






## Research Article

# A Malware Detection Scheme via Smart Memory Forensics for Windows Devices

**Muhammad Rashid Naeem** <sup>1</sup>, **Mansoor Khan** <sup>1</sup>, **Ako Muhammad Abdullah** <sup>2,3</sup>,  
**Fazal Noor** <sup>4</sup>, **Muhammad Ijaz Khan**,<sup>5</sup> **Muhammad Asghar Khan** <sup>6</sup>, **Insaf Ullah** <sup>6</sup>,  
**and Shah Room** <sup>7</sup>

<sup>1</sup>School of Electronic Information and Artificial Intelligence, Leshan Normal University, Leshan 614000, China

<sup>2</sup>University of Sulaimani, College of Basic Education, Computer Science Department, Sulaimani City, Kurdistan Region, Iraq

<sup>3</sup>Department of Information Technology, University College of Goizha, Sulaimaniyah, Kurdistan Region, Iraq

<sup>4</sup>Faculty of Computer and Information Systems, Islamic University of Madinah, Madinah 400411, Saudi Arabia

<sup>5</sup>Institute of Computing and Information Technology, Gomal University, DIK, Pakistan

<sup>6</sup>Hamdard Institute of Engineering & Technology, Islamabad, Pakistan

<sup>7</sup>Ghalib University, Herat, Afghanistan

Correspondence should be addressed to Mansoor Khan; [khan007\\_bet@yahoo.com](mailto:khan007_bet@yahoo.com) and Shah Room; [zai\\_fhs@foxmail.com](mailto:zai_fhs@foxmail.com)

Received 3 July 2022; Revised 20 August 2022; Accepted 5 September 2022; Published 3 October 2022

Academic Editor: Junaid Shuja

Copyright © 2022 Muhammad Rashid Naeem et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

With the introduction of 4G/5G Internet and the increase in the number of users, the malicious cyberattacks on computing devices have been increased making them vulnerable to external threats. High availability windows servers are designed to ensure delivery of consistent services such as business activities and e-services to their customers without any interruption. At the same time, a cyberattack on any of the clustered computer can put servers and customer devices in danger. A memory dump mechanism can capture the contents of memory in the event of a system or device crash such as corrupted files, damaged hardware, or irregular CPU power consumption. In this paper, we present a smart memory forensics scheme to recognize malicious attacks over high availability servers by capturing the memory dump of suspicious processes in the form of RGB visual images. Second, the local and global properties of malware images are captured using local binary patterns (LBP) and gray-level co-occurrence matrices (GLCM). A state-of-the-art t-distributed stochastic neighbor embedding scheme (t-SNE) is applied to reduce data dimensionality and improve the detection time of unknown malwares and their variants. An optimized CNN model is designed to predict malicious files harming servers or user devices. Throughout this study, we employed public data set of 4294 malicious samples covering malware variants and benign executables. A baseline is prepared to compare the performance of proposed model with state-of-the-art malware detection methods. The combined LBP + GLCM feature extraction along with t-SNE dimensionality reduction scheme further improved the detection accuracy by 98%, whereas the detection time is also increased by 73x. The overall performance shows that memory forensics is more effective for malware detection in terms accuracy and response time.

## 1. Introduction

Due to extreme and rapid development of Internet technology, the e-business and e-services has become a key part of daily life. Meanwhile, malicious software also known as malwares keep evolving and posing a security threat to the user devices. To overcome the security threats, the

establishment of a fast and secure malware detection system is essential for high services of services. The well-known and commonly used malware detection methods rely on signatures, which use the sequences of binary patterns to uniquely identify malwares [1]. Antivirus programs detect malwares of similar behavior by matching malicious signatures of the scanning files. Signature-based malware

detection methods generally provides outstanding accuracy with fewer false positives. When a new malicious program enters a victims' device, the extraction process takes time to extract and read signatures leaving connected devices vulnerable for some time. Researchers have also suggested heuristic measures to detect suspicious files that could harm computer devices in order to tackle such vulnerabilities [2]. Heuristic methods were developed to detect unknown malwares by preserving suspicious program features. For instance, if an unknown packer is being used to protect detection, it could be listed as a suspicious software attempting to hide or modify its original signatures. Such methods have the potential to detect new unknown malwares, but their false positive rate is relatively high. Since many commercial software developers use packers or similar tools to stop hackers from breaching or manipulating their products for free. Researchers have been developed many strategies to overcome limitations of malware detection such as static, dynamic, and hybrid strategies [3].

