf Mongoose Optimization Algorithm (DMOA). Experimental evaluations conducted on the Maling malware dataset demonstrate that the AIBMC-ODLFFA approach significantly outperforms existing methods in terms of accuracy and computational efficiency. The results indicate that this technique holds great potential for practical application in automated and precise malware detection systems.

Keywords: Malware Detection, Feature Fusion, Hyperparameter Tuning, Deep Belief Network, CapsNet, SimAM-EfficientNet.

1. INTRODUCTION

Recent research highlights the alarming surge in malicious software (malware), with cybercriminals employing sophisticated obfuscation techniques to evade detection [1]. As the internet becomes increasingly ubiquitous, the risk of cyber-attacks grows, necessitating early malware detection to prevent widespread damage. The evolution of malware from basic encrypted viruses to advanced polymorphic and metamorphic variants presents significant detection challenges [2]. One prominent threat is ransomware, which targets system files and demands substantial ransoms for decryption [3]. The rise of ransomware underscores the need for robust detection and mitigation strategies, particularly in vulnerable sectors like unattended automated systems and critical infrastructure [4].

The dynamic nature of cybercrime, coupled with limited awareness, fuels its expansion, with projected global cybercrime costs reaching staggering figures by 2025 [5]. It is essential to distinguish between the broader concept of cybercrime and the specific realm of computer misuse, which includes activities like hacking and virus-related offenses [6]. Despite consistent reporting of computer misuse crimes, non-financial and non-physical impacts are often underestimated, highlighting the need for further investigation and awareness [7]. Malware poses significant threats to mobile devices and the healthcare industry, exploiting vulnerabilities in operating systems and network infrastructure [8]. Android malware, ranging from worms to spyware, underscores the need for proactive user protection and robust cybersecurity measures. In the healthcare sector, ransomware attacks pose severe operational and financial risks, emphasizing the urgency of enhancing cybersecurity systems and staff training [9].

Detecting malware involves various methodologies, including static, dynamic, and hybrid analysis [10]. Static analysis examines code without execution, while dynamic analysis

observes software behavior during execution, and hybrid analysis combines both approaches for comprehensive detection [11]. Signature-based, heuristic-based, and specification-based detection are common approaches, each offering unique insights into malware behavior and characteristics [12].

Machine learning (ML) has revolutionized malware detection by automating predictions based on historical data, encompassing various paradigms and learning methods [13]. In Android malware detection, ML enhances accuracy by training models on diverse features, including permissions, API calls, app ratings, and download statistics [14]. Supervised ML classifiers, combined with algorithms like Decision Trees and Naïve Bayes, offer effective solutions for identifying and classifying new applications [15].

Deep learning (DL), a subset of AI, employs deep neural networks to emulate human brain learning processes, demonstrating remarkable success in various domains, including cybersecurity [16]. DL techniques, such as Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs), are particularly effective in Android malware detection, leveraging features like opcodes, APIs, and visual images [17]. The fusion of DL with traditional ML methods leads to innovative frameworks like "DL-Droid," showcasing superior performance in static and dynamic feature scenarios [18].

Image-based malware detection offers a promising avenue to combat evolving threats, circumventing the limitations of traditional signature-based approaches [19]. By integrating computer vision and machine learning, hybrid frameworks demonstrate superior performance in classifying malware, leveraging image-derived attributes to automate assessment [20]. Deep learning techniques, particularly CNNs, enable effective visualization and feature extraction from malware binaries, enhancing detection accuracy [21]. Efforts in malware classification through image processing have shown notable success, with techniques such as grayscale representations and CNN-based approaches achieving high accuracies [22].

This paper is structured as follows: Section 2 presents a comprehensive literature review, offering an in-depth survey of existing ML and DL-based malware classification techniques utilizing image representation. In Section 3, we discuss the proposed model, focusing on the design and development of the AIBMC-ODLFFA technique for malware image classification. Section 4 presents the results and discussion, showcasing the experimental outcomes of the AIBMC-ODLFFA approach tested on the Malimg malware dataset. Finally, Section 5 concludes the paper, summarizing the key findings and implications of the study.

2. LITERATURE REVIEW

Recent research in malware detection highlights several key findings and future directions. Nunes et al. [23] examined classifier bias in dynamic malware analysis, focusing on evasive behavior. They found that classifiers often misidentified ransomware as malicious due to significant evasive actions, suggesting the need for improved feature selection and malware simulators. Avci et al. [24] evaluated various LSTM network variants for malware prediction, finding that optimized vanilla and bidirectional LSTM models were most effective, offering valuable insights for cybersecurity practitioners. Dener et al. [25] addressed malware threats using a big data approach with the CIC-MalMem-2022 dataset, identifying Logistic Regression as the top performer with a 99.97% accuracy rate. Yang et al. [26] proposed an Android malware detection method based on permission complement and API calls, which effectively detected dynamic code loading malware. Aslan et al. [27] discussed the increasing complexity of malware attacks

and the necessity for innovative detection methods, emphasizing cloud-based solutions and the ongoing need for advancements in malware detection technologies.

Alhaidari et al. [28] introduced ZeVigilante, a framework for Zero-Day malware detection using machine learning and sandboxing within the Cuckoo sandbox, achieving superior accuracy. Gopinath & Sethuraman [17] emphasized deep learning-based malware detection as superior to traditional methods, providing a comprehensive review across various platforms and malware types. Hemalatha et al. [29] developed a highly accurate malware detection method using a Dense Net model, showing resilience against obfuscated malware. Daoudi et al. [22] evaluated DexRay, an image-based Android malware detection method, demonstrating high detection rates and robustness. Geremias et al. [30] used sophisticated image-based deep learning techniques for Android malware detection, improving accuracy through feature reduction and addressing image construction challenges.

Bakour and Ünver [31] presented Deep Vis Droid, a CNN model for Android malware classification, achieving a classification accuracy of 98.96% by combining hybrid image features and 1D-convolutional layers. Panda et al. [32] developed SE-AGM, an ensemble of neural network models, which attained a 99.43% accuracy rate on the Mallmg dataset, showcasing the effectiveness of lightweight models in malware detection. Mahindru and Sangal [14] created a robust malware detection framework for Android apps using refined feature selection techniques, achieving a 98.8% detection rate for known and unknown malware. Shatnawi et al. [33] proposed a static-based classification method using permission and API call features, with SVM achieving the highest accuracy rates of 94% and 83%, respectively. Akhtar and Feng [34] evaluated three ML algorithms for malware detection, finding that the Decision Tree (DT) achieved the highest accuracy of 99%, followed by CNN with 98.76% and SVM with 96.41%, with DT also showing the lowest false positive rate.

Vasan, Hammoudeh, and Alazab [35] developed GPU-free Image-based Malware Classification using Broad Learning (IMCBL), transforming malware binaries into images and achieving high accuracy and processing efficiency. Haile, Walle, and Belay [36] presented an ensemble model using SVM and LASSO feature selection, attaining 99.78% accuracy and computational efficiency. Bikku et al. [37] demonstrated the superior accuracy of Quantum Neural Networks (QNNs) for real-time malware analysis, emphasizing their potential in cybersecurity through advanced quantum computing techniques.

Godwin Olaoye [38] analyzed the role of AI and ML in identifying malware in cloud systems, highlighting the speed and accuracy improvements these technologies bring to malware detection. Hashmi [39] examined the use of Back Propagation in Enhanced Convolutional Neural Networks (ECNN) for Android malware detection, achieving notable accuracy and precision of up to 96.92%. Laouadi, Menacer, and Benatchba [40] employed CNN architectures like VGG, ResNet, and Inception with Transfer Learning for malware detection, finding Grayscale + ResNet to be the most effective strategy with a testing accuracy of 90.08%.

3. RESEARCH GAPS

Despite advancements in malware detection methodologies, several critical gaps persist that warrant further investigation. Firstly, while current studies have demonstrated the efficacy of deep learning-based approaches for malware classification, there is a noticeable lack of comprehensive comparative analyses with other contemporary models.

