Cluster Computing (2024) 27:9191–9220
https://doi.org/10.1007/s10586-024-04397-4

Deep learning vs. adversarial noise: a battle in malware image analysis

K. A. Asmitha[1] · Vinod Puthuvath[1,2] · K. A. Rafidha Rehiman[1] · S. L. Ananth[3]

Received: 10 October 2023 / Revised: 12 February 2024 / Accepted: 26 February 2024 / Published online: 17 April 2024

Abstract
The proliferation of malware variants has shown a steep increase, attributed to their enhanced sophistication and the utilization of the latest technologies. This constitutes a severe menace to smart gadgets and IT infrastructure. Malware visualization has emerged as an exceptionally attractive technique, primarily because it obviates the need for disassembly or code execution. In this approach, malicious executables are transformed into visual representations resembling images. This visual representation allows for the extraction of textural features using the Local Binary Pattern (LBP) technique. Subsequently, classification models are constructed using ResNet50, VGG16, and customized models tailored to the specific task. These model undergoes extensive evaluation through two benchmark datasets: the MalImg dataset (consisting of 9,342 instances of malware across 25 families) and the Malware Classification Challenge dataset (BIG2015) (with 10,868 labeled malware instances across nine families). Additionally, the model is validated on a self-made dataset, which we named Malhub, consisting of 26,452 executables comprising 20 families. Furthermore, we implemented a white-box adversarial attack using additive noise (Gaussian, Local Variable, Poisson, Salt and Pepper, Speckle). We observed an F1 score in the range of 0.992−0.993 for MalImg, 0.874−0.878 for BIG2015, and 0.014−0.992 for Malhub dataset. This proves that efforts are required to tune machine learning models to detect adversarial examples.

Keywords Malware classification · Malware visualization · Deep learning · Adversarial attack · Local binary patterns · Classifier fusion

1 Introduction

Cyber attacks are on the rise and have emerged as a disruptive force against cybersecurity, driven by the proliferation of malware and the increasing sophistication in their development. According to the PurpleSec Trend Report, cybercriminals have recorded a 600% surge in launching cybercrimes through malware during lockdown and the digital convergence in the pandemic.[1] Cyber threats were identified as the top human-caused risks, with an estimated financial loss of around $11.4 million per minute, as documented in Crowdstrike global threat report (2021) .[2] Furthermore, there has been an increase in malware attacks targeting various industries, including education, healthcare, technology, and government, as reported in Global industry sectors most targeted by malware incidents in 2020.[3]

Malware, a collection of malicious programs, consists of executable code that can infiltrate an organization's security. Commonly, all potentially unwanted software, such as viruses, trojans, worms, adware, ransomware, spyware, keyloggers, rootkits, etc., is referred to as malware.

✉ Vinod Puthuvath
vinod.puthuvath@unipd.it; vinod.p@cusat.ac.in

K. A. Asmitha
asmitha@pg.cusat.ac.in

K. A. Rafidha Rehiman
rafidharehimanka@cusat.ac.in

S. L. Ananth
ananthceh19@gmail.com

[1] Department of Computer Applications, Cochin University of Science & Technology, Kochi 682022, Kerala, India

[2] Department of Mathematics, University of Padua, Padua, Italy

[3] Cisco Systems India Pvt. Ltd., Bangalore, India

[1] Cyber Security Trends Report (2021): https://purplesec.us/cyber-security-trends-2021.

[2] Crowdstrike global threat report (2021): https://go.crowdstrike.-com/rs/281-OBQ-266/images/ Report2021GTR.pdf.

[3] Global industry sectors most targeted by malware incidents in 2020: https://www.statista.com/statistics/223517/ malware-infection-weekly-industries/

Generally, malware is classified depending on its functionality, mode of propagation, and impact on the system. Security of interconnected devices is paramount for modern organizations, as cyber threats continue to evolve and pose significant risks. Understanding the various types of malware and their infection mechanisms is crucial for organizations to protect their systems and data effectively. Malware analysis involves the techniques or tools for understanding its behavior to build defense strategies and prepare systems to withstand future attacks. Anti-malware vendors generally employ signature and heuristic-based approaches to detect and remove malicious files before affecting the system. The former method matches strings extracted from suspicious samples against a known signature repository. This method is not suitable for detecting new malware strains. Besides, managing a massive database of signatures is practically infeasible. Generally, experts categorize heuristic-based approaches into static, dynamic, and hybrid analysis. Static analysis examines the source code by decompiling the allegedly malicious software. This method fails to identify encrypted and obfuscated code as it defeats reverse engineering. Conversely, dynamic approaches perform analysis by unpacking and executing suspected binary code in a virtual machine or sandbox. However, such methods have limited code coverage and are computationally expensive.

The use of machine learning (ML)and deep learning (DL)algorithms have gained popularity for malicious software detection using attributes extracted from both static and dynamic approaches [1–3]. Malware detection using ML techniques gained popularity due to (1) the availability of labeled malware feeds and (2) a reduction in the cost of hardware. However, the solutions based on ML have shown remarkable acceptance by researchers from industries and academia. These methods involve substantial time and resources to extract relevant features through feature engineering. Researchers have explored image visualization techniques for recognizing the visual similarities within malware families, malware detection, and classification [4]. This approach departs from traditional methods that rely on intricate feature engineering and instead leverages the inherent visual patterns present in malware samples. This shift in approach offers several advantages [4], including (a) reduced reliance on domain-specific knowledge, (b) enhanced robustness against obfuscation, and (c) improved generalizability.

In this approach, malware binaries are visualized as grayscale or color images [5], enabling the application of deep learning techniques like Convolutional Neural Networks (CNNs), Auto Encoders (AEs), and Long Short Term Memory (LSTMs) for malware detection and classification. While these deep learning methods have demonstrated promising results, they exhibit a critical vulnerability to adversarial examples (AEs) [6, 7].

This paper aims to classify malicious executable variants into their respective families employing visualization and deep learning algorithms. The proposed approach utilizes Local Binary Pattern (LBP) to extract distinctive features from malware samples, transforming them into visual representations. These LBP-generated images are fed into pre-trained Convolutional Neural Networks (CNNs) like ResNet50 and VGG16, along with customized classifiers, to classify malware samples effectively. Furthermore, to assess the efficacy of the proposed approach, we conducted repeated experiments using images without employing LBP conversion. Using LBP, the proposed model can classify samples into corresponding families with an F1 score in the range of 0.90−0.995, 0.989−0.998, and 0.993−0.999 on BIG2015, MalImg, and Malhub datasets, respectively. Moreover, we supplied the model with adversarial samples generated using additive noise and observed a marginal reduction of 4.1% for MalImg, 11.9% for BIG2015, and 98.5% for Malhub in their respective maximum F1 scores. In general, the major contributions of this research work are as follows:

1. Developing a novel deep learning technique that integrates malware visualization and family categorization incorporating textural features of images using Local Binary Pattern (LBP) on grayscale images.
2. Providing a comprehensive analysis of different image dimensions on two well-known benchmark datasets: MalImg, and the Microsoft Malware Classification Challenge and Malhub dataset.
3. Conducting a rigorous analysis by executing a variety of conventional machine learning and cutting-edge deep learning designs using three different datasets.
4. Analyzing the performance through a comparative analysis of visualization-based methods for malware classification. The results show that our proposed approach can classify samples with the highest F1 score of 0.995(64x64), 0.998(128x128), and 0.999(128x128) for BIG2015, MalImg, and Malhub datasets, respectively, which is superior to other methods in the literature.
5. We also demonstrate white-box attacks by generating tainted examples. The experiments demonstrate a marginal drop in the performance with classifiers trained on MalImg(1.6−4.1%) using ResNet+CNN+Dense Layer. In the case of Fusion model, images in MalImg were resilient to adversarial samples except for Pepper noise(1%). On the contrary, ResNet50+CNN+DL trained on BIG 2015 samples were resilient to evasion attacks, and for the fused ResNet50||CNN||DL, we experienced a drop of a

maximum 11.9%. For the Malhub dataset, the drop is in the range of 0.7−98.5%.

The remaining part of the paper is arranged as follows: Sect. 2 presents the research status in malware classification. Section 3 presents a detailed explanation of the proposed method. Section 4 expounds on the evaluation measures and details of the results. Finally, section 5 summarizes our findings and provides directions for the future work.

2 Related works

Malware detection approaches can be categorized into three main types: static analysis-based, behavior analysis-based, and visualization-based. This section offers insights into existing malware detection techniques by reviewing related works within the above-mentioned categories.

2.1 Static analysis

Static malware analysis is a technique used to examine the code and properties of a suspected malware sample without executing it, eliminating the risk of infecting the analysis environment. Binary analysis tools such as IDAPro, PEStudio, etc., are used in static analysis techniques to extract static features like strings, file hashes, API calls, opcode sequences, etc., from executable binaries.

In [8], the authors utilized a list of DLLs, functions, function calls within DLLs, encoded strings, and byte sequences. Using the multinomial Naïve Bayes algorithm, they successfully attained 97.11% accuracy on a dataset consisting of 3265 malware samples and 1001 benign executables. However, this approach necessitates feature engineering tools and domain expertise. Moreover, the scarcity of benign examples compared to malware files introduces a divergence from the realistic scenarios.

In [9], Kolter et al. generated n-grams from executables and tested the accuracy of different machine-learning algorithms and their combined versions. They classified malware into multiple families and obtained the best accuracy for the Boosted j48. However, their approach failed to take into account the overhead time.

In [10], Santos et al. have proposed a semi-supervised learning method for malware detection, as it is strenuous to obtain labeled datasets. Byte n-gram distribution approaches with Local, Global Consistency (LGGC) have been used in their work and achieved 86% accuracy for detection. Even though the accuracy is less than other models, they could estimate the number of labeled samples required for learning while maintaining reasonable classification results. However, they considered a very small number of samples and disregarded the overhead time.

In [11], the authors used variable-length instruction sequences to categorize benign and malignant classes using machine learning. They obtained 96% accuracy using Random Forest and Decision Trees. A machine learning method was presented on n-opcode sequences in [12]. They employed a Support Vector Machine (SVM) classifier, achieving 98% accuracy. However, the proposed approach necessitates feature engineering techniques.

2.2 Dynamic analysis

In dynamic analysis-based techniques, researchers extract behavioral features by executing the samples within a controlled sandbox environment. In [13], authors proposed a novel method using Artificial Intelligence techniques to analyze behavior-based malware and classify malware into Worms and Trojans. CWSandbox and Anubis generated behavior profiles for collected samples. Their manual analysis-driven approach needed more scalability and feasibility for practical implementation.

The authors in [14] proposed an incremental method using clustering and classification to create behavioral profiles for malware execution changes. They reported an accuracy of 88% when using SVM. The single execution path limits their approach, and evasion affects it either by detecting the sandbox environment or mimicking different behaviors. In [15], the authors introduce an alternative approach that uses DNA sequence alignment algorithms to identify common API call sequence patterns. However, they must regularly update a list of trusted benign and malicious programs for their approach, which uses white-list and blacklist filtering. Additionally, the hooking process of the study only traces user-level APIs, so API call sequences can't be logged if malware uses kernel-level APIs. Anderson et al. [16] suggested a novel approach to detect malware by applying Markov Chain Graphs on instruction traces during its execution. They employed machine learning algorithms to categorize the data based on global and local similarity, achieving an accuracy of 96.41%. However, this approach necessitates additional computation overhead.

Malware detection and classification traditionally involve static analysis, requiring the disassembly of malware samples into.asm files to extract opcodes and operands. Subsequently, the generation of n-grams or function call graphs for analysis proves computationally expensive and time-consuming. In contrast, dynamic analysis executes samples in a virtualized environment, extracting system calls, memory artifacts, and malware traces. While dynamic analysis is also resource-intensive, time-sensitive, and demands human interaction. Therefore, image-based

malware detection offers an efficient alternative. Reading raw data to generate byte plots and utilizing CNN models for feature extraction, significantly reduces the time and resources required for analysis and providing a more streamlined and practical approach to malware detection and classification.

2.3 Visualization based techniques

Visualization-based malware analysis proves resilient against obfuscation techniques, focusing on the visual representation of malware files. This approach efficiently utilizes resources, requiring fewer computational assets than traditional static and dynamic methods. Its non-execution approach distinguishes it from dynamic analysis, minimizing the risk of triggering malicious behavior and ensuring a safer analytical environment. Leveraging Convolutional Neural Networks (CNNs) allows effective feature extraction from image representations, empowering the model to discern intricate patterns within the data. The visual and interpretable nature of image-based representation facilitates intuitive pattern identification by analysts. However, despite its strengths, image-based analysis encounters challenges. It may lack a comprehensive understanding of malware functionality, prioritizing patterns, and anomaly detection over detailed behavioral insights. The quality of generated images significantly impacts analysis effectiveness, posing a challenge for researchers. Vulnerability to adversarial attacks raises concerns about the reliability of the analysis. Addressing these challenges is crucial to align the analysis method with specific malware analysis goals and sample characteristics.

The authors in [17] proposed the initial work in malware visualization, utilizing self-organizing maps to visualize virus binaries. Natraj et al. [18] visualized grayscale images of malware binaries categorized into 25 families and extracted GIST features. They reported an accuracy of 97.18% for their solution on the K-Nearest Neighbour classifier. Evaluation on adversarial examples was not performed. Such modified samples can exploit the global image-based features to evade detection. Additionally, the researchers have applied deep learning techniques [19] on grayscale images of benign and malware binaries and demonstrated equal performance with machine learning models trained with GIST descriptors [20]. However, the experiment involved only a few malware samples and overlooked the overhead time. The authors in [21] integrated dynamic analysis with image processing techniques to identify unpacked and packed malware samples and concluded that it is not scalable as textural analsis.

In [19], the authors utilized grayscale image representations to visualize malware and benign binaries. They used deep learning techniques to train the model and achieved a test accuracy 95.66% on a dataset comprising 10,000 benign and 200 malware files. However, their system lacked detailed insight into the design and attributes of the malware, and notably, it failed to consider the execution overhead.

Detection of IoT malware using one-channel grayscale images was proposed in [22] and achieved a classification accuracy 94% for DDoS malware using CNN. The researchers applied a deep Convolutional Neural Network to the MalImg dataset in [23] and [24]. They obtain 94.5% and 98.48% accuracy, respectively. However, they designed a notably shallow network structure, and their samples were restricted to only two malware families. Another CNN-based approach proposed by the author in [25] obtained 97.02% accuracy. Researchers also investigated hybrid models such as CNN and bi-directional Gated Recurrent Units, as discussed in [26], and CNN-LSTM, as explored in [27], for classifying malware. However, the authors of the paper neglected to consider overhead time.

Recently, transfer learning [28, 29] was adopted to classify malware samples to its corresponding family. The deep CNNs are trained on natural images and utilize the extracted features to identify characteristic attributes of malware represented as images. In[30], transfer learning with InceptionV1 architecture was applied to malware detection using grayscale images from the Malimg and Microsoft Malware Classification datasets: the multi-class classification achieved 99.25% accuracy and 0.03% false positives on Malimg dataset. A dataset with 16,518 benign and 10,639 malware files used and obtained 99.67% accuracy for binary classification. However, concerns were raised about excluding small malware files and testing against known classes, suggesting improvements in experimental design to handle more realistic data. The authors in[31] investigated the effectiveness of DenseNet in image classification using the visual similarity of malware families. They employed DEAM (Depthwise Efficient Attention Module) and DenseNet for malware detection and family classification. Their results showed 98.5% accuracy for the MalImg dataset, 97.3% for the BIG 2015 dataset, and 99.3% accuracy for a custom dataset composed of both datasets.

