Computers, Materials & Continua

Tech Science Press

Doi:10.32604/cmc.2025.064053

ARTICLE

AI-Driven Malware Detection with VGG Feature Extraction and Artificial Rabbits Optimized Random Forest Model

Brij B. Gupta[1,2,3,4,*], **Akshat Gaurav**[5], **Wadee Alhalabi**[6], **Varsha Arya**[7,8], **Shavi Bansal**[9,10] and **Ching-Hsien Hsu**[1]

[1]Department of Computer Science and Information Engineering, Asia University, Taichung, 413, Taiwan
[2]Department of Medical Research, China Medical University Hospital, China Medical University, Taichung, 40447, Taiwan
[3]Symbiosis Centre for Information Technology (SCIT), Symbiosis International University, Pune, 412115, India
[4]School of Cybersecurity, Korea University, Seoul, 02841, Republic of Korea
[5]Computer Engineering, Ronin Institute, Montclair, NJ 07043, USA
[6]Department of Computer Science, Immersive Virtual Reality Research Group, King Abdulaziz University, Jeddah, 21589, Saudi Arabia
[7]Hong Kong Metropolitan University, Hong Kong SAR, China
[8]Center for Interdisciplinary Research, University of Petroleum and Energy Studies, Dehradun, 248007, India
[9]Department of Research and Innovation, Insights2Techinfo, Jaipur, 302001, India
[10]University Centre for Research and Development (UCRD), Chandigarh University, Chandigarh, 140413, India

*Corresponding Author: Brij B. Gupta. Email: bbgupta@asia.edu.tw

Received: 03 February 2025; Accepted: 14 May 2025; Published: 30 July 2025

ABSTRACT: Detecting cyber attacks in networks connected to the Internet of Things (IoT) is of utmost importance because of the growing vulnerabilities in the smart environment. Conventional models, such as Naive Bayes and support vector machine (SVM), as well as ensemble methods, such as Gradient Boosting and eXtreme gradient boosting (XGBoost), are often plagued by high computational costs, which makes it challenging for them to perform real-time detection. In this regard, we suggested an attack detection approach that integrates Visual Geometry Group 16 (VGG16), Artificial Rabbits Optimizer (ARO), and Random Forest Model to increase detection accuracy and operational efficiency in Internet of Things (IoT) networks. In the suggested model, the extraction of features from malware pictures was accomplished with the help of VGG16. The prediction process is carried out by the random forest model using the extracted features from the VGG16. Additionally, ARO is used to improve the hyper-parameters of the random forest model of the random forest. With an accuracy of 96.36%, the suggested model outperforms the standard models in terms of accuracy, F1-score, precision, and recall. The comparative research highlights our strategy's success, which improves performance while maintaining a lower computational cost. This method is ideal for real-time applications, but it is effective.

KEYWORDS: Malware detection; VGG feature extraction; artificial rabbits; optimization; random forest model

1 Introduction

The current cybersecurity landscape is increasingly characterized by the proliferation of malware, which poses significant threats to digital environments. Defined as harmful software meant to compromise computer systems, malware has changed significantly and is driving increased cyberattacks. Recent data show that during the 1990s, malware-related network security events increased by over 50% yearly, underscoring the critical necessity of efficient detection and mitigating techniques [1]. The fast development of digital technology and the related growth in vulnerabilities that cybercriminals use help explain this surge [2].

Sophisticated malware, including ransomware and banking trojans, which have proliferated in cyber-attack operations, are among the most alarming developments [2]. Ransomware has become well-known for its capacity to encrypt important data and demand payment, causing significant financial losses for businesses [3,4]. Renowned for starting Distributed Denial of Service (DDoS) assaults, the Mirai virus shows the terrible power of malware in compromising security and upsetting systems [5]. The growing complexity of malware, especially sophisticated obfuscation techniques, renders conventional detection approaches insufficient; therefore, stronger solutions must be developed [6].

The cybersecurity industry is looking to artificial intelligence (AI) and machine learning (ML) for malware identification and categorization more and more to fight these threats. These technologies enable the analysis of enormous volumes of data and the identification of trends suggestive of malicious conduct. In dynamic contexts where malware may evolve to avoid conventional defenses, AI-assisted methods have shown promise in improving the speed and accuracy of malware detection [7,8]. Furthermore, integrating hybrid analytic approaches, mixing static and dynamic techniques, has become increasingly important in the battle against malware [9].

Beyond just causing instantaneous security breaches, malware compromises data quality and availability, which has long-term effects for businesses and people, both individually [10]. Continuous research and innovation in malware detection and response techniques remain vital to protect digital infrastructures and lower the risks connected with malware assaults as the terrain of cyberthreats develops [11].

Ultimately, malware's emergence poses a significant obstacle in the contemporary cybersecurity scene and calls for a multifarious strategy combining a proactive attitude toward new hazards with enhanced detection methods. Improving defenses against malware's constantly changing character depends on including AI and ML in cybersecurity processes.

1.1 Contribution

In this context, this paper proposes a malware detection system that integrates a Visual Geometry Group 16 (VGG16)-based feature extraction approach with Artificial Rabbits Optimization (ARO). This method outperforms current strategies in terms of accuracy, computational complexity, and real-time application by offering a lightweight, accurate, and efficient solution for malware detection in IoT contexts, therefore addressing the constraints of conventional methods.

1.2 Organization

The rest of the paper is organized as follows: Section 2 presents the details of the past research, Section 3 gives the details of the proposed approach, Section 4 presents the results, and Section 5 concludes the paper.

2 Related Work

This section gives the details of the state-of-the-art models proposed by the researchers, and the summary of the models is presented in Table 1. Kumar et al. [12] propose an intelligent malware classification approach using a deep convolutional neural network (IMCNN) combined with honeypots in organizational networks. The method utilizes transfer learning with pre-trained CNN models (VGG16, VGG19, InceptionV3, and ResNet50) and ensemble learning for malware detection. Features are extracted and selected using ReLU layers, PCA (Principal Component Analysis), and SVD (Singular Value Decomposition), then classified using k-NN (k-Nearest Neighbors), SVM, and RF (Random Forest), with final predictions made through soft voting. The model demonstrates high accuracy on benchmark and real-world malware datasets. However, the model has limitations: it relies on computationally intensive deep learning models and requires substantial preprocessing, which may limit its scalability in real-time environments. Rao et al. [13] explore security challenges in cloud computing, focusing on flooding (DoS) and Sybil attacks that disrupt services and mislead users. It discusses existing security mechanisms to counter these threats and highlights the need for more potent defense strategies, especially for small-scale industries. However, the study lacks a novel detection or prevention approach and does not provide an in-depth performance evaluation of existing security schemes. Additionally, it offers a theoretical discussion without real-world implementation or case studies to validate its findings.