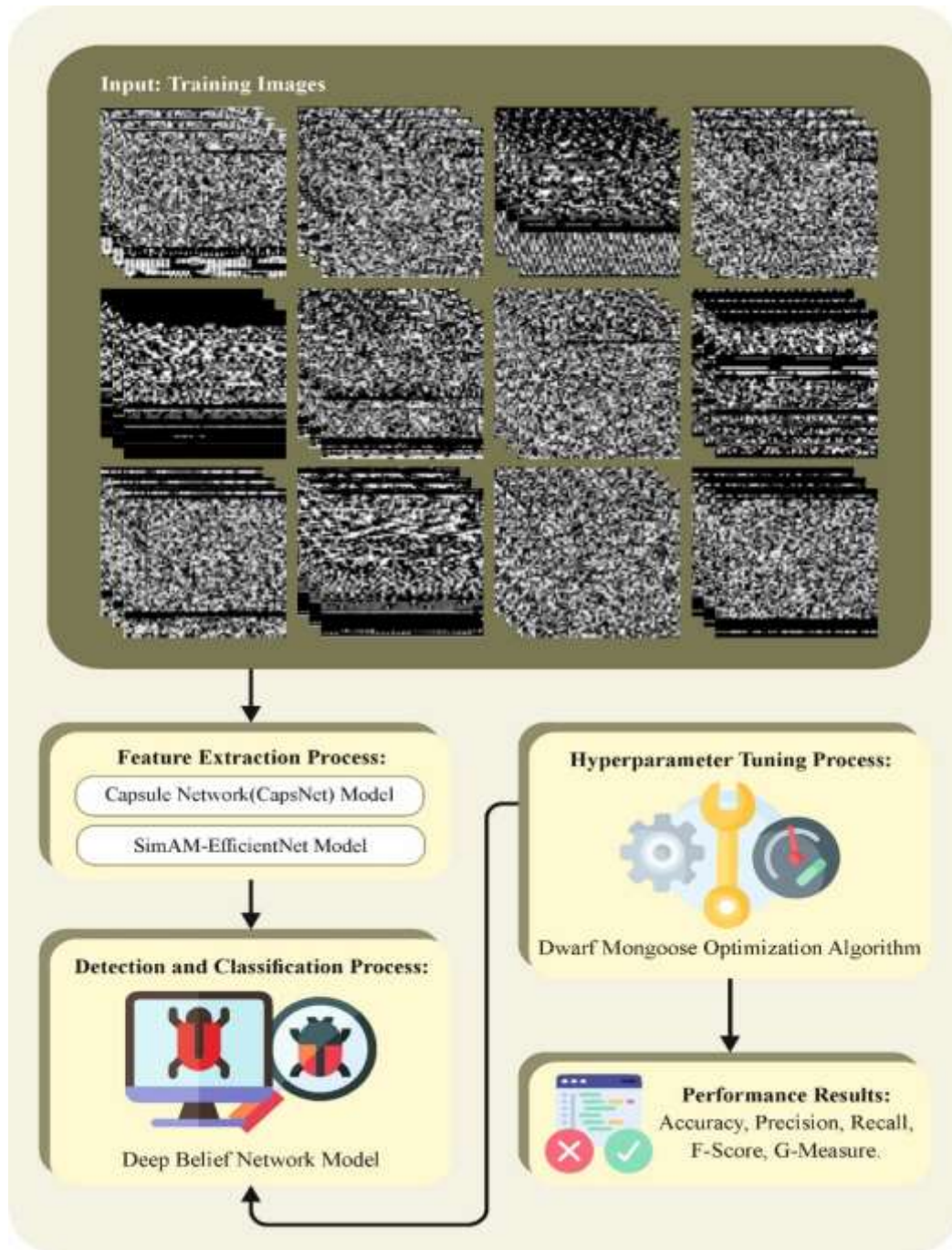
This gap is significant because a thorough comparative study under standardized conditions could provide a clearer understanding of the relative strengths and weaknesses of various models. Moreover, the scalability and real-world applicability of these models remain underexplored. Existing studies often focus on controlled experimental settings, which may not accurately reflect real-world scenarios where new, unseen malware variants frequently emerge. Future research should aim to deploy these models in live environments to assess their adaptability and performance in dynamic, real-world conditions.

Another major gap identified is the insufficient focus on handling advanced obfuscation and polymorphism techniques used by modern malware. Robust feature extraction and classification mechanisms are critical, yet current methodologies may fall short in effectively detecting highly obfuscated malware. Current techniques need enhancement to address these sophisticated evasion techniques comprehensively. One potential direction for future research is the integration of dynamic analysis or hybrid static-dynamic analysis methods to improve detection rates of obfuscated malware. Furthermore, the process of feature fusion in deep learning approaches, though innovative, can benefit from further optimization. Exploring various fusion strategies, such as weighted fusion or ensemble methods, could significantly enhance the robustness and accuracy of malware classification models.

Lastly, the discussion on post-classification analysis, such as interpretability of model outputs and handling false positives, is limited. While many studies provide a solid foundation for malware classification, they often lack detailed analysis on the interpretability of the results, which is crucial for practical applications and decision-making processes. Understanding why a model makes certain predictions can help in refining the model and making it more reliable. Additionally, the management of false positives and negatives is crucial, especially in cybersecurity, where false alarms can lead to resource wastage and missed detections can have severe consequences. Future studies should focus on developing techniques to enhance the interpretability of model outputs and strategies to effectively manage false positives and negatives, thereby improving the overall reliability and usability of malware detection systems.

4. PROPOSED MODEL

In this manuscript, we have focused on the design and growth of the AIBMC-ODLFFA technique for malware image classification. The AIBMC-ODLFFA methodology utilizes hyperparameter tuned DL approach for detecting and classifying the malware families. To accomplish this, the AIBMC-ODLFFA system deploys feature fusion based feature extractor, DBN based classification, and DMOA based hyperparameter tuning. Fig. 1 exemplifies the entire flow of AIBMC-ODLFFA system.

Fig. 1. Overall flow of AIBMC-ODLFFA methodology

4.1. Feature Fusion based Feature Extraction

The AIBMC-ODLFFA model integrates features extracted from CapsNet and SimAM-Efficient Net for enhanced malware detection. Let's signify the feature vectors extracted from CapsNet and SimAM-Efficient Net as F_{CapsNet} and $F_{\text{SimAM-EfficientNet}}$ correspondingly. The fusion procedure is described as mathematical notation. The feature extraction process of the CapsNet and SimAM-EfficientNet models are given as follows.

- CapsNet output: $F_{\text{CapsNet}} = [f_{\text{CapsNet},1}, f_{\text{CapsNet},2}, \dots, f_{\text{CapsNet},m}]$, where m denotes the amount of features extracted by CapsNet.
- SimAM-EfficientNet output: $F_{\text{SimAM-EfficientNet}} = [f_{\text{SimAM-EfficientNet},1}, f_{\text{SimAM-EfficientNet},2}, \dots, f_{\text{SimAM-EfficientNet},n}]$, where n signifies the amount of features extracted by SimAM-EfficientNet.

Here, a weighted fusion can be implemented, where each feature is multiplied by a weight and then totaled.

$$F_{\text{fusion}} = \alpha \cdot F_{\text{CapsNet}} + \beta \cdot F_{\text{SimAM-EfficientNet}} \quad (1)$$

where, α and β denote the weights allocated to the features of CapsNet and SimAM-EfficientNet, correspondingly. The weights can be defined via optimizer or pre-defined depending on the characteristics of features. The resulting F_{fusion} signifies the fused feature vector, which is then utilized for following tasks like malware classification. Regulating the fusion parameters permits flexibility in uniting the powers of CapsNet and SimAM-EfficientNet in capturing related designs for effectual malware classification.

4.1.1. CapsNet Model

Deep learning (DL) based CNN methods are commonly utilized for feature extractor, recognition, identification, etc [41]. The convolution processes in CNN make it very easy to resolve difficult issues. Furthermore, CNN does not reflect the direction of modules and relation in image feature space but only cautions regarding the occurrence of features. In 2017, Sabour et al developed CapsNet to diminish above-mentioned CNN tasks and signify a model of visual objects. Capsules in the CapsNet are united neurons that display the action vectors signifying the present pose parameter. The vector length directs the presence of a unit. One main issue of CNNs is connected to the pool layers. Therefore, CapsNet has swapped pool layers with an effective method termed routing by contract. Fig. 2 represents the infrastructure of CapsNet. Every capsule in the low layer forecasts the parent capsule output. If the forecast equals the parent capsule's result, then the coupling co-efficient is amplified. Let u_i denotes capsule output i and its forecast from parent capsules j has been stated as $s_j = \sum_i c_j \hat{u}_{j|i}$.