Static methods can easily recognize from other detection techniques since they need less malware detection time and do not require real-time malware execution. These methods typically analyze various aspects of suspicious programs such as sequences, byte patterns, opcode, API calls, and other relevant properties from portable executables (PEs) [4]. These aspects of suspicious program are collectively referred as signature, which is an algorithm or unique hash used to distinguish one malware from another such as malware families. As a result, no real-time execution or computational resource is required. However, static methods have some shortcomings such as code obfuscation or encryption, which can easily deceive malware detection strategy [5]. Therefore, an alternative and faster strategy is essential to detect obfuscated and encrypted malwares with a higher true positive.

Dynamic methods generate similar outcomes regardless of whether malware binaries are obfuscated or encrypted. Rather than examining the underlying features of a malicious code, a dynamic approach depends on examining discriminative behavior caused by real-time program execution. Dynamic methods generally execute programs in virtual environments such as sandboxes or virtual machines that are designed to monitor malware behavior. To be more specific, a dynamic method examines function calls to uncover suspicious anomalies in order to detect malware [6]. However, these malware activities are further classified into several observation strategies such as function parameter analysis, function call analysis, information workflow analysis, trace and tack analysis, and dynamic visual analysis of malware execution. Apart from that, numerous activity analyzer tools are also accessible online for dynamic analysis of malware binaries and executables. These activity analyzer tools include TT analyzer, CW Sandbox, and Anubis, etc. Dynamic methods detect malware more effectively with fewer false positives, but extracting dynamic malware characteristics is more complicated and more time consuming than static analysis. Therefore, many researchers have developed hybrid strategies to overcome the shortcomings of static and dynamic methods to generate high positives ratios in small amount of time.

Given the drawbacks of static and hybrid methods, cybersecurity businesses are shifting toward artificial intelligence [7]. Artificial intelligence algorithms are proven to be more successful in assessing malware patterns than static or dynamic methods. An AI algorithm uses smart visual representation to capture malware characteristics using DWT (discrete wavelet transform), SURF (speeded up robust feature), GIST, and SIFT descriptors. The DWT and GIST descriptors can be used to capture global characteristics such as structural patterns, while the SURF and SIFT descriptors can be employed to capture local information of malicious binaries. Few studies have adopted a combined approach to capture both local and global characteristics in order to enhance malware detection methods [8]. However, the cybercriminals are actively creating new malwares variants making their detection more challenging that may require new strategies to overcome security threats [9].

Recently, memory forensics such as examining the volatile memory has shown to be a more effective compared to static, dynamic, and signature-based malware detection methods. Memory forensic-based malware detection is determined into two phases. The first phase involves the transformation of virtual or physical memory data into memory dump files, while the second phase involves numerous analyzers to uncover anomalies or malicious behavior caused by malware execution. Lastly, the smart techniques like artificial intelligence are used to detect actual malware binaries [5]. According to Dai et al. [10], a malware executable in a volatile memory is most likely to be uncovered compared to mapping malicious signatures. Although some malwares can disguise themselves using encryption or packing, but eventually exposes their vital information such as code and data segmentation during real-time process execution. As a result, memory forensics can detect malware by examining victims' computer RAM. Furthermore, latest malware variants can escape detection by eradicating the evidence of their existence that may overshadowed by traditional detection methods. Fortunately, such malwares stay in volatile memory until malicious task is completed. A memory utilization strategy can effectively detect these malwares by obtaining memory dumps. Given the capability and flexibility of memory forensics, we can transform volatile memory binary data into RBG images as a source of information from dumped processes. The feature descriptors can further extract discriminating information for faster detection using artificial intelligence algorithms. The main contributions of this paper are listed as follows:

- (i) A malware detection architecture is proposed for windows devices that applies memory forensics to capture discriminating behavior of malware and benign samples instead of malware signatures, which can be obfuscated and encrypted to evade detection.
- (ii) We used RBGs of different resolutions such as  $224 \times 224$  and  $300 \times 300$  to capture the effects of memory dumps on malware samples. Few malware binaries consist of small resolutions compared to

others that may not capture sufficient discriminating information. Instead, we used different resolutions and 3-channel RGB images to extract malicious features from both malware and benign samples.

