IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received October 9, 2020, accepted October 27, 2020, date of publication November 6, 2020, date of current version November 25, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3036491

Intelligent Vision-Based Malware Detection and Classification Using Deep Random Forest Paradigm

S. ABIJAH ROSELINE[1], S. GEETHA[1], (Senior Member, IEEE), SEIFEDINE KADRY[2], (Senior Member, IEEE), AND YUNYOUNG NAM[3], (Member, IEEE)

[1]School of Computer Science and Engineering, Vellore Institute of Technology, Chennai Campus, Chennai 600127, India
[2]Department of Mathematics and Computer Science, Faculty of Science, Beirut Arab University, Beirut 1107 2809, Lebanon
[3]Department of Computer Science and Engineering, Soonchunhyang University, Asan 31538, South Korea

Corresponding author: Yunyoung Nam (ynam@sch.ac.kr)

This work was supported in part by the Ministry of Science and ICT (MSIT), South Korea, through the ICT Challenge and Advanced Network of HRD (ICAN) Program supervised by the Institute of Information and Communications Technology Planning and Evaluation (IITP), under Grant IITP-2020-0-01832, and in part by the Soonchunhyang University Research Fund.

ABSTRACT Malware is a rapidly increasing menace to modern computing. Malware authors continually incorporate various sophisticated features like code obfuscations to create malware variants and elude detection by existing malware detection systems. The classification of unseen malware variants with similar characteristics into their respective families is a significant challenge, even if the classifier is trained with known variants belonging to the same family. The identification and extraction of distinct features for each malware is another issue for generalizing the malware detection system. Features that contribute to the generalization capability of the classifier are difficult to be engineered with modifications in each malware. Conventional malware detection systems employ static signature-based methods and dynamic behavior-based methods, which are inefficient in analyzing and detecting advanced and zero-day malware. To address these issues, this work employs a visualization approach where malware is represented as 2D images and proposes a robust machine learning-based anti-malware solution. The proposed system is based on a layered ensemble approach that mimics the key characteristics of deep learning techniques but performs better than the latter. The proposed system does not require hyperparameter tuning or backpropagation and works with reduced model complexity. The proposed model outperformed other state-of-the-art techniques with a detection rate of 98.65%, 97.2%, and 97.43% for Malimg, BIG 2015, and MaleVis malware datasets, respectively. The results demonstrate that the proposed solution is effective in identifying new and advanced malware due to its diverse features.

INDEX TERMS Malware, malware classification, malware detection, malware images, malware variants, malware visualization.

I. INTRODUCTION

The internet has become a key aspect of our daily lives. Although making our lives convenient, the internet has made innocent users vulnerable to attacks. The rise of the internet and the emergence of social networks have triggered exponential growth in malware. The evolution of malware starts as a hobby of technical enthusiasts and is now pinned with the main motive of making money. Attackers commonly use phishing or email as infection vectors. The main targets of cyber-attacks are the food industry, logistics industry, and non-profit concerns [10]. Recent threats are posed by information [30], similar to the earlier banking Trojans. TrickBot and Emotet are significant droppers, which carry multiple malware modules for performing various malicious actions like sending spams, erasing sensitive data, crypto-wallet theft, etc. These malware variants focused a great deal of attention on ensnaring businesses, targeting profit from selling the most sensitive information.

The associate editor coordinating the review of this manuscript and approving it for publication was Yichuan Jiang.

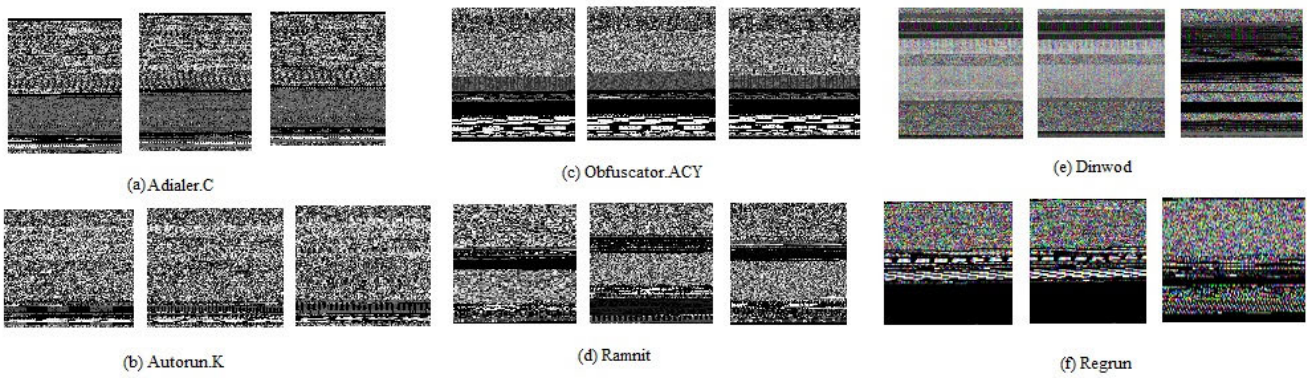

FIGURE 1. Some malware sample images belonging to distinct families of three malware datasets.

Malware detection and classification is one of the most significant problems in the area of cybersecurity. Signature-based methods are effective against known malware, but they are ineffective against advanced and unknown malware. Malware authors introduce evasion techniques like obfuscation, encryption, packing, etc. on existing malware to elude detection, leading to more number of new malware. Malware files with the same malicious behavior belong to the same malware family. These files are continuously modified using different tactics, which make them very distinct. These large volumes of files are grouped into their respective families by effective malware detection and classification process. Basically, most of the existing malware classifiers extract significant features from the malware files and train a machine learning model with those features. The trained model can distinguish between malware and a cleanware. This is at a very preliminary level. The classifiers face a lot of challenges because of the tactics like obfuscation, packing, and other subtle modifications the malware authors practice, in order to escape the malware classifiers. The features extracted from such modified malware, do not reveal any useful clue to the classifier and thus are causes for the failure of the malware detector. This is quite possible with any machine learning-based malware detector and classifier. Hence, the malware and anti-malware community holds an arms race situation and is a never-ending story in the cybersecurity domain. This creates a pressing need to approach the malware detection in an angle of extracting features that are robust to any sort of modifications practiced by the malware authors and could detect a malware in spite of any changes introduced in the malware script. Further, the variant of an existing malware should also be trapped by the system.

This research work makes such an attempt and is found to be succeeding to a greater extent. In this paper, binary file executables (malware & cleanware) are analyzed using vision- based analysis technique. The malware authors reuse the same malicious code segments to generate new malware variants. This is easily observed when we visualized the binary code of the malware. This approach is called the Vision-based Analysis Technique. Malware visualization has recently been used as an alternative and efficient approach for malware analysis. This approach does not require an in-depth analysis.

The similarity in malicious code variants is visualized and identified. Each malware family exhibited a particular texture in the respective malware images. These texture patterns in the images are found to be exhibiting significant visual similarities to malware belonging to the same class. Another advantage is that vision-based analysis does not need static disassembly or dynamic execution of binaries, unlike other traditional malware analysis techniques. Texture features could be extracted from the malware image and could be employed for training the classifier. This will not fail since, due to obfuscation or packing or reassembling, the texture will occur at a different position in the malware image. So the features can act as the telltale clue to detect the occurrence of the texture and thus help in finding the malware, their invariants, and any of their modified versions. This technique aids better performance for classifying malware and is resilient to obfuscation and other modification techniques.

The binary files of malware are converted as 8-bit vectors consisting of a string of zeros and ones and are organized into two-dimensional matrices forming grayscale images $g = f(m, n)$, where g is the gray level. The intensity levels vary from G_{min}, G_{max} to $[0, G-1]$, for $G_{min} \leq g = f(m, n) \leq G_{max}$. We represent $g = 0$ for black, $g = G-1$ for white, and intermediate values for shades of black to white. Malware authors alter a small part of the binary, and images visualize these small changes maintaining their global structure. Fig. 1 shows some of the image patterns of families in three datasets. Fig. 1(a) and (b) shows the sample images of Adialer.C and Autorun.K families in the Malimg dataset. Fig. 1(c) and (d) shows the sample images of Obfuscator.ACY and Ramnit families in Microsoft BIG 2015 dataset. Fig. 1(e) and (f) show the sample images of Dinwod and Regrun families in the MaleVis dataset. The similarity in the texture pattern is found to be more among the malware of the same family.

The next step in the process is to extract features of these patterns from the images and train a classifier with those features. Over the past decade, many research efforts have been taken to develop such a classifier with the malware features. Data mining and machine learning techniques are employed to develop intelligent malware detection and classification systems. Deep neural networks have reached significant success in diverse applications, especially in the field of computer vision. Although deep learning models are powerful, they have certain limitations in real-world detection tasks, especially in security domains. With the flow of zero-day and unlabeled malware, the detection performance using deep learning is also low. These deep models are very intricate and require high computational overhead. Also, they require a substantial number of hyperparameters, and performance improvement is achieved by tuning them appropriately.

Deep learning models are trained by backpropagation, whereas the proposed model is trained using tree learners. Considering all these factors, this research work proposes to employ an ensemble deep forest algorithm, which has many advantages over the existing machine learning and deep learning models. It has high generalization ability, improved detection accuracy & precision, and low computational overhead.

Various types of forests and random subspace sampling for high-dimensional data are adopted in order to promote diversity in the layered ensemble approach. The proposed malware detection and classification system is context-aware and generalizes well. As far as the model complexity is concerned, the deep forest approach uses less hyper-parameters compared with deep learning and performs well with default setting. Thus, it does not require hyper-parameter tuning. Deep forests work well even with small scale data where deep learning typically fails. The model does not require additional computational resources such as Graphical Processing Unit (GPU). Deep forest do not suffer from overfitting, as randomness is increased by the use of different grains of sliding windows, and ensemble forests generate class vectors using cross-validation.

The main contributions of this paper are as follows.

- Adaptation of Vision-based Malware Analysis Technique, where-in the malware executable files are converted as grayscale images to exploit global features. The risk of executing the malware for analysis as in Dynamic Analysis, and the requirement of intense knowledge in opcodes/assembly language to understand the malware codes as in Static Analysis & reverse Engineering are not involved at all. It involves a clean, straight forward, simple, and powerful visual feature engineering & extraction approach.
- Employed a novel ensemble deep forest approach [45] for malware detection and classification, which is competitive and superior over deep learning and other machine learning techniques. The proposed approach is data-independent and learns the discriminative representation from the data itself rather than depending on hand-crafted feature descriptors.
- Extensive empirical investigation on three benchmark malware datasets (Malimg, Microsoft, and MaleVis) to demonstrate the efficiency of the proposed method over traditional and contemporary methods.
- Superior accuracy than the existing schemes: A higher detection performance of 98.86% without requiring to extract local features, indicating the proposed method as a promising approach in detecting novel, zero-day, and even obfuscated malware.
- Low Model Complexity: The proposed system does not require hyper-parameter tuning or backpropagation and works with reduced model complexity.

The rest of the paper is organized as follows. Section II discusses the related works on various malware analysis and detection techniques. Section III describes the proposed model. Section IV presents the mathematical proof of the proposed deep forest-based malware detection system. Section V describes the details of the datasets used and the experimental setup. Section VI discusses the results and performance analysis with other works. Finally, Section VII presents the conclusion of the paper.

II. RELATED WORK

Malware analysis approaches include static, dynamic, hybrid, and vision-based approaches. The features are extracted by using any of the analysis approaches and based on these features, the file is classified as malware or cleanware by the trained classifier.

A. STATIC ANALYSIS

Static analysis analyzes the binary file executable by disassembling it without execution. Signature-based methods or pattern-matching methods detect malware by identifying a match with the malware signature database that is updated frequently. The features extracted from these methods include hash signatures, string sequences [17], [36], byte sequences, system resource information [36]. Portable Executable (PE) file features [34] like the list of DLLs, DLL function calls list, the number of different function calls within each DLL, Number of (standard sections, non-standard sections, Executable sections, Readable/Writable/Executable sections, entries in the IAT), and Entropies of the (PE header, code sections, data sections, and entire PE file). Many traditional signature-based methods like antivirus programs use hash values of binary files as signatures [49]. This method is ineffective for unknown zero-day and obfuscated malware. The exponential growth and scalability issues of malware signature databases are also a major drawback.

To combat these issues, automatic signature generation methods such as string sequences [17], function-based methods [37] constructing common sequences or segments were used as unique signatures. The semantics-aware malware detector [8] employs a template-based approach that is capable of detecting malware variants, using a unique template.

Their method is resilient to some obfuscations like instruction reordering, register renaming, and garbage insertion. The techniques like instruction replacement, equivalent functionality, and reordered memory accesses are not handled. Static analysis techniques do not generalize for unseen malware, are computationally expensive, require expertise, and are slow. It takes more time to discover it as malware, and by that time, the malware could have created the damage by performing its intended malicious actions.

B. DYNAMIC ANALYSIS

The dynamic analysis approach addresses the drawbacks of static analysis, with a great deal of attention towards tracking the runtime behavior of the executables. This analysis allows executables to run in a safe and virtual environment to prevent system outbreaks. The features extracted using dynamic analysis includes n-grams, operational code (opcode) sequences, API/system call sequences, control flow graphs. The work presented in [24] extracted byte code sequences from executables and converted them into n-gram features. The opcode sequences and their appearance frequencies [35] are extracted from executables to identify unknown malware. This method was observed ineffective in detecting packed malware. The Spatio-temporal features are extracted from API traces [3] and achieved a higher accuracy rate in identifying malware. The dynamic analysis approach is ineffective when malware authors design the same malware with different behaviors in different instances or behaviors exhibition by user triggers. This technique cannot efficiently detect malware in real-time.

1) GRAPH-BASED DETECTION

The dynamic malware behaviors, such as system calls and API calls, are visualized using dependency graphs [23], [13]. The dependencies are analyzed by tracing the input and output arguments of the system or API calls. The API dependency graphs [12] are constructed from malware files and common behavior graphs are extracted from them. A non-string feature space containing semantic relevance paths is developed [33] in terms of the Average Logarithmic Branching Factor (ALBF) metric. These semantically relevant paths are extracted from the system-level information flow. The 2-grams features are extracted from binary files to form a Markov chain [4], which is transformed into a graph. Then, a similarity matrix is generated to classify malware. Various analytical models were proposed to analyze the propagation dynamics of malware in networks using Spatio-temporal measures [7], difference equation [43], mean-field approximation [11]. Dynamic analysis techniques involve the execution of the malware in a controlled environment like Cuckoo sandbox or virtual machine, letting the malware to cause the damage and then analyzing its persistence mechanism, spreading mechanism and their vulnerability exploration and exploitation aspects. It requires an extremely safe controlled environment and an intense knowledge in analyzing the effects caused by the malware attack. Even a small ignorance and negligence in the environment could cause very serious damage to the network and systems.