A non-linear function is employed to analyze longer and shorter vectors to one and zero correspondingly. This is known as squash function which is intended to avoid the result vectors from beyond 1. Eq. (2) displays the non-linear function of squash.

$$v_j = \frac{\|s_j\|^2}{1 + \|s_j\|^2} \frac{s_j}{\|s_j\|} \quad (2)$$

From the above-mentioned formula, s_j denotes the input vector to the j th capsules and v_j refers to output vectors. CapsNet accepts non-linear function of squashing on resultant vectors () in every iteration. This displays the probability of the vector among (0,1) that means it squashes smaller vectors and keeps longer vectors within the unit length.

$$v_j \approx \|s_j\| s_j = 0 \approx \frac{\|s_j\|}{\|s_j\|} \quad (3)$$

The log-probability was upgraded within the routing procedure depending upon the contract among v_j for the point that the contract amid dual vectors will be enlarged and contain larger inner product. So, agreement a_{ij} for upgrading the coupling co-efficient and log-probability is definite as follows:

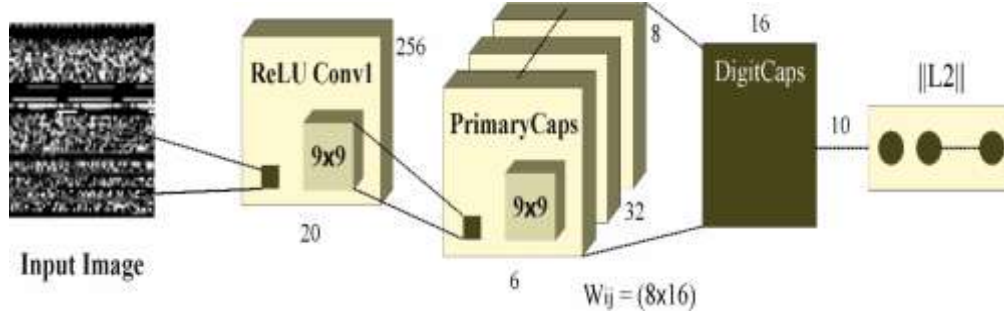
$$a_{ij} = v_j \hat{u}_{j|i} \quad (4)$$

In the last layer, Capsule k is related by a loss l_k . When the entity does not occur, this sets huge loss values with longer output instantiation parameters. The loss function l_k has been stated as,

$$l_k = T_k \max(0, m^+ - ||v_k||)^2 + \lambda(1 - T_k) \max(0, ||v_k|| - m^-)^2 \quad (5)$$

Whereas, the value of T_k is 1 when class k exists or else it is 0. The $+$, m^- , and λ denotes hyperparameters which are fixed previously in the learning procedure.

Fig. 2. Framework of CapsNet model



3.1.2. SimAM-EfficientNet Model

In several existing attention modules make 1D or 2D weights. Then, the created weights have been extended for spatial attention and channel [42]. Several modules contain two issues. The primary is that they can only able to remove features by network and space that mains to the elasticity of attention weights. Besides, the designs of many CNNs are difficult and inclined by a sequence of aspects. It develops 3D attention weights deprived of including any limits to the original system. Accurately, it describes an energy function depending upon theory of neuroscience and originates a solution which is rapidly joined. Ten lines of code execute this procedure. Another main benefit of SimAM is that it evades too many alterations to the design. So, SimAM is further modular, stretchy, and light-weight. In several states, SimAM is great when compared to CBAM and SE attention modules.

The SimAM element appears for significant neurons and describes an energy function. It employs dual labels and increases normal terms. Lastly, the minimum energy is intended as follows:

$$e_t^* = (4(\lambda + \sigma^2)) / ((t - u)^2 + 2\sigma^2 + 2\lambda) \quad (6)$$

Where

$$u_t = \frac{1}{M-1} \sum_{i=1}^{M-1} x_i, \sigma_t^2 = \frac{1}{M-1} \sum_{i=1}^{M-1} (x_i - u_t)^2 \quad (7)$$

Between them, μ_t denotes the mean. σ_t^2 refers to the variance. t signifies the target neuron. x_i represents other neurons of input feature. λ refers to the regularized coefficient. The amount of neurons is attained by $M = H \times W$. Lastly, the variation among neurons and peripheral neurons have been linked to energy. The importance of every neuron is considered by $1/e^*$. It employs a scaling operation to improve features. The complete modification stage of the SimAM unit is as follows:

$$X = X \cdot \text{sigmoid}\left(\frac{1}{E}\right) \quad (8)$$

Whereas, E clusters all e through the channel and spatial sizes. It employs sigmoid function to bind the extent of value from E .

The EfficientNet-B0 totally has 9 phases. The initial phase is a 3x3 convolution (conv) layer. The 2nd-8th phases are MBConv that is most significant. The final phase is collected of 1x1 a pooling layer, conv layer, and fully connected (FC) layer. MBConv contains 5 parts. The 1st part is a 1x1 conv layer. The 2nd part is a depth-wise conv which is surveyed by SE attention module. The 4th part is a 1x1 conv layer for dimensional decrease. Lastly, the

dropout layer is last one to improve over-fitting issue. The SimAM was included after the first conv layer to enhance spatial weights and channels. MBConv is a basic structure. The remaining structure is correctly adapted by it. Our technique includes the SimAM attention in order to enhance it. The final SimAM-EfficientNet has been attained afterward by adapting MBConv.

During this research, SimAM-EfficientNet is collected of 7 enhanced SimAM-MBConv components, 2 conv layers, 1 FC layer, and 1 pooling layer. To start with, the images with sizes $224 \times 224 \times 3$ are risen by the 3×3 conv layer. The sizes of the attained images with features are $112 \times 112 \times 32$. The following step is removing the image features by employing SimAM-Conv. After 1×1 pointwise conv, the original network is reinstated, and the FC layer has been employed for the main purpose of classification.

4.2. DBN based Malware Classification

At this stage, the DBN model can be employed for the recognition and classification of malware families. DBN is a probabilistic generalization model [43]. DBN is composed of a regression layer and multiple RBM connections and is generated by finetuning the resultant deep network via backpropagation (BP) and gradient descent to make it the optimum model. As the building block of DBN, RBM is a generalization random ANN, which can learn probability distribution within the input. The v visible layer (VL) and h hidden layer (HL) are two layers of RBM. The VL receives training dataset. The VL accepts the preferred location of the current robot joint angle and the end from the robot co-ordinate method. The input of HL is the resultant of VL that can utilized for feature extraction. The neurons of 2 layers contains inter-layer full-linked relationships and intra-layer no-connection. w^p is weight matrix between the VLs and HLs. b^v and b^h are the biases of VL and HL, correspondingly and v and h are the VL and HL vectors, correspondingly.

The BP layer and multi-layer RBM are stacked to form DBN. The initial layer of RBM holds v_1 (VL) and h_1 (HL). The v_2 (VL) of the second layer of RBM is the h_1 (HL) of the initial layer of RBM, viz., $v_1 = h_1$. The DBN realizes its layer-wise learning by stacking multiple RBM to extract the features. The final layer of DBN sets the BP model.

DBN training consists of fine-tuning and unsupervised pre-training processes. The greedy algorithm is used in the pre-training process. The outcome attained by the prior RBM training has been utilized as the input of succeeding RBM still each RBM is trained. The first parameter of every RBM is attained simultaneously. The energy function of RBM is defined by:

$$E(v, h; \theta) = - \sum_{i=1}^m b_i v_i - \sum_{j=1}^n c_j h_j - \sum_{i,j=1}^{m,n} w_{ij} v_i h_j \quad (9)$$

$$\theta = \{w_{ij}, b_i, c_j\} \quad (10)$$

Where c_j and h_j are the biases among neurons in the HL. m and n are the number of nodes in the VL and HL, correspondingly. v_j and b_i are the biases between neurons in the VL. w_{ij} is the connection weight value between i^{th} and j^{th} neurons in the VL and HL. The probability distribution can be obtained based on the energy function:

$$P(v, h; \theta) = \frac{1}{Z(\theta)} \exp(-E(v, h; \theta)) \quad (11)$$

In Eq. (11), $Z(\theta)$ is the normalized factor given as follows:

$$Z(\theta) = \sum_{v,h} \exp(-E(v, h; \theta)) \quad (12)$$

The state probability of HLs and VLs are:

$$P(h_j = 1|v; \theta) = \varphi \left(b_j + \sum_{i=1}^n w_{ij} v_i \right) \quad (13)$$

$$P(v_j = 1|h; \theta) \varphi \left(b_j + \sum_{i=1}^m w_{ij} h_i \right) \quad (14)$$

Where $\varphi = \frac{1}{1 + \exp(-x)}$, φ refers to the activation function.