- (iii) We used combined LBP + GLCM descriptors for feature extraction instead of single image descriptor. LBP has the ability to recognize unchanging pitch and variation against monotonic grayscale contrast with great precision. The smoothness, softness, and roughness characteristics of image can be used to measure the textural variations. Alternatively, the GLCM captures the spatial distributions of pixels that holds textual information. It further exploits degree of correlation between two adjacent gray pixels separated by distance at certain position to measure discrimination of global characteristics. Lastly, both local and global characteristics are fused to capture both types of textural features.
- (iv) We further applied t-SNE dimensionality reduction and visualization to transform high-dimensional and complex features into low-dimensional space. Since LBP and GLCM generate large number of features from a single malware image. Therefore, the artificial intelligence algorithms require an extensive amount of time over data training and prediction, which is improved by t-SNE algorithm.
- (v) An optimized fine-tuned CNN model is ensembled for a public data set of 4294 malware + benign executables to perform malware detection and variant classification. The proposed model is further compared to state-of-the-art models and related works to evaluate performance.

This study is structured as follows: Section 2 describes a literature review on malware detection strategies. Section 3 describes the overall architecture of malware detection covering data selection, feature analysis, extraction methods, and evaluation matrices. Section 4 examines experimental findings and state-of-the-art comparison. Section 5 concludes this research and outlines the future works.

## 2. Literature Review

Since the 5G network has been deployed worldwide, the demands for high-performance service delivery are rapidly increasing. The global Internet traffic via Internet devices has been grown to 54.8% since the first quarter of 2017 [11]. The number of malicious attacks is expected to grow further in future as the demand for Internet devices expands, which is also one of the major tasks in future Internet of things. Many studies on malware detection have been undertaken, but the malware detection via image processing has proven to be more effective. In this approach, the meta information of malware is retrieved from executables or binaries and then transformed into grayscale or color images to achieve predictive malware detection. This section discusses the pros and cons of some of these approaches.

Han et al. [12] used graph theory and information entropy-based associations to find distinguishing malware characteristics. Hidden patterns were first identified and transformed into grayscale images. Later, entropy graphs were generated to expose potential malware families. The empirical analysis of 1000 malware samples achieved around 97.9% overall accuracy. Barath et al. [13] further extracted textural patterns from malware images and conduct PCA assessment on the collected features. The principal components were then loaded into a nearest neighbor classifier to perform detection. The empirical analysis of 10,000 samples produced an average accuracy of 96% on eight malware families. Xiaofang et al. [14] retrieved localized hashing patterns from malware images using SIFT descriptor and achieved an accuracy of 85% on a data set of 8410 malware samples consisting of 25 families. Although, the SURF descriptor has proven to be robust than the SIFT descriptor while extracting local textual characteristics, but SURF is not resilient to pixel rotation and illumination.

Several studies exploited the capabilities of deep neural networks to overcome the limitations of simple machine learning methods. For instance, Bozkir et al. [15] analyzed malware and benign files using several neural network models. In the beginning, raw malware and benign binaries were collected from executables and transformed into colored images. Later, a public data set of 12,394 malicious files was partitioned into 8750 train and 3644 test images for performance analysis. The empirical study achieved 97.48% malware detection accuracy on the “DenseNet” framework. Natraj et al. [16] extracted local binary patterns from visual malwares using the GIST descriptor. First, malware binaries were transformed to grayscale images, and afterward classification algorithms were employed to analyze extracted patterns. Their proposed model obtained 97% accuracy on a data set of 9339 malware samples from 25 families.