Another approach presents the FDL-CADIS model, a fusion of deep learning-based models[32]. This approach uses two-dimensional malware images and applies them to MobileNetv2. The authors used the Black Widow

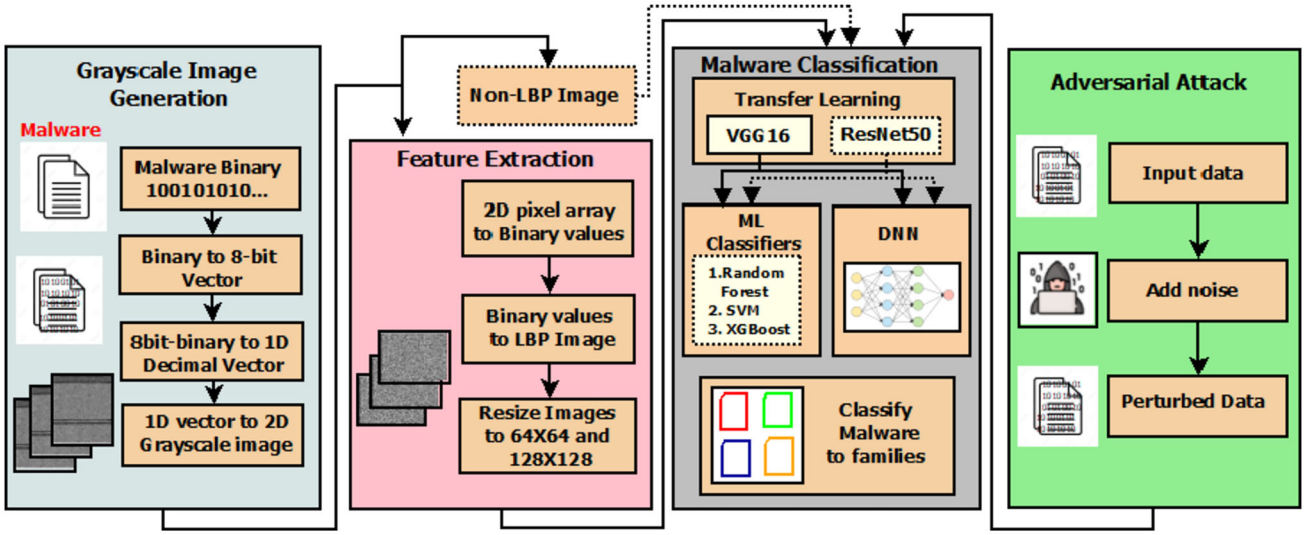


Fig. 1 Proposed architecture for malware family classification consists of **a** Grayscale image generation, **b** feature extraction using LBP and without LBP, **c** Malware classification, **d** adversarial attack on the model

Optimization for tuning hyperparameters. Furthermore, an ensemble of voting-based classifiers, incorporating Gated Recurrent Unit and Long Short-Term Memory techniques, achieves accuracies of 98.73% for the MalImg dataset and 98.83% for the BIG2015 dataset, respectively. However, the exclusive emphasis on malware detection and the absence of information about the utilized features hinder the assessment of their significance. The authors in [33] introduce a malware classification method based on deep learning, which relies on a one-dimensional representation of raw binary. However, these models mentioned in state-of-art were unable to evaluate the impact of adversarial attacks.

Machine learning's dominance in malware detection has made it a vulnerable target for hackers. By manipulating data distributions during training or testing, attackers can trick machine learning classifiers into misclassifying malicious software. This manipulation is called an adversarial attack. Recent works such as Ambra et al. [34] presented a case study of attack vectors against Android malware classifiers. Kathrin et al. [35] demonstrate techniques to evade deep neural networks. They utilized the DREBIN dataset of Android malware and claimed the misclassification of 69%. Similarly, Chen et al. crafted adversarial examples by poisoning syntactic features in [36]. They reported that the attack successfully defeated three popular Android malware classifiers: DREBIN, DroidAPIMiner, and MaMaDroid.

Unlike the above-presented methods, we researched by integrating pre-trained deep neural networks designed for object detection with Convolutional Neural Networks, known for extracting features automatically. We further supplemented this approach with dense layers to detect new visual images of malware. Additionally, to ensure the scalability of the proposed malware detector, we created a custom dataset containing real-time malware samples from 20 distinct malware families, in addition to utilizing benchmark datasets. Furthermore, we thoroughly examined how adversarial attacks, implemented through the introduction of various types of additive noises, impacted the performance of the proposed model. Our investigation centered on assessing whether even these fundamental noise types could result in misclassification of the samples.

3 Proposed method

In this section, we briefly discuss the architecture of the proposed solution (refer Fig. 1).

Table 1 The MalImg dataset comprises images exclusively in PNG format belonging to 25 malware families

No.	Malware Family name	Type	# of Samples	Average Resolution
1	Lolyda.AA 1	Password Stealer(PWS)	213	64 x 428
2	Allaple.A	Worm	2949	256 x 283
3	C2lop.P	Trojan	146	559 x 689
4	Lolyda.AA 2	Password Stealer(PWS)	184	126 x 280
5	C2lop.gen!G	Trojan	200	659 x 817
6	Lolyda.AT	Password Stealer(PWS)	159	64 x 385
7	Dialplatform.B	Dialer	177	64 x 218
8	Yuner.A	Worm	800	768 x 683
9	Alueron.gen!J	Trojan	198	299 x 343
10	Swizzor.gen!E	TDownloader	128	522 x 644
11	Adialer.C	Dialer	125	512 x 409
12	Lolyda.AA 3	Password Stealer(PWS)	123	512 x 478
13	Instantaccess	Dialer	400	384 x 450
14	Fakerean	Rogue	381	384 x 288
15	Malex.gen!J	Trojan	136	262 x 315
16	Rbot!gen	Backdoor	158	505 x 475
17	Wintrim.BX	TDownloader	97	512 x 798
18	Obfuscator.AD	TDownloader	142	384 x 424
19	Skintrim.N	Trojan	80	384 x 502
20	Autorun.K	Worm	106	768 x 683
21	Dontovo.A	TDownloader	162	115 x 313
22	Allaple.L	Worm	1591	128 x 451
23	Agent.FYI	Backdoor	116	64 x 251
24	Vb.AT	Worm	408	702 x 924
25	Swizzor.gen!I	TDownloader	132	513 x 623

3.1 Dataset preparation

The dataset includes malware executables collected from MalImg [18], the Big 2015 (Microsoft Malware Classification Challenge) [37]. Moreover, we created a dataset, which we will refer to as the Malhub dataset throughout this article. The MalImg dataset contains 9,339 malware executables from 25 different families. The transformation of individual byte values into pixels produced the grayscale images depicting each executable within this dataset. We utilized 10,868 tagged malware executables belonging to 9 distinct families from the Big 2015 dataset. Each file contains raw data, encompassing the hexadecimal representation of a binary file and associated metadata comprising strings, function calls, and opcodes. Finally, the Malhub dataset consists of 26,452 samples of 20 families collected from VirusShare[4] repository. Table 1, Table 2, and Table 3 respectively show the datasets along with the details such as family name and number of samples. To ensure the effectiveness and generalizability of our model, we divided each dataset into three subsets: a training set comprising 70% of the malware samples, a validation set containing 20% of the samples, and a test set comprising the remaining 10%. This approach enabled us to train the model on a substantial amount of data, evaluate its

[4] VirusShare: https://virusshare.com/

Table 2 The Microsoft Classification Challenge (BIG 2015) consists of images in PNG format, all with a resolution of 128x128 pixels categorized into 9 families

No.	Malware family name	Type	# of Samples
1	Kelihos_ver3	Backdoor	2942
2	Kelihos_ver1	Backdoor	398
3	Simda	Backdoor	42
4	Obfuscator.ACY	Obfuscated Malware	1228
5	Lollipop	Adware	2478
6	Vundo	Trojan	475
7	Tracur	Trojan Downloader	751
8	Gatak	Backdoor	1013
9	Ramnit	Worm	1541

Table 3 The Malhub dataset consists of images in PNG format, all with a resolution of 256x256 pixels, comprising real-time malware from 20 families

No.	Malware family name	Type	# of Samples
1	Onlinegames	Trojan	1293
2	Renos	Rogue	1312
3	Startpage	Trojan	1136
4	Vundo	Trojan	1793
5	Vobfus	Worm	936
6	Zbot	Trojan	1796
7	Obfuscator	Obfuscated Malware	1445
8	Winwebsec	Rogue	3651
9	Rbot	Backdoor	1017
10	Zeroaccess	Trojan	1129
11	Delfinject	Trojan	1146
12	Lolyda	Password Stealer	915
13	Cycbot	Backdoor	1029
14	Bho	Adware	1176
15	Ceeinject	Trojan	894
16	Hotbar	Adware	1501
17	Adload	Trojan	1050
18	Fakerean	Trojan Downloader	1063
19	Alureon	Trojan	1328
20	Agent	Backdoor	842

performance on a separate collection of samples, and assess its generalizability to unseen malware images.

3.2 Grayscale image generation

Malicious programs exist in the form of Portable Executables (PE), which have extensions such as.bin,.dll,.exe, and consist of code, data, and resource sections. In recent years, researchers have framed the malware classification task as a challenge in image categorization [18]. Obfuscation methods can undermine binary feature extraction, and gray-scale images consisting of textural patterns, can reveal these features through inspection. We represent the raw malware binary as an image to filter patterns from images.

The proposed method splits the raw files into 8-bit strings and maps them to a decimal value between 0 and 255. Thus, we organize each executable into a one-dimensional vector (1D) comprising decimal numbers. Using the decimal values extracted from the 1D array, we construct a 2-dimensional image (2D) with predetermined width and height dimensions. The process of gray-scale image generation is discussed in the Algorithm 1. Fig 2 shows the gray-scale image representation of samples from different families. We observed that variants of the same family exhibit homogeneous textural features. As the malware versions inherit code from the original sample (i.e., code reuse) [19, 22], texture analysis can reveal the hidden patterns to improve the performance of the model during the inference phase. Primarily, the attackers modify a small chunk of the original code to generate obfuscated variants; however, they inherit the code of base malware. Hence, we can conclude that code reuse is evident by looking at Fig. 2. For example, the variants of the Swizzor_gen family are visually similar. Conversely, samples of Agent_FYI family show differences in byte pattern comparing Swizzor_gen (refer Fig. 2).

Algorithm 1 Algorithm for Grayscale image generation

Input: Malware Executable Set, $E = \{e_1, e_2, \ldots, e_n\}$
$w \leftarrow$ width, $h \leftarrow$ height
Output: Set of grayscale Images $I^* = \{I_1, I_2, \ldots, I_n\}$
Decimal vector $D = \{\}$
`/* Initialization */`
$I^* \leftarrow \phi$
$Bytestream \leftarrow \phi$

for *each* $e \in E$ **do**
  $Bytestream \leftarrow Bytestream + e.\text{toByteStream}()$ `/* Converting each binary executable to bytestream */`
  $DV \leftarrow \phi$
  for *each* $b_i \in Bytestream$ **do**
    $DV \leftarrow DV + b_i \times 2^i$ `/* Calculate Decimal value */`
  end
  $D \leftarrow D \cup \{DV\}$ `/* Creating 1D vector of decimal numbers */`
  while $D.length() \neq 0$ **do**
    $I_i \leftarrow \text{img.show}(w, h, D)$ `/* Create Grayscale image */`
  end
  $I^* \leftarrow I^* \cup \{I_i\}$
end

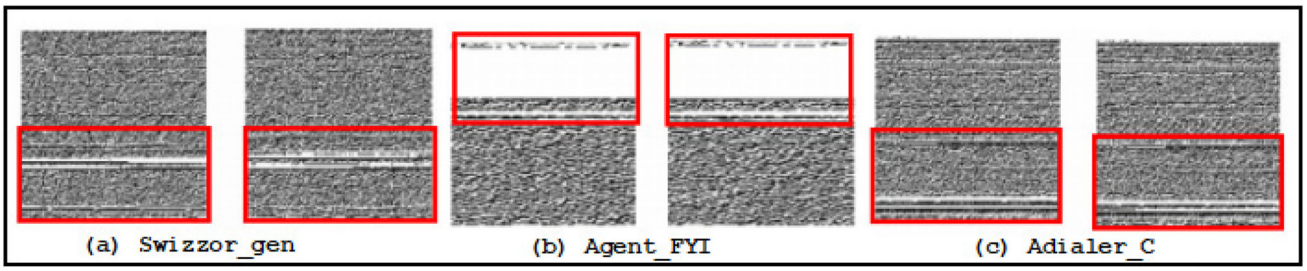


Fig. 2 The grayscale images of binary executables: the bounding box represents the similar patterns exhibited on family variants, and samples from different families exhibit a dissimilarity in patterns

3.3 Feature extraction

Traditional malware detection methods rely on extracting distinctive features from malicious executables to identify their family affiliations. Our approach utilizes Local Binary Patterns (LBP) to capture fine-grained details from grayscale images representing specific malware families. To evaluate the effectiveness of LBP-transformed images in malware classification, we conduct a comparative analysis against normal images (i.e., without LBP transformation).