After pretraining, finetuning uses the BP model for training the whole network, for the entire DBN is in optimum layer, which avoids the shortcomings of critical points and longer training time. Suppose the actual and desired outputs of the DBN are y and $\hat{y}$, correspondingly, the loss function of resultant state is given as follows:

$$F(\tau) = \frac{1}{2} \sum_1^N (\hat{y} - y)^2 \quad (15)$$

In Eq. (15), N denotes the amount of training samples and τ refers to iterations count. The weights between output layers and HLs of the final layer are iterated by the update function:

$$w_{out}(\tau + 1) - w_{out}(\tau) = -\eta \frac{\partial F(\tau)}{\partial w_{out}(\tau)} \quad (16)$$

In Eq. (8), η is iteration rate of the DE. The DBN has fault tolerance and strong robustness since the distributed data in the neurons could approximate any complicated non-linear system.

4.3. DMOA based Hyperparameter Tuning

To enhance the performance of the DBN methodology, the DMOA based hyperparameter tuning procedure can be implemented. DMOA is a population-based stochastic meta-heuristic method stimulated through searching and social behavior of dwarf mongoose, known as Helogale [44]. All dwarf mongooses hunt for food on their own because food searching cannot be a combined implementation, however searching will be accomplished owing to the seminomadic feature of such animals, the structure of a sleeping mound has been nearly a numerous food sources. The technique precisely develops the routine of this animal for resolving optimizer issues.

Every population-based optimizer method begins with random initialization. Then, all solutions collect near the global best optimum due to the diversification and intensification approaches. Additionally, the DMOA starts the performance by modifying the mongoose's candidate population. This population will be created randomly among specific problems with upper and lower limitations.

$$X = \begin{bmatrix} x_{1,1} & x_{1,2} & \dots & x_{1,d-1} & x_{1,d} \\ x_{2,1} & x_{2,2} & \dots & x_{2,d-1} & x_{2,d} \\ \vdots & \vdots & x_{ij} & \vdots & \vdots \\ x_{n,1} & x_{n,2} & \dots & x_{n,d-1} & x_{n,d} \end{bmatrix} \quad (17)$$

where d refers to the dimension of the problem, x_{ij} represents the location of the j th dimension, X denotes the set of the candidates' existing population to be produced arbitrarily via Eq. (18), and n specifies the population size.

$$x_{ij} = \text{unifrnd}(\text{VarMin}, \text{VarMax}, \text{VarSize}) \quad (18)$$

Here, *unifrnd* denotes a random number that can be distributed consistently, *VarMin* and *VarMax* describes lower and upper boundaries, *VarSize* has been the size of the difficulty. A better solution at all rounds becomes the best solution gained.

Similar to all meta-heuristic methods, 2 phases exist in the DMOA: exploitation (specific mongoose executes a detailed pursuit in every search space), known as intensification as

well as exploration (a randomized search for a novel plenty of food source and novel sleeping mound) or diversification. A 3 main social architecture of the DMOA performs the actions of 2 stages described namely the babysitters, alpha group, and scout group. The alpha female (α) has the family part controller and is chosen by applying Eq. (19).

$$\alpha = \frac{fit_i}{\sum_{i=1}^n fit_i} \quad (19)$$

The count of babysitters could be represented by bs , $n - bs$ compares the amount of mongooses in the alpha group, and peep signifies the sound of α to the direction of other part followers.

The sleeping mound could be found in numerous food as exhibited in Eq. (20).

$$X_{i+1} = X_i + phi * peep \quad (20)$$

whereas phi denotes a random evenly allocated number $[-1 \text{ and } 1]$, then all rounds, it have assessment of the sleeping mound as given in Eq. (21).

$$sm_i = \frac{fit_{i+1} - fit_i}{\max\{fit_{i+1}, fit_i\}} \quad (21)$$

where a sleeping mound has been determined average value that can be derived by Eq. (22)

$$\varphi = \frac{\sum_{i=1}^n sm_i}{n} \quad (22)$$

When the babysitter interchange criteria can be accomplished, the following stage has the scouting that estimates the subsequent sleeping mound defined by alternative food sources.

As mongoose has recognized not to reappearance at a previous sleeping mound, the scout collection performs exploring in the following sleeping mound for confirmed exploration. The Mongoose has been identified to search as well as scout concurrently in DMOA with the validation to be more unit search, the probability of determining the other sleeping mound. It can be evaluated by Eq. (23).

$$X_{i+1} = \begin{cases} X_i - CF * phi * rand * [X_i - \vec{M}] & \text{if } \varphi_{i+1} > \varphi_i \\ X_i + CF * phi * rand * [X_i - \vec{M}] & \text{else} \end{cases} \quad (23)$$

whereas $rand$ denotes a random number among $[0,1]$, $CF = \left(1 - \frac{iter}{Max_{iter}}\right)^{\left(2 \frac{iter}{Max_{iter}}\right)}$ that represents the parameter to be direct the collective-volatile measure of the group of mongoose that linearly minimizes repetitions. $\vec{M} = \sum_{i=1}^n \frac{X_i \times sm_i}{X_i}$ signifies the vector that stimulates the movement of mongoose for a novel sleeping mound.

The group of babysitters residues with the juveniles while the searching and scouting group hunts for food sources and sleeping mounds. The count of the followers of such a set could be extracted from the overall candidate population counts as it does not carry to scout and search till the babysitter interchange parameter is seen in Eq. (23).

The DMOA model originates a fitness function (FF) in order to acquire better classification result. It states an optimistic integer to signify the better performance of the candidate solution. During this research, the minimizing of the error rate of classification is measured as the FF, as provided in Eq. (24).

$$\begin{aligned} fitness(x_i) &= ClassifierErrorRate(x_i) \\ &= \frac{No. of misclassified instances}{Total no. of instances} * 100 \end{aligned} \quad (24)$$

5. Results and Discussion

In this section, the experimental result of the AIBMC-ODLFFA approach has been tested utilizing the Maling malware dataset [45]. The Maling Dataset comprises 9,339 malware

byteplot images from 25 various families such as Worm-Allapple.L (1591), Worm-Allapple.A (2949), Worm-Yuner.A (800), PWS*-Lolyda.AA 1 (213), PWS*-Lolyda.AA 2 (184), PWS*-Lolyda.AA 3 (123), Trojan-C2Lop.P (146), Trojan-C2Lop.gen!G (200), Dialer-Instantaccess (431), Trojan Downloader-Swizzor.gen!I (132), Trojan Downloader-Swizzor.gen!E (128), Worm-VB.AT (408), Rogue-Fakerean (381), Trojan-Alueron.gen!J (198), Trojan-Malex.gen!J (136), PWS*-Lolyda.AT (159), Dialer-Adialer.C (125), Trojan Downloader-Wintrim.BX (97), Dialer-Dialplatform.B (177), Trojan Downloader-Dontovo.A (162), Trojan Downloader-Obfuscator.AD (142), Backdoor-Agent.FYI (116), Worm:AutoIT-Autorun.K (106), Backdoor-Rbot!gen (158), and Trojan-Skintrim.N (80). Table 1 represents a detailed description of dataset.