Malware attacks are not only affecting Windows PCs but also the Internet and industries. As a result, Ullah et al. [17] developed a CNN model to identify malware attacks on industries over the Internet. For visualization, the Internet malware binaries were first transformed into colored images. Hemalatha et al. [18] proposed an efficient DenseNet neural network model for malware detection based on grayscale images. Experimental evaluation of four public malware data sets Maling [16], BIG-2015 [19], MaleVis [15], and Malicia [20] produces 98.23%, 98.46%, 98.21%, and 89.48%, respectively. Multi neural networks can optimize the performance of single neural network [21]. Ullah et al. [22] used a hybrid multimodel image representation for malware classification to improve prediction time and accuracy. Their multimodel approach able to produce 98% to 99.4% accuracy on two different malware data sets. Federated learning allows devices to combinedly learned and shared prediction model. Rey et al. [23] federated learning approach able to produce 99.92% accuracy on known devices and 98.59% accuracy on unknown devices, respectively.

Memory forensic has recently gained more popularity for malware detection. Dai et al. [24] used HOG descriptor to extract features based on memory dumps. The extracted features were then used to train deep neural networks on

grayscale images. Grayscale images greater than 4 MB were resized to 4096 pixel width using the bicubic interpolation method. As per the empirical study, the accuracy performance was improved by 96.7% on malware detection. In another study, Bozkir et al. [25] employed memory forensics and computer vision to detect malware and their variants. Instead of grayscale images with limited characteristics, a malware data set consisting of 4294 colored images is prepared with public access. They further used UMAP feature reduction strategy and GIST + HOG descriptors to improve detection accuracy on different predictive models. The empirical evaluation achieved up to 96.39% detection accuracy with an average detection time of 3.56 seconds. HOG is a well-studied descriptor that performs well on human-based detection but also sensitive toward image rotation as it uses only one direction for each image pixel. It is quite possible that feature extraction process may lose some discriminating information as many malwares are obfuscated or encrypted to avoid detection. In our study, we use LBP that uses eight directions for each pixel. Therefore, LBP can effectively map discriminating behavior regardless of obfuscation or encryption. Furthermore, GLCM extract spatial relationship among different pairs of pixels. Hence, a smart memory forensic approach with combined LBP + GLCM descriptor and t-SNE dimensionality reduction can overcome such limitations.

### 3. Malware Detection via Memory Forensics for Windows Devices

The malware detection architecture for windows is designed to examine memory dump files for the identification of potential malware from interconnected user devices. The memory dump files can capture malware anomalies and execution process in real time. As a result, we can identify the discriminating behavior of malware and benign executables more efficiently. This section briefly explains the data set and descriptors along with dimension reduction strategy used to implement malware detection architecture. Figure 1 presents the overall structure of malware detection framework based on memory forensics. In the first phase, we focused on generating dump files from volatile memory of malicious process using a virtual machine. The second phase focused on binary file representation of malware into  $224 \times 224$  and  $300 \times 300$ -dimensional RGB images. Next, feature extraction process is carried out using LBP and GLCM descriptors to extract discriminating characteristics of malware and benign samples. Malware visual images consist of both local and global textures. The local features refer to those patterns that depict distinct structure such as an edge, a point, or a small patch. Alternatively, the global features refer to those patterns associated to the whole image shape or structure. Since a single descriptor can only capture the limited information. Therefore, a feature fusion strategy can facilitate neural network to learn image features from their rich internal and external information. The resulting feature set can effectively classify both malware and benign samples regardless of computational complexity and obfuscation. Next, the t-SNE dimensional reduction and data

visualization are also applied to reduce high-dimensional data into low-dimensional feature space. Lastly, several state-of-the-art algorithms were utilized to evaluate malware detection and classification performance. In this study, we utilized memory forensics instead of traditional signature-based methods. Memory forensics analyze the process under execution rather than matching signature of malicious structure or code of malware executable. The memory forensic process transforms physical memory data into dump files to uncover anomalies or malicious behavior of a malware sample throughout its execution. Any malware in a volatile memory is most likely to be exposed compared to signature matching. Since many malwares disguise themselves via encryption or packing. However, such malwares expose their vital information such as code and data segmentation during real-time execution. As a result, memory forensic process detects malware by examining RAM data of a computer or user device. The advanced malware variants can avoid detection by erasing traces of their activity, enabling them to escape detection from traditional methods. These malwares remain in volatile memory until the malicious process is executed. A memory forensic method can detect such malwares by obtaining memory dumps of terminated processes.