Table 1: Comparative analysis of state-of-art models

Paper	Methodology	Strengths	Limitations
[12]	IMCNN with transfer learning & ensemble learning	High accuracy, robust feature selection	Computationally expensive, requires preprocessing
[13]	Cloud security analysis (DoS, Sybil attacks)	Comprehensive threat analysis, identifies key risks	Lacks novel detection mechanism, theoretical focus
[14]	SimHash & CNN-based malware classification	Effective classification, high accuracy on imbalanced data	Vulnerable to obfuscation, lacks dynamic analysis
[15]	Federated learning with VGG16	Preserves data privacy, high accuracy	Limited to MNIST dataset, lacks real-world validation
[16]	Hybrid deep learning & visualization	Improves detection accuracy with hybrid approach	High computational cost, relies on data quality
[17]	LSTM-GRU model optimized with EOA (Earthworm Optimization Algorithm)	99% accuracy, outperforms traditional models	Lacks real-world validation, ignores adversarial robustness
[18]	Grayscale image-based malware detection	Effective against obfuscation-based attacks	Computationally expensive, struggles with complex obfuscation
[19]	Serverless malware detection with AWS & n-grams	Scalable cloud-based approach, promising results	Latency & cost concerns, lacks adversarial analysis
[20]	CNN-based IIoT malware detection using visualization	Improved accuracy for IIoT malware detection	High computational demand, real-time performance concerns
[21]	Few-shot classification with multi-view image generation	Enhances generalization in few-shot learning	Lacks deployment validation, adversarial robustness untested
[22]	Dynamic feature extraction for online malware detection	Dynamic analysis, outperforms traditional methods	Scalability & advanced evasion concerns
[23]	APT evasion techniques with EMRF	Demonstrates effectiveness of APT evasion	No mitigation strategies, lacks practical security solutions

(Continued)

Table 1 (continued)

Paper	Methodology	Strengths	Limitations
[24]	Review of malware visualization techniques	Identifies key gaps in malware visualization	No experimental validation or impact assessment
[25]	Lightweight CNN-based malware classification (IMCLNet)	High accuracy, lightweight model	Limited to small images, lacks data augmentation
[26]	Transfer learning-based malware classification (IVMCT)	Good performance against polymorphic malware	Image conversion may miss malware characteristics
[27]	Binary file to image-based malware classification	Effective malware family classification	Struggles with highly obfuscated malware

Ni et al. [14] propose a malware classification algorithm called MCSC (Malware Classification using SimHash and CNN), which converts disassembled malware codes into gray images using SimHash and classifies them using a convolutional neural network. Techniques such as multi-hash, primary block selection, and bilinear interpolation are employed to enhance performance. The model achieves high classification accuracy, even for unevenly distributed samples. However, the model has the following limitations: it relies on static features, which may be vulnerable to evasion techniques like obfuscation. It does not incorporate dynamic analysis, potentially missing behavior-based detection. Sharma et al. [15] propose a federated learning (FL)-based framework for training a VGG16-based CNN model without sharing client data, addressing security concerns in critical infrastructure. The model updates occur locally by utilizing federated averaging, ensuring data privacy while achieving high accuracy, as demonstrated on the MNIST dataset. Additionally, the paper discusses the benefits and challenges of FL, security vulnerabilities, potential attacks, and mitigation strategies. However, the study is limited to the MNIST dataset, which may not fully represent real-world critical infrastructure scenarios. It lacks comparative analysis with other FL models or alternative deep learning architectures.

Venkatraman et al. [16] propose a novel hybrid deep learning and visualization approach for malware detection, utilizing image-based techniques to detect suspicious system behavior and hybrid deep learning architectures for effective malware classification. The model incorporates various similarity measures and cost-sensitive deep learning architectures to improve performance, demonstrating high accuracy on large public and private datasets. However, the model has the following limitations: it requires substantial computational resources for image processing and deep learning training, and the quality and completeness of data representation may limit its reliance on behavior visualization. Gupta et al. [17] present an Android malware detection model that integrates long short-term memory (LSTM) and gate recurrent unit (GRU) networks optimized using the Earthworm Optimization Algorithm, with a random forest employed for feature selection. The model achieves 99% accuracy, outperforming traditional methods such as GRU, LSTM, RNN (Recurrent Neural Network), Logistic Regression, and SVM, demonstrating its effectiveness in enhancing cybersecurity for Android devices. However, the study lacks real-world deployment validation and focuses solely on accuracy without addressing computational efficiency, adversarial robustness, or model generalizability across diverse malware datasets.

Mercaldo and Santone [18] propose a novel method for malware detection in mobile environments that classifies malware families and variants by converting executable samples into grayscale images and extracting features for classification. This approach addresses the limitations of signature-based methods and performs well against obfuscation techniques. However, the model has the following limitations: it may still face challenges with highly complex or novel obfuscation methods and requires image processing, which can be computationally demanding for large datasets. Mishra et al. [19] introduce CloudIntellMal, a serverless computing-based intelligent malware detection framework for Android applications. It leverages AWS (Amazon Web Services) (EC2, S3) for log storage, preprocessing, and machine learning-based malware detection. A Bag of n-grams-based feature extraction algorithm is developed to enhance feature representation, and a trained decision model is deployed on EC2 for classification. Experimental validation using the Drebin dataset shows promising results. However, the study relies on cloud-based infrastructure, which may introduce latency and cost concerns, and lacks comparative analysis with on-device detection models or adversarial robustness assessments.

Naeem et al. [20] propose a novel architecture for detecting malware attacks on the Industrial Internet of Things (IIoT) using color image visualization and a deep convolutional neural network (CNN). This approach enhances malware analysis and improves predictive time and detection accuracy compared to existing methods. However, the model has the following limitations: it may have high computational demands due to image-based deep learning techniques, which could impact real-time performance in resource-constrained IIoT environments. Chai et al. [21] introduce DMMal, a novel few-shot malware classification framework that addresses the challenge of classifying new or unknown malware. It enhances classification at the data and model levels by proposing a multi-channel malware image generation method based on multi-view techniques to enrich semantic information. Additionally, it employs adaptive sharpness-aware minimization to improve model generalization by optimizing both loss and sharpness simultaneously. Experimental results on two few-shot malware classification datasets demonstrate improved performance. However, the paper does not explore real-world deployment feasibility, and the effectiveness of the proposed approach against adversarial attacks or evolving malware patterns remains unaddressed.

Ghahramani et al. [22] propose a novel method for online malware detection in IoT devices by monitoring malware behavior, extracting dynamic features, and converting them into sparse binary images for analysis. They compare clustering, probabilistic, and deep learning methods to identify the most effective approach, demonstrating that deep learning outperforms others in seven out of eight metrics. However, the model has the following limitations: the binary image transformation may not capture all malware behavior nuances, and the approach's scalability and effectiveness against advanced evasion techniques need further validation. Sharma et al. [23] present the Evasive Manoeuvres Re-Engineering Framework (EMRF) to systematically analyze and demonstrate evasive techniques used by Advanced Persistent Threats (APT) malware across multiple platforms, including Windows, Linux, macOS, Android, IoT, and ICS/SCADA devices. By leveraging a dataset of 4403 APT malware samples, the framework examines evasion strategies such as stealth, covert communication, and anti-analysis mechanisms, revealing that non-zero-day payloads can effectively evade modern security solutions with an evasion rate between 36% and 96%. This research challenges the over-reliance on zero-day exploit detection and highlights the growing advantage of attackers in bypassing existing defenses. However, the study does not propose a concrete countermeasure or defense mechanism and primarily focuses on demonstrating evasion rather than mitigating it, limiting its immediate applicability for enhancing security solutions.

Brosolo et al. [24] evaluate the effectiveness of malware visualization techniques in detecting and classifying malware by categorizing studies based on information retrieval, visualization, feature extraction, classification, and evaluation. This comprehensive review identifies key challenges in visualization-based methods and offers valuable insights into current progress and future opportunities in the field. However, the study has the following limitations: it primarily focuses on reviewing existing methods without providing experimental validation or direct comparisons of the visualizations' impact on detection performance.

Zou et al. [25] propose IMCLNet, a lightweight malware classification model that uses malware images without relying on feature engineering or domain knowledge. IMCLNet integrates Coordinate Attention, Depthwise Separable Convolution, and Global Context Embedding to enhance accuracy while minimizing computational costs and parameters. The model demonstrates high classification accuracy on MalImg and BIG2015 datasets, outperforming mainstream lightweight models like MobileNetV3, ShuffleNetV2, and MixNet. However, the model has the following limitations: it is primarily optimized for small image sizes (32 × 32), which may limit its applicability to more complex malware images, and it lacks data enhancement, potentially affecting generalization to diverse malware types.