2) MACHINE LEARNING-BASED DETECTION

Data mining and machine learning techniques identify malware based on features extracted from static analysis, dynamic analysis, vision-based analysis, or a combination of the analysis methods. Data mining classifiers are trained using various features to classify unknown malware. Some works employ feature ranking [42] using machine learning classifier for effective malware classification. The framework presented by [36] is based on static features that work for three learning algorithms, such as an inductive rule-based learner, a probabilistic method, and a multi-classifier system. Their approach was limited to large-scale real-world samples. Reference [24] implemented machine-learning techniques such as naive Bayes, decision trees, support vector machines, and boosting based on n-gram features. The authors inferred that boosted decision trees outperformed other methods. Their approach is limited to code obfuscations and determining the functional properties of malware. The static PE file features are extracted from executables [34] and experiments on machine learning classifiers such as Naive Bayes, J48 decision tree, Bagged J48, KNN, Multi-Layer Perceptron (MLP) and Entropy Threshold are conducted to classify packed and unpacked samples. Reference [35] constructed vectors from opcode sequences of executables and trained machine learning classifiers. Their approach is limited to packed executables. The instruction sequences are extracted from PE files [14] and generated malicious sequential patterns to perform feature representation. The authors performed malware detection using All-Nearest-Neighbor (ANN) classifier. However, the success of these machine learning-based detectors depends heavily on the features extracted and the capability of the machine learning model selected. In case the features could not give any clue because of the modifications like packing, obfuscation, subtle changes in sequencing done on them by the malware authors, the classifier will become ineffective. Also, frequent tuning of the trained classifier with the latest advanced malware samples is needed for effective detection.

C. VISION-BASED ANALYSIS

Malware researchers employ a vision-based analysis approach for analyzing malware binaries by visualizing them as images. Then, malware is characterized using image-based features. This involves either taking the whole image as a single vector or taking the local and global features extracted from the image. The images are then classified into their respective families using machine learning classifiers. Reference [31] first proposed the visualization of malware binaries as images and classification was based on global image features. They inferred that attackers could obfuscate the malware images by section relocation, insertion of largely redundant data, etc. To overcome these attacks, many local feature extraction techniques are employed for vision-based

malware analysis to identify the unique features of malware binaries and their original code segments.

The texture features such as GIST [32], Gray Level Co-occurrence Matrix (GLCM) [9], Wavelet-based features [21] are extracted from the images to classify malware variants. The three types of features, such as Intensity-based, Wavelet-based, and Gabor based features are extracted to classify malware using an SVM classifier. Reference [15] extracted global features like texture and color features, and local features like code and data sections from color images of malware binaries.

Malware researchers used image features to classify malware using machine learning and deep learning classifiers [20], [25], [22]. Experiments were conducted for malware classification using machine learning classifiers such as Logistic Regression (LR) [40], Naïve Bayes (NB) (Vinayakumar *et al.*, 2019), Decision Tree (DT) [40], Random Forest (RF) [40], [15], K-Nearest Neighbor (KNN) [31] and Support Vector Machine (SVM) [31], [40]. Reference [26] proposed an approach that extracts Local Binary Patterns (LBP), and dense Scale-Invariant Feature Transform (SIFT) features from malware image. Reference [40] presented a scalable and hybrid deep learning model called ScaleMalNet based on malware image processing to detect new malware. They assessed the performance of machine learning and deep learning models on all available malware datasets for malware detection and classification. They inferred that the proposed deep learning model outperformed the machine learning techniques in detecting malware. The whole binary image features are extracted automatically [9] for classifying malware using Convolutional Neural Networks (CNN). The authors used a data equilibrium approach based on a bat algorithm to solve the data imbalance problem among various malware classes. A hybrid approach [1] was presented by combining deep learning and SVM algorithms to classify malware from the Malimg dataset. However, these approaches involve heavy computational complexity and are subject to a huge amount of hyperparameter tuning of the deep learning models.

D. HYBRID ANALYSIS

The hybrid analysis approach involves the extraction of integrated features by combining any of the other malware analysis approaches to detect and classify malware. The combinations of static and dynamic features improve detection performance [24], [47]. Reference [19] combined static features such as function length-frequency vectors and printable string information vectors and dynamic API features as a single vector. The opcode sequences and system call features are extracted from binaries using dynamic analysis. Then these dynamic features [44], [18] are visualized as images. A similarity estimation approach [18], [48] based on the vector representation of image matrices is constructed from static and dynamic features. The samples are represented using a novel approach [39] by forming clusters based on similarity functions of each resource types such as file names, registry names, mutexes and domain names. They used machine learning classifier to classify malware from cleanware. These methods lacked generalization ability which is needed for malware detection, since new and variant of existing malware are always expected to flow in the network.

III. PROPOSED MODEL

A. OVERVIEW

Fig. 2 shows the overall design of the proposed malware detection system. The proposed model consists of two phases, analysis and classification phase. In the analysis phase, each PE binary file is transformed into a 2D array and visualized as a grayscale image. The second phase involves the classification of malware into their corresponding classes based on the image patterns. This phase uses the deep forest approach for malware detection and classification. The deep forest approach includes two stages, namely, sliding window scanning and cascade layering. Motivated by the concept of Convolutional Neural Networks (CNNs), the sliding window scanning stage is adopted to preserve the spatial relationship between raw pixels. Each binary image in the training set is given as input to the classification phase. The first stage begins with scanning the input image with parallel processing of two sliding windows. A set of smaller instances are obtained from each sliding window and are trained with two ensembles. The predictions from both the ensemble forests return class probability vectors. These class vectors from both sliding windows are concatenated to form multidimensional feature vectors that enable the model to boost the performance.

The resulting transformed feature vectors from sliding window scanning are passed as input to the cascade layering stage. This stage consists of sequential layers, with each layer comprising of four ensemble forests. The deep forest model adaptively chooses the number of layers based on the input data leading to lower model complexity. Each forest generates class vectors using k-fold cross validation to minimize the problem of overfitting. In every layer, the input feature vector is trained k-1 times using k-1 forests and each forest outputs k-1 class vectors. Each layer generates final class vector by averaging the k-1 class vectors. Here 5-fold cross validation is used and the instances are trained four times using four ensemble forests. The feature representation at each cascade layer is performed using class vector probability of each ensemble forest. The class vector probabilities are estimated by averaging the class vectors across all decision trees in the ensemble. The layering process terminates when there is no improvement in accuracy. The ensemble outputs the prediction probabilities of different classes and the average probability values for every class are calculated from each of the ensembles. The class with maximum average probability value is identified as the corresponding class of the input sample. The steps of the proposed model are given in Algorithm 1.

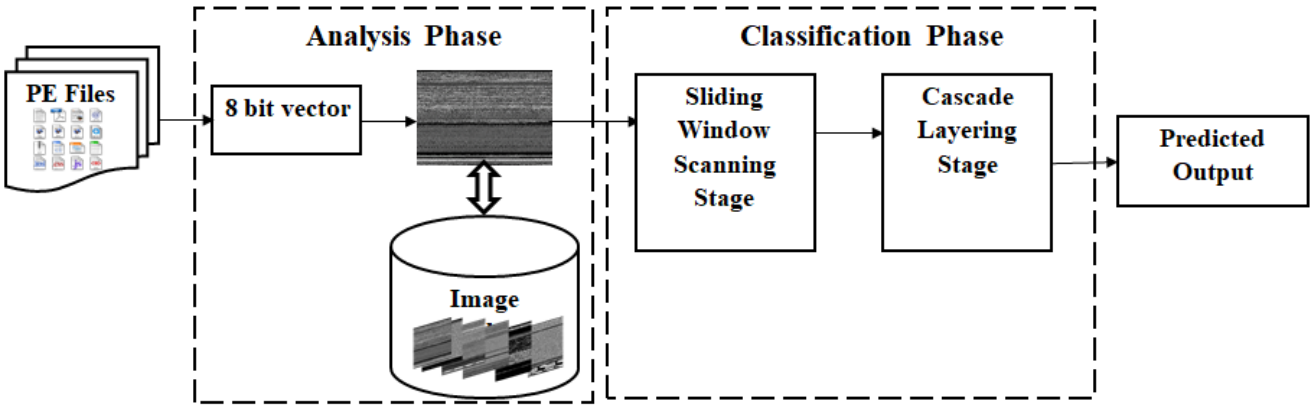

FIGURE 2. Proposed malware detection model.

Algorithm 1 Proposed Malware Detection and Classification Model

Input: $P = \{(x_f, y_f)\}_{f=1}^{N}$: the training set
Output: Predicted class y_f

repeat
  for each PE file $f \in P$ **do**
    Visualize as grayscale image: $g \leftarrow f(m, n)$
    Resize $g : m \times n$
    Set sliding windows $W \in \{w_1, w_2$ of size $i \times j$.
    repeat
      for each W **do**
        Sliding Window Scanning Phase ()
      end for
    until W
    Cascade Layering Phase ()
    Compute the average of class probabilities $v_f = \frac{1}{E}\sum_{e=1}^{E} v_{f,e}$
    Return y_f
  end for
until the last binary file $f = (x_N, y_N)$

B. PREPROCESSING PHASE

The PE binary files (malware or cleanware) are given as input to the proposed model. Each file comprises the hexadecimal representation of its binary content. Each binary is converted using a function that processes hexadecimal into images in.png format. First, each line of a binary is scanned and every set of eight characters are stored in an array. Then, each byte is converted to its decimal equivalent and stored it in another array. This conversion process is repeated for all the lines of the binary file. The array with decimal values is converted to visualize binary images using a Python Imaging Library (PIL) package.

Experiments were conducted by giving squared images (aspect ratio 1:1) as input to the proposed classification model. It was observed that the model did not show good performance. By keeping aspect ratio of the image as 1:3 (various sizes such as 10×30, 20×60, 30×90), the model showed better performance, but required higher computations and memory. This can be attributed to the PE files having certain format which restricts the width of the files while the height of the program is generally large. Also, images with smaller dimensions lead to loss of sensitive information. In order to minimize the cost in terms of time, space and information loss, the image dimensions are resized to 30×90 for the experiments.

C. SLIDING WINDOW SCANNING STAGE

Each input binary image is scanned using two sliding windows of different sizes such as 10×10 and 30×30. Based on the input dimensions, sliding window size, and stride value, the number of smaller image instances are generated for further processing. The dimensions of the input image is $g : m \times n \rightarrow 30 \times 90$. The sliding window is denoted as $i \times j$ and stride $s = 1$. The number of smaller images S are generated by using the formula, $S = v_1 \times v_2$, where $v_1 = m - i + s$ and $v_2 = n - j + s$. For window $w_1 : i \times j \rightarrow 10 \times 10$, $v_1 = 21$, $v_2 = 81$, resulting in $S = 1701$. Thus, for 10×10 window size, 1701 smaller image instances are generated. Similarly, for the window size of 30×30 denoted as w_2, 61 smaller image instances are generated.

Then, the smaller images of each sliding window are processed by two ensembles such as Random Forest (RF) and Completely Random Forests (CRF) classifiers. Both RF and CRF forests take decisions from multiple decision trees. Each ensemble determines a probability distribution over classes for densely sampled instances from the input image. The class distributions are estimated at the leaf nodes into which the concerned instance falls. The final feature vectors are obtained by concatenating class vectors from the two sliding windows. The aggregation is based on the probability rather than majority voting. The probability values are observed to be reflecting the class label of the malware family, more precisely. The chances of 50/50 predictions are very low and in such cases, the predictions of CRF model are given more

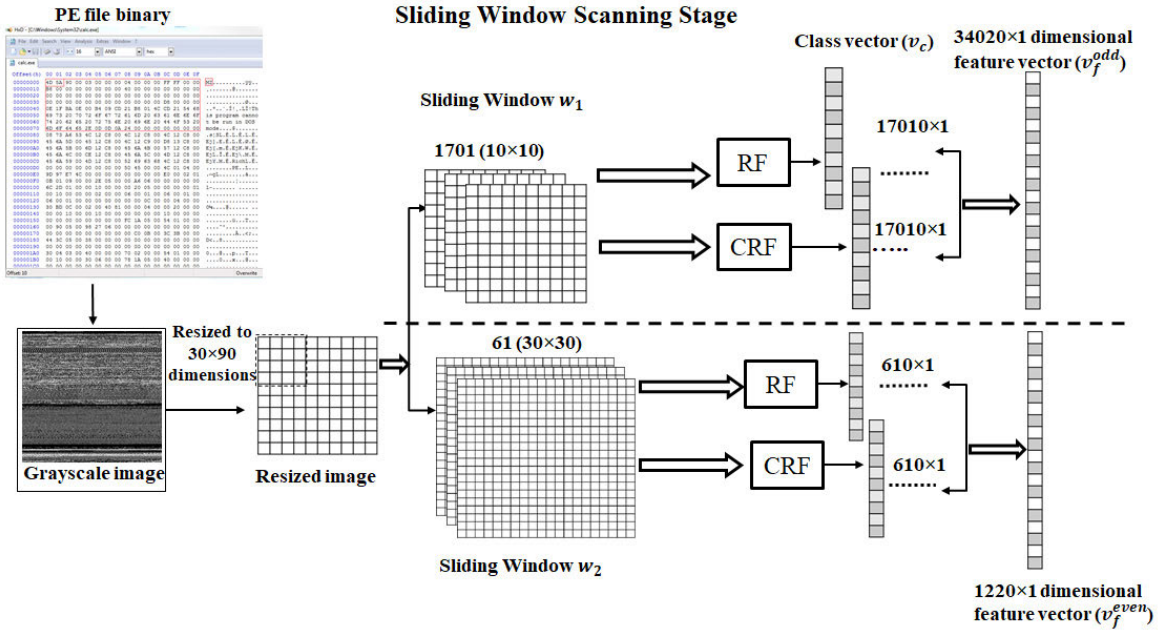

FIGURE 3. Sliding window scanning stage.

weightage, since on an average they provide performance superior to naïve RF model.