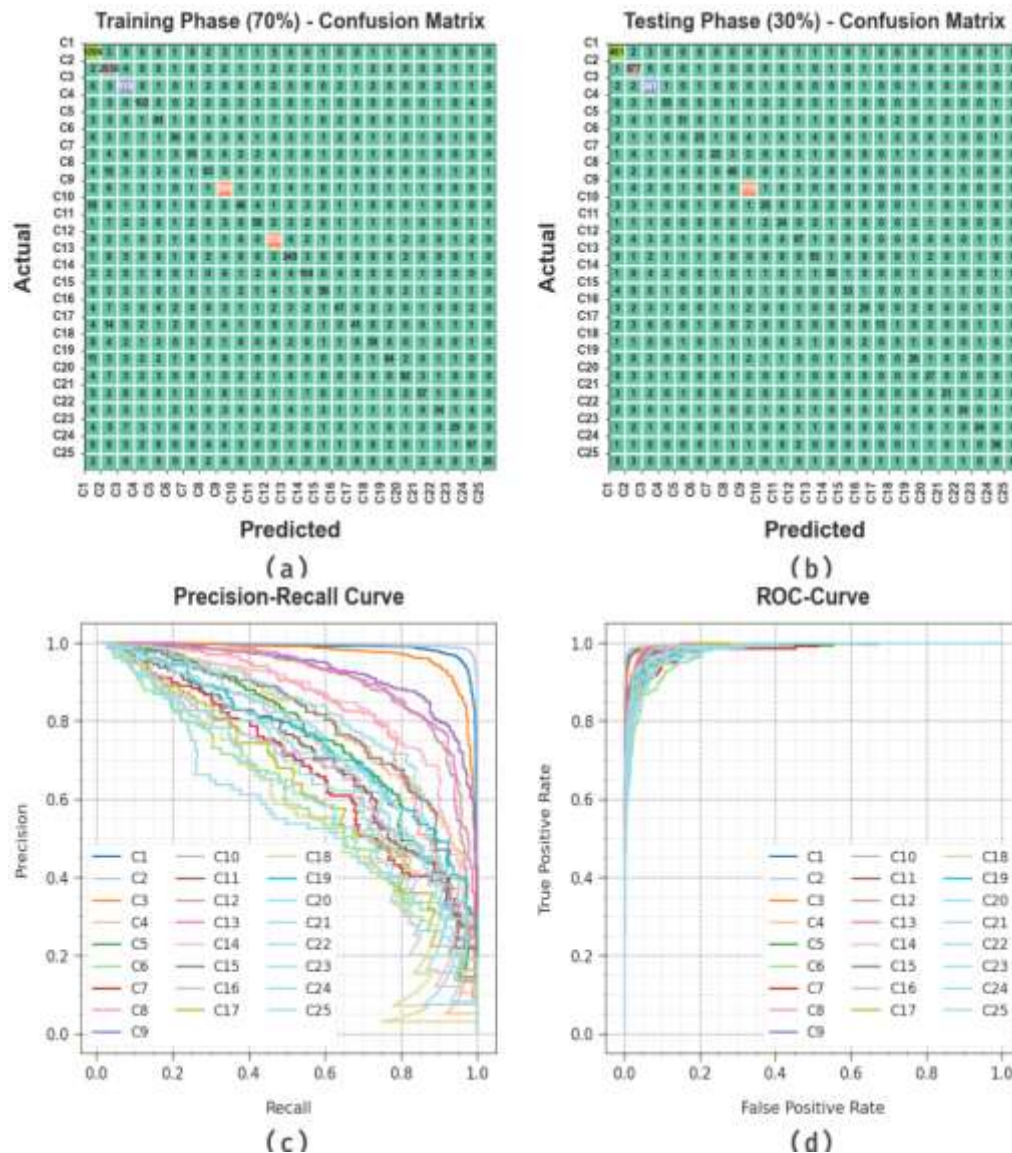
Table 1 Details on dataset

Class Labels	Family	Family Name	No. of Instances
C1	Worm	Allapple.L	1591
C2	Worm	Allapple.A	2949
C3	Worm	Yuner.A	800
C4	PWS*	Lolyda.AA 1	213
C5	PWS*	Lolyda.AA 2	184
C6	PWS*	Lolyda.AA 3	123
C7	Trojan	C2Lop.P	146
C8	Trojan	C2Lop.gen!G	200
C9	Dialer	Instantaccess	431
C10	Trojan Downloader	Swizzor.gen!I	132
C11	Trojan Downloader	Swizzor.gen!E	128
C12	Worm	VB.AT	408
C13	Rogue	Fakerean	381
C14	Trojan	Alueron.gen!J	198
C15	Trojan	Malex.gen!J	136
C16	PWS*	Lolyda.AT	159
C17	Dialer	Adialer.C	125
C18	Trojan Downloader	Wintrim.BX	97
C19	Dialer	Dialplatform.B	177
C20	Trojan Downloader	Dontovo.A	162
C21	Trojan Downloader	Obfuscator.AD	142
C22	Backdoor	Agent.FYI	116
C23	Worm:AutoIT	Autorun.K	106
C24	Backdoor	Rbot!gen	158
C25	Trojan	Skintrim.N	80
Total No. of Instances			9342

Fig. 3 displays the classifier outcomes of the AIBMC-ODLFFA system with test dataset. Figs. 3a-3b demonstrations the confusion matrices acquired by the AIBMC-ODLFFA method under 70:30 of TRPH/TSPH. This experimental result indicated that the AIBMC-ODLFFA system can be identified and categorized with all 25 class labels

appropriately. Next, Fig. 3c establishes the PR outcome of the AIBMC-ODLFFA method. The simulation result described that the AIBMC-ODLFFA motherlodes obtain higher PR analysis under every class. Lastly, Fig. 3d shows the ROC curve of the AIBMC-ODLFFA methodology. The outcome revealed that the AIBMC-ODLFFA system provides efficient solution with increased ROC outcomes on diverse classes.

Fig. 3. (a-b) Confusion matrices of 70% of TRPH and 30% of TSPH (c) PR-curve and (d) ROC-curve



In Table 2, a detailed malware classification analysis of the AIBMC-ODLFFA system is investigated under 70% of TRPH. The experimental outcome indicates that the AIBMC-ODLFFA methodology offers enhanced classification of 25 malware families. For instance, with C1 class, the AIBMC-ODLFFA technique provides $accu_y$ of 97.89%, $prec_n$ of 90.64%, $reca_l$ of 97.77%, F_{score} of 94.07%, and $G_{measure}$ of 94.13%. Besides, with C2 class, the AIBMC-ODLFFA method offers $accu_y$ of 97.71%, $prec_n$ of 94.30%, $reca_l$ of 98.69%, F_{score} of 96.45%, and $G_{measure}$ of 96.47%. Additionally, based on C3 class, the AIBMC-ODLFFA

algorithm gets $accu_y$ of 98.27%, $prec_n$ of 86.36%, $reca_l$ of 94.36%, F_{score} of 90.18%, and $G_{measure}$ of 90.27%. Finally, with C4 class, the AIBMC-ODLFFA methodology achieves $accu_y$ of 98.87%, $prec_n$ of 73.91%, $reca_l$ of 72.86%, F_{score} of 73.38%, and $G_{measure}$ of 73.38%, respectively.

Table 2 Malware classification outcome of AIBMC-ODLFFA model under 70% of TRPH

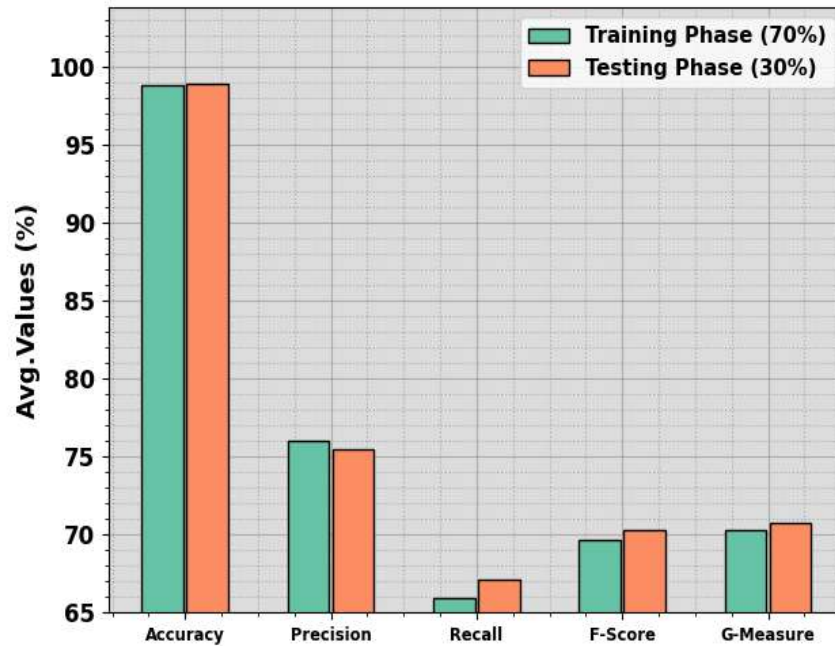
Class Labels	$Accu_y$	$Prec_n$	$Reca_l$	F_{score}	$G_{measure}$
TRPH (70%)					
C1	97.89	90.64	97.77	94.07	94.13
C2	97.71	94.30	98.69	96.45	96.47
C3	98.27	86.36	94.36	90.18	90.27
C4	98.87	73.91	72.86	73.38	73.38
C5	98.82	73.55	66.42	69.80	69.89
C6	99.08	65.45	46.75	54.55	55.32
C7	99.02	78.57	52.88	63.22	64.46
C8	98.84	72.66	69.40	70.99	71.01
C9	98.32	77.23	89.63	82.97	83.20
C10	99.02	68.66	51.69	58.97	59.57
C11	99.07	68.24	63.04	65.54	65.59
C12	98.29	77.08	88.10	82.22	82.41
C13	98.41	77.92	86.96	82.19	82.32
C14	99.04	75.36	78.20	76.75	76.77
C15	99.25	78.38	63.74	70.30	70.68
C16	98.81	65.69	60.91	63.21	63.25
C17	99.04	74.55	45.56	56.55	58.27
C18	99.16	69.64	50.65	58.65	59.39
C19	99.04	84.85	63.64	72.73	73.48
C20	99.05	78.10	67.77	72.57	72.75
C21	98.99	74.03	55.34	63.33	64.00
C22	99.07	65.38	44.16	52.71	53.73
C23	99.17	67.44	42.03	51.79	53.24
C24	98.91	71.28	60.36	65.37	65.59
C25	99.45	90.91	37.04	52.63	58.03
Average	98.82	76.01	65.92	69.64	70.29