Kumar [26] propose the IVMCT framework, which utilizes transfer learning to classify malware using grayscale images derived from malware binaries. This approach addresses the limitations of traditional signature-based and machine learning methods that struggle with polymorphic and metamorphic malware variants. The framework is evaluated on the MalImg dataset and demonstrates superior performance compared to existing techniques. However, the model has the following limitations: it relies on image conversion, which may not capture all malware characteristics, and its effectiveness against highly obfuscated malware variants needs further evaluation.

Han et al. [27] propose a novel malware family classification method that converts binary files into images and entropy graphs to detect similarities among malware variants and classify malware families. This approach leverages the repeated modules within malware variants to improve detection and classification accuracy. However, the model has the following limitations: it may not fully capture the behavioral characteristics of malware, and its effectiveness could be limited when dealing with highly obfuscated or novel variants that do not share common modules.

3 Proposed Approach

Three main phases define the proposed appraoch for AI-driven malware detection: data collecting, feature extract, and hyperparameter optimization, as represented in Fig. 1. Malware data is collected from the Kaggle [28]. In the feature extractiom phase, the malware pictures are given to a VGG16 model, which acts as a feature extractor, spotting important characteristics from the malware images. After that a Random Forest model uses the extracted features for the calssficaiton. Also, ARO technique is used to fine-tunes hyperparameters of randome forest.

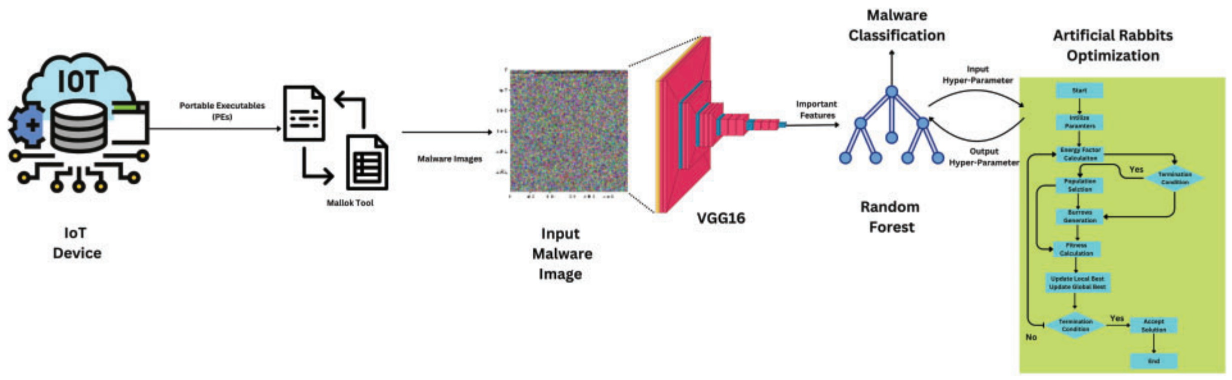


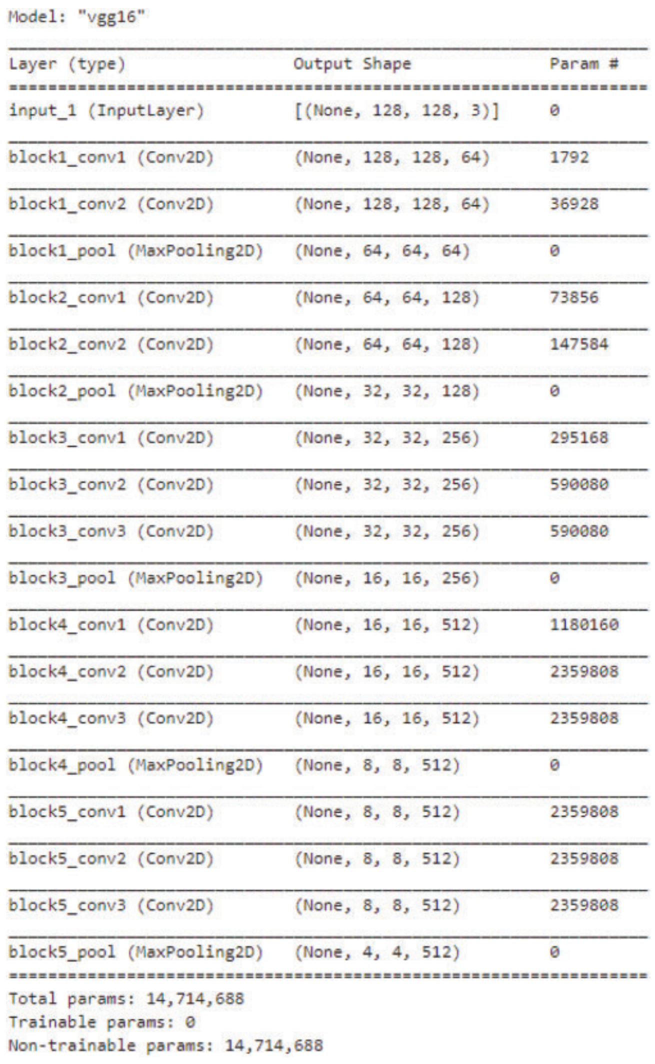
Figure 1: Proposed approach

Our selection of ARO was motivated by its improved exploration-exploitation balance since it essentially inhibits premature convergence compared to genatic algorithm (GA) and Particle Swarm Optimization

(PSO) while using adaptive energy-based mobility for various solution generation. While Bayesian Optimization (BO) is computationally expensive and ARO is more appropriate for real-time applications, ARO shows faster convergence than GA and PSO regarding computational efficiency. Furthermore, ARO dynamically adapts to various search environments and provides superior scalability and resilience to GA and PSO, which reduces the hyperparameter changes needed.

3.1 Feature Selection by VGG

The VGG16 model (Fig. 2) is a deep convolutional neural network (CNN) used for feature extraction from images. It consists of multiple convolutional layers followed by pooling layers. VGG16 retrieves spatial and texture-based characteristics from input malware images. Notably, although deeper layers extract higher-level abstract information, including repeating patterns, density changes, and shape distributions, which are useful for identifying malware families, the bottom levels of VGG16 detect edges, corners, and simple textures. These extracted properties assist in classification since different kinds of malware show distinct structural patterns depending on packing, encryption, and obfuscation methods.



```
Model: "vgg16"
_________________________________________________________________
Layer (type)                Output Shape              Param #
=================================================================
input_1 (InputLayer)        [(None, 128, 128, 3)]     0

_________________________________________________________________
block1_conv1 (Conv2D)       (None, 128, 128, 64)      1792
_________________________________________________________________
block1_conv2 (Conv2D)       (None, 128, 128, 64)      36928
_________________________________________________________________
block1_pool (MaxPooling2D)  (None, 64, 64, 64)        0
_________________________________________________________________
block2_conv1 (Conv2D)       (None, 64, 64, 128)       73856
_________________________________________________________________
block2_conv2 (Conv2D)       (None, 64, 64, 128)       147584
_________________________________________________________________
block2_pool (MaxPooling2D)  (None, 32, 32, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)       (None, 32, 32, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)       (None, 32, 32, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)       (None, 32, 32, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)  (None, 16, 16, 256)       0
_________________________________________________________________
block4_conv1 (Conv2D)       (None, 16, 16, 512)       1180160
_________________________________________________________________
block4_conv2 (Conv2D)       (None, 16, 16, 512)       2359808
_________________________________________________________________
block4_conv3 (Conv2D)       (None, 16, 16, 512)       2359808
_________________________________________________________________
block4_pool (MaxPooling2D)  (None, 8, 8, 512)         0
_________________________________________________________________
block5_conv1 (Conv2D)       (None, 8, 8, 512)         2359808
_________________________________________________________________
block5_conv2 (Conv2D)       (None, 8, 8, 512)         2359808
_________________________________________________________________
block5_conv3 (Conv2D)       (None, 8, 8, 512)         2359808
_________________________________________________________________
block5_pool (MaxPooling2D)  (None, 4, 4, 512)         0
=================================================================
Total params: 14,714,688
Trainable params: 0
Non-trainable params: 14,714,688
```