The methodology is explained with Microsoft Malware dataset along with cleanware class, containing ten classes (9 malware classes + 1 cleanware class). For w_1, the RF classifier outputs 10×1701 class vectors, obtained from processing 1701 image subsets with respect to ten classes. Likewise, the CRF classifier generates 10×1701 class vectors. The resulting feature vectors from both the ensembles are concatenated to form 34020-dimensional feature vectors (v_f^{even}). Similarly for w_2, the RF and CRF classifiers generate 10×61 feature vectors each, forming 1220-dimensional feature vectors (v_f^{odd}). Fig. 3 shows the flow of sliding window scanning stage. The procedure of sliding window scanning stage is given in Algorithm 2.

Algorithm 2 Sliding Window ScanningStage

Input: g: resized grayscale image.
Output: v_f^{odd} and v_f^{even}: feature vectors.

1: $g : m \times n$; $W \in \{w_1, w_2 : i \times j$; s: stride; $I \in \{I_1, \ldots, I_S$: set of all smaller images; E $\in \{e_1, e_2$: set of ensembles.
2: Generate smaller images $S = v_1 \times v_2$
Where $v_1 \leftarrow m - i + s$; $v_2 \leftarrow n - j + s$.
3: **repeat**
4: **for** each $e_1, e_2 \in E$ **do**
5: Train I
6: Obtain class probability vectors v_C
Where $v_C \in \{v_{C1}, v_{C2}$: set of class probability vectors; $v_{c1} = (v_{I_1,1}, \ldots v_{I_S,C})$ and $v_{c2} = (v_{I_1,1}, \ldots v_{I_S,C})$
7: **end for**
8: **until** no element in E
9: Obtain feature vectors $v_f^{odd} \leftarrow v_{C1} \& v_{C2}$ and $v_f^{even} \leftarrow v_{C1} \& v_{C2}$
Where v_{f1}: $2S_1 \times 1$ dimensions, for $w_1 \in W$;
v_{f2}: $2S_2 \times 1$ dimensions, for $w_2 \in W$;
S_1 and S_2 – number of smaller images from w_1 and w_2 respectively.

D. CASCADE LAYERING STAGE

The input to this stage is the feature vectors v_f^{even} and v_f^{odd} obtained by processing smaller images of sliding windows w_1 and w_2. The first layer processes the feature vector v_f^{even} from first sliding window of the first stage. Each ensemble generates class vectors and is concatenated with the feature vector v_f^{odd} obtained from the second sliding window. This resulting feature vector from the first layer is given as input to the four ensemble forests in the second layer. The class vectors generated from each ensemble are concatenated with the feature vector obtained from the first sliding window. This process repeats in an alternative manner. (i.e) Odd layers take feature vector obtained from the first sliding window, and even layers take feature vector obtained from the second sliding window. The feature vectors from the two sliding windows are alternatively concatenated with the class vectors generated from the four ensembles of each layer.

The process is explained with the first setting, where all the four ensembles are CRFs. The first layer takes the feature vector v_f^{even} and processes it using the four ensembles, resulting in 4×10 dimensional class vectors. Then, the 40-dimensional class vectors are concatenated with v_f^{even} to serve as input to

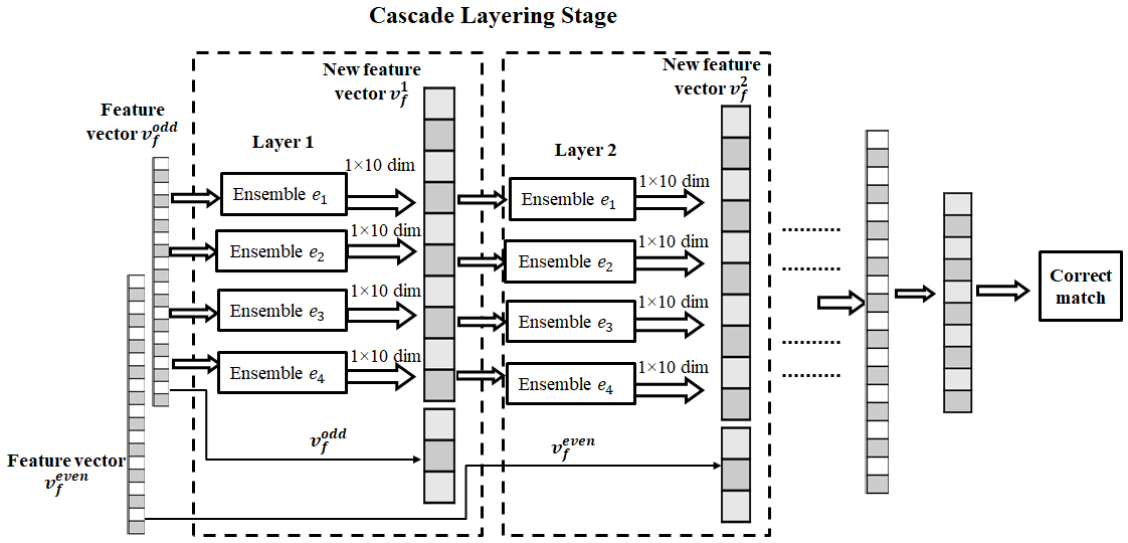

FIGURE 4. Cascade layering stage. e_2,e_2, e_3, e_4 denote ensembles.

the next layer. This layering process repeats by taking the output from the previous layer, sending the new feature vector to be processed by the ensembles, and concatenating with the feature vector v_f^{odd}. Fig. 4 shows the flow of the cascade layering stage.

Likewise, alternatively feature vectors v_f^{even} and v_f^{odd} are concatenated with the class vectors of the current layer to form new feature vectors. At each layer, 5-fold cross-validation is carried out to check the accuracy achieved and decide whether this process should proceed further or not. If there is an increase in accuracy, the next layer is processed. If there is no improvement in accuracy, the layering stops. At the optimum layer, the class probability vectors from the four forests of the final layer are averaged to obtain the final prediction probability vector with 10 × 1 dimensions. The class, which shows the highest probability, is the correct matching class for the given input sample. The steps in the cascade layering phase are given in Algorithm 3.

Algorithm 3 Cascade Layering Phase

Input: v_f^{odd} and v_f^{even}: feature vectors from $w_1, w_2 \in W$
Output: v_f^n: new feature vector; acc_n: accuracy of current layer

$E \in \{e_1, e_2, e_3, e_4\}$: set of ensembles; n: layer variable; $acc_{n-1} = 0$.
for $n = 1$ until $acc_{n-1} < acc_n$
  $acc_{n-1} = acc_n$
  if $n = 1$
    call Layering $(v_f^{odd}, 0)$
    $n = n + 1$
  end if
  if $n = even$
    call Layering (v_c^n, v_f^{even})
    else
      call Layering (v_c^n, v_f^{odd})
  end if
$n = n + 1$
end for
function Layering $(v_1, v_2 \rightarrow v_f^n, acc_n)$:
  for each layer $L_n(n = 1)$ **do**
    Train v_f^{n-1} with all elements of E
    for each $e_1, e_2, e_3, e_4 \in E$ and C classes **do**
      Generate class probability vectors $E \rightarrow v_c^n$
    Where $v_c \leftarrow v_c^{e_1} \& v_c^{e_2} \& v_c^{e_3} \& v_c^{e_4}$: $e \times C$-dimensional class vectors; $v_c^{e_1}, v_c^{e_2}, v_c^{e_3}, v_c^{e_4}$-class vectors obtained from e_1, e_2, e_3, e_4 respectively; e – no. of ensembles ($e = 4$).
    end for
  end for
  Obtain new feature vector $v_f^n \leftarrow v_1 \& v_2$ and acc_n
end function

IV. MATHEMATICAL PROOF OF THE PROPOSED MALWARE DETECTION SYSTEM

Given a set of N training PE file samples $P = \{(a_f, b_f)\}_{f=1}^{N}$, in which $a_f \in v_f$ is a feature vector and $b_f \in \{1, \ldots, C\}$ refers to the class of the associated instances. For each PE file represented as a binary image with size m × n, sliding window size i × j, and stride s, the smaller images are generated using the formula,

$$(m \times n) * (i \times j) = (m - i + s) \times (n - j + s) \quad (1)$$

Likewise, the smaller images for an input binary image are obtained for W sliding windows simultaneously. For each sliding window $W \in \{w_1, w_2\}$, the smaller images are generated using Eqn. 1. The smaller images are fed to e ensembles where $E \in \{e_1, e_2\}$. Each forest gives an estimate of the class probability distribution $d = (d_1, \ldots, d_C)$ based on the percentage of distinct classes of training samples at the leaf node where the appropriate sample belongs. These probabilities are computed as the ratio of

the number of smaller images of class c, which reaches l to the total number of smaller images in set I reaching l. For each tree t ∈ T, class c ∈ C, and at each leaf node l ∈ F, the posterior probabilities are given by,

$$(\mathrm{P_{t,l}}(\mathrm{Y}(\mathrm{I}) = \mathrm{c})) \tag{2}$$

where T is the tree set, C is the class set, F is the leaf set of any tree t, Y(I) is the class label c for smaller image I.

Each image in I is classified as belonging to class C, which is of highest probability. By taking an average of the probabilities of Eqn. 2, the smaller images are classified using,

$$\mathrm{Y}(\mathrm{I}) = \arg\max \frac{1}{\mathrm{T}} \sum\nolimits_{\mathrm{t}=1}^{\mathrm{T}} (\mathrm{P_{t,l}}(\mathrm{Y}(\mathrm{I}) = \mathrm{c})) \tag{3}$$

The class probabilities $\mathrm{v_c}=(\mathrm{v}_{I_S,1}, .., \mathrm{v}_{I_S,\mathrm{C}})$ for every forest classifier is calculated by finding the average of all class probability distributions over all trees. Let all ensembles have the same number of decision trees, each layer in the layering phase consists of e ensembles, and the number of layers is denoted by L. The current layer produces e class vectors $\mathrm{v}_{I_S,1}, \ldots \mathrm{v}_{I_S,\mathrm{e}}$ and then concatenated alternatively with the original vector v_f^{odd} or v_f^{even} obtained from $\mathrm{w}_1, \mathrm{w}_2 \in W$ to serve as input to the next subsequent layer. Thus, the training dataset for the next layer is given by,

$$\mathrm{Tr}^* = \{ \left(\left(\mathrm{a_I}, \mathrm{v_{I,1}}, \ldots, \mathrm{v_{I,e}} \right), \mathrm{b_I} \right), \mathrm{I} = 1, \ldots, \mathrm{S} \} \tag{4}$$

At each level l, the layer building process is decided automatically based on accuracy obtained from cross-validation for all smaller images. If the accuracy of one layer l is higher than the previous layer $l-1$, the final prediction is determined at the current level. Otherwise, the process continues through the next level. The prediction of any sample $\left(\left(\mathrm{a_f}, \mathrm{v_{f,1}}, \ldots, \mathrm{v_{f,e}} \right), \mathrm{b_f} \right)$ is calculated by finding the average vector of the class probabilities,

$$\mathrm{v_f} = \frac{1}{\mathrm{E}} \sum\nolimits_{\mathrm{e}=1}^{\mathrm{E}} \mathrm{v_{f,e}} \tag{5}$$

The final prediction is given by,

$$\mathrm{X} = \begin{cases} 1, & \max\left(\mathrm{v_{f,1}} \ldots, \mathrm{v_{f,C}} \right) \\ 0, & \text{for other classes.} \end{cases} \tag{6}$$

For a PE sample, if X = 1, the final prediction is reached at the current layer, which is given by,

$$\mathrm{y_f} = \arg_{\mathrm{p}=1,\ldots,\mathrm{C}} \max \mathrm{v_{f,p}} \tag{7}$$

A. GENERALIZATION ERROR OF ENSEMBLE ESTIMATOR

Consider the smaller image set $\mathrm{t^S} = \{\mathrm{t_1}, \ldots, \mathrm{t_S}\}$ where $\mathrm{t_m} = (\mathrm{a_m}, \mathrm{b_m})$. The basic relationship among $(\mathrm{a_m}, \mathrm{b_m})$ is given by,

$$\mathrm{b_m} = \mathrm{h}(\mathrm{a_m}) + \varepsilon \tag{8}$$

where h(a) is the unknown target function and ε is the additive noise with mean $\mathrm{E}(\varepsilon) = 0$ and variance $\mathrm{V}(\varepsilon) = \sigma^2 (< \infty)$. $\mathrm{t^S}$ is a realization of a random sequence $\mathrm{T^S} = \{\mathrm{T_1}, \ldots, \mathrm{T_S}\}$ whose mth component consists of a random vector $\mathrm{T_m} = (\mathrm{A_m}, \mathrm{B_m})$. Given $\mathrm{t^S}$, the parameter estimation is given by,

$$\hat{\theta}\left(\mathrm{t^S}\right) = argmin_\theta \sum\nolimits_{m=1}^{S} (\mathrm{b_m} - p(\mathrm{a_m}; \theta))^2 / S \tag{9}$$

The outcome of the estimator p for an input a is given as $p(a; \mathrm{t^S})$. Consider a new random vector $\mathrm{T_0} = (\mathrm{A_0}, \mathrm{B_0}) \in \mathrm{R^{g+1}}$, with a distribution similar and independent (for all m) of $\mathrm{T_m}$. Then, the mean squared error averaged over all possible realizations of $\mathrm{T^S}$ and $\mathrm{T_0}$ is the generalization error (denoted by $\mathrm{G_{err}}$) of the estimator p. It is given by,

$$\mathrm{G_{err}}(\mathrm{p}) = \{\mathrm{E_{T^S}} \mathrm{E_{T_0}} \{[\mathrm{B_0} - \mathrm{p}(\mathrm{A_0}; \mathrm{T^S})]^2\}\} \tag{10}$$

where $\mathrm{E_{T_0}}\{\}$ and $\mathrm{E_{T^S}}\{\}$ denote expectation in terms of $\mathrm{T_0}$ and $\mathrm{T^S}$ respectively. From Eqn. (2), it is identified that $\mathrm{G_{err}}(\mathrm{p})$ does not depend on the smaller image set $\mathrm{t^S}$ or an unknown malware sample $\mathrm{t_0}$, but it depends on sample size S and the estimator model used.