In Table 3, a wide-ranging malware classification analysis of the AIBMC-ODLFFA system can be examined under 30% of TSPH. The accomplished findings specify that the AIBMC-ODLFFA method provides increased classification of 25 malware families. According to C1 class, the AIBMC-ODLFFA system offers $accu_y$ of 98.14%, $prec_n$ of 91.83%, $reca_l$ of 97.67%, F_{score} of 94.66%, and $G_{measure}$ of 94.71%. In addition, with C2 class, the AIBMC-ODLFFA method gains $accu_y$ of 98%, $prec_n$ of 94.91%, $reca_l$ of 98.98%, F_{score} of 96.91%, and $G_{measure}$ of 96.93%. Also, with C3 class, the AIBMC-ODLFFA methodology gets $accu_y$ of 98.29%, $prec_n$ of 86.07%, $reca_l$ of 96.40%, F_{score} of 90.94%, and $G_{measure}$ of 91.09%. Finally, based on C4 class, the AIBMC-ODLFFA algorithm achieves $accu_y$ of 98.86%, $prec_n$ of 79.71%, $reca_l$ of 75.34%, F_{score} of 77.46%, and $G_{measure}$ of 77.50%, correspondingly.

Table 3 Malware classification analysis of AIBMC-ODLFFA system on 30% of TSPH

Class Labels	$Accu_y$	$Prec_n$	$Reca_l$	F_{score}	$G_{measure}$
TSPH (30%)					
C1	98.14	91.83	97.67	94.66	94.71
C2	98.00	94.91	98.98	96.91	96.93
C3	98.29	86.07	96.40	90.94	91.09
C4	98.86	79.71	75.34	77.46	77.50
C5	98.89	72.09	62.00	66.67	66.86
C6	98.89	74.19	50.00	59.74	60.91
C7	99.00	73.33	52.38	61.11	61.98
C8	98.97	81.36	72.73	76.80	76.92
C9	98.36	78.29	90.15	83.80	84.01
C10	98.97	69.44	58.14	63.29	63.54
C11	99.29	75.00	66.67	70.59	70.71
C12	98.18	74.05	85.09	79.18	79.37
C13	98.72	79.49	88.57	83.78	83.91
C14	99.11	83.33	76.92	80.00	80.06
C15	99.39	86.84	73.33	79.52	79.80
C16	99.04	80.56	59.18	68.24	69.05
C17	99.07	76.47	37.14	50.00	53.29
C18	99.14	35.71	25.00	29.41	29.88
C19	98.82	65.00	57.78	61.18	61.28
C20	99.11	71.05	65.85	68.35	68.40
C21	99.11	75.00	53.85	62.69	63.55
C22	99.22	77.42	61.54	68.57	69.02
C23	99.25	75.00	64.86	69.57	69.75
C24	99.14	73.47	76.60	75.00	75.02
C25	99.14	57.14	30.77	40.00	41.93
Average	98.88	75.47	67.08	70.30	70.78

In Fig. 4, an average malware classifier outcome of the AIBMC-ODLFFA algorithm can be provided. The attained outcomes demonstrate that the AIBMC-ODLFFA technique properly classified distinct classes of malware families. With 70% of TRPH, the AIBMC-ODLFFA method offers average $accu_y$ of 98.82%, $reca_l$ of 76.01%, $reca_l$ of 65.92%, F_{score} of 69.64%, and $G_{measure}$ of 70.29%. In addition, with 30% of TSPH, the AIBMC-ODLFFA method offers average $accu_y$ of 98.88%, $reca_l$ of 75.47%, $reca_l$ of 67.08%, F_{score} of 70.30%, and $G_{measure}$ of 70.78%.

Fig. 4. Average of the AIBMC-ODLFFA method under 70:30 of TRPH/TSPH

The $accu_y$ curves for training (TR) and validation (VL) indicated in Fig. 5 for the AIBMC-ODLFFA algorithm offer valued insights into its effectiveness on different epochs. Mostly, it can be a consistent upgrading in the TR and TS $accu_y$ with raised epochs, representing the proficiency of model in learning and identifying patterns from the both data of TR and TS. The increasing trend in TS $accu_y$ underscores the model's flexibility for the dataset of TR and abilities for producing correct predictions on unseen data, emphasizing capabilities of robust generalization.

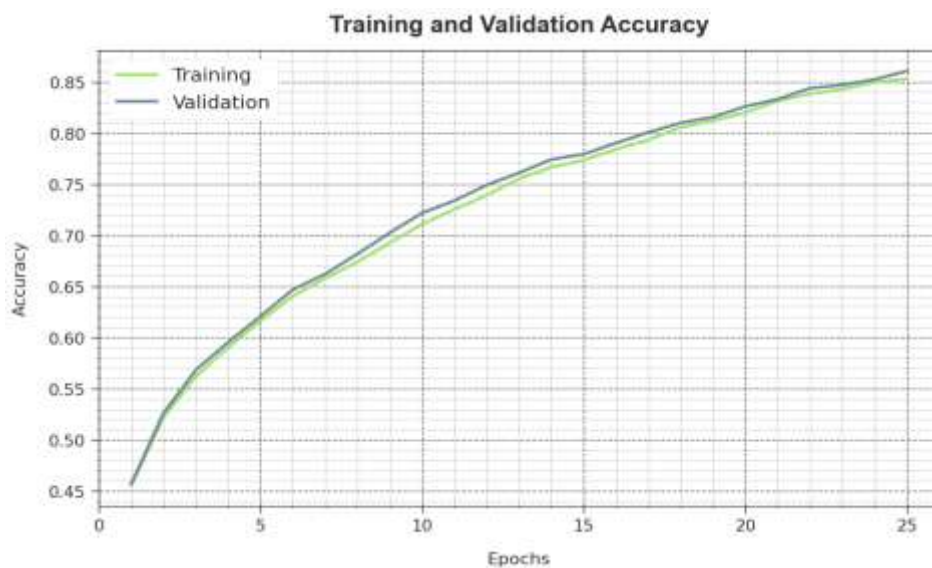
Fig. 5. $Accu_y$ curve of the AIBMC-ODLFFA system

Fig. 6. Loss curve of the AIBMC-ODLFFA algorithm

Fig. 6 illustrates an extensive overview of the TR and TS loss values for the AIBMC-ODLFFA system over numerous epochs. The TR loss incessantly reduces as a model refines weights to decrease classification errors under both datasets. These loss curves establish the model's alignment with the data of TR, underscoring its ability for proficiently capturing patterns. Significant is the continual improvement of parameters in the AIBMC-ODLFFA method, aimed at minimalizing discrepancies among predictions and actual TR labels.

To ensure the better performance of the AIBMC-ODLFFA technique, a wide range of simulations were performed in Table 4 [46]. Fig. 7 investigates a comparison $accu_y$ outcome of the AIBMC-ODLFFA method. The experimental values demonstrate that the AIBMC-ODLFFA technique accomplishes enhanced $accu_y$ values. Based on $accu_y$, the AIBMC-ODLFFA technique offers increased $accu_y$ of 98.88% while the GIST+SVM, Byte 1-gram, VGG16+Softmax, VGG16+SVM, Light-weight DL, NSGA-II, and Google+Softmax models obtain decreased $accu_y$ of 89.43%, 83.70%, 90.77%, 92.29%, 94.00%, 97.60%, and 98.00%, correspondingly.