Algorithm 2 Algorithm for LBP image generation

Input: Grayscale Image Set, $I^* = \{I_1, I_2, \ldots, I_n\}$
$w \leftarrow$ width, $h \leftarrow$ height, $r \leftarrow$ radius
Output: Set of LBP Images $L^* = \{L_1, L_2, \ldots, L_n\}$
/* Initialization */
$L^* \leftarrow \phi$

for *each* $I_m \in I^*$ **do**
  for $s := 1$ *to* w **do**
    for $t := 1$ *to* h **do**
      LBP $\leftarrow \phi$
      block $\leftarrow$ readBlock(I_m, g_c, s, t, r) /* Read each 3×3 blocks */
      $G \leftarrow G \cup$ block.getNeighborhood_Pixel(block, g_c) /* G is the set of neighbourhood pixels */
      while $G.length() \neq 0$ **do**
        $k \leftarrow (g_p - g_c)$ /* g_p and g_c are gray levels for neighbourhood and center pixel */
        if $k \geq 0$ **then**
          $BV \leftarrow 1$ /* BV is the binary value */
        end
        else
          $BV \leftarrow 0$
        end
        /* calculate LBP Value */
        LBP $\leftarrow$ LBP $+ BV \times 2^p$
      end
      $LV \leftarrow LV \cup$ LBP /* LV is a 2D array containing the corresponding LBP value for each pixel */
    end
  end
  while $LV.length() \neq 0$ **do**
    /* Create LBP image */
    $L_i \leftarrow$ img.show(w, h, LV)
  end
  $L^* \leftarrow L^* \cup \{L_i\}$
end

3.3.1 LBP image generation

In 1994, T. Ojala introduced the concept of local texture features [38]. LBP is a robust technique for extracting texture features from grayscale images and exhibits resilience to object rotation. LBP generates a sequence of binary bits for a given image by applying a thresholding operation. In a 2D pixel matrix, the center pixel is the threshold against its surrounding pixels in a circular neighborhood. A value less than the center pixel is encoded as zero (0), while any other value is encoded as one (1). The resulting binary sequence is read in a clockwise direction (as illustrated in Fig. 3). Finally, we convert the sequence of bits into a decimal value, representing the LBP code for the center pixel. The grayscale textural characteristic, T, can be considered as the joint spread of gray shades for p neighborhood pixels represented as:

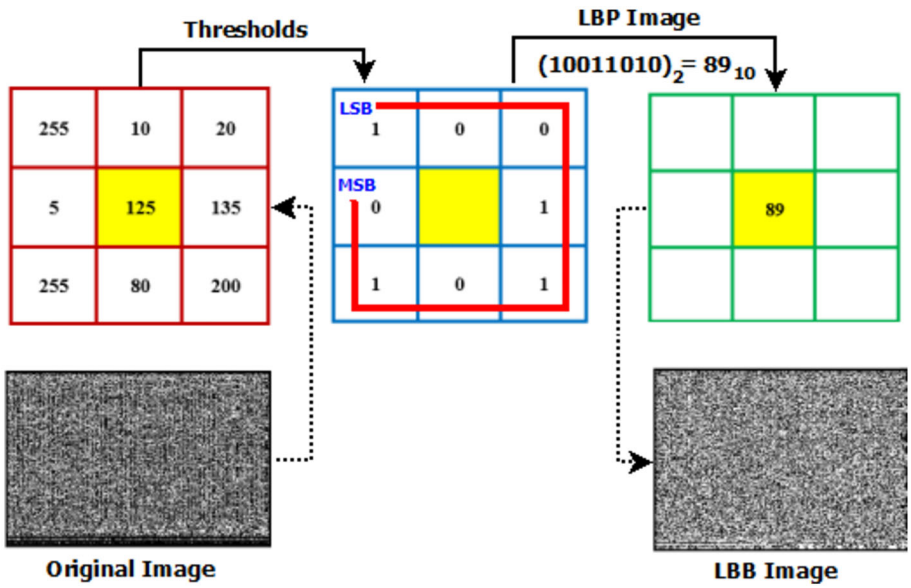


Fig. 3 Generation of LBP Image

$$T = \tau\ (g_c, g_0, g_1 \ldots, g_{p-1}) \tag{1}$$

Where p represents the count of a circularly symmetric set of neighboring pixels (i.e., p=0,1,2...7 for a 3 × 3 block) for a ring of radius r $(r > 0)$, g_p is the p^{th} neighborhood pixel, with g_c as the center pixel. To determine T, we subtract the central pixel's grey level from the surrounding pixels' grey level.

$$T = \tau(g_0 - g_c, g_1 - g_c \ldots, g_{p-1} - g_c) \tag{2}$$

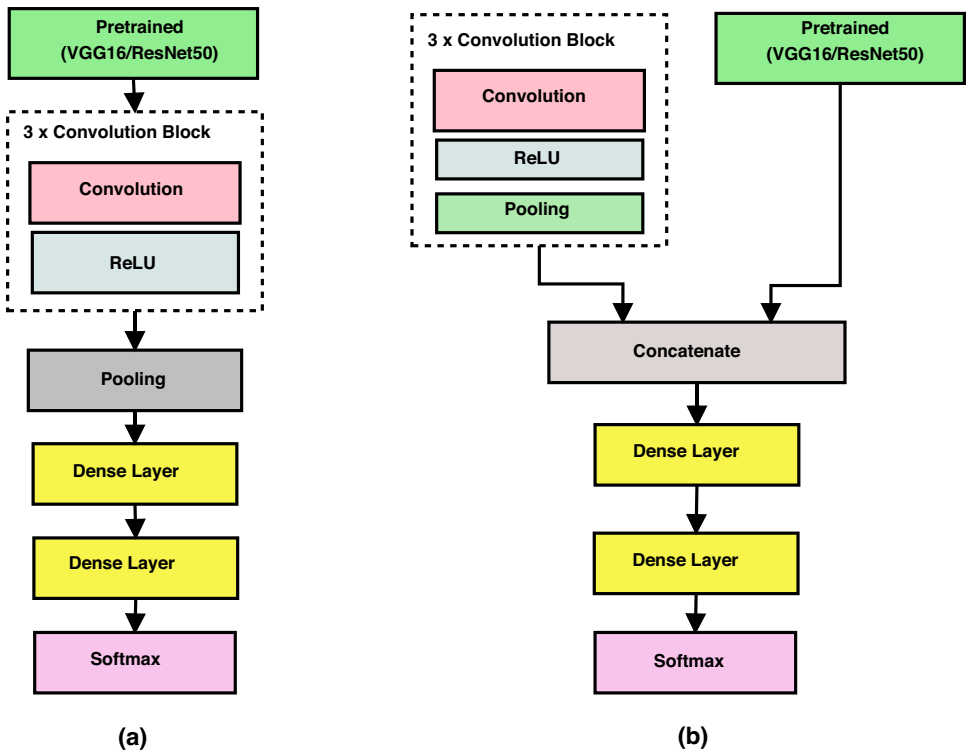


Fig. 4 Deep learning model architecture for malware family classification (proposed model) (**a**) It comprises ResNet50 followed by three convolutional layers with ReLU and a pooling layer, two dense layers, and a final output layer with a Softmax activation function. **b** The setup consists of three convolutional blocks incorporating ReLU activation and pooling layers, connected with ResNet50. Additionally, it includes two dense layers, culminating in a final output layer featuring a Softmax activation function

$$LBP_{p,r} = \sum_{p=1}^{\#neighborhood} S(g_p - g_c)2^{p-1} \quad (3)$$

$$S(x) = \begin{cases} 1 & if\ x \geq 0 \\ 0 & otherwise \end{cases} \quad (4)$$

$S(x)$ is the binary threshold function. Figure 3 depicts the LBP operation with eight neighboring pixels. Here, we consider the center pixel value, which is 125, as the threshold. The neighbor pixels are assigned 1 if greater than or equal to 125 and otherwise 0. The LBP value is calculated as $1 * 2^0 + 0 * 2^1 + 0 * 2^2 + 1 * 2^3 + 1 * 2^4 + 0 * 2^5 + 1 * 2^6 + 0 * 2^7 = 89$.

We create LBP images in both 64x64 and 128x128 sizes. For each pixel in the target image, we calculate the LBP value using equations 3 and 4. Algorithm 2 outlines the entire process of LBP image generation.

3.4 Classification model

In our strategy for visualizing malware, we leverage a sophisticated deep neural network architecture. This architecture incorporates a pre-trained base model, as demonstrated in Fig. 4. LBP and non-LBP images developed in the feature extraction phase are input to a pre-trained model [39].

3.4.1 ResNet50

In visualization-based classification, researchers have widely utilized Convolutional Neural Networks (CNN) [40] because CNNs can automatically extract distinctive local features from visual representations. The convolutional layers serve as filters in detection, identifying particular patterns within images. Within a CNN, the initial layers extract basic features, while deeper layers progressively capture more complex features. Towards the end of the CNN architecture, fully connected layers combine all the specific attributes from the preceding layers to produce the final output.

Deeper neural networks are incapable of increasing performance as the depth of the network increases. The accuracy begins to saturate, and performance degrades, known as the vanishing gradient problem. To address this challenge, we adopted ResNet50, a deep convolutional neural network that incorporates shortcut/skip connections between convolutional layer blocks, utilizing the ReLU activation function in the subsequent layer. The ResNet50 architecture incorporates a bottleneck residual block that includes three convolutional layers (1×1, 3×3, and 1×1 convolutions) with 3×3 filters, allowing the model to acquire the capability to learn an identity function. The 1×1 layer is in charge of shrinking and then expanding

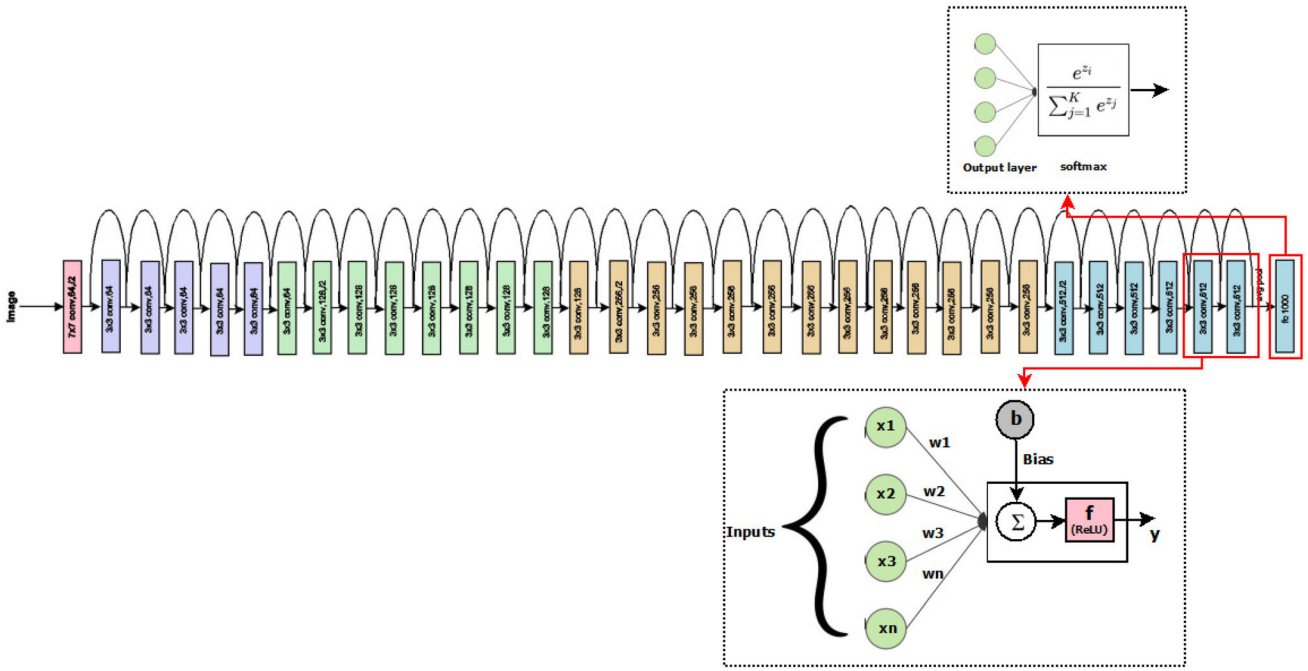


Fig. 5 The ResNet50 architecture incorporates residual blocks, each composed of two convolutional layers, followed by a batch normalization layer and ReLU activation. ResNet skips the residual block and forwards the input directly before the final ReLU activation function through the residual connections

Table 4 Dropout and activation for the layers

Layer	Dropout	Activation	Output
Global Average Pooling	0.2	NA	NA
Dense layer 1	0.2	ReLU	256
Dense layer 2	0.2	ReLU	128
Output layer	NA	Softmax	The number of classes (9 for BIG2015 dataset 22 for MalImg dataset and 20 for Malhub dataset)

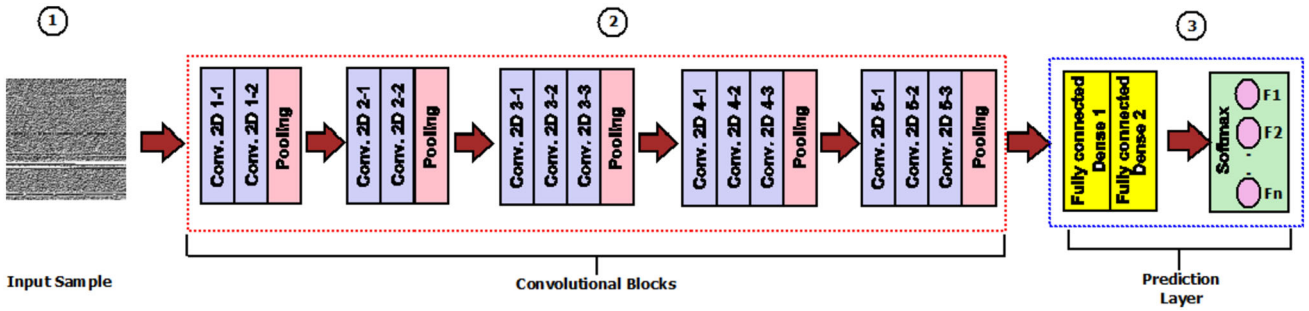


Fig. 6 VGG16 feature extraction illustration: (1) Depicts the input of the VGG16. (2) The convolutional blocks along with pooling (3) Prediction layer include flattening the output of final convolution block followed by Softmax layer

dimensions, leaving the 3 × 3 layer as a bottleneck with reduced parameters of input or output. Convolutional layers with stride length 2 perform downsampling and batch normalization before applying ReLU. The design directives for the residual blocks are as follows: (1) allocate equal filters to produce output feature maps of comparable sizes. (2) Double the filters as the feature map's size is reduced by half. The network has 50 weighted layers with a global average pooling layer and a final Softmax layer (Fig. 5).

ResNet50 is a transfer learning [41] approach to improve neural network performance by transferring the learned knowledge from a similar task. The initial layer discovers minute details of images, while the deeper layer extracts generic byte patterns representative of the family. ResNet50 effectively empowers training on substantially smaller target datasets without encountering overfitting issues.

We evaluated the performance of ResNet50 by presenting its output to (a) a convolutional neural network with a pooling layer with two dense layers and (b) a fused model as represented in Fig. 4, three convolutional blocks with pooling concatenated with ResNet50 pre-trained model and followed by two dense layers and a final output layer, and (c) various ML algorithms like SVM, RF, and XGBoost each of the cases above assessed using images of dimensions (64x64) and (128x128) respectively. We present the set of optimal hyperparameters selected during the learning process in Table 4.

3.4.2 VGG16

Leveraging the pre-trained VGG16 deep CNN and its 16-layer architecture, we conducted an investigation using the network's weights and the architecture proposed in Fig. 4 in conjunction with various machine learning algorithms. Figure 6 demonstrates how the VGG16 network is employed for malware classification.

Understanding grayscale images generated from malware PE files poses a challenge, but CNNs, known for their effectiveness in image-based classification, offer a robust solution. Notably, established CNN architectures like ResNet50 and VGG16 efficiently extract crucial features, saving time and resources. The selection of these architectures is motivated by their ability to leverage pre-trained weights and biases from the ImageNet database, simplifying deployment without retraining on specific datasets. Despite differences between natural and malware byte-plot images, transferring VGG16 parameters enhances malware image recognition effectiveness through transfer learning. The preference for VGG16 in this study stems from its ability to handle large-scale datasets through deeper

network layers and smaller filters, contributing to enhanced performance.

Similarly, ResNet-50, recognized for its excellence in the ImageNet classification challenge, achieves high accuracy in transferring knowledge to malware images. ResNet-50 facilitates training networks with thousands of layers previously deemed impractical. ResNet-50 achieves this by using residual blocks and skip connections, ensuring the preservation of information from earlier layers. This architectural choice is motivated by the desire to address challenges like the vanishing gradient problem and enable the training of extremely deep neural networks. Thus, the network learns more accurate representations of input data, leading to improved predictions. The researchers in [39] addressed the vanishing gradient problem by using extra connections between non-contiguous convolutional layers.