Figure 2: Pre-trained VGG16 model

3.1.1 Convolution Operation

The convolution operation in VGG16 involves sliding a filter (kernel) across the input image or feature map to compute the dot product between the filter weights and the input values. The convolution operation for a single filter can be expressed as:

$$X_{ijc}^{(l)} = \sigma\left(\sum_{k=1}^{K}\sum_{m=1}^{M}\sum_{n=1}^{N} W_{mnk}^{(l)} \cdot X_{(i+m)(j+n)k}^{(l-1)} + b_c^{(l)}\right) \tag{1}$$

where:

- $X_{ijc}^{(l)}$: Output at location (i, j) and channel c of the l-th layer.
- $W_{mnk}^{(l)}$: Weight of the filter at position (m, n) for the k-th channel of the l-th layer.
- $X_{(i+m)(j+n)k}^{(l-1)}$: Input value from the previous layer at location $(i + m, j + n)$ and channel k.
- $b_c^{(l)}$: Bias term for the c-th output channel of the l-th layer.
- K: Number of input channels.
- M and N: Dimensions of the filter (kernel size).
- σ: Activation function, typically ReLU.

3.1.2 ReLU Activation Function

The ReLU activation function introduces non-linearity, allowing the network to model complex patterns:

$$\sigma(x) = \max(0, x) \tag{2}$$

This activation function is applied element-wise after each convolution operation.

3.1.3 Pooling Layer (Max Pooling)

Pooling layers downsample the feature maps to reduce the spatial dimensions, preserving important features while reducing computation. For max pooling, the equation is:

$$P_{ijc}^{(l)} = \max_{(m,n)\in\mathscr{R}} X_{(2i+m)(2j+n)c}^{(l-1)} \tag{3}$$

where:

- $P_{ijc}^{(l)}$: Output value at location (i, j) and channel c in the pooling layer.
- $\mathscr{R}$: Pooling region (e.g., 2×2 window).
- $X_{(2i+m)(2j+n)c}^{(l-1)}$: Input values within the pooling region from the previous layer.

3.1.4 Sequential Layer Blocks in VGG16

VGG16 consists of five main convolutional blocks, each containing multiple convolutional layers followed by a max pooling layer. The pattern is:

- Block 1: 2 Conv layers + 1 Max Pooling
- Block 2: 2 Conv layers + 1 Max Pooling
- Block 3: 3 Conv layers + 1 Max Pooling
- Block 4: 3 Conv layers + 1 Max Pooling
- Block 5: 3 Conv layers + 1 Max Pooling

3.1.5 Fully Connected Layers (Omitted in Feature Extraction)

In classification tasks, the extracted features are usually fed into fully connected layers; however, the output from the last pooling layer is used directly for feature extraction.

3.1.6 Output Feature Map

The final output of the feature extraction process is a set of feature maps that encapsulate hierarchical information about the input image. In the proposed approach, the downstream Random Forest model in the proposed approach.

By systematically applying these convolutional, activation, and pooling operations, VGG16 effectively captures spatial hierarchies and patterns from the input images, making it an excellent choice for feature extraction in various applications, including malware detection.

3.2 Hyper-Parameter Optimization by ARO

The ARO [29] algorithm is used to optimize the hyperparameters of the Random Forest model by navigating the search space and finding the optimal settings.

The optimization process for the Random Forest model begins with a set of baseline hyperparameter values, which serve as a reference point for performance evaluation. Initially, the model is configured with 100 estimators, a maximum depth of 10, and a minimum sample split of 2. To enhance model performance, a hyperparameter search is conducted within predefined search space constraints: the number of estimators is varied between 50 and 500, the maximum depth is adjusted within the range of 5 to 50, and the minimum samples required for a split are explored between 2 and 10.

3.3 Initialization

The ARO algorithm initializes a population of artificial rabbits (R_i) with random positions (X_i) within the defined search space. Each position represents a potential solution for the hyperparameters of the Random Forest model.

$$X_i = \text{random}(X_{\min}, X_{\max}) \tag{4}$$

3.4 Fitness Evaluation

Each rabbit evaluates its position based on a fitness function $\mathrm{f}(\mathrm{X}_\mathrm{i})$, which in this context is the performance metric of the Random Forest model (e.g., accuracy, precision).

$$f(X_i) = \text{evaluate\_RF}(X_i) \tag{5}$$

3.5 Energy Factor Calculation

The algorithm calculates the energy factor of each rabbit based on its current fitness. Lower energy values correspond to better solutions.

$$E_i = \frac{1}{1 + f(X_i)} \tag{6}$$

3.6 Position Update

Rabbits update their positions based on their local and global best solutions, adjusting their direction and magnitude to move toward optimal hyperparameters.

$$X_i^{t+1} = X_i^t + \alpha \cdot (X_{\text{global best}} - X_i) + \beta \cdot \text{random}(-1, 1) \tag{7}$$

where:

- X_i^{t+1}: Position of the rabbit at the next iteration.
- X_i^t: Current position of the rabbit.
- $X_{\text{global best}}$: Position of the best solution found so far.
- α and β: Coefficients that control the influence of the global best and randomness.
- $\text{random}(-1, 1)$: A random value between -1 and 1.

3.7 Termination

The process continues iteratively until a termination condition is met, such as reaching a maximum number of iterations or achieving the desired fitness level.

These equations provide the core mechanics of how ARO optimizes the hyperparameters of the Random Forest model, guiding the search towards configurations that enhance the model's performance.

4 Results and Discussion

Our model tested on a dataset derived from Kaggle [28] consists of visual representations of Portable Executables (PEs) produced by Mallook. Two groups define these PEs: benign and malignant. While the "malicious" PEs were acquired from publicly accessible malware repositories, more significantly, the zoo-the "benign" PEs were taken from reliable programs featured in PC Magazine's "The Best Free Software of 2020."

Each executable file was converted into graphics using several resolutions, 120, 300, 600, and 1200 DPI. Two interpolation techniques, nearest and Lanczos, were used to improve the visual display of the data, thereby guaranteeing a comprehensive depiction of the structural variations between benign and malicious files.

Fig. 3 provides benign software examples with usually more homogeneous and less organized visual patterns. Fig. 4, on the other hand, shows instances of malicious software that emphasize the fundamental variations in the executable code by showing frequently clear patterns, lines, and erratic color distributions.