*3.1. Dumpware10 Data set.* Several public malware data sets have been widely studied in the literature to detect and validate different types of malwares and their variants. Although these data sets have greatly improved malware detection strategies, they are restricted by two fundamental limitations. The first limitation is the unavailability of portable executables due to security concerns of data sharing. The second limitation is the lack of benign samples, which is essential for discriminating between actual malware and benign samples apart from malware variants. The negative samples are essential in classification tasks. For instance, benign samples serve as negative samples in malware detection process; alternatively, malware classification without benign samples is regarded as a close-set problem [25]. The primary objective of our study is to develop a classification model capable of identifying suspicious processes whether malicious or not. This study focuses on employing memory forensics to identify suspicious processes, which may simply overlook by traditional signature-based malware detection techniques possibility due to obfuscation and encryption. As a solution, we selected a public malware data set Dumpware10, which is entirely based on memory forensics and contains 3686 malware and 608 benign samples. We further partitioned the data set into 80% train, 20% validation, and 20% test sets for malware detection and classification.

*3.2. LBP Feature Descriptor.* The LBP descriptor is one of the simplest and most reliable methods for extracting local features from textural images such as malware representation. LBP has the ability to recognize grayscale micro-patterns from visual images with great precision [26]. In computer vision, the texture properties of an image are used