3.5 Attacks on machine learning algorithms

Researchers have reported that malware with fingerprinting abilities can evade detection [42]. Deep Neural Network algorithms are vulnerable to different types of attacks, and based on the adversary's goal, it can be generally categorized as a poisoning attack or an evasion attack [43]. The former is an attack that permits an attacker to inject or recast spurious samples into a training set. These fictitious examples may cause the trained classifier to predict incorrect labels, increasing misclassification. Such attacks are common and evolve if an adversary accesses the training data. In the latter case, the opponents are not permitted to alter the classifier or its parameters but provide some false examples to avoid detection. Attacks are created by adding minimal perturbation to an input indistinguishable from a human, forcing the model to make incorrect decisions. The commonly used attack model for creating adversarial examples is known as the additive model [44]. Here, an attacker utilizes a linear operator that adds perturbation to the input to deceive the classifier.

The knowledge gained about the target system (i.e., training data, feature set, learning algorithms, parameters, etc.) categorizes the attacks into three types: (1) white box attacks, (2) black box attacks, and (3) gray box attacks. In the white-box method, the adversary operates under the assumption of having full access to the model's algorithm and parameters. Conversely, in the black-box method, the adversary lacks knowledge of the target model's parameters and architecture and resorts to querying the model to initiate an attack. Unlike other approaches, gray-box attacks only obtain access to the model during the training phase by generating adversarial samples using a generative model[45]. Evaluating the resilience of classification models to adversarial attacks motivated our use of a white-box attack approach in this study. To demonstrate the model's robustness against adversarial attacks, we crafted tainted binaries by introducing various types of noise: Gaussian, Speckle, Salt and Pepper, Local Variable, and Poisson noise.

Let us decide classification model with decision function $f : M \rightarrow Y$, that assigns each sample in the set $M = \{m_1, m_2, ...m_n\}$ to a label in a set $Y = \{y_1, y_2...y_n\}$, where y_i denotes the true class of a malware. We train the model on a dataset $D = \{m_i, y_i\}_{i=1}^{n}$ where m_i, y_i represents the image of the malicious sample and corresponding true label respectively. During training, the parameter θ is determined which minimizes the loss ($\mathcal{L}$) where,

$$\mathcal{L}_{min}(g(m_i, \theta), y_i) \tag{5}$$

$g(m_i, \theta)$ is the prediction of the input m_i.

A set $\tilde{D} = \{\tilde{m_i}, y_i\}_{i=1}^{n_s}$ is consisting of adversarial examples generated using Eq. 6 for n_s malware images which are drawn from the test set. Thus, we create the adversarial samples using Eq. 6.

$$\tilde{m_i} = m_i + \delta \tag{6}$$

where $\tilde{m_i}$ is the tainted version of m_i. δ is the noise (perturbation) added to m_i. The intrinsic restriction lies in the minimum allowable perturbation, denoted as $d(m_i, \tilde{m_i})$, between m_i and $m_i^{\sim}$ that causes misclassification while preserving semantic integrity (i.e., if m exhibits maliciousness, then $\tilde{m}$ should also exhibit maliciousness). In an untargeted attack (refer Eq. 7), our goal is to induce the model to misclassify the sample into an incorrect class. We then feed the adversarial samples back into the model to estimate the target class.

$$g(m_i + \delta, \theta) = \tilde{y_i}, \quad \tilde{y_i} \neq y_i \tag{7}$$

where $\tilde{y_i}$ is the prediction for tainted sample $\tilde{m_i}$. We used the *random_noise()* function from the "skimage" Python library [5] to generate adversarial malware images. The arguments of the function are: (i) `image`: n-dimensional input image (ii) `mode`: type of noise (iii) `seed`: random seed (iv) `clip`: to maintain proper range for image data. In the following paragraphs, we present the different types of noise used in this experiment.

(a) **Gaussian noise:** This statistical noise exhibits a probability density function equivalent to the normal distribution. The probability distribution function of a Gaussian distribution is a bell-shaped, represented as:

$$n(g) = \sqrt{\frac{1}{2\pi\sigma^2}} e^{-\frac{(g-\mu)^2}{2\sigma^2}} \tag{8}$$

[5] scikit-image: https://scikit-image.org/

where g is the gray scale value, σ is standard deviation and μ represents the mean respectively.

(b) **Speckle noise:** It is a type of granular noise that occurs naturally in images and lowers their quality. Multiplying arbitrary pixel values with distinct pixels in a picture introduces the Speckle noise in the image. The probability density function of speckle noise is as follows:

$$f(g) = \frac{g^{\alpha-1} e^{\frac{-g}{a}}}{(\alpha-1)! a^{\alpha}} \tag{9}$$

where g is the pixel's gray level and a^{α} is the variance.

(c) **Salt and Pepper noise:** It is applied only to grayscale images where "salt" represents white pixels, and "pepper" denotes black pixels. This involves distributing random bright pixels (representing a value of 255) and random dark pixels (representing a value of 0) throughout the entire representation. This model earns the nickname "data drop noise" because it statistically reduces the original values.

$$I(x,y) = \begin{cases} 0 & \text{Pepper noise} \\ 255 & \text{Salt noise} \end{cases} \tag{10}$$

where (x, y) are a pixel's co-ordinates.

(d) **Local Variable:** It is a type of Gaussian-distributed additive noise or zero-mean Gaussian white noise. Here, we define local variance in terms of pixel intensity values at each point of an image.

(e) **Poisson noise:** The Poisson or Shot noise appears in images due to the statistical nature of x-rays, visible lights, etc., and follows the Poisson distribution as in Eq. 11.

$$p(k) = \frac{\lambda^k e^{-\lambda}}{k!} \tag{11}$$

where λ and k represent the mean and the expected value, respectively.

4 Experiments and results

This section discusses our dataset, assessment criteria, and outcomes. To substantiate the efficacy of the proposed deep learning model, we performed a comprehensive evaluation as listed below:

1. *The performance of various machine learning and deep learning models on the benchmark and self-made datasets (i.e., Malhub)*
2. *Evaluation of proposed model on obfuscated samples*
3. *Evaluation of proposed model on evasion attack*
4. *A comparative analysis of the proposed system with the state-of-the-art approaches*

Table 5 Hyper parameters for the model tuned using KerasTuner

Model	Learning rate	Epochs
MalImg—64 × 64	0.05	40
MalImg—128 × 128	0.05	40
Big 2015—64 × 64	0.05	40
Big 2015—128 × 128	0.05	40
Malhub—64 × 64	0.001	10
Malhub—128 × 128	0.001	10

4.1 Dataset description

In this paper, we assess the performance of the proposed model using two publicly available benchmark datasets: the MalImg dataset [18], which includes 9,342 malware samples spanning 25 different families, and BIG2015 [37], containing 10,868 malicious samples categorized into nine families. Furthermore, we evaluate the performance of the proposed model on the Malhub dataset, which consists of 26,452 samples across 20 families. In each experiment, we allocated 70% of the samples for the training set, 20% for the validation set, and reserved the remaining 10% for the testing phase. The work's implementation source code was crafted using Python, employing Keras−2.4.3 alongside TensorFlow−2.4.1 as the backend. As the pre-trained model requires a fixed-size input image, we resized each file to different dimensions, specifically 64 x 64 and 128 x 128 pixels. Malware binaries have variable lengths, and transforming large binaries into small images may discard essential information. Conversely, expanding small binaries into large images may introduce unnecessary padding, potentially affecting detection accuracy. We used *RandomizedSearchCV*[6] and *KerasTuner*[7] as hyperparameter tuners for ML-classifiers and deep learning, respectively (refer to Table 5). The XGBoost parameters include *n_estimator*, representing the number of boosting rounds or trees, and *random_state*, which initializes the random number generator with a specific seed value. In SVM parameters, *c* regulates the balance between minimizing training and testing errors. At the same time, *cache_size* sets the kernel cache size to enhance training speed. The *decision_function_shape* determines the strategy for multi-class classification, with *ovr* indicating the use of separate

[6] https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.RandomizedSearchCV.html

[7] https://keras.io/keras_tuner/

Table 6 List of the experiments

Pre-trained model	SVM	RF	XGBoost	Dense (2-layers)	CNN+Dense Layers	Fused model
VGG16	✔	✔	✔	✔	✔	✔
ResNet50	✔	✔	✔	✔	✔	✔

binary classifiers for each class against the rest. Other SVM parameters, such as *degree*, *gamma*, *kernel*, *shrinking*, and *tol*, control various aspects of the model's behavior and convergence. RF parameters include *criterion*, *min_samples_leaf*, *min_samples_split*, *n_estimators*, and *random_state*, influencing decision tree construction and randomness.

We evaluated the model's performance by employing metrics like F1 Score (F1), Accuracy (A), Precision (P), and Recall (R). The aforementioned metrics are derived using the following:

- True Positive(TP): indicates that the model has correctly identified an instance as belonging to the class it genuinely belongs to.
- False Positive(FP): True class is negative, and predicted class is positive
- False Negative (FN): True class is positive, and predicted class is negative
- True Negative (TN): is an instance that is correctly classified as not belonging to a particular class. It indicates that the model has correctly identified an instance as belonging to the class it does not genuinely belong to.

Accuracy: It is the fraction of samples predicted correctly.

$$Accuracy(A) = \frac{TP + TN}{TP + TN + FP + FN} \tag{12}$$

Precision: It is the fraction of predicted positive events that are actually positive.

$$Precision(P) = \frac{TP}{TP + FP} \tag{13}$$

Recall: It is the fraction of positive events that are predicted correctly.

$$Recall(R) = \frac{TP}{TP + FN} \tag{14}$$

F1 score: It is the harmonic mean of precision and recall.

$$F1score(F1) = 2 * \frac{P * R}{P + R} \tag{15}$$

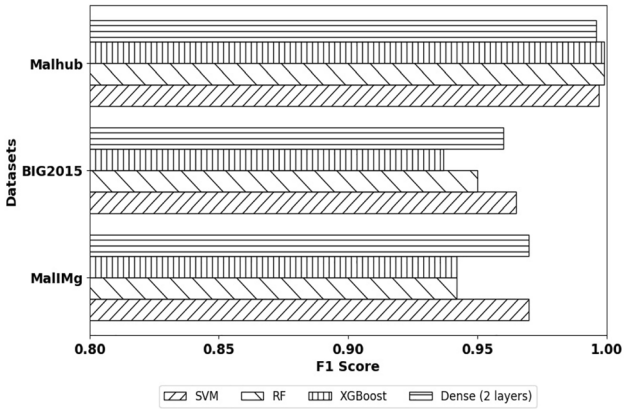


Fig. 7 Performance of ML classifiers and Dense(2 layers) using LBP images VGG16 as pre-trained network (size = 128 x 128)

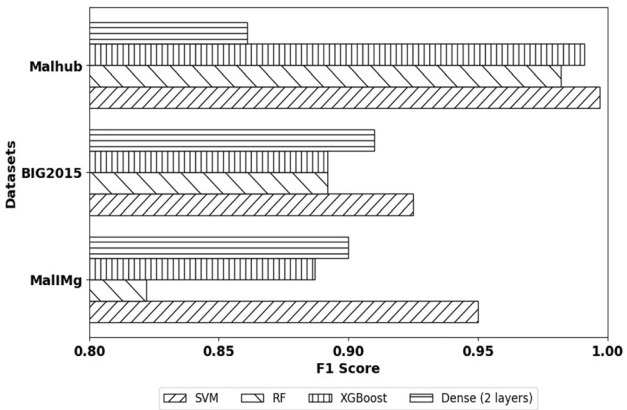


Fig. 8 Machine learning classifiers and Dense(2 layers) performance utilizing LBP images with ResNet50 pre-trained Network (128x128)

4.2 The performance of classification models on benchmark datasets

To verify the effectiveness of the proposed deep learning model in classifying malicious samples into different families, we experimented with different combinations of classifiers. Table 6 presents the list of experiments conducted for evaluation, we present F1 score for all the

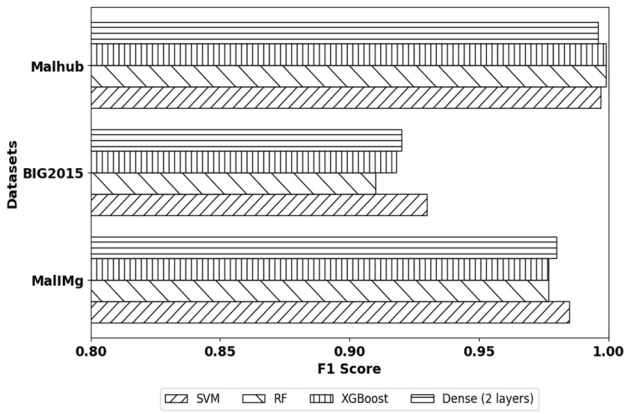


Fig. 9 Evaluation of Non-LBP images (Size: 128 x 128) using ML classifiers and dense (2 Layers) with pre-trained VGG16

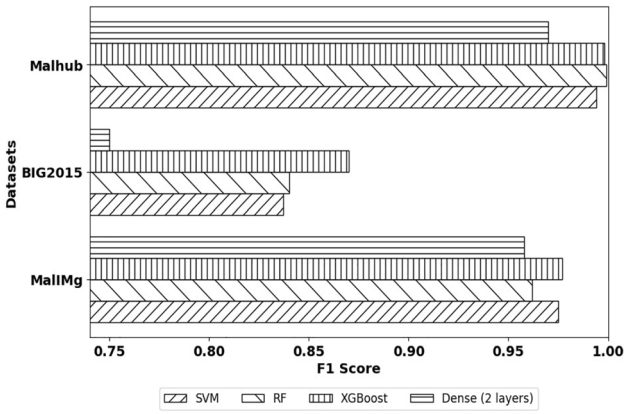


Fig. 10 The performance of machine learning classifiers and dense(2 layers)utilizing pre-trained ResNet50 with non-LBP images (size = 128 x 128)

models (due to unbalanced nature of the dataset). First, we conducted experiments to evaluate the capability of different conventional ML algorithms in classifying malware into associated classes. Figure 7 and Fig. 8 present the performance of machine learning classifiers using LBP images (detailed results are presented in the Appendix Section A, more specifically in Table 15 and Table 16). We ran the experiments a total of five times and then derived the average results. As seen in Fig. 7, with the VGG16 and MalImag dataset, we obtain an F1 score in the range of 0.942−0.97. An identical trend in results is obtained for the BIG2015 dataset. All classifiers achieved an F1 score above 0.99 for files in the Malhub dataset. On the contrary, combining ResNet50 with the machine learning classifier for MalImg resulted in F1 score in the range of 0.822−0.95 as can be seen in Fig. 8. For BIG2015, F1 score is between 0.892−0.925, and for Malhub F1 score is in the range of 0.861−0.997.

In addition to LBP images, we also experimented with Non-LBP images (original images). From Fig. 9, VGG16+SVM provides an F1 score of 0.985 (MalImg), 0.93 (BIG 2015) and 0.999 (Malhub) respectively. Further, using the ResNet50 model the F1 score is between 0.75 to 0.998(Refer Fig. 10). These experiments suggest that the appropriate choice of classification algorithms during the inference phase can remarkably influence the results. Table 17 and Table 18, respectively, report the performance measures in terms of accuracy. Table 7 presents the configurations for ML classifiers in the experiment.