Eqn. (10) is expressed using the bias-variance decomposition that is given by,

$$\mathrm{G_{err}}(\mathrm{p}) = \mathrm{E_{A_0}} \left\{ \mathrm{V}\{\mathrm{p} \mid \mathrm{A_0}\} + \mathrm{Bias}\{\mathrm{p} \mid \mathrm{A_0}\}^2 \right\} + \sigma^2 \tag{11}$$

where $\mathrm{V}\{\mathrm{p} \mid \mathrm{A_0} = \mathrm{a_0}\}$ and $\mathrm{Bias}\{\mathrm{p} \mid \mathrm{A_0} = \mathrm{a_0}\}$ are the conditional variance and conditional bias (given $\mathrm{A_0} = \mathrm{a_0}$).

$$\mathrm{V}\{\mathrm{p} \mid \mathrm{A_0}\} = \mathrm{E_{T^S}}(\mathrm{p}\left(\mathrm{A_0}; \mathrm{T^S}\right) - \mathrm{E_{T^S}}\{(p\left(\mathrm{A_0}; \mathrm{T^S}\right))\}^2 \tag{12}$$

$$\mathrm{Bias}\{\mathrm{p} \mid \mathrm{A_0}\} = \mathrm{E_{T^S}} - \mathrm{h}(\mathrm{A_0}) \tag{13}$$

Let $\mathrm{p_1}, \ldots, \mathrm{p_E}$ denote E estimators, where the eth estimator is individually trained on $\mathrm{t^S_{(e)}}$, $\mathrm{e} = 1, 2, \ldots, \mathrm{E}$. Let S be the sample size of each smaller image set $\mathrm{t^S_{(e)}}\}$, which is a realization of a random sequence $\mathrm{T^S_{(e)}} = \{\mathrm{T_1}, \ldots, \mathrm{T_S}$ which has the same distribution and are not always mutually independent. For an input file a, the ensemble estimator output is defined as the average of outcomes of individually trained E estimators for a. It is given by,

$$\mathrm{f^{(E)}_{en}}(\mathrm{a}) = \frac{1}{\mathrm{E}} \sum\nolimits_{\mathrm{e}=1}^{\mathrm{E}} \mathrm{f_e}(\mathrm{a}; \mathrm{t^S_{(e)}}) \tag{14}$$

$$\mathrm{G_{err}}\left(\mathrm{f^{(E)}_{en}}\right) = \mathrm{E_{a_0}}\{\frac{1}{\mathrm{E}} \overline{V}(\mathrm{A_0}) + \left(1 - \frac{1}{\mathrm{E}}\right) \overline{\mathrm{Cov}}(\mathrm{A_0}) + \overline{\mathrm{Bias}} + \sigma^2\} \tag{15}$$

where $\mathrm{G_{err}}(\mathrm{f^{(E)}_{en}})$ indicate the generalization error of the ensemble estimator given in Eqn. (14), $\overline{\mathrm{V}}(\mathrm{A_0})$ denote average conditional variance, $\overline{\mathrm{Cov}}(\mathrm{A_0})$ denote average conditional covariance, and $\overline{\mathrm{Bias}}(\mathrm{A_0})$ denote average conditional bias that is averaged over E estimators.

$$\overline{\mathrm{V}}(\mathrm{A_0}) = \frac{1}{\mathrm{E}} \sum\nolimits_{\mathrm{e}=1}^{\mathrm{E}} \mathrm{V}\{\mathrm{p_e} | \mathrm{A_0}\} \tag{16}$$

$$\overline{\mathrm{Cov}}(\mathrm{A_0}) = \frac{1}{\mathrm{E}(\mathrm{E}-1)} \sum\nolimits_{\mathrm{e}} \sum\nolimits_{\mathrm{e}' \neq \mathrm{e}} \mathrm{Cov}\{\mathrm{p_e}, \mathrm{p_{e'}} | \mathrm{A_0}\} \tag{17}$$

$$\overline{\mathrm{Bias}}(\mathrm{A_0}) = \frac{1}{\mathrm{E}} \sum\nolimits_{\mathrm{e}=1}^{\mathrm{E}} \mathrm{Bias}\{\mathrm{p_e} | \mathrm{A_0}\} \tag{18}$$

where $\mathrm{p_e}$ denotes $\mathrm{p_e}(\mathrm{A_0}; \mathrm{T^S_{(e)}})$.

The average generalization error averaged over E estimators is given by,

$$\overline{G_{err}} = \frac{1}{E}\sum_{e=1}^{E}\left(E_{a_0}\{V\{p_e|A_0 + Bias\}\{p_e A_0\}^2 + \sigma^2\}\right) \quad (19)$$

$G_{err}\left(f_{en}^{(E)}\right)$ and $\overline{G_{err}}$ can be related as given below.

$$\begin{aligned} G_{err}\left(f_{en}^{(E)}\right) &= \frac{1}{E}\overline{G_{err}} + \left(1-\frac{1}{E}\right)\sigma^2 + E_{a_0}\{\left(1-\frac{1}{E}\right)\overline{Cov}\,(A_0) \\ &\quad + \frac{1}{E^2}\sum_{e}\sum_{e'\neq e} Bias\{f_e|A_0\}Bias\{f_{e'}|A_0\}\} \end{aligned} \quad (20)$$

Eqn. 11 is rewritten by replacing p using $f_{en}^{(E)}$ as,

$$G_{err}\left(f_{en}^{(E)}\right) = E_{a_0}\left\{V\left\{f_{en}^{(E)} \mid A_0\right\} + Bias\left\{f_{en}^{(E)} \mid A_0\right\}^2\right\} + \sigma^2 \quad (21)$$

The term $f_{en}^{(E)}$ depends on $t_{(1)}^{S}, \ldots, t_{(E)}^{S}$. The conditional variance of the ensemble forest is given by,

$$\begin{aligned} V\left\{f_{en}^{(E)} \mid A_0\right\} &= E_{T_{(1)}^S,\ldots,T_{(E)}^S}\{[\frac{1}{E}\sum_{e=1}^{E} f_e \\ &\quad - E_{T_{(1)}^S,\ldots,T_{(E)}^S}\{\frac{1}{E}\sum_{e=1}^{E} p_e\}]^2\} \\ &= \frac{1}{E^2}\sum_{e=1}^{E} E_{T_{(e)}^S}\left\{\left[p_e - E_{T_{(e)}^S}\{p_e\}\right]^2\right\} \\ &\quad + \frac{1}{E^2}\sum_{e}\sum_{e'\neq e} E_{T_{(e)}^S, T_{(e')}^S} \\ &\quad \{[p_e - E_{T_{(e)}^S}\{p_e\}][p_{e'} - E_{T_{(e')}^S}\{p_{e'}\}]\} \end{aligned} \quad (22)$$

$$= \frac{1}{E}\overline{V}\,(A_0) + (1-\frac{1}{M})\overline{Cov}(A_0)$$

$$\begin{aligned} Bias\left\{f_{en}^{(E)} \mid A_0\right\} &= E_{T_{(1)}^S,\ldots..T_{(E)}^S}\left\{\frac{1}{E}\sum_{e=1}^{E} p_e\right\} - h \\ &= \frac{1}{E}\sum_{e=1}^{E} E_{T_{(e)}^S}\{p_e - h\} \\ &= \overline{Bias}(A_0) \end{aligned} \quad (23)$$

B. DIVERSITY MEASURE

Given a set of E trained classifiers $Z = \{e_m(a)\}_{m=1}^{E}$, where each classifier e_m: $A \rightarrow \{-1, +1\}$ is a mapping from the feature space A to the class label set $\{-1, +1\}$, the decision function is defined as,

$$f_d\,(a; Z) = \frac{1}{E}\sum_{m=1}^{E} e_m(a) \quad (24)$$

The diversity of classifier set $Z = \{e_m(a)\}_{m=1}^{E}$ is given by,

$$div\,(Z) = 1 - \frac{1}{\sum_{1\leq m\neq o\leq E} 1}\sum_{1\leq m\neq o\leq E} dif(e_m, e_o) \quad (25)$$

The diversity is calculated using the average of pairwise differences between two classifiers. The larger the value of div(Z), the higher is the diversity of the classifier set Z.

$$dif\,(e_m, e_o) = \frac{1}{S}\sum_{q=1}^{S} e_m(a_q)e_o(a_q) \quad (26)$$

where dif(., .) denotes the pairwise differences.

Consider the set of classifiers $Z = \{e_m(a)\}_{m=1}^{E}$ and a set of smaller images $I = \{(a_m, b_m)\}_{m=1}^{S}$. Let the output of the decision function be $f_d = [f\,(a_1; Z), \ldots, f(a_S; Z)]^T$. On dataset I, if $div(Z) \geq k$, then

$$||f_d||_1 \leq S\sqrt{1/E + (1-1/E)(1-k)} \quad (27)$$

According to basic algebra,

$$\begin{aligned} ||f_d||_2^2 &= \sum_{m=1}^{S}\left(\frac{1}{E}\sum_{d=1}^{E} e_d(a_m)\right)^2 \\ &= \sum_{m=1}^{S}\left(\frac{1}{E} + \frac{1}{E^2}\sum_{1\leq o\neq x\leq E} e_o(a_m)e_x(a_m)\right) \\ &= S(1/E + (1 - div(Z))(1 - 1/E)) \geq 0 \end{aligned} \quad (28)$$

Since the term, $1/E + (1 - 1/E)(1 - k)$ is always positive, based on the inequality $||f_d||_1 \leq \sqrt{S}\,||f_d||_2$, the eqn. (27) is obtained.

C. RELATIONSHIP BETWEEN DIVERSITY AND GENERALIZATION

Let H denote the function space such that for every $f \in H$, there exist a set of E classifiers $Z = \{e_m(a)\}_{m=1}^{E}$ which satisfies $f\,(a) = \frac{1}{E}\sum_{m=1}^{E} e_m(a)$ and $div\,(Z) \geq k$ for any independent and identically distributed sample I of size S, then for any ϵ, it holds

$$\begin{aligned} \log_2 P_\infty(H, \varepsilon, S) \leq \frac{36(1+\ln E)}{\varepsilon^2}\log_2(2S \\ \left\lceil 4\sqrt{1/E + (1-1/E)\,(1-k)}/\varepsilon + 2\right\rceil + 1) \end{aligned} \quad (29)$$

Initially, assume that $\varepsilon \leq 1$, since the result is trivial if $\varepsilon \geq 1$. The interval $[-1-\varepsilon/2, 1+\varepsilon/2]$ is split into $p = \lceil 4/\varepsilon + 2\rceil$ sub-intervals, each of size lesser than $\varepsilon/2$, and θ_o be the boundaries of the sub-intervals so that $\theta_o - \theta_{o-1} \leq \varepsilon/2$ for all o. Let $o_{le}(m)$ denote the maximum index of θ_o such that $f\,(a_m) - \theta_{o_{le}(m)} \geq \varepsilon/2$ and $o_{ri}(m)$ denote the maximum index of θ_o such that $f\,(a_m) - \theta_{o_{ri}(m)} \leq -\varepsilon/2$.

Let $e_m = [e_1\,(a_m), \ldots, e_T\,(a_m)]^T$, $e_m' = [e_m, -\theta_{o_{le}(m)}]^T$ and $e_m'' = [-e_m, \theta_{o_{ri}(m)}]^T$. Then, the covering number $P_\infty(H, \varepsilon, I)$ is no more than the number of possible values of the vector α,
where

$$\alpha = g_w(\sum_{m=1}^{S} h_m e_m' + \sum_{m=1}^{S} q_m e_m'') \quad (30)$$

and $g_w(u)$ is a component-wise function which maps each component u_m of u to w. $sign(u_m)|u_m|^{w-1}$ with $w \geq 2$, and h_m and q_m are positive integers satisfying the condition,

$$\sum_{m=1}^{S}(h_m + q_m) \leq 36(1+\ln E)/\varepsilon^2 \quad (31)$$

There is a one-to-one mapping between e_m' and e_m'', thus the number of possible values of e_m' and e_m'' equals to the

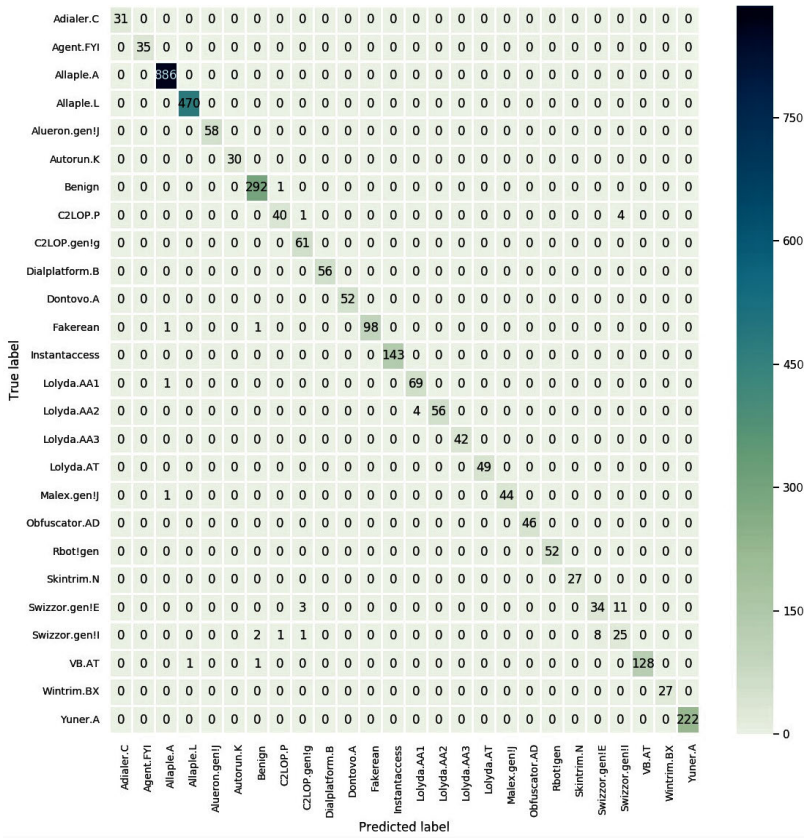

FIGURE 5. Confusion matrix for Malimg dataset.

number of possible values of $e_m{}'$. Let $f = [f(a_1), \ldots f(a_S)]^T$ be f's outputs on I. Based on the definition of $\theta_{O_{le}(S)}$, the number of possible values of $e_m{}'$ is no more than $S\left\lceil 4\sqrt{1/E + (1-1/E)(1-k)}/\varepsilon + 2\right\rceil$. Consequently, from Eqn. (30) and Eqn. (31), the number of possible values of (α, ρ) is upper-bounded by

$$\left(2S\left\lceil 4\sqrt{1/E + (1-1/E)(1\text{-}k)}/\varepsilon + 2\right\rceil + 1\right)^{36(1+lnE)/\varepsilon^2} \tag{32}$$

Hence the Eqn. (32) proves Eqn. (29).