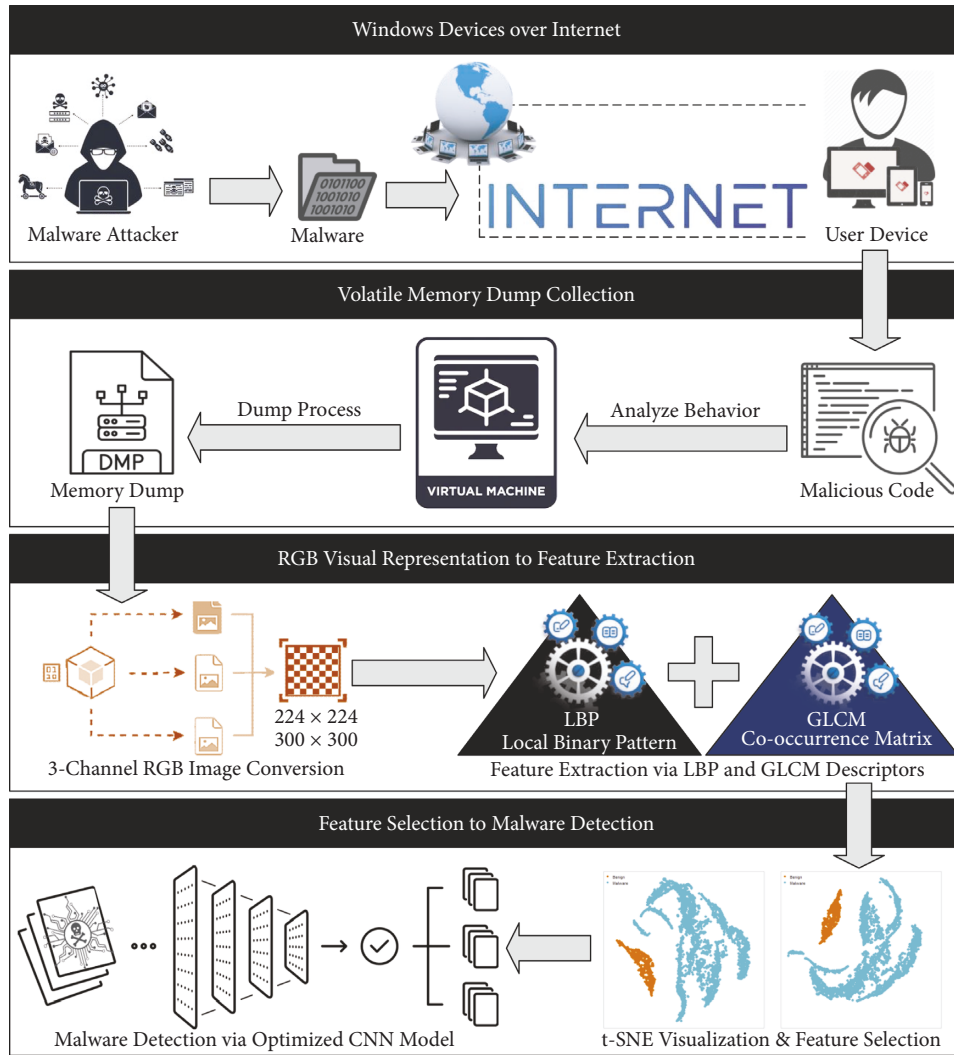


FIGURE 1: An overall malware detection framework for windows devices via smart memory forensics.

to characterize the degree of variations or spatial changes in textual patterns. The smoothness, direction, softness, and roughness characteristics of each surface can be used to measure textural variations. Such patterns required a strong descriptor to extract all distinctive features. We adopt the LBP descriptor to capture the unchanging pitch or variations produced by irregular malware patterns.

The LBP descriptor measures intensity in a digital image using a threshold value for adjacent pixels. Following that, LBP further assigns a decimal label to that threshold pixel. Figure 2 presents a  $3 \times 3$ -pixel block to demonstrate how the LBP descriptor extracts discriminating characteristics from a malware image. A  $3 \times 3$ -pixel block is chosen to examine each pixel intensity, with the central pixel of the block acting as the threshold value for neighboring pixels, i.e., 130. Subsequently, all pixels in the block with values greater than the threshold have been assigned a bit value of 1, while all remaining pixels in the block with values less than the threshold have been assigned a bit value of 0. In this way, each neighboring pixel has an assigned value to it is either 0 or 1. Next, the LBP descriptor locates all values clockwise other than the central

pixel and returns an 8 bit binary number, which in our case is 10101101. Lastly, the central pixel's 8 bit value is transformed to a decimal number, which is 173 for the binary number 10101101 used in this example. Once, labels are assigned to pixels, the final feature set is calculated from pixel values in the form of a histogram.

The LBP descriptor has proven to be very successful in visual malware analysis; however, resultant images may have large dimensions. As a result, the characteristic of the larger structures cannot be retrieved using only  $3 \times 3$ -pixel block. To overcome this problem, we used LBP descriptor with different radiuses depending on the size of malware images. The final LBP feature set was constructed and merged after calculating histograms for each radius. The computational execution of LBP descriptor is further detailed in the form of four-step procedure:

- (i) For each pixel on the  $x$ - and the  $y$ -axes, select surrounding pixels  $P$  within a defined radius  $R$ .
- (ii) Determine difference in the intensity between the current pixel on the  $x$  and  $y$  axes and the surrounding pixels  $P$ .

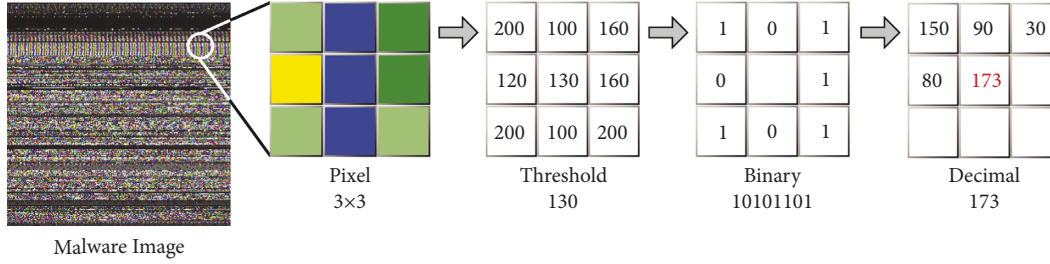


FIGURE 2: Example of LBP estimation for  $3 \times 3$ -pixel block from RGB malware image.