4.3 Performance evaluation of the proposed models

We developed various classification models to evaluate whether particular variations of the model demonstrate better performance compared to those discussed previously in Sect. 4.2. Each model was trained and tested independently on size 64x64 and 128x128 malware images. In particular, we created combinations like (a) ResNet50+CNN+Dense layers, (b) VGG16+CNN+Dense layers, and (c) fusion of models (ResNet50||CNN||Dense layers and VGG16||CNN||Dense layers). The results obtained with ResNet50+CNN+Dense Layers for LBP and Non-LBP images are reported in Table 8 and Table 9 respectively. Table 10 presents the

Table 7 Hyper-parameters for ML classifiers

Dataset	Hyper-parameters		
	XGBoost	SVM	RF
MalImg BIG2015 Malhub	n_estimators=50 degree: 3	c: 1.0 cache_size: 200 decision_function_shape: ovr random_state: 42 gamma: scale shrinking': True kernel: linear tol: 0.001	random_state: 42 min_samples_leaf: 1 n_estimators: 50 min_samples_split: 2 criterion: gini

Table 8 Performance of ResNet50+CNN+DL on benchmark datasets with LBP images

Dataset	Image size	A	P	R	F1	Execution time (s)
Big 2015	64x64	0.995	0.996	0.995	0.995	0.0009
Big 2015	128x128	0.989	0.989	0.989	0.989	0.001
MalImg	64x64	0.974	0.965	0.964	0.96	0.001
MalImg (25 class)	128x128	0.980	0.980	0.980	0.979	0.0015
MalImg (22-class)	128x128	0.998	0.998	0.998	0.998	0.015
Malhub	64x64	0.997	0.997	0.997	0.997	0.0005
Malhub	128x128	0.992	0.993	0.992	0.993	0.0016

Table 9 F1 score of proposed model (ResNet50+CNN+DL) with Non-LBP images

Dataset	Image dimension			
	64 × 64		128 × 128	
	A	F1	A	F1
BIG2015	0.959	0.959	0.961	0.957
MalImg	0.965	0.964	0.90	0.902
Malhub	0.995	0.995	0.996	0.996

Table 10 The F1 score of VGG16+CNN+DL model on different datasets

Dataset	LBP	Non-LBP
BIG2015 (64 × 64)	0.938	0.911
MalImg (128 × 128) (25 class)	0.972	0.908
MalImg (128 × 128) (22 class)	0.982	0.994

results obtained for VGG16+CNN+Dense Layers using LBP and Non-LBP images. We obtained less F1 score with VGG16 model trained using Malhub dataset. Similar trends in the results are also reported by authors in [29].

Figure 11 depicts that six of nine families achieved perfect classification with an F1 score of 1.0 for the BIG2015 dataset. The model misclassified two samples of Lollipop as Gatak and two samples of Tracur as Obfuscator.ACY and Vundo. The model also incorrectly classified one sample of Kelihos_Ver1 as Obfuscator.ACY. Furthermore, 21 of the 25 malware families in the MalImg dataset achieved accurate classification with an F1 score of 1.0. Among the 25 families, four (C2LOP.gen!g, C2LOP.P, Swizzor.gen!, Swizzor.gen!E) exhibit identical in behavior as reported in [18]. Therefore, these families are combined into a single class and retrained the model using 128x128 sized images. When examining the confusion matrix in Fig. 13, it is evident that the model successfully classified 21 out of 22 families, with only one misclassification, i.e., Wintrim.BX wrongly identified as Allaple.A. From Table 9, the F1 score of the Non-LBP image is 0.90 for the MalImg dataset (128x128) and 0.957 (128x128) for the BIG2015 (128x128) dataset, respectively. At the same time, for the LBP image, it is 0.998 for MalImg (22 classes) and 0.995 for the BIG2015

Table 11 F1 score of proposed model(ResNet50||CNN||DL) on LBP images (fusion model)

Dataset	A	P	R	F1	Execution time (s)
BIG2015(64x64)	0.943	0.918	0.88	0.900	0.0009
BIG2015(128x128)	0.949	0.0010	0.873	0.949	0.001
MalImg(64x64)	0.987	0.983	0.979	0.981	0.001
MalImg(128x128)	0.995	0.993	0.992	0.993	0.0015
Malhub(64x64)	0.997	0.997	0.997	0.997	0.0005
Malhub(128x128)	0.999	0.999	0.999	0.999	0.0010

Table 12 Evaluation results of the adversarial attack on proposed ResNet50+CNN+DL model

Noise type	BIG2015		MalImg		Malhub	
	A	F1	A	F1	A	F1
Gaussian	0.995	0.995	0.974	0.97	0.992	0.992
Local variable	0.995	0.995	0.961	0.957	0.47	0.58
Poisson	0.995	0.995	0.986	0.982	0.254	0.241
Salt	0.995	0.995	0.972	0.966	0.254	0.243
Pepper	0.995	0.995	0.972	0.966	0.25	0.243
S & P	0.995	0.995	0.972	0.966	0.12	0.12
Speckle	0.995	0.995	0.972	0.966	0.12	0.12

Table 13 Performance of ResNet50||CNN||DL model with adversarial examples

Noise type	BIG2015		MalImg		Malhub	
	A	F1	A	F1	A	F1
Gaussian	0.947	0.876	0.996	0.993	0.992	0.992
Local Variable	0.946	0.874	0.995	0.993	0.556	0.624
Poisson	0.949	0.878	0.995	0.993	0.556	0.625
Salt	0.944	0.874	0.996	0.993	0.32	0.32
Pepper	0.945	0.875	0.995	0.992	0.32	0.32
S & P	0.946	0.874	0.995	0.993	0.014	0.05
Speckle	0.948	0.877	0.995	0.993	0.053	0.014

Table 14 Comparative analysis with state-of-the-art approaches (BIG2015 and MalImg datasets)

No.	Author	Year	citation	Algorithm	Dataset	Performance	
						Acc	F1
1	Nataraj et al.[18]	2011	1290	KNN	MalImg	0.98	-
2	Cui et al.[23]	2018	616	CNN	MalImg	0.945	-
3	Le et al.[33]	2018	200	CNN-BiLSTM	BIG2015	0.982	0.960
4	Agarap et al.[48]	2019	91	GRU-SVM	MalImg	0.84	0.85
5	Gibert et al.[24]	2019	192	CNN	MalImg	0.984	0.974
					BIG2015	0.984	0.94
6	Yifei et al.[49]	2021	43	SEResNet50+Bi-LSTM	BIG2015	0.983	0.983
7	Wang et al. [31]	2021	30	DenseNet	BIG2015	0.973	0.954
8	Deng et al.[50]	2023	5	CNN	BIG2015	0.994	0.992
9	Alzubi et al.[32]	2023	27	LSTM+GRU	BIG2015	0.982	0.988
10	Proposed method			ResNet50+CNN+DL	MalImg	0.998	0.995
				ResNet50+CNN+DL	BIG2015	0.999	0.998
				ResNet50||CNN||DL	Malhub	0.995	0.999

Table 15 ML classifiers and Dense(2-layers) performance using LBP images VGG16 as pre-trained network (size = 128 x 128)

Dataset	VGG16							
	SVM		RF		XGBoost		Dense (2 layers)	
	A	F1	A	F1	A	F1	A	F1
MalIMg	0.969	0.97	0.945	0.942	0.942	0.942	0.97	0.97
BIG2015	0.968	0.965	0.946	0.95	0.932	0.937	0.966	0.96
Malhub	0.997	0.997	0.999	0.999	0.999	0.999	0.996	0.996

Table 16 Performance of ML classifiers and Dense(2-layers)using LBP images ResNet50 as pretrained network (size = 128 x 128)

Dataset	ResNet50							
	SVM		RF		XGBoost		Dense (2 layers)	
	A	F1	A	F1	A	F1	A	F1
MalIMg	0.945	0.95	0.825	0.822	0.885	0.887	0.909	0.900
BIG2015	0.92	0.925	0.885	0.892	0.902	0.892	0.918	0.91
Malhub	0.997	0.997	0.982	0.982	0.991	0.991	0.854	0.861

Table 17 Evaluation results of ML classifiers and Dense(2-layers) using Non-LBP images (size = 128 x 128) using pretrained VGG16

Dataset	VGG16							
	SVM		RF		XGBoost		Dense (2 layers)	
	A	F1	A	F1	A	F1	A	F1
MalIMg	0.98	0.985	0.974	0.977	0.972	0.977	0.984	0.98
BIG2015	0.92	0.93	0.90	0.91	0.912	0.918	0.92	0.92
Malhub	0.997	0.997	0.999	0.999	0.999	0.999	0.996	0.996

Table 18 Assesment of ML classifiers and Dense(2-layers) using Non-LBP images (size = 128 x 128) using pretrained ResNet50

Dataset	ResNet50							
	SVM		RF		XGBoost		Dense (2 layers)	
	A	F1	A	F1	A	F1	A	F1
MalIMg	0.977	0.975	0.96	0.962	0.972	0.977	0.959	0.958
BIG2015	0.83	0.837	0.84	0.84	0.87	0.87	0.759	0.75
Malhub	0.995	0.994	0.999	0.999	0.998	0.998	0.972	0.97

dataset. The Malhub dataset demonstrated improved performance with LBP and Non-LBP images, achieving an F1 score of 0.996 and above. LBP images exhibited superior performance at a dimension of 64x64, while non-LBP images performed better at 128x128. Thus, it is evident that LBP images capture textural patterns specific to a malware family, resulting in an improved classifier outcome.

To further enhance the classification performance, we devised an alternative model that integrated the outputs of ResNet50 and CNN, utilizing dense layers to distinguish malware images across three datasets. The findings are summarized in Table 11. Consistent with prior experiments, the analysis was conducted employing variable image dimensions. Confusion matrices are depicted in Fig. 14, Fig. 15 and Fig. 16. Both models, model1 (ResNet 50+CNN+DL) and model2 (ResNet50||CNN||DL), demonstrated comparable F1 scores for MalImg and Malhub datasets across image dimensions 64x64 and 128x128. This suggests that both models can effectively distinguish malware images, regardless of size. On the contrary for BIG2015, model1 (ResNet50+CNN+DL) exhibited the highest F1 score of 0.995, and model2 (ResNet50||CNN||DL) obtained an F1 score of 0.949. Seven out of nine malware families in the BIG2015 dataset exhibited a detection rate exceeding 90%, with the exception of Simda (shown in Fig. 14). This disparity is likely attributed to the dataset's imbalanced nature, as Simda only contains 42 samples, accounting for a mere 0.386% of the entire dataset. Nonetheless, we believe that further improvement can be achieved by leveraging generative models like Generative Adversarial Networks (GANs) to augment and balance the classes with fewer

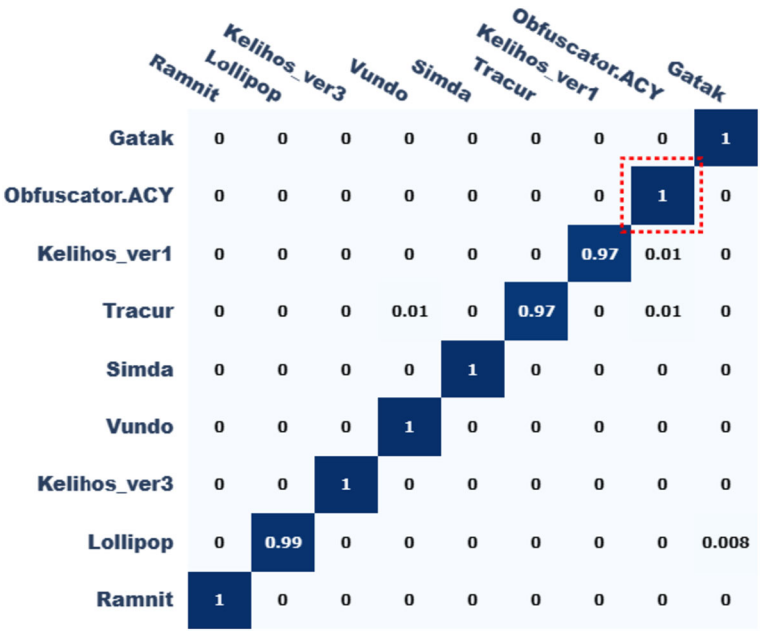


Fig. 11 BIG2015 dataset: Confusion matrix for 64x64 images obtained with ResNet50+CNN+DL model. The two Lollipop samples were misclassified as Gatak, one sample of Tracur is misclassified as Obfuscator.ACY and Vundo each, and one sample of Kelihos_Ver1 is misclassified as Obfuscator.ACY respectively

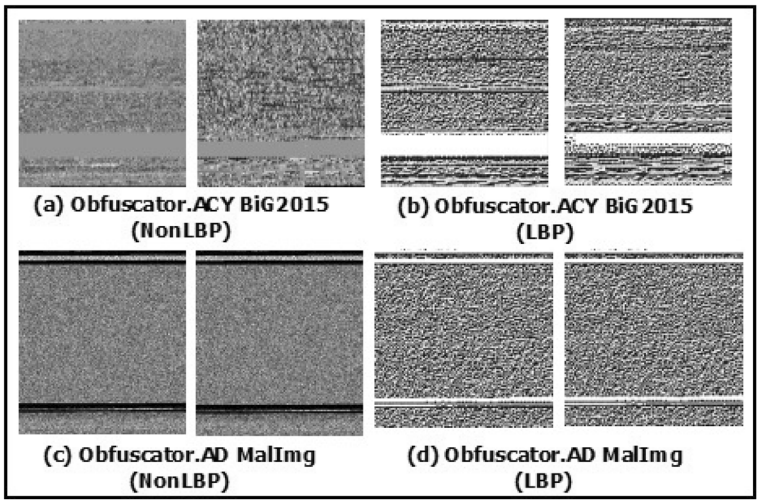


Fig. 12 Sample images of obfuscated samples: **a** non-lbp images of Obfuscator.ACY family of BIG2015 dataset (**b**) corresponding LBP images of Obfuscator.ACY class of BIG2015. **c** sample Non-LBP images of Obfuscator.AD of MalImg dataset. **d** LBP images of Obfuscator. AD samples from MalImg

samples in the training set. Figure 15 illustrates that an impressive 17 out of 22 malware families achieved a remarkable 100% detection rate. For the remaining five classes, the detection rates were still substantial, ranging from 93.7% to 99.8%. Moreover, we can observe from Fig. 16 that 13 out of 20 families depicted 100% detection rate, and for the remaining seven families, detection rates are between 99.5% and 99.7%. This demonstrates the remarkable efficacy of our proposed model in distinguishing malware families based on their visual features. Details of ROC curves (refer Figures Fig. 18, Fig. 19 and 20) for best-performing models are shown in Appendix A.