From Eqn. (29), the relationship between diversity and generalization voting performance is obtained with probability at least $1-\beta$, for any $\theta > 0$, every function $f \in H$ satisfies the following bound

Bartlett's Lemma [5]

$$G_{err}(f) \leq I^{\theta}_{err}(f) + \sqrt{\frac{2}{S}\left(lnP_{\infty}\left(H, \frac{\varepsilon}{2}, 2S\right) + ln\frac{2}{\beta}\right)} \tag{33}$$

Applying Eqn. (29) in Eqn. (33), the result is obtained.

V. EXPERIMENTS AND RESULTS

A. DATASETS AND EXPERIMENT SETUP

The four benchmark malware datasets used to analyze the performance of the proposed deep forest-based malware detection system are the Malimg dataset [31] and the BIG 2015 dataset [2], MaleVis dataset [6] and Malware dataset [38]. The cleanware samples are self-collected containing 1044 files, and they were checked using the VirusTotal [41] portal for confirming their legitimacy. The Malimg dataset contains 9,339 malware images belonging to 25 malware classes, with each class comprising of a varying number of samples. The malware classes include Adialer.C, Agent.FYI, Allaple.A, Allaple.L, Alueron.gen!J, Autorun.K, Benign, C2LOP.P, C2LOP.gen!g, Dialplatform.B, Dontovo.A, Fakerean, Instantaccess, Lolyda.AA1, Lolyda.AA2, Lolyda.AA3, Lolyda.AT, Malex.gen!J, Obfuscator.AD, Rbot!gen, Skintrim.N, Swizzor.gen!E, VB.AT, Wintrim.BX, and Yuner.A.

Kaggle's Microsoft BIG 2015 dataset consists of training and testing malware samples belonging to nine different classes, classified based on their varying features. There are 21741 samples contained in this dataset with a size of nearly half-terabyte uncompressed, of which 10868 are training samples (200 GB), and the remaining 10873 are testing samples (200 GB). The malware classes include Ramnit, Lollipop, Kelihos_ver3, Vundo, Simda, Tracur, Kelihos_ver1, Obfuscator.ACY and Gatak. Each sample consists of a unique class ID and a class to which it belongs to. The class ID is a 20 character hash value that identifies the malware sample and a class labeled with an integer value denoting

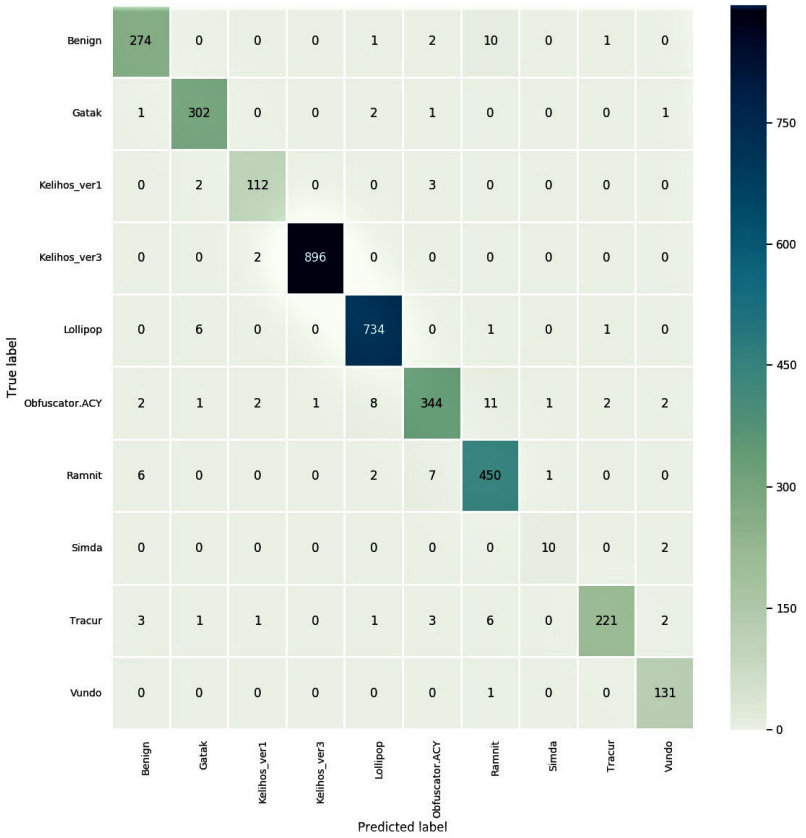

FIGURE 6. Confusion matrix for BIG2015 dataset.

any of the nine malware classes. The distribution of the number of samples in all classes varies and is non-uniform. Each binary sample consists of two files, such as bytes file and disassembled asm file. The byte's file provides the hexadecimal representation of a binary file, and asm file contains metadata obtained from the binary content like function calls, instruction sequences, strings, registers, etc.

MaleVis (Malware Evaluation with Vision) dataset is an open image dataset consisting of 9100 RGB images for training and 5126 RGB images for testing belonging to 25 malware classes and a cleanware class. Malware classes include Adposhel, Agent-fyi, Allaple.A, Amonetize, Androm, AutoRun-PU, BrowseFox, Dinwod!rfn, Elex, Expiro-H, Fasong, HackKMS.A, Hlux!IK, Injector, InstallCore.C, MultiPlug, Neoreklami, Neshta, Regrun.A, Sality, Snarasite.D!tr, Stantinko, VBA/Hilium.A, VBKrypt, and Vilsel. Each class contains 350 samples for training, and each class contains varying samples for testing.

The detection ability of the proposed algorithm has to be tested on a totally new unseen malware family to ensure efficient performance. Hence, we have chosen Malicia dataset [46] to test the efficiency of the proposed malware detector. It contains 9670 malware samples belonging to 8 classes such as cleaman, winwebsec, zbot, zeroaccess, cridex, harebot, smarthdd, securityshield. This dataset is not trained with the deep forest model. The samples from this dataset are not exposed to the model during training and validation.

The experiments were performed on Ubuntu 18.04 64-bit system with 32 GB RAM and 1 TB Hard Drive. The implementation was done with Python programming language, with the necessary packages included. The experiments are conducted with four settings, where each setting is based on different types of ensemble forest combinations. Each level in CL phase consists of 4 Completely Random Forests (CRF) in the first setting, 4 Random Forests (RF) in the second setting, 4 XGBoost (XGB Forest) in the third setting, whereas the fourth setting uses a combination of all the forests (Mixed forest). The generation of class vectors is done using three-fold cross-validation. The number of cascade levels is determined automatically based on the data. The levels are generated until no performance improvement is observed. The experiments are trained on 80% and validated on 20% of the training set. The proposed model involves few hype-parameters and the default settings produce superior performance. The optimal choice of hyper-parameters used in the sliding window scanning stage are: the number of forests is 2, number of trees in each forest is 10 and number of sliding windows used is 2. In cascade layering stage, the number of forests is 4 for each layer and the number of trees of each forest is 10.

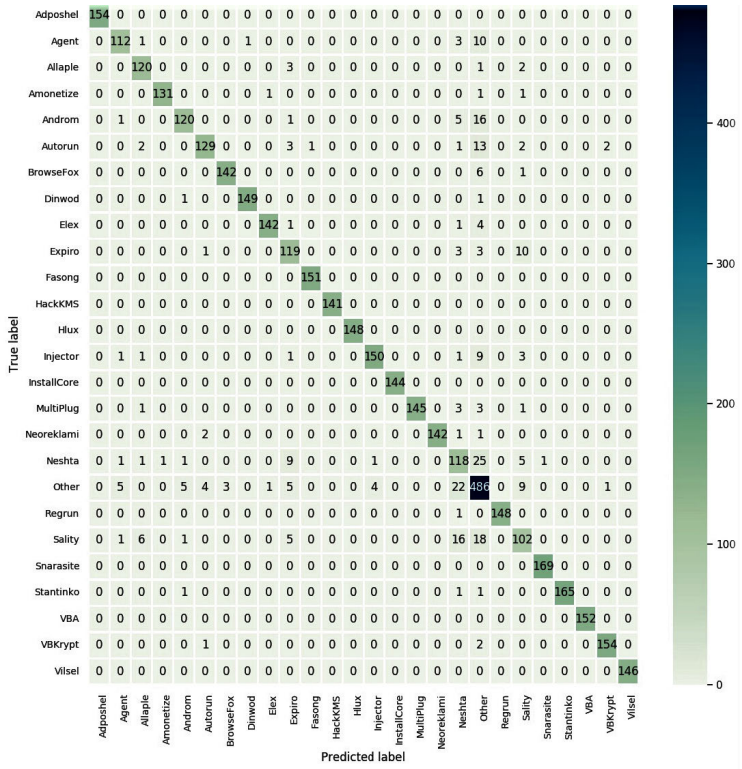

FIGURE 7. Confusion matrix for MaleVis dataset.

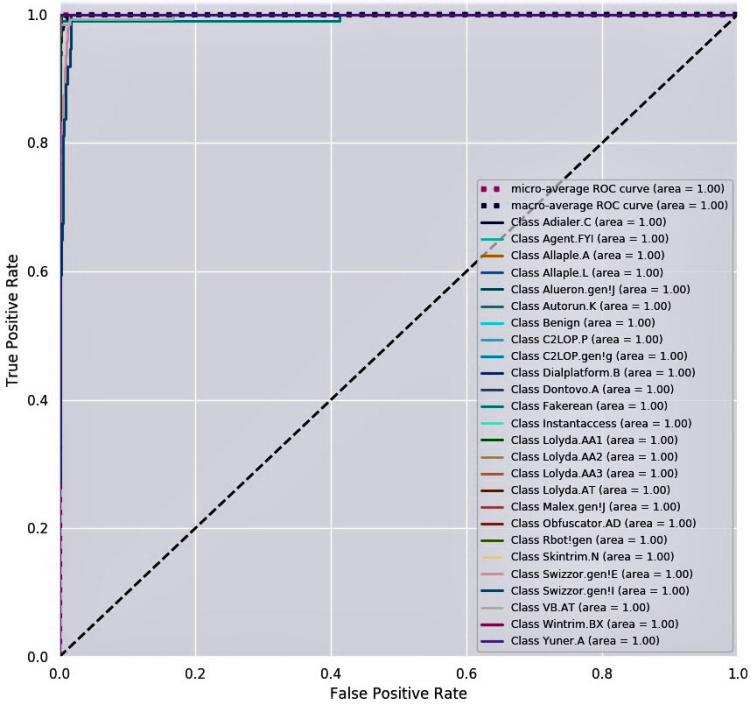

FIGURE 8. ROC Curve for Malimg dataset.

B. RESULTS AND DISCUSSION

The test results for all four models are summarized in Table 1, Table 2, Table 3, and Table 4. With 2 sliding windows and 10 trees in a forest, the proposed malware detector showed optimal results. The proposed deep forest based malware detector achieves highly competitive performance

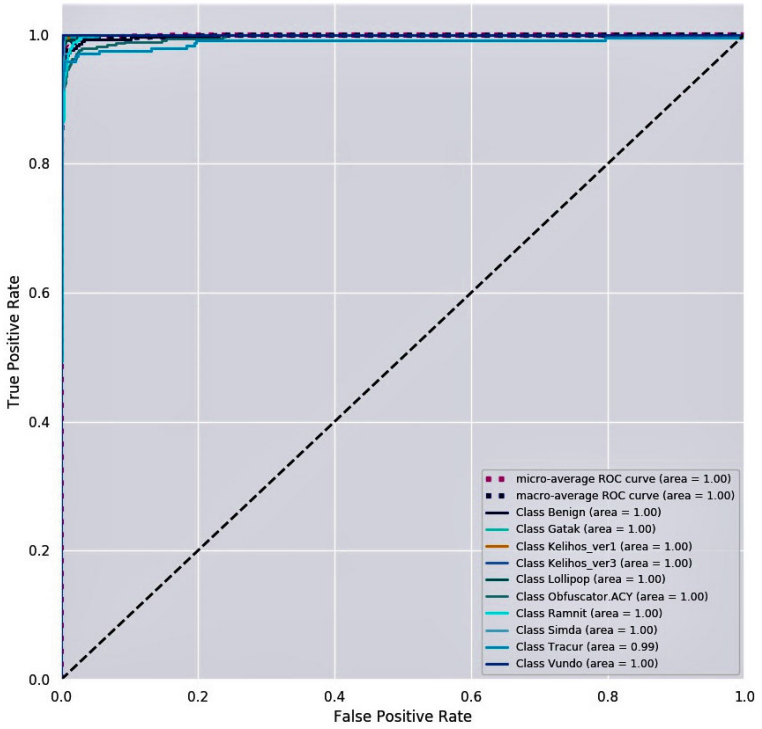

FIGURE 9. ROC Curve for BIG2015 dataset.

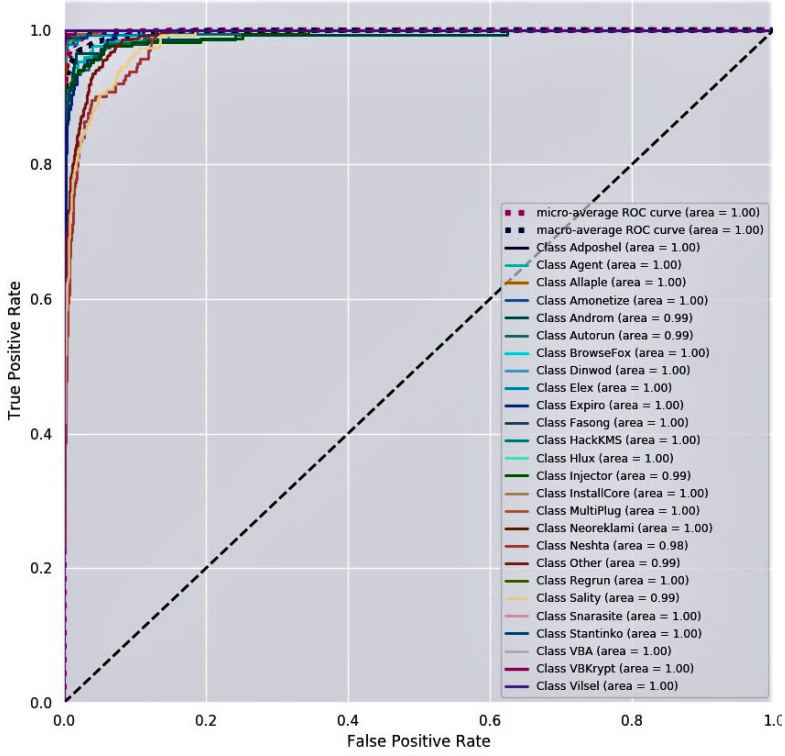

FIGURE 10. ROC Curve for MaleVis dataset.

with default settings. It is inferred that the fourth setting, which is a mixed forest model, shows the highest accuracy of 98.65%, 97.20%, and 97.43% on each of the three datasets, respectively. Table 2 and Table 4 shows results on three malware datasets and also unseen dataset.