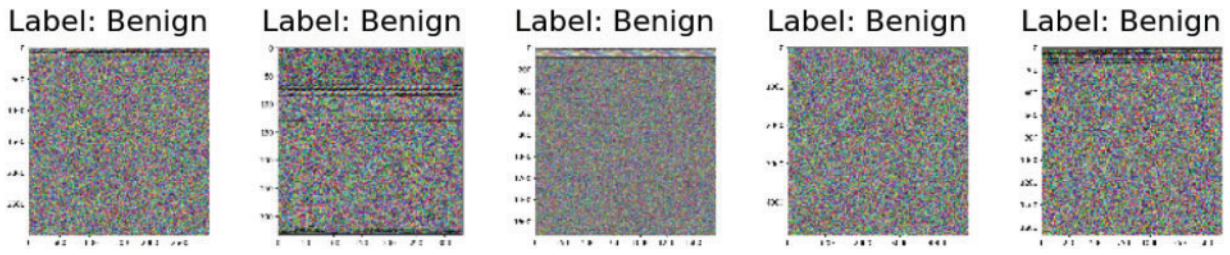


Figure 3: Benign samples

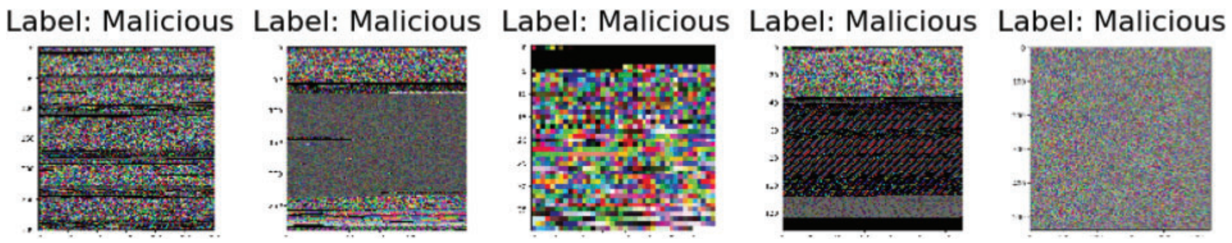


Figure 4: Malicious samples

4.1 Performance of Artificial Rabbits Optimizer (ARO)

Aiming to get optimum performance, the Artificial Rabbits Optimizer (ARO) [29] was used to fine-tune the hyperparameters of the Random Forest model after feature extraction from the picture representations using VGG16. The performance measures of ARO throughout many iterations are shown below, therefore stressing essential features of its optimization mechanism:

Fig. 5, the runtime chart, shows, in seconds, the ARO algorithm's running time throughout 100 iterations. The differences in runtime point to the computational complexity and the iteratively performed corrections. Early fluctuations balance out around the middle, indicating that exploration and exploitation are in balance when the system adjusts its settings. The latter rounds show a regular pattern, suggesting a convergence phase wherein the algorithm stabilizes its search space, lowering the running variance.

Showcasing in Fig. 6, the diversity measurement follows the exploration capacity of the algorithm throughout iterations. High initial diversity lets the algorithm search the parameter space broadly, enabling it to investigate many possible answers. The variety decreases dramatically as the iterations proceed, suggesting a move toward exploitation in which the algorithm focuses only on the most likely areas of the search space to modify the hyperparameters efficiently.

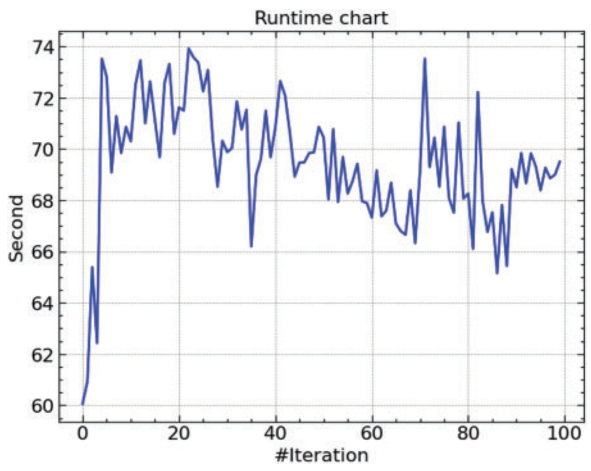


Figure 5: Runtime of ARO

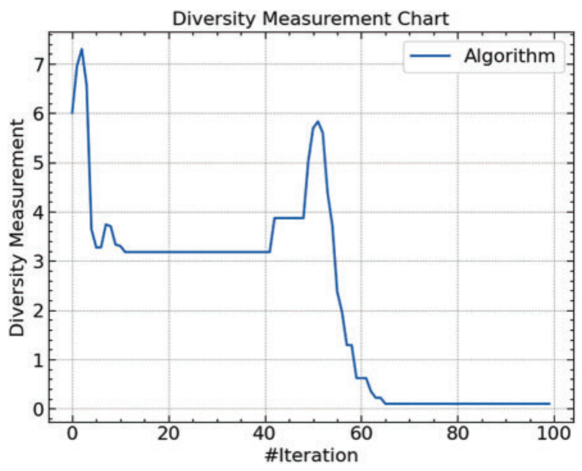


Figure 6: Diversity of ARO

Fig. 7 emphasizes the dynamic balance between exploration and exploitation throughout the optimization process. High exploration percentages in the early phases of the method highlight the search for many solutions. With time, exploitation takes front stage and shows how the process moves toward optimal parameter refinement. This change is absolutely necessary for the ARO to converge to an ideal set of hyperparameters.

As shown in Fig. 8, the Global Objectives Chart shows the objective value over the iterations. An apparent decline at iteration 40 denotes a significant performance gain in the model, most likely resulting from the ARO algorithm focusing on a very optimum parameter range. Depending on the goal set, the following flat line shows the stability of the objective function, implying either successful minimization of the error or maximization of the performance measure.

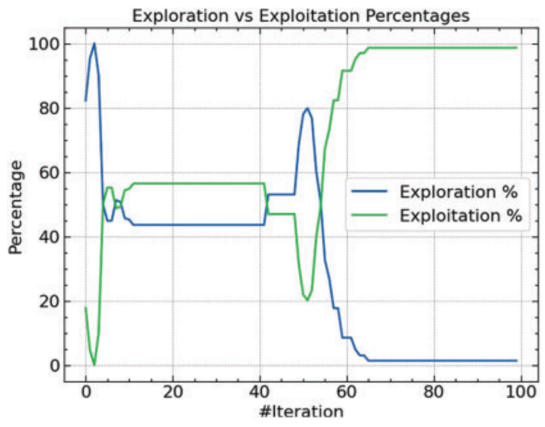


Figure 7: Exploration and exploitation of ARO

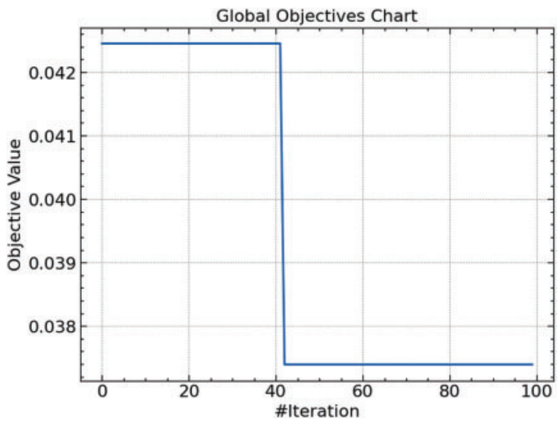


Figure 8: Global best fitness ARO

4.2 *Performance of Random Forest*

Following the Artificial Rabbits Optimizer (ARO) to find the ideal hyperparameters, the Random Forest model was trained and tested on the dataset with performance assessed using the confusion matrix shown in Fig. 9. The confusion matrix shows how precisely the model distinguishes between benign and dangerous software, therefore offering a thorough analysis of its classification outcomes. The model correctly classed 47 cases as benign (True Negatives), suggesting effective detection of non-harmful software; it also accurately recognized 59 instances of malware (True Positives), exhibiting good competence in spotting malicious files. There were no false positives, which emphasizes the excellent specificity of the model as it did not falsely identify any innocuous program as dangerous. However, the model misclassified four instances of malware as benign (False Negatives), suggesting considerable potential for development in identifying all hostile events. With excellent accuracy in identifying benign and harmful events, low false alarms, and good dependability in malware detection, the confusion matrix generally shows the resilience of the ARO-optimized Random Forest model.