4.4 Evaluation of proposed model on obfuscated samples

Software obfuscation encompasses altering the arrangement of a program's code while upholding its intended behavior. This technique employs various tactics, including unconditional jumps, control flow interlacing, conditional

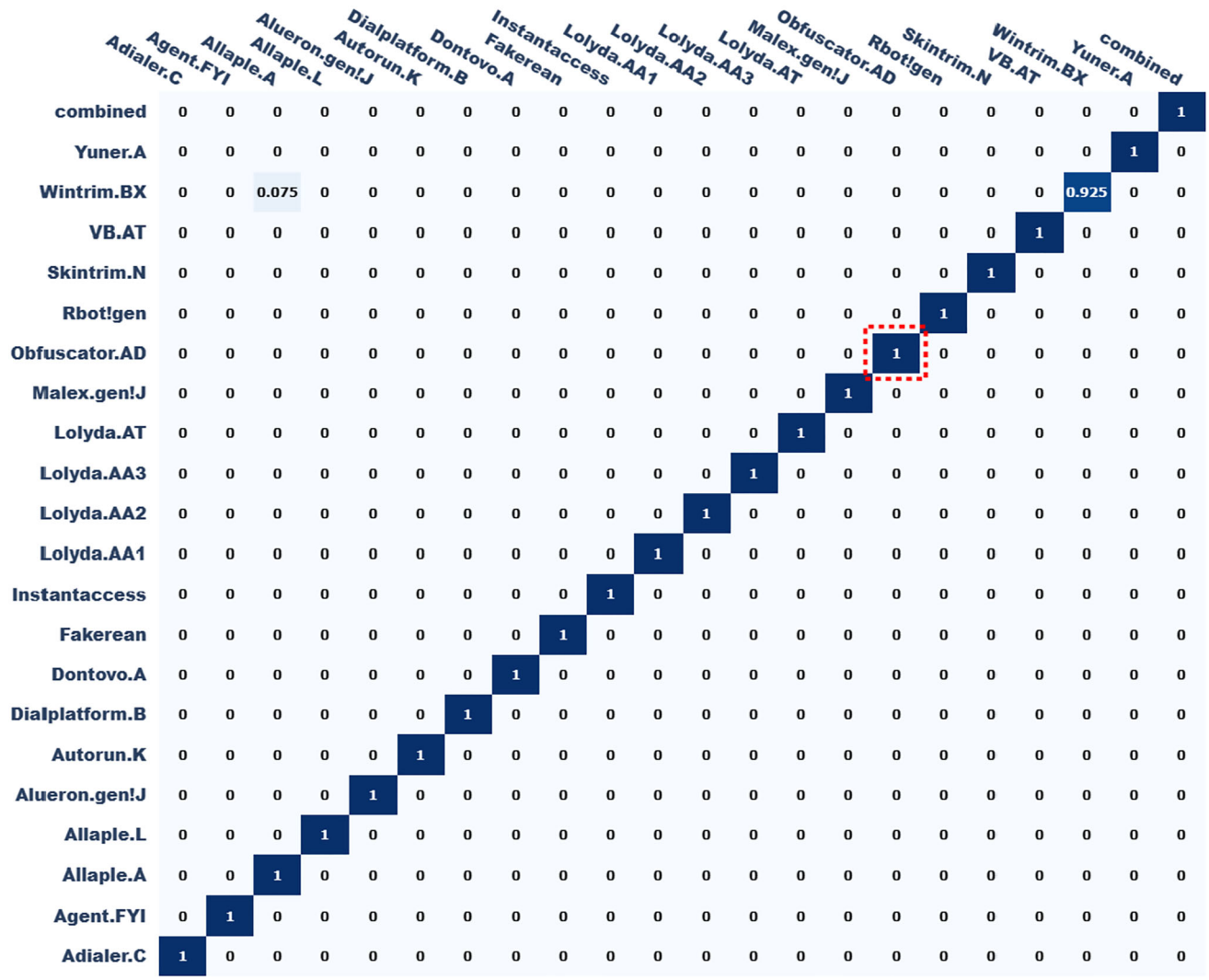


Fig. 13 MalImg dataset (22 class): Confusion matrix for 128 × 128 sized images after merging similar classes

jumps, transparent branching, merging local integers, introducing randomized dead code, reassigning variables, generating fictitious intermediate-level code, encoding strings, and suppressing constants [46]. These obfuscation methods hinder the comprehension and reverse engineering of the program's functionality. According to statistics, approximately 80% of malware executables cleverly employ obfuscation techniques to escape from detection [47]. Specifically, we concentrated on identifying obfuscated malware from the benchmark dataset. Notably, Obfuscator.ACY and Obfuscator.AD families of BIG2015 and MalImg datasets harbor such obfuscated executable (refer Fig. 11 and Fig. 13).

Next, we discuss the efficiency of the ResNet50+CNN+DL model in classifying obfuscated malware. For this, we use the obfuscator family from both datasets, which offers a variety of obfuscated malware employing different sophisticated code obfuscations. Figure 11 distinctly demonstrates that the Obfuscator.ACY variants family within the BIG2015 dataset achieves precise classification, boasting a detection rate of 100%. Also, 98.7% of files in the same class were detected by ResNet50||CNN||DL model (refer Fig. 14). Similarly, Fig. 13 and Fig. 15 demonstrates that the Obfuscator.AD family in the MalImg dataset also achieves a detection rate of 100%. The classification models proved adept at identifying obfuscated executables as they could precisely

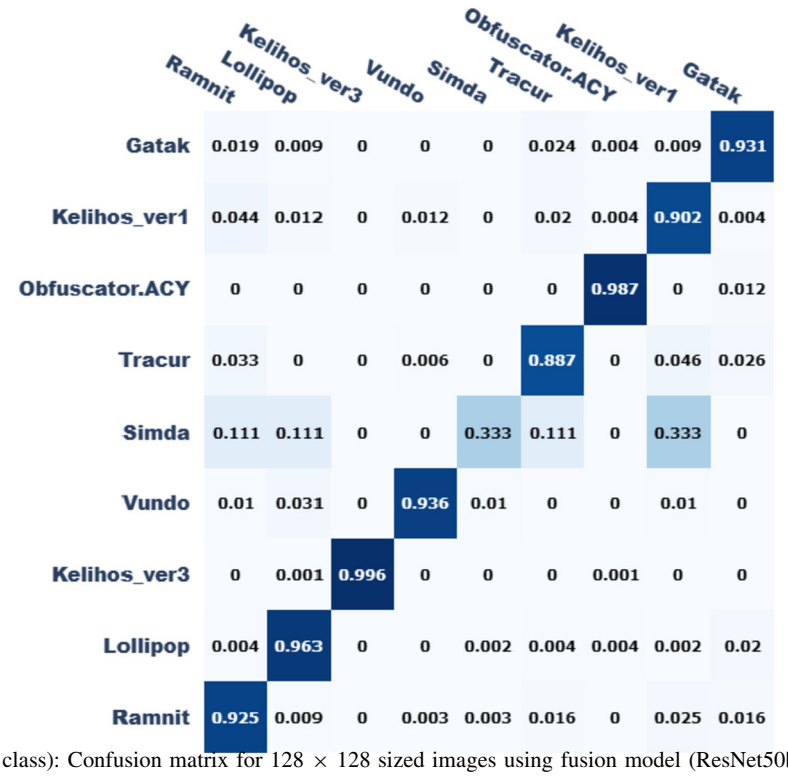


Fig. 14 BIG2015 dataset (9 class): Confusion matrix for 128 × 128 sized images using fusion model (ResNet50||CNN||DL)

identify image patches corresponding to obfuscated regions, refer Fig. 12. Additonally, within the MalImg dataset, families like Yuner.A, VB.AT, Malex.gen!j, Autorun.k, and Rbot.gen employ UPX packing. As depicted in Fig. 13, the model exhibits a 100% detection rate in classifying these families.

4.5 Performance evaluation of proposed model on evasion attack

We also performed evasion attacks to measure the robustness of classification models. We conducted the experiments by creating adversarial examples from test samples by applying additive noise. The results obtained after the attack on model1 (ResNet50+CNN+DL) are shown in Table 12. Here, 10% malware images in the test set were perturbed. As seen in Table 12, creating an adversarial example using Poisson noise results in a maximum drop of 1.68% in F1 score. Further, from Fig. 17, it can be seen that 19 out of 22 families of samples are well classified with an F1 score of 1.0. One sample of Obfuscator.AD is misclassified as LoydaAA1 and one sample of Allaple.A is misclassified as Malexgen!A, respectively. Furthermore, all variants of Adlair.C are wrongly labeled completely as AgentFYI. Overall, the F1 score has reduced to 0.957, representing a 4.19% decrease when applying the Local Variance method, which is less compared to the performance obtained with the ResNet50+CNN+Dense Layers model. Interesting evasion attacks launched with adversarial samples, developed with different additive noise methods, failed to lower the detection rate of the classification model(ResNet50+CNN+DL) on BIG2015 dataset. Furthermore, we conducted an identical experiment on model2 (ResNet50||CNN||DL) (Table 13). We observed a decrease in F1 scores for BIG2015, ranging from 0.874 to 0.878. For MalImg, F1 scores varied between 0.992 and 0.993. Additionally, F1 scores for Malhub encompassed a range from 0.014 to 0.992. This finding suggests that attackers can manipulate the malware code by adding specific bytes without altering its operational capabilities, ultimately evading detection by the machine learning classifier. A few solutions to deal with adversarial attacks are (a) *defense in depths*: which is to have multiple layers of protection (signature/non-signature based approach, machine learning based techniques, or automatic sandboxing), (b) *data diversification and augmentation*: use of GAN to create synthetic malware samples and augment the training set, (c) *Explainable AI*: develop

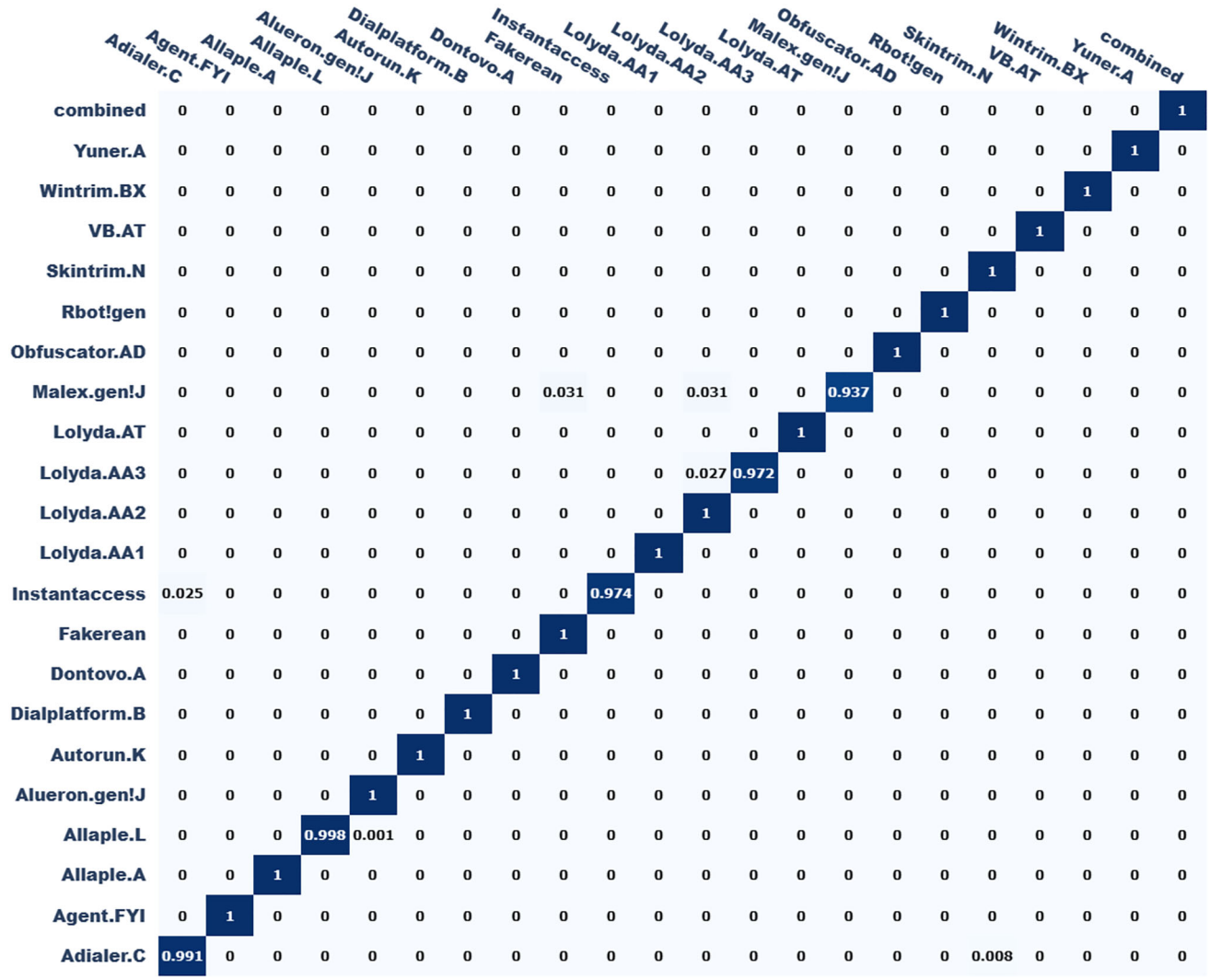


Fig. 15 MalImg dataset: Confusion matrix for 128 × 128 sized images using fusion model (ResNet50||CNN||DL)

techniques to provide explanations about the decisions made by algorithms. This will try to uncover patterns in the model that the adversary might use to deceive detection, (d) *Robustness-based training*: is to retrain the classier by modifying the training set, optimize the training algorithm to learn diverse data points, and using regularization techniques, (e) *Open science*: Open-source sharing of datasets, tools, and research findings can foster collaboration and accelerate the pace of innovation.

4.6 Comparative analysis of the proposed system with the state-of-the-art approaches

Most malware specimens adopt obfuscation techniques to evade detection, as mentioned in the previous sections, and the texture-based approach can withstand such techniques, which leads to enhanced accuracy. The homogeneous malware versions exhibit visual resemblance, whereas distinct visual traits belong to different families.