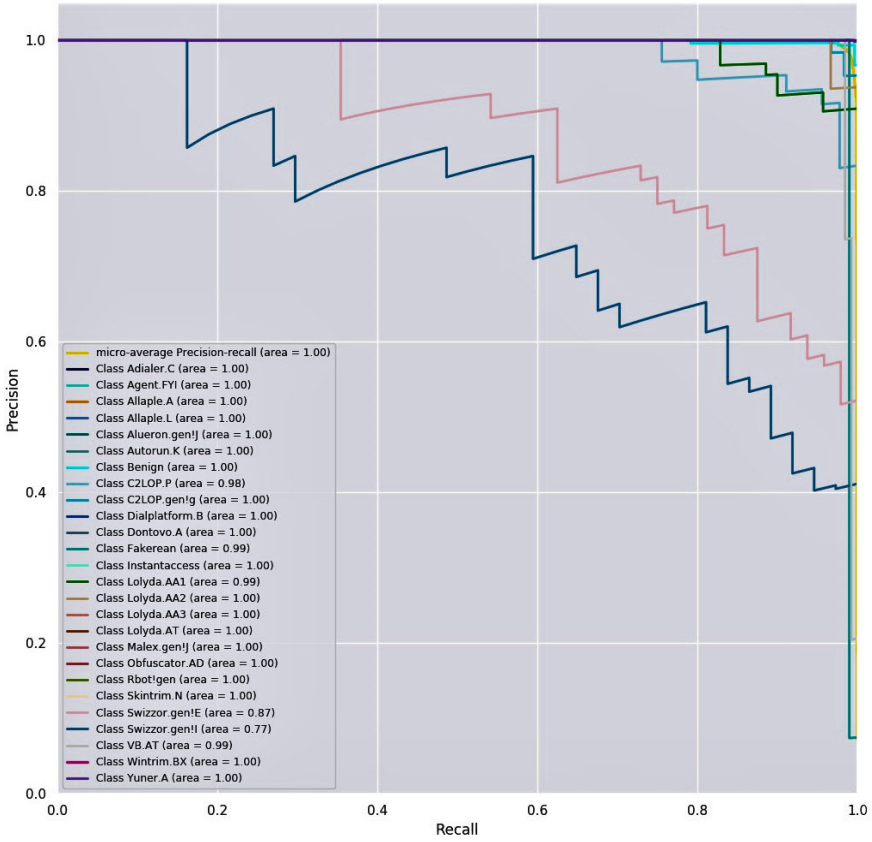

FIGURE 11. Precision recall curve for Malimg dataset.

TABLE 1. Performance comparison of the proposed model with other methods in malware detection for the three datasets.

Methods	DL / Non - DL	Malimg Dataset				BIG 2015 Dataset				MaleVis Dataset			
		Accuracy (%)	Precision	Recall	F-score	Accuracy (%)	Precision	Recall	F-score	Accuracy (%)	Precision	Recall	F-score
Agarap, 2017	DL and Non-DL	84.92	0.8547	0.8464	0.8505	80.51	0.8135	0.7986	0.8060	79.36	0.8022	0.7845	0.7933
Vinayakumar et al., 2019	DL	96.3	0.963	0.9582	0.9606	91.27	0.9221	0.9132	0.9176	86.29	0.8685	0.8628	0.8656
Cui et al., 2018	DL	94.5	0.9464	0.9431	0.9447	93.4	0.9328	0.9354	0.9341	92.13	0.9209	0.9189	0.9199
Singh, 2019	DL	96.08	0.9576	0.9616	0.9596	94.24	0.9423	0.9289	0.9356	93	0.9287	0.9167	0.9227
Nataraj et al., 2011	Non - DL	97.18	0.9657	0.9685	0.9671	96.48	0.9646	0.9544	0.9595	91.69	0.9236	0.8958	0.9095
Luo et al., 2017	DL	93.72	0.9413	0.9254	0.9333	93.57	0.9447	0.9268	0.9357	92.24	0.9179	0.9096	0.9137
Ma et al., 2019	DL	96.63	0.9732	0.9654	0.9693	96.09	0.962	0.9595	0.9607	91.31	0.9126	0.9087	0.9106
Narayanan et al., 2016	Non - DL	97.34	0.9671	0.9704	0.9687	96.6	0.9708	0.9479	0.9592	91.24	0.9118	0.9074	0.9096
Gibert, 2016	DL	95.33	0.9502	0.9476	0.9489	94.64	0.9555	0.9351	0.9452	90.59	0.9143	0.8979	0.9060
Proposed Model	**Non - DL**	**98.65**	**0.9886**	**0.9863**	**0.9874**	**97.2**	**0.9761**	**0.9679**	**0.9720**	**97.43**	**0.9753**	**0.9732**	**0.9742**

The confusion matrices for the three malware datasets with the mixed forest model are shown in Fig. 5, Fig. 6, and Fig. 7. For the Malimg dataset, the off-principal diagonal values show low values. Though Swizzorgen! E and Swizzorgen! I show higher misclassifications, as they both are similar families with less variation in pattern. For the BIG 2015 dataset, the major number of misclassifications is identified from samples of the Obfuscator.ACY class as belonging to the Ramnit class. For MaleVis dataset, there is more number of misclassifications observed for cleanware class misclassified as other classes and vice versa.

Receiver Operating Characteristic (ROC) curve is an important evaluation metric for examining the classifier performance. The ROC curve is plotted with True Positive

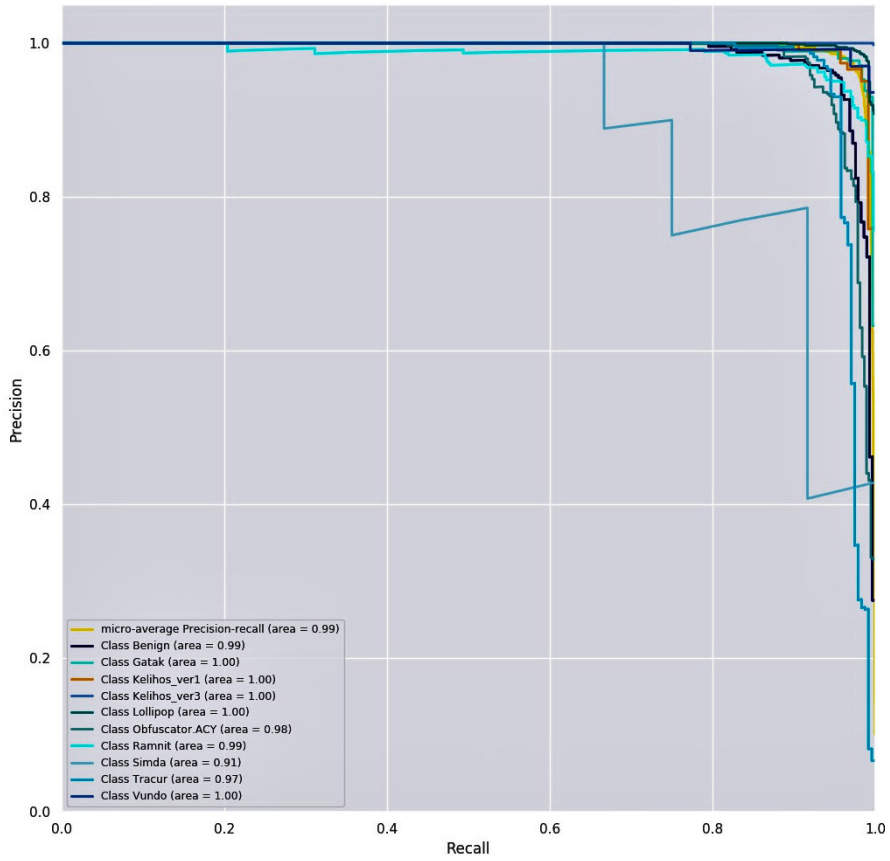

FIGURE 12. Precision recall curve for BIG2015 dataset.

Rate (TPR) against the False Positive Rate (FPR), where TPR is on the y-axis, and FPR is on the x-axis. In Fig. 8, Fig. 9 and Fig. 10, for N number of classes, the N number of ROC Curves is plotted. In the Malimg dataset, there are 26 classes. The plot consists of one ROC for class 1 classified against the other 25 classes, another ROC for class 2 classified against the other remaining classes, and so forth.

The curves for each class to every other class shows TPR as nearly one and FPR as nearly zero. A larger area under the curve is observed in Fig. 8 and Fig. 9. Fig. 10 shows a higher false-positive rate when compared to Fig. 8 and Fig. 9. Thus, the proposed model shows higher performance in classifying malware into various classes.

The Precision-Recall Curve for the three datasets is shown in Fig. 11, Fig. 12, and Fig. 13. The effectiveness of the proposed model is assessed using the precision-recall curve, which shows the trade-off between precision and recall at different thresholds for the three malware datasets. High scores for both metrics show that the classifier is returning accurate results (high precision), as well as returning a majority of all positive results (high recall).

For the Malimg dataset, the precision value is 0.9886, and the recall is 0.9863. For the BIG 2015 dataset, the precision value is 0.9761, and the recall is 0.9679. For the MaleVis dataset, the precision value is 0.9753, and the recall is 0.9732. These results show that the proposed model shows high precision and high recall. This indicates that the proposed results have shown almost correct predictions.

C. COMPARISON RESULTS

Table 1 shows the results for the comparison of the proposed model and other methods for malware detection and classification for the three datasets. Performance metrics, such as accuracy, precision, recall, and F-score, are recorded. The results are taken for the three malware datasets to analyze the efficiency of the proposed method. The malware detection approaches employ Deep Learning (DL) and Non-Deep Learning (Non-DL) algorithms or a combination of both. The proposed model is a Non-DL model showing an accuracy of 98.65% for the malimg dataset, 97.2% for the Microsoft dataset, and 97.43% for the MaleVis dataset. The precision, recall, and f-score measures are also higher compared to the other methods. This comparison shows the effectiveness of the proposed Non-DL model over the other DL and Non-DL methods.

Table 2 compares the performance of the proposed deep forest variant models with machine learning models for the three datasets. The results are taken for studying the effectiveness of the proposed models with various machine learning

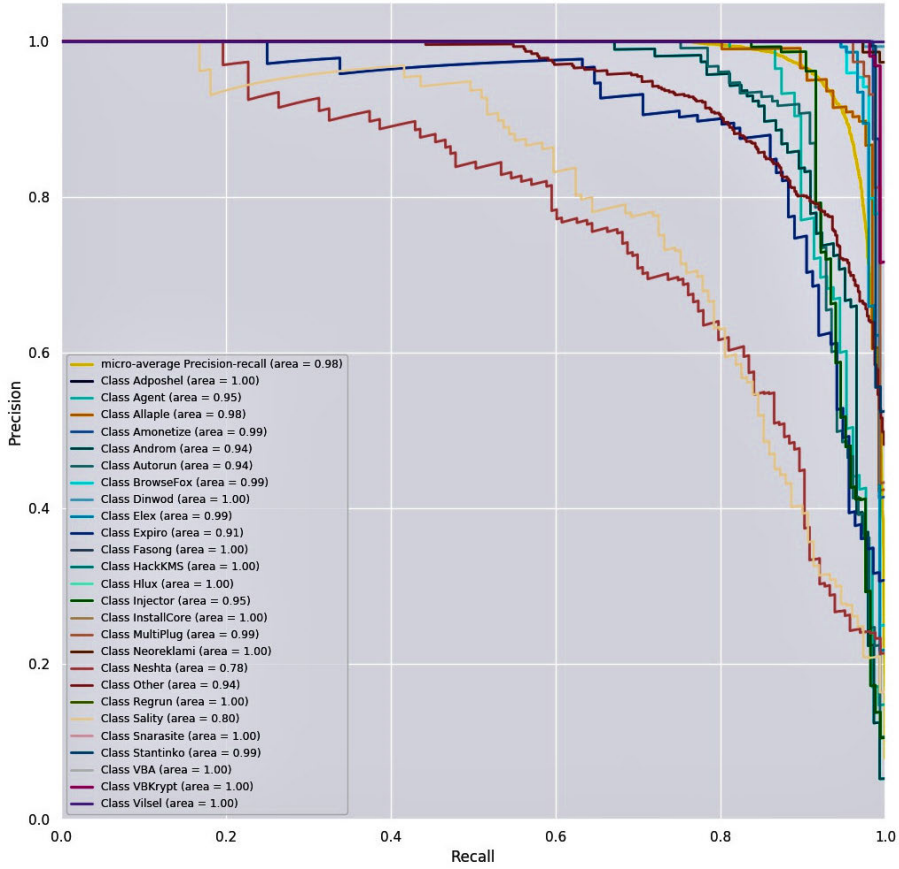

FIGURE 13. Precision recall curve for MaleVis dataset.

models such as K-Nearest Neighbor (KNN), Logistic Regression (LR), Support Vector Machine (SVM), Naïve Bayes (NB), Random Forest (RF), and Adaboost. The proposed models outperform the machine learning models in terms of accuracy, precision, recall, and f-score. The mixed forest model shows the highest performance compared to other proposed variant models and machine learning models. The CRF Forest model shows an accuracy of 98.62% for the malimg dataset, 97.06% for the Microsoft dataset, and 96.92% for the MaleVis dataset, which is slightly lesser than the mixed forest model. LR, SVM, and NB models show low performance in classifying malware for all the three datasets.

Table 3 shows the comparison of the proposed model with the previous works in malware detection for the three datasets. The performance analysis is done based on the criteria such as features used, classifier used, number of features, number of samples, number of classes, and accuracy. Various features such as GIST, Gray-Level Co-Occurrence Matrix (GLCM), Gabor, Wavelet, Intensity, and LBP were extracted and used for classification of malware. The proposed system uses global features and deep forest as a classifier to classify malware with three malware datasets. The proposed system outperforms other works extracting distinct features.

Table 4 shows the performance comparison of the proposed deep forest model with deep learning models such as CNN, VGG16, VGG19, Inception-v3, Resnet-50, Densenet-121, Densenet-169, Densenet-201, and Xception for the three datasets. The proposed model outperforms the deep learning models in terms of accuracy, precision, recall, and f-score.