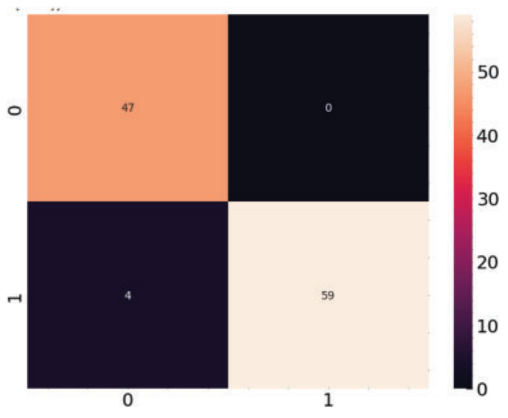


Figure 9: Confusion matrix

The ARO optimized Random Forest model's performance was systematically assessed using the classification report in Table 2. Using essential metrics: precision, recall, F1-score, and support for benign (class 0) and malicious (class 1) software, this paper offers thorough insights into the categorization capabilities of the model. Accuracy of the model's optimistic forecasts is gauged by precision. The accuracy of benign software was 0.92, meaning most cases labeled benign were accurate. With flawless accuracy of 1.00, the model proved that all expected malware cases were precisely detected, free from false positives.

Table 2: Classification report

	Precision	**Recall**	**F1-score**	**Support**
Benign	0.92	1.00	0.96	47
Malicious	1.00	0.94	0.97	63
Accuracy			0.96	110
Macro avg	0.96	0.97	0.96	110
Weighted avg	0.97	0.96	0.96	110

Recall tests the model's capacity to identify every relevant example of every class. With a recall of 1.00 for benign software, the model proved to have precisely detected all benign events without missing any. At 0.94, the recall for malware was somewhat lower, indicating that false negatives resulted from a limited number of undetectable malware occurrences. Reflecting the general correctness and usefulness of the model in categorizing the dataset, the F1-score, which strikes a balance between precision and recall, was high for both classes: 0.96 for benign and 0.97 for malware.

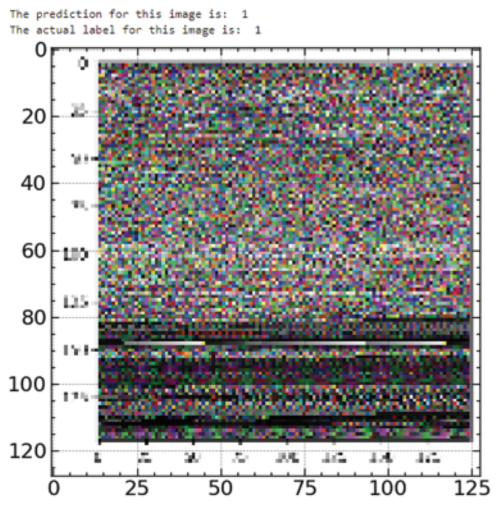
4.3 Result Prediction

We evaluated our ARO-optimized Random Forest model on a collection of random photos from the dataset to further assess its efficacy. These numbers show the actual and expected labels for every picture, proving the model's capacity to accurately categorize benign and dangerous software depending on their visual forms.

Fig. 10 displays a successfully categorized harmful picture; the prediction and the actual label indicate '1' (malicious). The model effectively determined that the intricate patterns in the picture mirror the sophisticated architecture often seen in dangerous software.

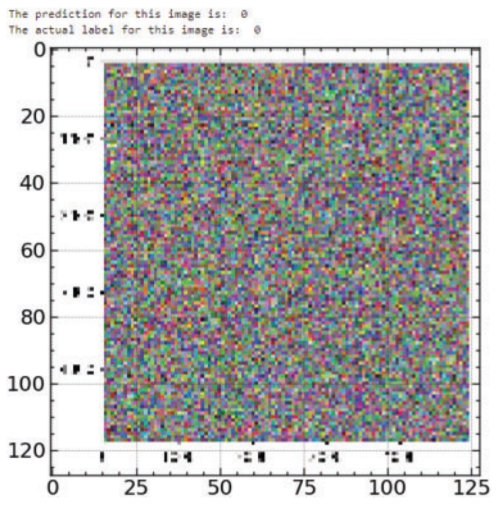
Fig. 11 shows a benign picture; the expected and actual labels are '0'. This image's homogeneous and less regimented pattern fits benign software's usual traits, highlighting the model's accuracy in spotting non-malicious events.

Figs. 12 and 13 support the accuracy of the model by providing further instances of accurately identified harmful pictures where both the actual labels and the predictions are "1." Regardless of the particular visual depiction, the variances in these photos highlight the model's resilience in spotting many kinds of malevolent trends.



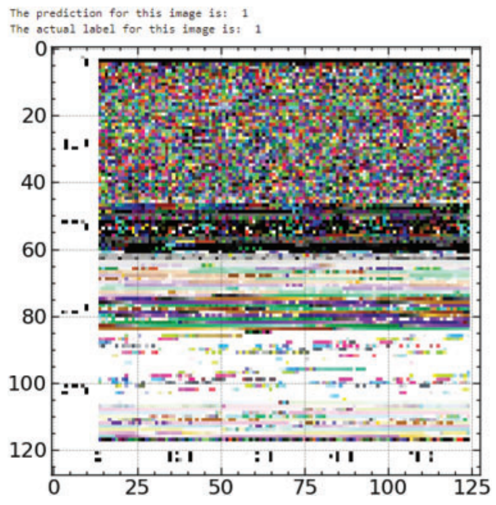
```
The prediction for this image is:  1
The actual label for this image is:  1
```

Figure 10: Test sample 1



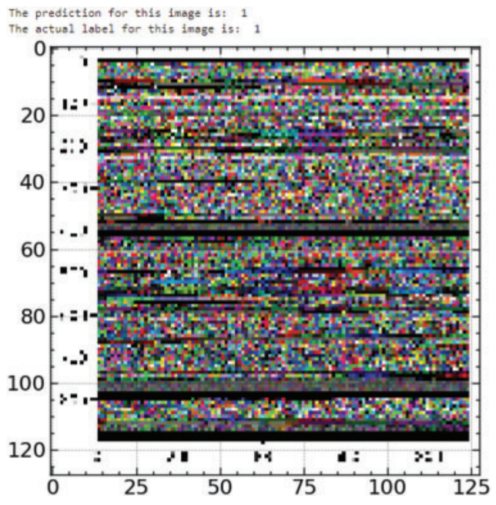
```
The prediction for this image is:  0
The actual label for this image is:  0
```

Figure 11: Test sample 2



```
The prediction for this image is:  1
The actual label for this image is:  1
```

Figure 12: Test sample 3



```
The prediction for this image is:  1
The actual label for this image is:  1
```

Figure 13: Test Sample 4

These findings, seen in the figures, show the great predictive power of the model, which often matches the real labels with the predictions. Reiterating its dependability in real-world malware detection, the proper classifications show the efficiency of the VGG16 feature extraction coupled with the ARO-optimized Random Forest model.

We used t-Distributed Stochastic Neighbour Embedding (t-SNE) to depict the predictions, as shown in Fig. 14, thus obtaining further understanding of the performance of the ARO-optimized Random Forest model. By mapping high-dimensional data to a two-dimensional space, t-SNE is a dimensionality reduction method that facilitates visualization of the data points based on the model's predictions and helps explain the separation and grouping of the data points.