Visualization of malware binaries as image preserves variations in code structure introduced by malware authors. In this study, we mainly focused on comparing our effective solution with other popular texture-based malware categorization schemes reported in [18, 23, 48, 51]. Each

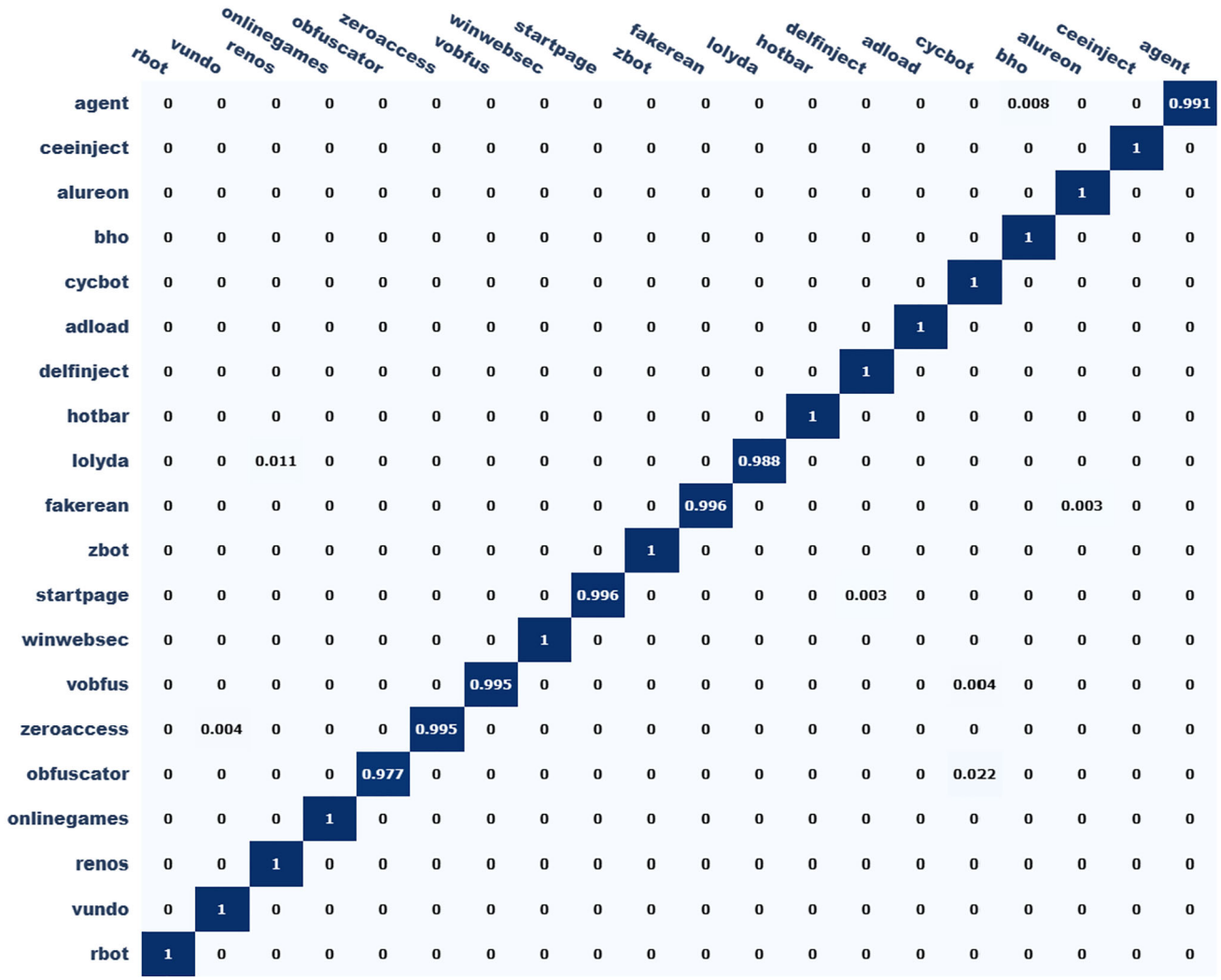


Fig. 16 Malhub dataset(20 families): confusion matrix for 128 × 128 sized images using fusion model (ResNet50||CNN||DL)

method applied a machine learning algorithm or deep learning, extracting features from malware images for classification. We have tabulated the results in Table 14. Our proposed model successfully classifies samples with an F1 score of 0.998 with 0.998 accuracy in the MalImg dataset. Besides, we attained the highest F1 score of 0.995 with an accuracy of 0.995 for the BIG2015 dataset. Additionally, we obtained an F1 score in the range of 0.993−0.999 on Malhub dataset using LBP images. Furthermore, we performed an adversarial attack on the model generated to measure the robustness.

4.7 Discussion and limitations

Our outcomes suggest that the proposed malware family classification scheme is superior to current approaches in terms of the following factors: (1) For both the benchmark datasets, the majority of the malware families show 100% F1 score (2) Deeper networks of ResNet50 classifies class-specific features (3) Texture-based images of binaries preserves the generic pattern of a family. In both datasets, we can observe some misclassifications; this may be because such samples share some generic features that overlap with other classes. Hence, such files will be visually similar. For Example, in Fig. 11, the Kelihos_version3 is misclassified as Obfuscator.ACY because they are both Trojans and share common patterns. Thus, the classifier is unable to discriminate samples and wrongly classify them.

Various pre-trained deep neural models were used to categorize the malware images. It is evident from the results that the generic attributes obtained by augmenting pre-trained models with CNN+DL precisely detected

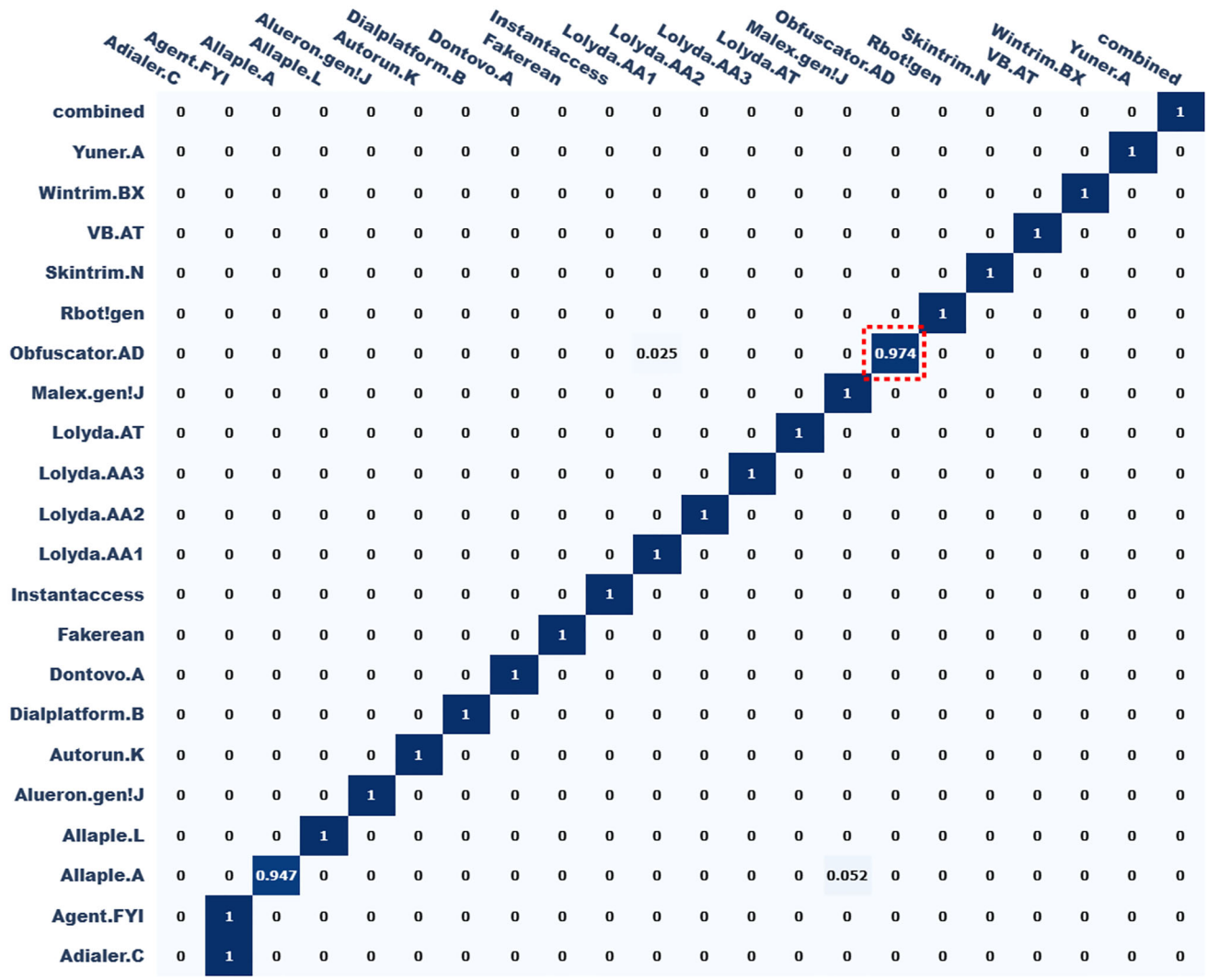


Fig. 17 MalImg dataset with adversary attack on ResNet50+CNN+DL model: confusion matrix of 128 × 128 sized images with Poisson random noise

samples. The experiments conducted by authors in [52] demonstrate the superior performance of ResNet50 in classifying the Imagenet dataset than VGG16. We also obtained similar trends while classifying malicious binaries. Deeper networks like ResNet50 can produce semantically enriched features, making them generalizable for other classification problems than shallower models like VGG16.

Moreover, we investigated the significance of image dimensions in family classification by experimenting with different image dimensions (64x64, 128x128). We noted that models trained on the image size 64x64 outperformed the others, achieving an F1 score of 0.995 for the BIG 2015 dataset, while 128x128 performed well for the MalImg dataset with an F1 score of 0.979 (LBP image of 25 class). Therefore, we can infer that the images with small dimensions cannot make fine distinctions as they cannot retain relevant information about the family. The only exception is BIG2015 (executables with stripped header). In contrast, a high image dimension can only escalate the computation time without contributing much to overall performance. Furthermore, we conducted extensive experiments to test the robustness of our approach against a couple of adversarial attacks. We noticed that we obtained

some misclassifications (see Fig. 17), but the model could accurately identify the families of MalImg and BIG2015, however, it could not identify malware files of Malhub dataset. The adversary may also try to relocate the parts of code to modify the malware image to deceive detection. However, such modifications might negatively impact the performance of our model, as convolutional networks are skillful enough to learn features invariant to relocation. This would be useful in identifying malicious files obfuscated using the control flow obfuscation technique. The source code and hashes of malware files are shared on the github page.[8]

5 Conclusions and future scope

Conventional antivirus solutions heavily rely on machine learning (ML) techniques to protect digital information from malware attacks, and these methods have demonstrated efficacy in identifying new malware strains. However, creating robust features for ML algorithms is demanding regarding time and expertise. Deep learning architectures have also emerged as promising tools for malware detection, offering remarkable performance in classifying malware samples. The proposed deep learning model achieved exceptional results, consistently high F1 scores of 0.99 across 64x64 and 128x128 sized malware images. This success was consistently observed on two publicly available datasets (MalImg and Big 2015) and a self-created dataset, highlighting the model's versatility and robustness. Additionally, we have validated the effectiveness of extracting features from malware executable images to classify them into their respective families and employing transfer learning to group new malware samples. This approach significantly reduces computational costs and resource requirements when dealing with new datasets. However, we also observed that the classification models exhibit vulnerability to tainted examples crafted by adversaries, except adversarial examples generated using Gaussian noise.

Despite achieving competitive results compared to state-of-the-art approaches, we believe there is still significant room for improvement. We intend to conduct more extensive experiments using various executable files in the future. We also plan to delve into the performance of the proposed visualization approach in the context of concept drift, a phenomenon where changes in data patterns can affect the model's accuracy. Additionally, we aim to develop techniques to enhance the detection rates of malware scanners trained on image-based features. Further, we will explore methods for effectively fusing deep learning models based on the attributes they extract. Towards this goal, we plan to augment our proposal by implementing model interpretation techniques, such as class activation maps, which provide insights into how the model makes decisions. Finally, we acknowledge that the representation of 2D images and how Convolutional Neural Networks (CNNs) operate might lead to information loss between bytes appearing in the right corner and the beginning of the left edge. To address this, we intend to investigate the behavior of CNNs on 1D images, which could potentially capture more contextual information.

Appendix A

This section presents the detailed evaluation results of ML classifiers and Dense (2-Layers). Additionally, we include the ROC curve for the proposed classification model.

See Figs. 18, 19, and 20.

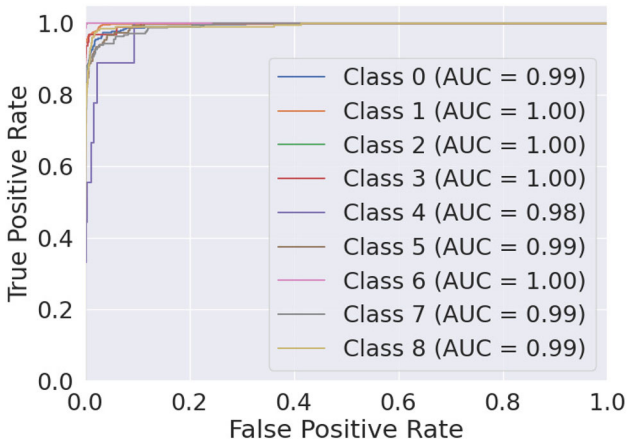


Fig. 18 ROC curve for BIG2015 dataset using proposed fusion model(ResNet50||CNN||DL)

[8] Github link: https://github.com/OPTIMA-CTI/DL-Adversarial-Noise

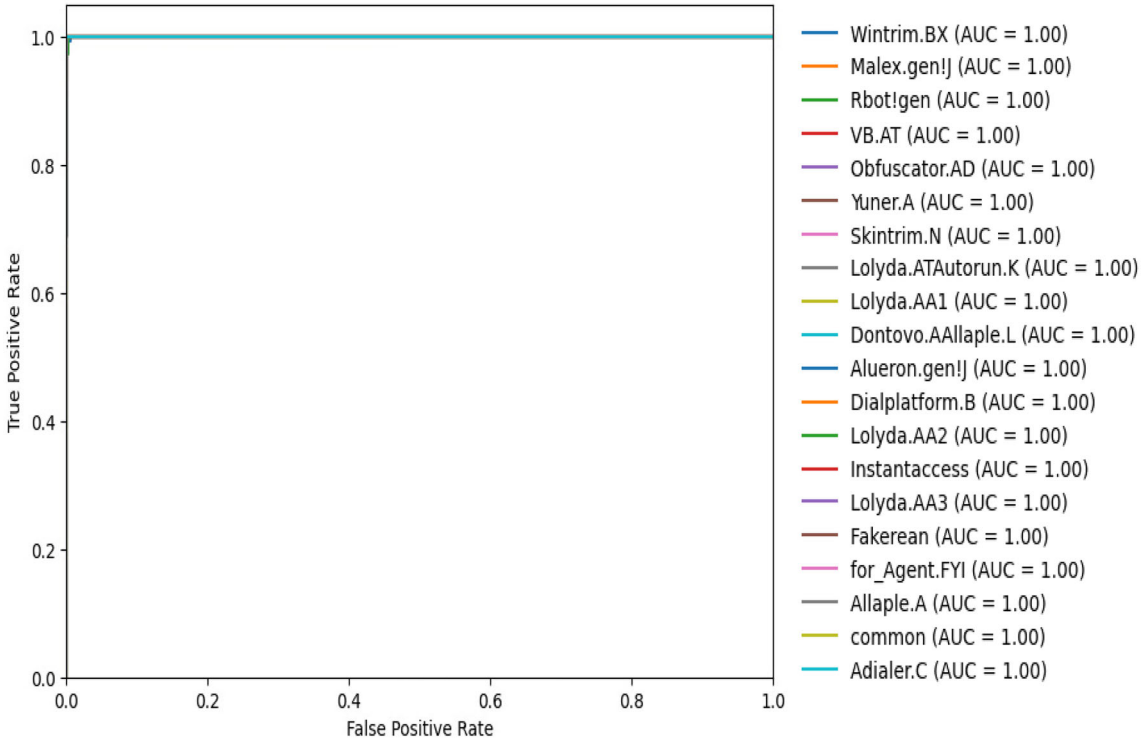
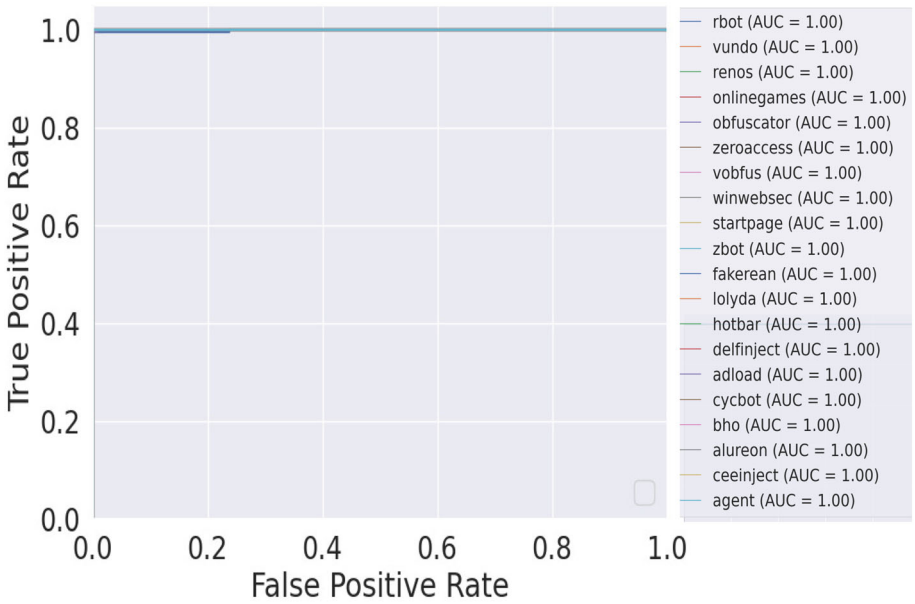


Fig. 19 ROC curve for MalImg dataset using proposed fusion model(ResNet50||CNN||DL)

Fig. 20 ROC curve for Malhub dataset using proposed fusion model(ResNet50||CNN||DL)



Supplementary Information The online version contains supplementary material available at https://doi.org/10.1007/s10586-024-04397-4.