Table 4 Comparison outcome of the AIBMC-ODLFFA model with other systems

Method	$Accu_y$	$Prec_n$	$Reca_l$
GIST+SVM	89.43	64.15	64.07
Byte 1-gram	83.70	67.61	63.30
VGG16+Softmax	90.77	67.78	65.48
VGG16+SVM	92.29	61.75	66.36
Light-weight DL	94.00	60.14	66.28
NSGA-II	97.60	63.67	66.61

Google+Softmax	98.00	62.98	65.30
AIBMC-ODLFFA	98.88	75.47	67.08

Fig. 7. Accu_y of the AIBMC-ODLFFA model compared with other algorithms

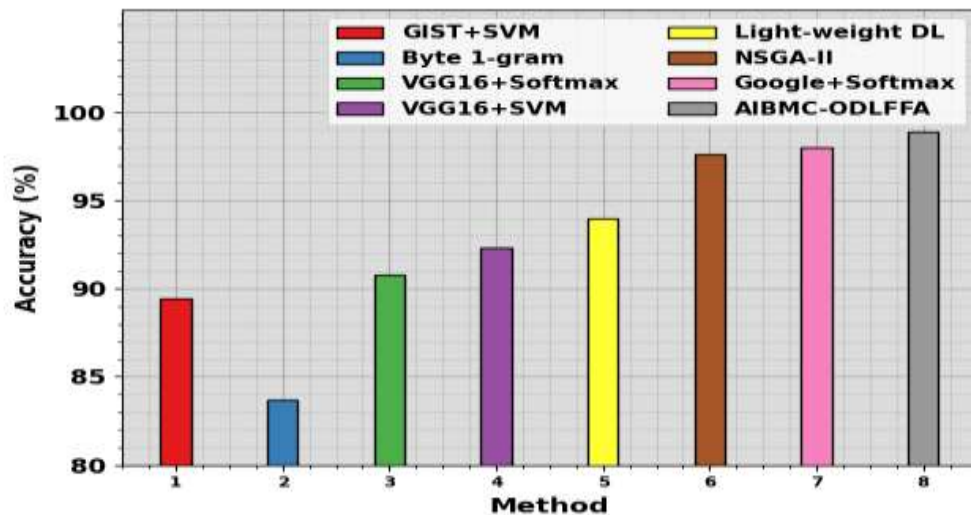
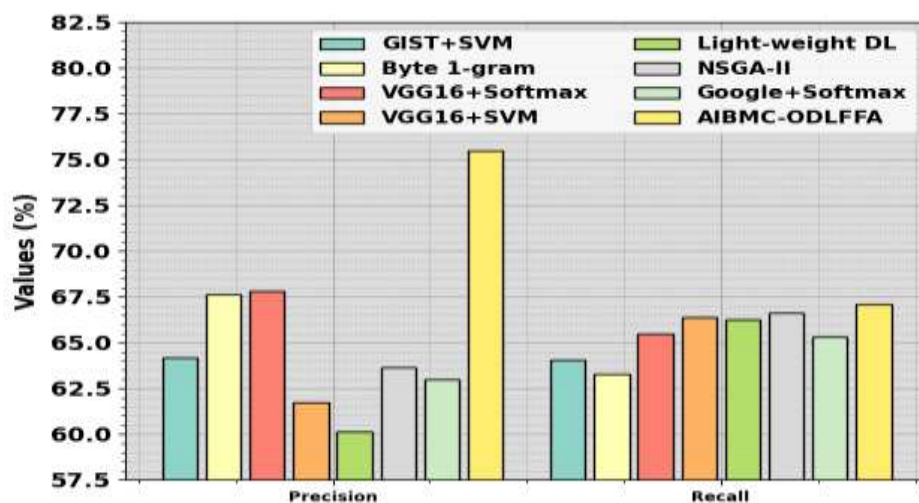


Fig. 8 examines a comparative $prec_n$ and $reca_l$ analysis of the AIBMC-ODLFFA system. The experimental results exhibit that the AIBMC-ODLFFA method achieves increased $prec_n$ and $reca_l$ values. According to $prec_n$, the AIBMC-ODLFFA technique gains higher $prec_n$ of 75.47% whereas the GIST+SVM, Byte 1-gram, VGG16+Softmax, VGG16+SVM, Light-weight DL, NSGA-II, and Google+Softmax algorithms acquire lessened $prec_n$ of 64.15%, 67.61%, 67.78%, 61.75%, 60.14%, 63.67%, and 62.98%. Similarly, based on $reca_l$, the AIBMC-ODLFFA methodology gets boosted $reca_l$ of 67.08% but, the GIST+SVM, Byte 1-gram, VGG16+Softmax, VGG16+SVM, Light-weight DL, NSGA-II, and Google+Softmax systems get diminished $reca_l$ of 64.07%, 63.30%, 65.48%, 66.36%, 66.28%, 66.61%, and 65.30% respectively.

Fig. 8. Comparison analysis of the AIBMC-ODLFFA technique with other systems



Therefore, the AIBMC-ODLFFA technique can be employed for accurate and automated malware classification process.

6. Conclusion

In this manuscript, we have focused on the design and growth of the AIBMC-ODLFFA algorithm for malware image classification. The AIBMC-ODLFFA technique uses hyperparameter-tuned DL model to detect and classify the malware families. To accomplish this, the AIBMC-ODLFFA technique employs feature fusion-based feature extractor, DBN classification, and DMOA-based hyperparameter tuning process. The feature fusion procedure delivers a fused feature vector, which is then utilized for following tasks like malware classification. Regulating the fusion parameters permits flexibility in uniting the powers of CapsNet and SimAM-EfficientNet in capturing related designs for effectual malware classification. Finally, the DMOA with DBN model classifies the presence of different malware families. The experimental evaluation of the AIBMC-ODLFFA system has been tested utilizing malware image dataset. The obtained results stated that the AIBMC-ODLFFA technique reaches better solution with compared approaches with respect to various aspects. Therefore, the developed technique can be applied to automated and accurate malware classification process.

Implications and Future Work:

The findings from the current research on image-based malware classification using deep learning methodologies have significant implications for the field of cybersecurity. The enhanced accuracy and robustness of the proposed approaches demonstrate the potential for these models to be integrated into practical malware detection systems. By leveraging advanced feature fusion techniques and deep learning architectures, the research provides a pathway for developing more sophisticated and effective malware detection tools. This has direct implications for improving the security of critical infrastructures, enterprise systems, and individual user devices, thereby contributing to a more secure digital environment. The necessity for continuous improvement and adaptation of malware detection models is underscored by the evolving nature of cyber threats, highlighting the importance of integrating dynamic analysis capabilities and optimizing feature fusion processes to enhance the detection of sophisticated and obfuscated malware.

Future research should focus on several key areas to enhance malware detection systems. Comparative analyses with other advanced deep learning models under standardized conditions will help identify strengths and weaknesses, guiding improvements. Testing these models in real-world environments is crucial to assess their performance against new malware variants and ensure scalability for large datasets. Additionally, advanced techniques for detecting obfuscated malware, such as combining dynamic and static analysis methods, should be explored. Future work can also enhance the flexibility of AIBMC-ODLFFA by integrating dynamic updating devices for continuous learning and combining additional DL architectures or innovative feature fusion approaches. Optimizing the feature fusion process and improving post-classification analysis will make these models more reliable and practical, reducing false positives and negatives. Addressing these areas will advance the field and develop more effective and reliable malware detection systems.

References

- [1] Ö. A. Aslan and R. Samet, "A comprehensive review on malware detection approaches," *IEEE Access*, vol. 8, pp. 6249-6271, 2020.