The running efficiency of the proposed model is appreciable. Sliding window scanning will increase the cost of the proposed deep forest model. However, the different grains of scanning are inherently parallel. Table 5 shows the computational time efficiency of the proposed model over the deep learning models for the three datasets. The training time taken by the proposed model with the Malimg dataset is 4165 seconds. With BIG 2015 and MaleVis datasets, the proposed model takes 2496 seconds and 8974 seconds for training. Deep learning variant models show a higher training time comparably, with GPU. But the proposed malware detection model takes a lesser training time without GPU.

D. RESILIENCE TO OBFUSCATION

The proposed model is robust, even if the malware is obfuscated. The BIG 2015 malware dataset contains Obfuscator.ACY class consisting of 1228 obfuscated binary samples. The detection accuracy of the proposed model on this dataset

TABLE 2. Performance comparison of the proposed models with machine learning models for the four datasets.

Models	Malimg Dataset				Microsoft Dataset				MaleVis Dataset				Malicia Dataset (Unseen)			
	Accuracy (%)	Precision	Recall	F-score	Accuracy (%)	Precision	Recall	F-score	Accuracy (%)	Precision	Recall	F-score	Accuracy (%)	Precision	Recall	F-score
KNN	82.4	0.8146	0.8233	0.8189	85.28	0.8614	0.8464	0.8538	84.36	0.8531	0.8357	0.8443	76.75	0.7753	0.7618	0.7685
LR	69.2	0.6955	0.668	0.6815	62.59	0.6421	0.6235	0.6327	66.47	0.6686	0.6575	0.6630	56.33	0.5779	0.5612	0.5694
SVM	75.1	0.7459	0.7534	0.7496	89.25	0.9042	0.8846	0.8943	88.38	0.8779	0.8748	0.8763	80.33	0.8138	0.7961	0.8049
Naïve Bayes	56.25	0.5678	0.5547	0.5612	52.14	0.5158	0.5223	0.5190	55.62	0.5622	0.5474	0.5547	46.93	0.4642	0.4701	0.4671
Decision Tree	88.47	0.8798	0.8721	0.8759	86.41	0.8632	0.8575	0.8603	87.35	0.8787	0.8663	0.8725	77.77	0.7769	0.7718	0.7743
Random Forest	90.75	0.9127	0.8963	0.9044	91.22	0.9179	0.9064	0.9121	90.28	0.8985	0.9055	0.9020	82.10	0.8261	0.8158	0.8209
Adaboost	74.36	0.7463	0.7286	0.7373	83.68	0.8512	0.8244	0.8376	76.44	0.7564	0.7652	0.7608	75.31	0.7661	0.742	0.7538
CRF Forest	98.62	0.9856	0.9847	0.9851	97.06	0.9724	0.9647	0.9685	96.92	0.9704	0.9696	0.9700	87.35	0.8752	0.8682	0.8717
RF Forest	98.39	0.9846	0.9774	0.9810	96.67	0.9644	0.9572	0.9608	96.53	0.9669	0.9650	0.9659	87.00	0.868	0.8615	0.8647
XGB Forest	98.43	0.9824	0.9826	0.9825	96.53	0.9733	0.9637	0.9685	96.35	0.9636	0.9634	0.9635	86.88	0.876	0.8673	0.8716
Mixed Forest	**98.65**	**0.9886**	**0.9863**	**0.9874**	**97.2**	**0.9761**	**0.9679**	**0.9720**	**97.43**	**0.9753**	**0.9732**	**0.9742**	**87.48**	**0.8785**	**0.8711**	**0.8748**

TABLE 3. Performance comparison of the proposed model with previous works in malware detection based on features for the three datasets.

Features	GIST [31]	GLCM [9]	GLCM [9]	Gabor + Wavelet + Intensity [21]	Global (texture and color) and Local features [15]	GIST [29]	LBP [27]	**Global features**	**Global features**	**Global features**
Classifier	KNN	KNN	SVM	SVM	RF	ANN	CNN	Proposed Deep Forest	Proposed Deep Forest	Proposed Deep Forest
Number of features	320	-	-	534	157	320	-	-	-	-
Number of Samples	9458	9342	9342	15000	7087	3131	12348	9339	10868	14226
Number of Classes	25	25	25	-00	15	24	32	26	10	26
Accuracy (%)	97.18	92.5	93.2	95.95	97.47	96.35	93.92	**98.65**	**97.2**	**97.43**

is 97.2%, which indicates the resiliency of the proposed system.

E. MODEL COMPLEXITY

The proposed malware detection model uses a minimal number of layers to train and test the data. The model complexities of the proposed deep layered forest method for the three datasets are shown in Fig. 14, Fig. 15, and Fig. 16. The model builds layers based on the data. For the malimg dataset, the model grows until 5 layers. The test accuracy starts improving at layer 2. At layer 5, there is no improvement, and the layer building process stops. Similarly, for the BIG2015 dataset, the proposed method builds 11 layers, and the test accuracy does not improve after layer 9. For the MaleVis dataset, the training accuracy does not show any improvement from layer 2.

Deep forest model involves sliding-window application and layering as the key stages by employing traditional forests as a subroutine. In the first stage, four forests are used for two sliding windows. In the second stage, each layer uses 4 forests. Each forest generates class vector probabilities using 10 decision trees. The computational complexity is higher for the proposed malware detection method, since deep forest allows more computations than malware detectors

TABLE 4. Performance comparison of the proposed models with deep learning models for the four datasets.

Models	Malimg Dataset				Microsoft Dataset				MaleVis Dataset				Malicia Dataset (Unseen)			
	Accuracy (%)	Precision	Recall	F-score	Accuracy (%)	Precision	Recall	F-score	Accuracy (%)	Precision	Recall	F-score	Accuracy (%)	Precision	Recall	F-score
CNN	97.59	0.9761	0.9748	0.9754	95.67	0.9573	0.9570	0.9571	94.38	0.9441	0.9438	0.9439	71.42	0.722	0.7061	0.7139
VGG16	97.44	0.9754	0.9742	0.9748	88.61	0.8872	0.8861	0.8866	96.18	0.9644	0.9576	0.9610	77.66	0.7817	0.7765	0.7791
VGG19	97.51	0.9765	0.9753	0.9759	88.82	0.8886	0.8879	0.8882	96.27	0.9637	0.9627	0.9632	82.92	0.8288	0.827	0.8279
Inception-v3	97.65	0.9870	0.9864	0.9867	93.29	0.9336	0.9328	0.9332	95.32	0.9568	0.9499	0.9533	83.7	0.8358	0.825	0.8304
Resnet-50	97.68	0.9761	0.9768	0.9764	88.52	0.8868	0.8852	0.8860	90.36	0.9063	0.8994	0.9028	82.52	0.8312	0.8062	0.8185
Densenet-121	98.15	0.9808	0.9814	0.9811	96.77	0.9672	0.9675	0.9673	95.27	0.9532	0.9515	0.9523	83.02	0.8261	0.8186	0.8224
Densenet-169	97.82	0.9778	0.9783	0.9780	96.26	0.9628	0.9626	0.9627	95.65	0.9586	0.9562	0.9574	82.18	0.8213	0.8178	0.8196
Densenet-201	97.67	0.9768	0.9725	0.9746	96.46	0.9658	0.9684	0.9621	97.21	0.9746	0.9720	0.9733	82.12	0.8206	0.8167	0.8186
Xception	98.03	0.9796	0.9803	0.9799	96.78	0.9680	0.9673	0.9676	97.49	0.9757	0.9738	0.9747	81.53	0.8229	0.8081	0.8154
Proposed Model	**98.65**	**0.9886**	**0.9863**	**0.9874**	**97.2**	**0.9761**	**0.9679**	**0.9720**	**97.43**	**0.9753**	**0.9732**	**0.9742**	**87.68**	**0.8778**	**0.8759**	**0.8768**

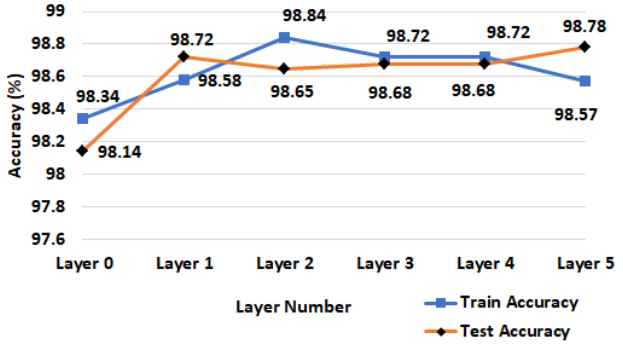

FIGURE 14. Model complexity of the proposed method for Malimg dataset.

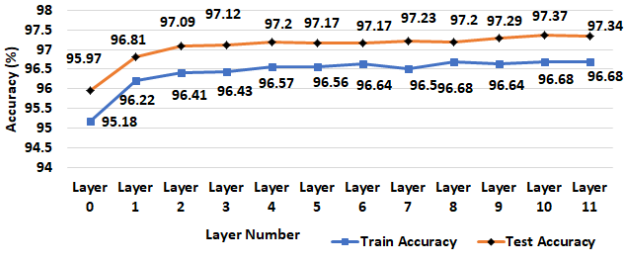

FIGURE 15. Model complexity of the proposed method for BIG2015 dataset.

based on machine learning models. Unlike most deep neural networks with fixed model complexity, the proposed classification model automatically decides its model complexity by terminating the training process when reaching the stopping criterion. Thus, the computational complexity of the proposed

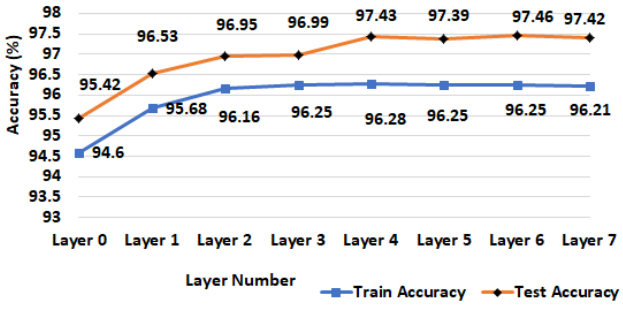

FIGURE 16. Model complexity of the proposed method for MaleVis dataset.

TABLE 5. Comparison of proposed model with deep learning models based on training time.

Models	Training time (in seconds)		
	Malimg dataset	BIG 2015 dataset	MaleVis dataset
CNN	6140	4406	10946
VGG16	5174	3652	12721
VGG19	5363	3870	15144
Inception-v3	5604	4146	11379
Resnet-50	6097	4712	8861
Densenet-121	6574	5259	8328
Densenet-169	6710	5416	11461
Densenet-201	7391	6197	13126
Xception	5674	4226	10448
Proposed Model	**4165**	**2496**	**8974**

deep forest based malware detector is lower than the deep learning based malware detection system.

The sliding window scanning stage consists of smaller instances of the input image for each sliding window which demands more memory. Random subsampling is used to handle this high dimensional space. In the second stage of layering, four forests are generated with 10 decision trees each. With more number of training data, the proposed approach obviously requires more memory compared to simple random forest machine learning approach. Compared to deep learning models, deep forest based malware detection system is better in terms of space complexity.

VI. CONCLUSION

Although there are several ongoing research efforts to malware detection and classification, the malware remains a serious threat in the cyberworld. Malware detection is bypassed using evasion techniques such as code obfuscations, packing, etc., making detection methods ineffective. This work proposes a diverse deep forest model for effective malware detection and classification. The system is focused on enhancing the existing malware detection systems in three aspects. Firstly, the PE binary files are converted to 2D grayscale images. Secondly, the images are processed in two phases, namely, sliding window scanning phase and cascade layering phase. The sliding window scanning phase is similar to convolutional neural networks where each pixel is processed using sliding windows, which allow considering critical features aiding for better prediction. The cascade layering phase consists of layers like deep learning but generates feature vectors without backpropagation. Thirdly, cross-validation performance is used to decide the layering process, whether to continue or stop.

The proposed model showed a detection rate of 98.65%, 97.2%, and 97.43% for Malimg, BIG 2015, and MaleVis malware datasets, respectively, which is better than existing systems discussed in literature. The proposed approach is characterized by its deep ensemble layering and low model complexity and outperforms deep neural networks in detecting malware. The proposed model identifies unknown malware samples belonging to the trained malware families. The malware samples, which belong to other families that are not trained by the model, may show incorrect predictions. Future work may include identifying unknown samples of untrained families by using a threshold.

ACKNOWLEDGMENT

The authors would like to thank Mr. J. Kesavardhanan, Founder and Chairman, K7 Computing Pvt., Ltd., and the CEO of The Association of Anti-Virus Asia Researchers (AVAR) for providing malware samples from their laboratory for analysis, and for the discussions, helping us out with malware research directions.

REFERENCES

[1] A. F. Agarap, "Towards building an intelligent anti-malware system: A deep learning approach using support vector machine (SVM) for malware classification," 2017, *arXiv:1801.00318*. [Online]. Available: http://arxiv.org/abs/1801.00318

[2] M. Ahmadi, D. Ulyanov, S. Semenov, M. Trofimov, and G. Giacinto, "Novel feature extraction, selection and fusion for effective malware family classification," in *Proc. 6th ACM Conf. Data Appl. Secur. Privacy (CODASPY)*, 2016, pp. 183–194.

[3] F. Ahmed, H. Hameed, M. Z. Shafiq, and M. Farooq, "Using spatio-temporal information in API calls with machine learning algorithms for malware detection," in *Proc. 2nd ACM Workshop Secur. Artif. Intell. (AISec)*, 2009, pp. 55–62.

[4] B. Anderson, D. Quist, J. Neil, C. Storlie, and T. Lane, "Graph-based malware detection using dynamic analysis," *J. Comput. Virol.*, vol. 7, no. 4, pp. 247–258, Nov. 2011, doi: 10.1007/s11416-011-0152-x.

[5] P. L. Bartlett, "The sample complexity of pattern classification with neural networks: The size of the weights is more important than the size of the network," *IEEE Trans. Inf. Theory*, vol. 44, no. 2, pp. 525–536, Mar. 1998, doi: 10.1109/18.661502.

[6] A. S. Bozkir, A. O. Cankaya, and M. Aydos, "Utilization and comparision of convolutional neural networks in malware recognition," in *Proc. 27th Signal Process. Commun. Appl. Conf. (SIU)*, Apr. 2019, pp. 1–4.

[7] Z. Chen and C. Ji, "Spatial-temporal modeling of malware propagation in networks," *IEEE Trans. Neural Netw.*, vol. 16, no. 5, pp. 1291–1303, Sep. 2005, doi: 10.1109/tnn.2005.853425.