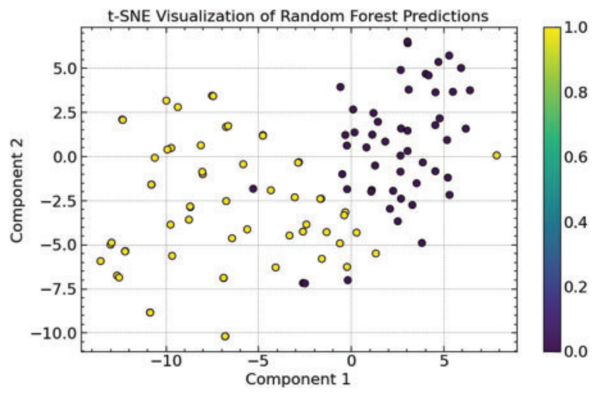


Figure 14: t-SNE visualization of random forest predictions

Fig. 10 shows the t-SNE plot of the model's predictions, where every point is a sample from the dataset and the colors reflect the prediction probability, ranging from 0 (benign) to 1 (malicious). The benign and malicious examples are separated in the plot, and different clusters develop around certain areas. The dark purple spots indicate harmful samples, while the yellow points indicate benign ones.

The differing grouping of benign and malicious data suggests that the model efficiently catches the fundamental trends separating the two groups. While the dispersal of benign points reflects the varied character of benign software, the close grouping of dangerous samples demonstrates great confidence in the model's capacity to identify malware. Apart from verifying the Random Forest model's correctness, this image shows its strength in managing intricate dataset patterns.

4.4 Comparative Analysis

The findings unequivocally show, based on comparing our suggested strategy with other models, that it constantly performs well across many assessment criteria, including accuracy, F1-score, precision, and recall.

Comparatively to Naive Bayes (87.27%) and SVM (88.18%), our suggested method attained an accuracy of 96.36%, as shown in Fig. 15. Though usually more sophisticated and computationally demanding, needing longer training durations and more resources, models like Gradient Boosting, AdaBoost, XGBoost, Logistic Regression, and k-NN also obtained comparable accuracies of 96.36%.

Fig. 16 shows the performance of our suggested method with an F1-score of 96.38%, which is on par with the highest-scoring models, including Gradient Boosting, AdaBoost, and k-NN. However, models like Naive Bayes (87.34%) and SVM (88.24%) fared much worse, suggesting that our model not only preserves high accuracy and recall but also fairly balances both measures.

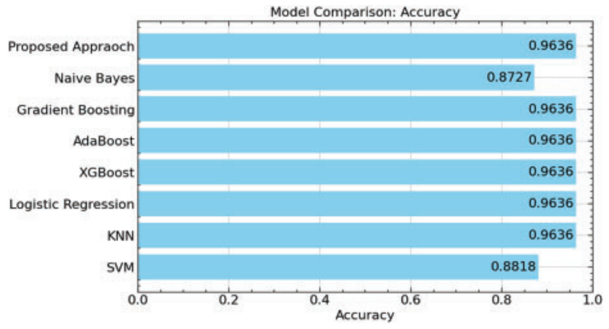


Figure 15: Accuracy comparison

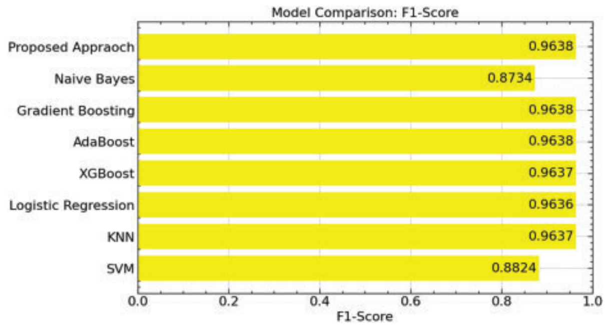


Figure 16: F1-score comparison

Our suggested model's 96.65% precision, which is on par with Gradient Boosting, AdaBoost, and k-NN, and above Naive Bayes (87.92%), and SVM (88.66%), as shown in Fig. 17. This great accuracy guarantees that our model reduces false positives, guaranteeing dependability in essential uses.

Our model attained a recall of 96.36% (Fig. 18), which aligns with other high-performance models such as XGBoost, Gradient Boost, and AdaBoost. The poor recall rates of Naive Bayes (87.27%) and SVM (88.18%) underline even more the resilience of our method in accurately spotting positive situations.

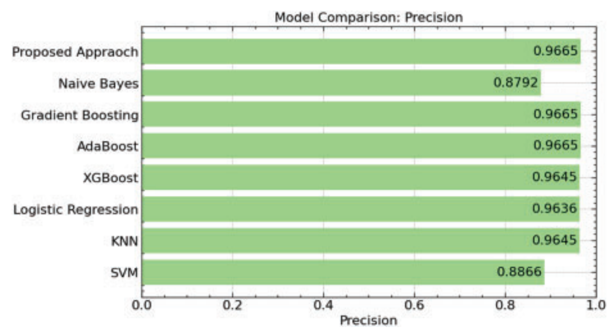


Figure 17: Precision comparison

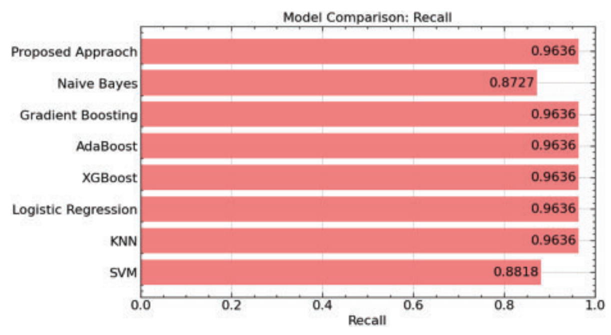


Figure 18: Recall comparison

Although our suggested method has a simpler structure, resulting in reduced computational cost, it meets or surpasses the performance of more complicated models across all measurements. This efficiency distinguishes it from other, more complicated models that need longer training cycles and greater computing capability, making it better for practical uses where speed and resource economy are crucial.

4.5 Computational Complexity Analysis

A balance between computational efficiency and predictive performance depends on choosing a suitable machine learning model. Based on results from Singh [30], Random Forest is quite an effective supervised learning model because of its favorable trade-off between training difficulty, inference speed, and generalization capabilities. Built on Random Forest, our model gains from these features; hence, it is a good solution for large-scale projects. The paper [30] shows that Random Forest guarantees scalability as the dataset size rises by attaining a training complexity of $O(knlog(n))$, while k is the number of trees. Random Forest preserves computational feasibility while providing strong performance, unlike Support Vector Machines (SVM), which have a significantly greater training complexity of $O(n^2d^2)$ and become intractable for big datasets. The complexity of random forest also makes it more efficient than models such as k-Nearest Neighbors (k-NN), which suffers from a considerable inference cost of $O(klog(n)d)$. Moreover, unlike decision trees, which sometimes overfit, random forests use an ensemble of trees to lower variance and improve predictive accuracy. Random Forest is more suited to manage non-linearity and high-dimensional data than more basic models, such as logistic regression and naïve Bayes. With these improvements, our model shows an ideal balance between accuracy and efficiency and fits the computational gains reported in previous work rather nicely. Thus, we find from theoretical complexity analysis and empirical explanation that our approach is computationally efficient, scalable, and appropriate for practical applications needing high-performance learning.

5 Conclusion

This work uses VGG16 for feature extraction paired with a Random Forest model optimized using the ARO algorithm to provide an efficient AI-driven malware detection method. While reducing computational complexity, the suggested model performs better than conventional models, with high accuracy, F1-score, precision, and recall. Using modern feature extraction and optimization methods, this method efficiently solves the difficulties of malware detection and offers a strong and quick solution fit for real-time applications. The results show the value of combining artificial intelligence-driven feature extraction with ideal machine learning models, setting a new bar for malware detection in challenging surroundings. This study offers a scalable and effective methodology for malware detection in smart devices, greatly improving cybersecurity measures.

Acknowledgement: This research work was funded by Institutional Fund Projects under grant no. (IFPDP-261-22). Therefore, the authors gratefully acknowledge technical and financial support from the Ministry of Education and Deanship of Scientific Research (DSR), King Abdulaziz University (KAU), Jeddah, Saudi Arabia.