Acknowledgements We gratefully acknowledge the support provided by the HORIZON Europe Framework Programme for the project "OPTIMA - Organization-specific Threat Intelligence Mining and Sharing" (No. 101063107).

Author contributions KAA: software, validation, investigation, data curation, writing—original draft, writing—review and editing. VP: conceptualization, investigation, supervision, writing—original draft,

writing—review and editing. KARR: supervision, writing—original draft, writing—review and editing. SLA: software, validation, investigation, writing—original draft, writing—review and editing.

Funding Not applicable.

Data availability Not applicable.

Declarations

Conflict of interest The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Ethical approval Not applicable.

References

1. Shijo, P., Salim, A.: Integrated static and dynamic analysis for malware detection. Procedia Comput. Sci. **46**, 804–811 (2015)
2. Alzaylaee, M.K., Yerima, S.Y., Sezer, S.: Dl-droid: Deep learning based android malware detection using real devices. Comput. Secur. **89**, 101663 (2020)
3. Islam, R., Tian, R., Batten, L.M., Versteeg, S.: Classification of malware based on integrated static and dynamic features. J. Netw. Comput. Appl. **36**(2), 646–656 (2013)
4. Ni, S., Qian, Q., Zhang, R.: Malware identification using visualization images and deep learning. Comput. Secur. **77**, 871–885 (2018)
5. Fu, J., Xue, J., Wang, Y., Liu, Z., Shan, C.: Malware visualization for fine-grained classification. IEEE Access **6**, 14510–14523 (2018)
6. Grosse, K., Papernot, N., Manoharan, P., Backes, M., McDaniel, P.: Adversarial perturbations against deep neural networks for malware classification. arXiv preprint arXiv:1606.04435 (2016)
7. Al-Dujaili, A., Huang, A., Hemberg, E., O'Reilly, U.-M.: Adversarial deep learning for robust detection of binary encoded malware. In: 2018 IEEE Security and Privacy Workshops (SPW), pp. 76–82 (2018). IEEE
8. Schultz, M.G., Eskin, E., Zadok, F., Stolfo, S.J.: Data mining methods for detection of new malicious executables. In: Proceedings 2001 IEEE Symposium on Security and Privacy. S &P 2001, pp. 38–49 (2000). IEEE
9. Kolter, J.Z., Maloof, M.A.: Learning to detect malicious executables in the wild. In: Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 470–478 (2004)
10. Santos, I., Nieves, J., Bringas, P.G.: Semi-supervised learning for unknown malware detection. In: International Symposium on Distributed Computing and Artificial Intelligence, pp. 415–422 (2011). Springer
11. Siddiqui, M., Wang, M.C., Lee, J.: Detecting internet worms using data mining techniques. J. Syst. Cybernetics Inform. **6**(6), 48–53 (2009)
12. Kang, B., Yerima, S.Y., McLaughlin, K., Sezer, S.: N-opcode analysis for android malware classification and categorization. In: 2016 International Conference on Cyber Security and Protection of Digital Services (cyber Security), pp. 1–7 (2016). IEEE
13. Peter, E., Schiller, T.: A Practical Guide to Honeypots. Washington Univerity, Washington, DC (2011)
14. Rieck, K., Holz, T., Willems, C., Düssel, P., Laskov, P.: Learning and classification of malware behavior. In: International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment, pp. 108–125 (2008). Springer
15. Ki, Y., Kim, E., Kim, H.K.: A novel approach to detect malware based on api call sequence analysis. Int. J. Distrib. Sens. Netw. **11**(6), 659101 (2015)
16. Anderson, B., Quist, D., Neil, J., Storlie, C., Lane, T.: Graph-based malware detection using dynamic analysis. J. Comput. Virol. **7**(4), 247–258 (2011)
17. Yoo, I.: Visualizing windows executable viruses using self-organizing maps. In: Proceedings of the 2004 ACM Workshop on Visualization and Data Mining for Computer Security, pp. 82–89 (2004)
18. Nataraj, L., Karthikeyan, S., Jacob, G., Manjunath, B.S.: Malware images: visualization and automatic classification. In: Proceedings of the 8th International Symposium on Visualization for Cyber Security, pp. 1–7 (2011)
19. Choi, S., Jang, S., Kim, Y., Kim, J.: Malware detection using malware image and deep learning. In: 2017 International Conference on Information and Communication Technology Convergence (ICTC), pp. 1193–1195 (2017). IEEE
20. Yajamanam, S., Selvin, V.R.S., Di Troia, F., Stamp, M.: Deep learning versus gist descriptors for image-based malware classification. In: Icissp, pp. 553–561 (2018)
21. Nataraj, L., Yegneswaran, V., Porras, P., Zhang, J.: A comparative assessment of malware classification using binary texture analysis and dynamic analysis. In: Proceedings of the 4th ACM Workshop on Security and Artificial Intelligence, pp. 21–30 (2011)
22. Su, J., Vasconcellos, D.V., Prasad, S., Sgandurra, D., Feng, Y., Sakurai, K.: Lightweight classification of iot malware based on image recognition. In: 2018 IEEE 42Nd Annual Computer Software and Applications Conference (COMPSAC), vol. 2, pp. 664–669 (2018). IEEE
23. Cui, Z., Xue, F., Cai, X., Cao, Y., Wang, G.-G., Chen, J.: Detection of malicious code variants based on deep learning. IEEE Trans. Ind. Inform. **14**(7), 3187–3196 (2018)
24. Gibert, D., Mateu, C., Planes, J., Vicens, R.: Using convolutional neural networks for classification of malware represented as images. J. Comput. Virol. Hack. Tech. **15**(1), 15–28 (2019)
25. Mourtaji, Y., Bouhorma, M., Alghazzawi, D.: Intelligent framework for malware detection with convolutional neural network. In: Proceedings of the 2nd International Conference on Networking, Information Systems & Security, pp. 1–6 (2019)
26. Venkatraman, S., Alazab, M., Vinayakumar, R.: A hybrid deep learning image-based analysis for effective malware detection. J. Inf. Secur. Appl. **47**, 377–389 (2019)
27. Akarsh, S., Simran, K., Poornachandran, P., Menon, V.K., Soman, K.: Deep learning framework and visualization for malware classification. In: 2019 5th International Conference on Advanced Computing & Communication Systems (ICACCS), pp. 1059–1063 (2019). IEEE
28. Vasan, D., Alazab, M., Wassan, S., Safaei, B., Zheng, Q.: Image-based malware classification using ensemble of cnn architectures (imcec). Comput. Secur. **92**, 101748 (2020)
29. Vasan, D., Alazab, M., Wassan, S., Naeem, H., Safaei, B., Zheng, Q.: Imcfn: Image-based malware classification using fine-tuned convolutional neural network architecture. Comput. Netw. **171**, 107138 (2020)
30. Chen, L.: Deep transfer learning for static malware classification. arXiv preprint arXiv:1812.07606 (2018)
31. Wang, C., Zhao, Z., Wang, F., Li, Q.: A novel malware detection and family classification scheme for iot based on deam and densenet. Secur. Commun. Netw. **2021**, 1–16 (2021)
32. Alzubi, O.A., Qiqieh, I., Alzubi, J.A.: Fusion of deep learning based cyberattack detection and classification model for intelligent systems. Clust. Comput. **26**(2), 1363–1374 (2023)

33. Le, Q., Boydell, O., Mac Namee, B., Scanlon, M.: Deep learning at the shallow end: Malware classification for non-domain experts. Digit. Invest. **26**, 118–126 (2018)
34. Demontis, A., Melis, M., Biggio, B., Maiorca, D., Arp, D., Rieck, K., Corona, I., Giacinto, G., Roli, F.: Yes, machine learning can be more secure! a case study on android malware detection. IEEE Trans. Depend. Secure Comput. **16**(4), 711–724 (2017)
35. Grosse, K., Papernot, N., Manoharan, P., Backes, M., McDaniel, P.: Adversarial examples for malware detection. In: European Symposium on Research in Computer Security, pp. 62–79 (2017). Springer
36. Chen, S., Xue, M., Fan, L., Hao, S., Xu, L., Zhu, H., Li, B.: Automated poisoning attacks and defenses in malware detection systems: An adversarial machine learning approach. computers & security 73, 326–344 (2018)
37. Ronen, R., Radu, M., Feuerstein, C., Yom-Tov, E., Ahmadi, M.: Microsoft malware classification challenge. arXiv preprint arXiv:1802.10135 (2018)
38. Ojala, T., Pietikäinen, M., Harwood, D.: A comparative study of texture measures with classification based on featured distributions. Pattern Recogn. **29**(1), 51–59 (1996)
39. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 770–778 (2016)
40. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. Advances in neural information processing systems **25** (2012)
41. Olivas, E.S., Guerrero, J.D.M., Martinez-Sober, M., Magdalena-Benedito, J.R., Serrano, L., et al.: Handbook of Research on Machine Learning Applications and Trends: Algorithms, Methods, and Techniques: Algorithms, Methods, and Techniques. IGI global (2009)
42. Bulazel, A., Yener, B.: A survey on automated dynamic malware analysis evasion and counter-evasion: Pc, mobile, and web. In: Proceedings of the 1st Reversing and Offensive-oriented Trends Symposium, pp. 1–21 (2017)
43. Xu, H., Ma, Y., Liu, H.-C., Deb, D., Liu, H., Tang, J.-L., Jain, A.K.: Adversarial attacks and defenses in images, graphs and text: a review. Int. J. Autom. Comput. **17**(2), 151–178 (2020)
44. Laidlaw, C., Feizi, S.: Functional adversarial attacks. Advances in neural information processing systems **32** (2019)
45. Vivek, B., Mopuri, K.R., Babu, R.V.: Gray-box adversarial training. In: Proceedings of the European Conference on Computer Vision (ECCV), pp. 203–218 (2018)
46. You, I., Yim, K.: Malware obfuscation techniques: A brief survey. In: 2010 International Conference on Broadband, Wireless Computing, Communication and Applications, pp. 297–300 (2010). IEEE
47. Schiffman, M.: A brief history of malware obfuscation: Part 2 of 2. Cisco Blog (2010)
48. Agarap, A.F.: Towards building an intelligent anti-malware system: a deep learning approach using support vector machine (svm) for malware classification. arXiv preprint arXiv:1801.00318 (2017)
49. Jian, Y., Kuang, H., Ren, C., Ma, Z., Wang, H.: A novel framework for image-based malware detection with a deep neural network. Comput. Secur. **109**, 102400 (2021)
50. Deng, H., Guo, C., Shen, G., Cui, Y., Ping, Y.: Mctvd: A malware classification method based on three-channel visualization and deep learning. Comput. Secur. **126**, 103084 (2023)
51. Shaid, S.Z.M., Maarof, M.A.: Malware behavior image for malware variant identification. In: 2014 International Symposium on Biometrics and Security Technologies (ISBAST), pp. 238–243 (2014). IEEE
52. Bianco, S., Cadene, R., Celona, L., Napoletano, P.: Benchmark analysis of representative deep neural network architectures. IEEE Access **6**, 64270–64277 (2018)

K. A. Asmitha is presently a Full Time Research Scholar at the Department of Computer Applications, Cochin University of Science & Technology, Cochin, Kerala, India. Prior to this, she was a Junior Research Fellow in the Department of Applied Mathematics & Computational Sciences at PSG College of Technology, Coimbatore, India. She received a Master of Technology in Computer Science and Engineering with specialization in Information Systems from SCMS School of Engineering & Technology, Mahatma Gandhi University. Her main research interests are Malware Analysis, Android Mobile security and privacy, Machine Learning, and Data Mining. She is currently focused on Malware Analysis and Explainable AI.

Vinod Puthuvath is presently a Marie Curie fellow at University of Padua in and a Professor in the Department of Computer Applications at Cochin University of Science & Technology, Cochin, Kerala, India. He is a Postdoctoral Researcher at the Department of Mathematics, University of Padua, Italy, where he is part of the HORIZON Europe Framework Programme project named OPTIMA. He was also a Postdoctoral Researcher at the Department of Mathematics, University of Padua, Italy, where he was part of the EU-H2020 project named TagitSmart. He was also a Postdoctoral researcher at Malaviya National Institute of Technology, Jaipur, Rajasthan, India, under the ISEA project on Mobile Security. He holds his Ph.D. in Computer Engineering from Malaviya National Institute of Technology, Jaipur, India. In 2020, he was awarded the Seal of Excellence for a Marie Skłodowska-Curie Individual Fellowship by the European Commission. He has numerous research articles published in peer-reviewed Journals and International Conferences. He is a reviewer of a number of security journals such as IEEE Transactions of Information Forensics, IEEE Communication Surveys and Tutorials, Elsevier Computer Communications, and is also serving as a programme committee member in International Conferences related to Computer and Information Security. His area

of interest is Adversarial Machine Learning, Malware Analysis, Context-aware privacy-preserving Data Mining, and Natural Language Processing.

K. A. Rafidha Rehiman is presently an Assistant Professor at the Department of Computer Application, Cochin University of Science & Technology, Cochin, Kerala, India. She holds Ph.D. in Data security, M.Tech in Information system security, and Masters in Computer Applications. She has several research articles published in peer-reviewed Journals and International Conferences. She is also supervising Ph.D. scholars in privacy-preserving data sharing and distributed machine learning. Her research interests include Cryptography, Information Security, and Cyber forensics Analysis. Her research work focuses on Ransomware Analysis and Cyber Threat Intelligence. She has been instrumental in organizing various workshops on security at International/national and state-level institutions.

S. L. Ananth received an M.Tech in Software Systems with a specialization in Data Science from BITS Pilani. He has more than 20 years of progressive experience in software analysis, design, and coding. His research interests are malware analysis and machine learning.