- [2] S. Alam, R. N. Horspool, I. Traore, and I. Sogukpinar, "A framework for metamorphic malware analysis and real-time detection," *Computers & Security*, vol. 48, pp. 212-233, 2015.
- [3] P. Sharma, S. Zawar, and S. B. Patil, "Ransomware analysis: Internet of Things (IoT) security issues, challenges, and open problems in the context of worldwide scenario of security of systems and malware attacks," in *International Conference on Recent Innovation in Engineering and Management*, vol. 2, no. 3, pp. 177-184, 2016.
- [4] R. G. Brody, H. U. Chang, and E. S. Schoenberg, "Malware at its worst: death and destruction," *International Journal of Accounting & Information Management*, vol. 26, no. 4, pp. 527-540, 2018.
- [5] I. Ahmad, S. Iqbal, S. Jamil, and M. Kamran, "A systematic literature review of e-banking frauds: Current scenario and security techniques," *Linguistica Antverpiensia*, vol. 2, no. 2, pp. 3509-3517, 2021.
- [6] S. van de Weijer, R. Leukfeldt, and S. Van der Zee, "Reporting cybercrime victimization: determinants, motives, and previous experiences," *Policing: An International Journal*, vol. 43, no. 1, pp. 17-34, 2020.
- [7] M. Button, D. Blackburn, L. Sugiura, D. Shepherd, R. Kapend, and V. Wang, "From feeling like rape to a minor inconvenience: Victims' accounts of the impact of computer misuse crime in the United Kingdom," *Telematics and Informatics*, vol. 64, p. 101675, 2021.
- [8] A. Qamar, A. Karim, and V. Chang, "Mobile malware attacks: Review, taxonomy & future directions," *Future Generation Computer Systems*, vol. 97, pp. 887-909, 2019.
- [9] L. Branch, W. Eller, T. Bias, M. McCawley, D. Myers, B. Gerber, and J. Bassler, "Trends in malware attacks against United States healthcare organizations, 2016-2017," *Global Biosecurity*, vol. 1, no. 1, 2019.
- [10] A. A. Elhadi, M. A. Maarof, and A. H. Osman, "Malware detection based on hybrid signature behaviour application programming interface call graph," *American Journal of Applied Sciences*, vol. 9, no. 3, pp. 283-289, 2012.
- [11] R. Tahir, "A study on malware and malware detection techniques," *International Journal of Education and Management Engineering*, vol. 8, no. 2, p. 20, 2018.
- [12] K. Griffin, S. Schneider, X. Hu, and T. C. Chiueh, "Automatic generation of string signatures for malware detection," in *Recent Advances in Intrusion Detection: 12th International Symposium, RAID 2009, Saint-Malo, France, September 23-25, 2009. Proceedings*, vol. 12, pp. 101-120, Springer Berlin Heidelberg, 2009.
- [13] K. Liu, S. Xu, G. Xu, M. Zhang, D. Sun, and H. Liu, "A review of Android malware detection approaches based on machine learning," *IEEE Access*, vol. 8, pp. 124579-124607, 2020.
- [14] A. Mahindru and A. L. Sangal, "MLDroid—framework for Android malware detection using machine learning techniques," *Neural Computing and Applications*, vol. 33, no. 10, pp. 5183-5240, 2021.
- [15] S. Y. Yerima, S. Sezer, and I. Muttik, "Android malware detection using parallel machine learning classifiers," in *2014 Eighth International Conference on Next Generation Mobile Apps, Services and Technologies*, pp. 37-42, Sept. 2014.
- [16] U. E. H. Tayyab, F. B. Khan, M. H. Durad, A. Khan, and Y. S. Lee, "A survey of the recent trends in deep learning based malware detection," *Journal of Cybersecurity and Privacy*, vol. 2, no. 4, pp. 800-829, 2022.
- [17] M. Gopinath and S. C. Sethuraman, "A comprehensive survey on deep learning based malware detection techniques," *Computer Science Review*, vol. 47, p. 100529, 2023.
- [18] M. K. Alzaylaee, S. Y. Yerima, and S. Sezer, "DL-Droid: Deep learning based android malware detection using real devices," *Computers & Security*, vol. 89, p. 101663, 2020.

- [19] I. You and K. Yim, "Malware obfuscation techniques: A brief survey," in 2010 International Conference on Broadband, Wireless Computing, Communication and Applications, Nov. 2010, pp. 297-300.
- [20] S. O'Shaughnessy and S. Sheridan, "Image-based malware classification hybrid framework based on space-filling curves," *Computers & Security*, vol. 116, p. 102660, 2022.
- [21] Y. Ding, X. Zhang, J. Hu, and W. Xu, "Android malware detection method based on bytecode image," *Journal of Ambient Intelligence and Humanized Computing*, pp. 1-10, 2020.
- [22] N. Daoudi, J. Samhi, A. K. Kabore, K. Allix, T. F. Bissyandé, and J. Klein, "Dexray: a simple, yet effective deep learning approach to Android malware detection based on image representation of bytecode," in *Deployable Machine Learning for Security Defense: Second International Workshop, MLHat 2021, Virtual Event, August 15, 2021, Proceedings*, vol. 2, pp. 81-106, Springer International Publishing, 2021.
- [23] M. Nunes, P. Burnap, P. Reinecke, and K. Lloyd, "Bane or Boon: Measuring the effect of evasive malware on system call classifiers," *Journal of Information Security and Applications*, vol. 67, p. 103202, 2022.
- [24] C. Avci, B. Tekinerdogan, and C. Catal, "Analyzing the performance of long short-term memory architectures for malware detection models," *Concurrency and Computation: Practice and Experience*, vol. 35, no. 6, pp. 1-1, 2023.
- [25] M. Dener, G. Ok, and A. Orman, "Malware detection using memory analysis data in big data environment," *Applied Sciences*, vol. 12, no. 17, p. 8604, 2022.
- [26] J. Yang, J. Tang, R. Yan, and T. Xiang, "Android malware detection method based on permission complement and API calls," *Chinese Journal of Electronics*, vol. 31, no. 4, pp. 773-785, 2022.
- [27] O. Aslan, M. Ozkan-Okay, and D. Gupta, "A review of cloud-based malware detection system: Opportunities, advances and challenges," *European Journal of Engineering and Technology Research*, vol. 6, no. 3, pp. 1-8, 2021.
- [28] F. Alhaidari et al., "ZeVigilante: detecting zero-day malware using machine learning and sandboxing analysis techniques," *Computational Intelligence and Neuroscience*, vol. 2022, 2022.
- [29] J. Hemalatha, S. A. Roseline, S. Geetha, S. Kadry, and R. Damaševičius, "An efficient densenet-based deep learning model for malware detection," *Entropy*, vol. 23, no. 3, p. 344, 2021.
- [30] J. Geremias, E. K. Viegas, A. O. Santin, A. Britto, and P. Horchulhack, "Towards multi-view android malware detection through image-based deep learning," in 2022 International Wireless Communications and Mobile Computing (IWCMC), May 2022, pp. 572-577.
- [31] K. Bakour and H. M. Ünver, "DeepVisDroid: Android malware detection by hybridizing image-based features with deep learning techniques," *Neural Computing and Applications*, vol. 33, pp. 11499-11516, 2021.
- [32] P. Panda, O. K. CU, S. Marappan, S. Ma, and D. Veasani Nandi, "Transfer Learning for Image-Based Malware Detection for IoT," *Sensors*, vol. 23, no. 6, pp. 3253, 2023.
- [33] A. S. Shatnawi, Q. Yassen, and A. Yateem, "An android malware detection approach based on static feature analysis using machine learning algorithms," *Procedia Computer Science*, vol. 201, pp. 653-658, 2022.
- [34] M. S. Akhtar and T. Feng, "Malware analysis and detection using machine learning algorithms," *Symmetry*, vol. 14, no. 11, pp. 2304, 2022.
- [35] D. Vasan, M. Hammoudeh, and M. Alazab, "Broad learning: A GPU-free image-based malware classification," *Applied Soft Computing*, p. 111401, 2024.

- [36] M. B. Haile, Y. M. Walle, and A. J. Belay, "Enhanced image-based malware multiclass classification method with the ensemble model and SVM," *Open Information Science*, vol. 8, no. 1, p. 20240003, 2024.
- [37] T. Bikku, S. B. Chandolu, S. P. Praveen, N. R. Tirumalasetti, K. Swathi, and U. Sirisha, "Enhancing real-time malware analysis with quantum neural networks," *Journal of Intelligent Systems and Internet of Things*, vol. 12, no. 1, pp. 57-7, 2024.
- [38] E. F. Godwin Olaoye, "Role of Machine learning and AI in cloud malware detection," *Journal Name (if applicable)*, vol. (if applicable), no. (if applicable), pp. (if applicable), 2024.
- [39] S. A. Hashmi, "Malware Detection and Classification on Different Dataset by Hybridization of CNN and Machine Learning," *International Journal of Intelligent Systems and Applications in Engineering*, vol. 12, no. 6s, pp. 650-667, 2024.
- [40] A. Laouadi, D. E. Menacer, and K. Benatchba, "A Machine Learning Approach for Malware Detection based on Image Conversion," *Journal Name (if applicable)*, vol. X, no. X, pp. X-X, 2024.
- [41] Opoku, M., Weyori, B.A., Adekoya, A.F. and Adu, K., 2023. CLAHE-CapsNet: Efficient retina optical coherence tomography classification using capsule networks with contrast limited adaptive histogram equalization. *Plos one*, 18(11), p.e0288663.
- [42] You, H., Lu, Y. and Tang, H., 2023. Plant disease classification and adversarial attack using SimAM-EfficientNet and GP-MI-FGSM. *Sustainability*, 15(2), p.1233.
- [43] Tao, Y., Liu, H., Chen, S., Lan, J., Qi, Q. and Xiao, W., 2023. An Off-Line Error Compensation Method for Absolute Positioning Accuracy of Industrial Robots Based on Differential Evolution and Deep Belief Networks. *Electronics*, 12(17), p.3718.
- [44] Akinola, O.A., Ezugwu, A.E., Oyelade, O.N. and Agushaka, J.O., 2022. A hybrid binary dwarf mongoose optimization algorithm with simulated annealing for feature selection on high dimensional multi-class datasets. *Scientific Reports*, 12(1), p.14945.
- [45] <https://paperswithcode.com/dataset/malimg>
- [46] Vasan, D., Alazab, M., Wassan, S., Safaei, B. and Zheng, Q., 2020. Image-Based malware classification using ensemble of CNN architectures (IMCEC). *Computers & Security*, 92, p.101748.