Funding Statement: This research work was funded by Institutional Fund Projects under grant no. (IFPDP-261-22). Therefore, the authors gratefully acknowledge technical and financial support from the Ministry of Education and Deanship of Scientific Research (DSR), King Abdulaziz University (KAU), Jeddah, Saudi Arabia.

Author Contributions: Final manuscript revision, funding, supervision: Brij B. Gupta, Ching-Hsien Hsu; study conception and design, analysis and interpretation of results, methodology development: Akshat Gaurav, Varsha Arya; data collection, draft manuscript preparation, figure and tables: Wadee Alhalabi, Shavi Bansal. All authors reviewed the results and approved the final version of the manuscript.

Availability of Data and Materials: All data generated or analysed during this study are included in this published article.

Ethics Approval: This article contains no studies with human participants or animals performed by any authors.

Conflicts of Interest: The authors declare no conflicts of interest to report regarding the present study.

References

1. Singh A. Classification of malware in https traffic using machine learning approach. El-Cezeri. 2022;9(2):644–55.
2. Hussain S. Deep learning based hybrid analysis of malware detection and classification: a recent review. J Cyber Secur Mobil. 2023;13(1):91–134. doi:10.13052/jcsm2245-1439.1314.
3. Agrawal AK, Sah S, Khatri P. Forensic analysis of a ransomware. Int J Innov Technol Explor Eng. 2020;9(3):3618–22.
4. Feng T, Cui Y. Particle swarm algorithm for smart contract vulnerability detection based on semantic web. Int J Semant Web Inform Syst (IJSWIS). 2024;20(1):1–33. doi:10.4018/ijswis.342850.
5. Palla T, Tayeb S. Intelligent mirai malware detection for IoT nodes. Electronics. 2021;10:1241. doi:10.3390/electronics10111241.
6. Aslan Ö, Ozkan-Okay M, Gupta D. A review of cloud-based malware detection system: opportunities, advances and challenges. Eur J Eng Technol Res. 2021;6:1–8. doi:10.24018/ejers.2021.6.3.2372.
7. Hassan S. Detection of malicious software and control using artificial intelligence. Int J Electron Crime Investig. 2023;7:17–34. doi:10.54692/ijeci.2023.0703159.
8. Zhou Q, Wang Z. A network intrusion detection method for information systems using federated learning and improved transformer. Int J Semant Web Inform Syst (IJSWIS). 2024;20(1):1–20. doi:10.4018/ijswis.334845.
9. Sihwail R, Omar K, Ariffin K. A survey on malware analysis techniques: static, dynamic, hybrid and memory analysis. Int J Adv Sci Eng Inform Technol. 2018;8:1662–71. doi:10.18517/ijaseit.8.4-2.6827.

10. García L, Bermejo R. A method for malware analysis by virtual machine introspection technique. Res Comput Sci. 2018;147:11–20.
11. Cha H. Intelligent anomaly detection system through malware image augmentation in iiot environment based on digital twin. Appl Sci. 2023;13:10196. doi:10.3390/app131810196.
12. Kumar S, Janet B, Neelakantan S. IMCNN: intelligent malware classification using deep convolution neural networks as transfer learning and ensemble learning in honeypot enabled organizational network. Comput Commun. 2024;216:16–33. doi:10.1016/j.comcom.2023.12.036.
13. Rao BS, Sharma V, Rathore N, Prasad D, Anandaram H, Soni G. A secure framework to prevent three-tier cloud architecture from malicious malware injection attacks. Int J Cloud Appl Comput (IJCAC). 2023;13(1):1–22. doi:10.4018/ijcac.317220.
14. Ni S, Qian Q, Zhang R. Malware identification using visualization images and deep learning. Comput Secur. 2018;77:871–85.
15. Sharma A, Singh SK, Chhabra A, Kumar S, Arya V, Moslehpour M. A novel deep federated learning-based model to enhance privacy in critical infrastructure systems. Int J Softw Sci Comput Intell (IJSSCI). 2023;15(1):1–23. doi:10.4018/ijssci.334711.
16. Venkatraman S, Alazab M, Vinayakumar R. A hybrid deep learning image-based analysis for effective malware detection. J Inf Secur Appl. 2019;47:377–89. doi:10.1016/j.jisa.2019.06.006.
17. Gupta BB, Gaurav A, Arya V, Bansal S, Attar RW, Alhomoud A, et al. Earthworm optimization algorithm based cascade LSTM-GRU model for android malware detection. Cyber Secur Appl. 2025;3:100083. doi:10.1016/j.csa.2024.100083.
18. Mercaldo F, Santone A. Deep learning for image-based mobile malware detection. J Comput Virol Hacking Tech. 2020;16:157–71. doi:10.1007/s11416-019-00346-7.
19. Mishra P, Jain T, Aggarwal P, Paul G, Gupta BB, Attar RW, et al. CloudIntellMal: an advanced cloud based intelligent malware detection framework to analyze android applications. Comput Electr Eng. 2024;119:109483. doi:10.1016/j.compeleceng.2024.109483.
20. Naeem H, Ullah F, Naeem M, Khalid S, Vasan D, Jabbar S, et al. Malware detection in industrial internet of things based on hybrid image visualization and deep learning model. Ad Hoc Netw. 2020;105:102154. doi:10.1016/j.adhoc.2020.102154.
21. Chai Y, Qiu J, Yin L, Zhang L, Gupta BB, Tian Z. From data and model levels: improve the performance of few-shot malware classification. IEEE Trans Netw Serv Manage. 2022;19(4):4248–61. doi:10.1109/tnsm.2022.3200866.
22. Ghahramani M, Taheri R, Shojafar M, Javidan R, Wan S. Deep image: a precious image based deep learning method for online malware detection in IoT Environment. arXiv:2204.01690. 2022.
23. Sharma A, Gupta BB, Singh AK, Saraswat V. Orchestration of APT malware evasive manoeuvers employed for eluding anti-virus and sandbox defense. Comput Secur. 2022;115:102627. doi:10.1016/j.cose.2022.102627.
24. Brosolo M, Puthuvath V, Ka A, Rehiman R, Conti M. SoK: visualization-based malware detection techniques. In: Proceedings of the 19th International Conference on Availability, Reliability and Security; 2024; New York, NY, USA. p. 1–13.
25. Zou B, Cao C, Tao F, Wang L, Zou B, Cao C, et al. IMCLNet: a lightweight deep neural network for image-based malware classification. J Inf Secur Appl. 2022;70:103313. doi:10.1016/j.jisa.2022.103313.
26. Kumar MR. IVMCT: image visualization based multiclass malware classification using transfer learning. Math Stat Eng Appl. 2022;71(2):42. doi:10.17762/msea.v71i2.65.
27. Han KS, Lim JH, Kang B, Im EG. Malware analysis using visualized images and entropy graphs. Int J Inf Secur. 2015;14:1–14. doi:10.1007/s10207-014-0242-0.
28. Matthew M. Malware as Images. 2020. [cited 2024 Jan 30]. Available from: https://www.kaggle.com/datasets/matthewfields/malware-as-images.
29. Wang L, Cao Q, Zhang Z, Mirjalili S, Zhao W. Artificial rabbits optimization: a new bio-inspired meta-heuristic algorithm for solving engineering optimization problems. Eng Appl Artif Intell. 2022;114:105082. doi:10.1016/j.engappai.2022.105082.
30. Singh J. Computational complexity and analysis of supervised machine learning algorithms. In: Next Generation of Internet of Things: Proceedings of ICNGIoT 2022; 2022; Singapore: Springer. p. 195–206.