[8] M. Christodorescu, S. Jha, S. A. Seshia, D. Song, and R. E. Bryant, "Semantics-aware malware detection," in *Proc. IEEE Symp. Secur. Privacy (S&P)*, May 2005, pp. 32–46.

[9] Z. Cui, F. Xue, X. Cai, Y. Cao, G.-G. Wang, and J. Chen, "Detection of malicious code variants based on deep learning," *IEEE Trans. Ind. Informat.*, vol. 14, no. 7, pp. 3187–3196, Jul. 2018, doi: 10.1109/tii.2018.2822680.

[10] (2019). *Cylance 2019 Threat Report*. Accessed: Nov. 13, 2019. [Online]. Available: https://www.cylance.com/content/dam/cylance-web/en-us/resources/knowledge-center/resource-library/reports/Cylance-2019-Threat-Report.pdf

[11] A. Dadlani, M. Kumar, K. Kim, and K. Sohraby, "Stability and immunization analysis of a malware spread model over scale-free networks," *IEEE Commun. Lett.*, vol. 18, no. 11, pp. 1907–1910, Nov. 2014, doi: 10.1109/lcomm.2014.2361525.

[12] Y. Ding, X. Xia, S. Chen, and Y. Li, "A malware detection method based on family behavior graph," *Comput. Secur.*, vol. 73, pp. 73–86, Mar. 2018, doi: 10.1016/j.cose.2017.10.007.

[13] D. Du, Y. Sun, Y. Ma, and F. Xiao, "A novel approach to detect malware variants based on classified behaviors," *IEEE Access*, vol. 7, pp. 81770–81782, 2019, doi: 10.1109/access.2019.2924331.

[14] Y. Fan, Y. Ye, and L. Chen, "Malicious sequential pattern mining for automatic malware detection," *Expert Syst. Appl.*, vol. 52, pp. 16–25, Jun. 2016, doi: 10.1016/j.eswa.2016.01.002.

[15] J. Fu, J. Xue, Y. Wang, Z. Liu, and C. Shan, "Malware visualization for fine-grained classification," *IEEE Access*, vol. 6, pp. 14510–14523, 2018, doi: 10.1109/access.2018.2805301.

[16] D. Gibert, "Convolutional neural networks for malware classification," M.S. thesis, Univ. Rovira i Virgili, Tarragona, Spain, Oct. 2016.

[17] K. Griffin, S. Schneider, X. Hu, and T. C. Chiueh, "Automatic generation of string signatures for malware detection," in *Proc. Int. Workshop Recent Adv. Intrusion Detection*. Berlin, Germany: Springer, Sep. 2009, pp. 101–120.

[18] K. Han, B. Kang, and E. G. Im, "Malware analysis using visualized image matrices," *Sci. World J.*, vol. 2014, pp. 1–15, Jul. 2014, doi: 10.1155/2014/132713.

[19] R. Islam, R. Tian, L. M. Batten, and S. Versteeg, "Classification of malware based on integrated static and dynamic features," *J. Netw. Comput. Appl.*, vol. 36, no. 2, pp. 646–656, Mar. 2013, doi: 10.1016/j.jnca.2012.10.004.

[20] M. Kalash, M. Rochan, N. Mohammed, N. D. B. Bruce, Y. Wang, and F. Iqbal, "Malware classification with deep convolutional neural networks," in *Proc. 9th IFIP Int. Conf. New Technol., Mobility Secur. (NTMS)*, Feb. 2018, pp. 1–5.

[21] K. Kancherla and S. Mukkamala, "Image visualization based malware detection," in *Proc. IEEE Symp. Comput. Intell. Cyber Secur. (CICS)*, Apr. 2013, pp. 40–44.

[22] R. U. Khan, X. Zhang, and R. Kumar, "Analysis of ResNet and GoogleNet models for malware detection," *J. Comput. Virol. Hacking Techn.*, vol. 15, no. 1, pp. 29–37, Mar. 2019, doi: 10.1007/s11416-018-0324-z.

[23] C. Kolbitsch, P. M. Comparetti, C. Kruegel, E. Kirda, X.-Y. Zhou, and X. Wang, "Effective and efficient malware detection at the end host," in *Proc. USENIX Secur. Symp.*, Aug. 2009, vol. 4, no. 1, pp. 351–366.

[24] J. Z. Kolter and M. A. Maloof, "Learning to detect malicious executables in the wild," in *Proc. ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining (KDD)*, 2004, pp. 470–478.
[25] Q. Le, O. Boydell, B. M. Namee, and M. Scanlon, "Deep learning at the shallow end: Malware classification for non-domain experts," *Digit. Invest.*, vol. 26, pp. S118–S126, Jul. 2018, doi: 10.1016/j.diin.2018.04.024.
[26] Y.-S. Liu, Y.-K. Lai, Z.-H. Wang, and H.-B. Yan, "A new learning approach to malware classification using discriminative feature extraction," *IEEE Access*, vol. 7, pp. 13015–13023, 2019, doi: 10.1109/access.2019.2892500.
[27] J.-S. Luo and D. C.-T. Lo, "Binary malware image classification using machine learning with local binary pattern," in *Proc. IEEE Int. Conf. Big Data (Big Data)*, Dec. 2017, pp. 4664–4667.
[28] X. Ma, S. Guo, H. Li, Z. Pan, J. Qiu, Y. Ding, and F. Chen, "How to make attention mechanisms more practical in malware classification," *IEEE Access*, vol. 7, pp. 155270–155280, 2019, doi: 10.1109/access.2019.2948358.
[29] A. Makandar and A. Patrot, "Malware analysis and classification using artificial neural network," in *Proc. Int. Conf. Trends Automat., Commun. Comput. Technol. (I-TACT)*, Dec. 2015, pp. 1–6.
[30] (2019). *Malwarebytes 2019 Threat Report*. Accessed: Nov. 13, 2019. [Online]. Available: https://resources.malwarebytes.com/files/2019/01/Malwarebytes-Labs-2019-State-of-Malware-Report-2.pdf
[31] L. Nataraj, S. Karthikeyan, G. Jacob, and B. S. Manjunath, "Malware images: Visualization and automatic classification," in *Proc. 8th Int. Symp. Visualizat. Cyber Secur. (VizSec)*, 2011, pp. 1–7.
[32] B. N. Narayanan, O. Djaneye-Boundjou, and T. M. Kebede, "Performance analysis of machine learning and pattern recognition algorithms for malware classification," in *Proc. IEEE Nat. Aerosp. Electron. Conf. (NAECON), Ohio Innov. Summit (OIS)*, Jul. 2016, pp. 338–342.
[33] S. Naval, V. Laxmi, M. Rajarajan, M. S. Gaur, and M. Conti, "Employing program semantics for malware detection," *IEEE Trans. Inf. Forensics Security*, vol. 10, no. 12, pp. 2591–2604, Dec. 2015, doi: 10.1109/tifs.2015.2469253.
[34] R. Perdisci, A. Lanzi, and W. Lee, "Classification of packed executables for accurate computer virus detection," *Pattern Recognit. Lett.*, vol. 29, no. 14, pp. 1941–1946, Oct. 2008, doi: 10.1016/j.patrec.2008.06.016.
[35] I. Santos, F. Brezo, X. Ugarte-Pedrero, and P. G. Bringas, "Opcode sequences as representation of executables for data-mining-based unknown malware detection," *Inf. Sci.*, vol. 231, pp. 64–82, May 2013, doi: 10.1016/j.ins.2011.08.020.
[36] M. G. Schultz, E. Eskin, F. Zadok, and S. J. Stolfo, "Data mining methods for detection of new malicious executables," in *Proc. IEEE Symp. Secur. Privacy. (S&P)*, May 2000, pp. 38–49.
[37] A. Shabtai, E. Menahem, and Y. Elovici, "F-sign: Automatic, function-based signature generation for malware," *IEEE Trans. Syst., Man, Cybern. C, Appl. Rev.*, vol. 41, no. 4, pp. 494–508, Jul. 2011, doi: 10.1109/tsmcc.2010.2068544.
[38] A. Singh, A. Handa, N. Kumar, and S. K. Shukla, "Malware classification using image representation," in *Proc. Int. Symp. Cyber Secur. Cryptogr. Mach. Learn.* Cham, Switzerland: Springer, Jun. 2019, pp. 75–92.
[39] J. Stiborek, T. Pevný, and M. Rehák, "Multiple instance learning for malware classification," *Expert Syst. Appl.*, vol. 93, pp. 346–357, Mar. 2018, doi: 10.1016/j.eswa.2017.10.036.
[40] R. Vinayakumar, M. Alazab, K. P. Soman, P. Poornachandran, and S. Venkatraman, "Robust intelligent malware detection using deep learning," *IEEE Access*, vol. 7, pp. 46717–46738, 2019, doi: 10.1109/access.2019.2906934.
[41] (2019). *VirusTotal*. Accessed: Dec. 4, 2019. [Online]. Available: https://www.virustotal.com/
[42] M. Wadkar, F. Di Troia, and M. Stamp, "Detecting malware evolution using support vector machines," *Expert Syst. Appl.*, vol. 143, Apr. 2020, Art. no. 113022, doi: 10.1016/j.eswa.2019.113022.
[43] S. Wen, W. Zhou, J. Zhang, Y. Xiang, W. Zhou, W. Jia, and C. C. Zou, "Modeling and analysis on the propagation dynamics of modern email malware," *IEEE Trans. Dependable Secure Comput.*, vol. 11, no. 4, pp. 361–374, Jul. 2014, doi: 10.1109/tdsc.2013.49.
[44] J. Zhang, Z. Qin, H. Yin, L. Ou, S. Xiao, and Y. Hu, "Malware variant detection using opcode image recognition with small training sets," in *Proc. 25th Int. Conf. Comput. Commun. Netw. (ICCCN)*, Aug. 2016, pp. 1–9.
[45] Z.-H. Zhou and J. Feng, "Deep forest," 2017, *arXiv:1702.08835*. [Online]. Available: http://arxiv.org/abs/1702.08835
[46] A. Nappa, M. Z. Rafique, and J. Caballero, "The MALICIA dataset: Identification and analysis of drive-by download operations," *Int. J. Inf. Secur.*, vol. 14, no. 1, pp. 15–33, Feb. 2015, doi: 10.1007/s10207-014-0248-7.
[47] H.-J. Li, Z. Wang, J. Pei, J. Cao, and Y. Shi, "Optimal estimation of low-rank factors via feature level data fusion of multiplex signal systems," *IEEE Trans. Knowl. Data Eng.*, early access, Aug. 13, 2020, doi: 10.1109/TKDE.2020.3015914.
[48] H.-J. Li, L. Wang, Y. Zhang, and M. Perc, "Optimization of identifiability for efficient community detection," *New J. Phys.*, vol. 22, no. 6, Jun. 2020, Art. no. 063035, doi: 10.1088/1367-2630/ab8e5e.
[49] K. Grifn, S. Schneider, X. Hu, and T.-C. Chiueh, "Automatic generation of string signatures for malware detection," in *Proc. Int. Workshop Recent Adv. Intrusion Detection*. Berlin, Germany: Springer, 2009, pp. 101–120.

S. ABIJAH ROSELINE received the B.E. degree in computer science and engineering from the Vel's Srinivasa College of Engineering and Technology (Affiliated to Anna University), Chennai, India, in 2008, and the M.E. degree in computer science and engineering from the MNM Jain Engineering College (Affiliated to Anna University), Chennai, in 2011. She is currently pursuing the Ph.D. degree with the School of Computer Science and Engineering, Vellore Institute of Technology, Chennai Campus, Chennai. She has published papers in reputed international conferences. Her research interests include cybersecurity, computer vision, and machine learning.

S. GEETHA (Senior Member, IEEE) received the B.E. degree in computer science and engineering from Madurai Kamaraj University, India, in 2000, and the M.E. degree in computer science and engineering and the Ph.D. degree from Anna University, Chennai, India, in 2004 and 2011, respectively. She has more than 18 years of rich teaching and research experience. She is currently a Professor and the Associate Dean with the School of Computer Science and Engineering, Vellore Institute of Technology, Chennai Campus, India. She has published more than 80 papers in reputed international conferences and refereed journals. Her research interests include steganography, steganalysis, multimedia security, intrusion detection systems, machine learning paradigms, and information forensics. She joins the Review Committee and the Editorial Advisory Board of journals, such as IEEE TRANSACTIONS ON INFORMATION FORENSICS AND SECURITY and IEEE TRANSACTIONS ON IMAGE PROCESSING, *Multimedia Tools and Security* (Springer), and *Information Sciences* (Elsevier). She has published four books. She has given many expert lectures, keynote addresses at international, and national conferences. She has organized many workshops, conferences, and FDPs. She was a recipient of the University Rank and Academic Topper Award from her B.E. and M.E. degrees, in 2000 and 2004, respectively. She was also the proud recipient of the ASDF Best Academic Researcher Award 2013, the ASDF Best Professor Award 2014, the Research Award in 2016, and the High Performer.

SEIFEDINE KADRY (Senior Member, IEEE) received the bachelor's degree from Lebanese University, in 1999, the M.S. degree from Reims University, France, in 2002, and the EPFL, Lausanne, the Ph.D. degree from Blaise Pascal University, France, in 2007, and the HDR degree from Rouen University, in 2017. His current research interests include data science, education using technology, system prognostics, stochastic systems, and probability and reliability analysis. He is an ABET Program Evaluator of computing and an ABET Program Evaluator of Engineering Tech.

YUNYOUNG NAM (Member, IEEE) received the B.S., M.S., and Ph.D. degrees in computer engineering from Ajou University, South Korea in 2001, 2003, and 2007, respectively. He was a Senior Researcher with the Center of Excellence in Ubiquitous System, Stony Brook University, Stony Brook, NY, USA, from 2007 to 2010, where he was a Postdoctoral Researcher, from 2009 to 2013. He was a Research Professor with Ajou University, from 2010 to 2011. He was a Postdoctoral Fellow with the Worcester Polytechnic Institute, Worcester, MA, USA, from 2013 to 2014. He was the Director of the ICT Convergence Rehabilitation Engineering Research Center, Soonchunhyang University, from 2017 to 2020. He has been the Director of the ICT Convergence Research Center, Soonchunhyang University, since 2020, where he is currently an Assistant Professor with the Department of Computer Science and Engineering. His research interests include multimedia database, ubiquitous computing, image processing, pattern recognition, context-awareness, conflict resolution, wearable computing, intelligent video surveillance, cloud computing, biomedical signal processing, rehabilitation, and healthcare systems.