- (iii) Choose a threshold value for the surrounding pixels  $P$  and use intensity difference to assign 0 and 1 as single bit values.
- (iv) Transform the bit sequence of the surrounding pixels  $P$  to decimal values and replace the original intensity value of current pixel with the computed decimal value.

The LBP descriptor decimal value for each pixel is computed as follows:

$$\text{LBP}(P, R) = \sum_{p=0}^{P-1} f(g_p - g_c) 2^p. \quad (1)$$

Here  $g_c$  and  $g_p$  are the intensity differences between the current pixel and its surrounding pixels, whereas  $P$  is the number of surrounding pixels for a given radius  $R$ .

**3.3. GLCM Feature Descriptor.** Haralick et al. [27] proposed the gray level co-occurrence matrix (GLCM) descriptor for extracting global features from digital images. The GLCM descriptor primarily built on the concept that the spatial distribution of pixels holds the textural information of the image. GLCM exploits a co-occurrence matrix to estimate the joint probability distribution of two gray pixels separated by distance  $d$  at certain position in the image. GLCM employs three important aspects: direction ( $\theta$ ), variation amplitude ( $d$ ), and neighboring interval or gray level to extract integrated information from a grayscale image. The direction ( $\theta$ ) refers to the change in grayscale angle, which includes the primary direction of texture changes such as  $0^\circ$ ,  $45^\circ$ ,  $90^\circ$ , and  $135^\circ$  among others. The offset ( $d$ ) is the distance between two pixels in a grayscale image, whereas two adjacent pixels indicate the gray level orientation and offset value of pixels equal to 1. The gray-level orientation of pixels indicates maximum grayscale value plus 1, which is used to signify grayscale compression. Figure 3 demonstrates the feature extraction process of an RGB image with offset value 1 in the direction of 0 degree and gray level 3. The first step in the GLCM descriptor is to transform an RGB-colored image to grayscale in order to apply compression. The grayscale compression is useful for reducing matrix dimensions and feature extraction time for larger images. In order to generate co-occurrence matrix, pixel pairs were selected on position  $i$  and  $j$  for each element in the matrix. The initial co-occurrence value for a matching pair is 1 which is incremented by 1 when the next identical matching pair is

recognized. The numbers are then multiplied by 2 to reconstruct the co-occurrence matrix in the opposite direction. The GLCM descriptor offers a significant number of co-occurrence matrices that cannot be used as a final feature set. As a result, we simply employed four co-occurrence matrix properties to build the final feature. The selected GLCM properties of the visual images are energy, contrast, correlation, and homogeneity indicators. The final GLCM feature set consists of 20 columns, with each indicator comprising of 5 directions, resulting in 4 indicators  $\times$  5 directions.

LBP and GLCM descriptors are quite effective in terms of malware classification with few minor limitations. For instance, LBP is not insensitive to pixel rotations. The size of feature set increases the number of neighbors, which may also increase the computational complexity of descriptor. The structural information of LBP is also restricted as it only captures the pixel variations. Alternatively, the GLCM descriptor characterizes textural patterns by assessing the occurrences of pixel pairs in specific spatial relationship. A large number of computational resources are required as multiple matrices are computed to identify pixel pairs. Furthermore, GLCM features are also not insensitive to pixel rotation as well as changes in textual scaling. Although computational complexity is a significant issue when descriptors applied to large number of images. To overcome this limitation, we resized malware images into  $200 \times 200$  and  $300 \times 300$  dimension to resolve computation cost overhead in effective manner.

**3.4. t-SNE Visualization and Dimensionality Reduction.** The t-distributed stochastic neighbor embedding (t-SNE) is a nonlinear dimension reduction strategy used to visualize high-dimensional data in two or three low-dimensional planes with apparent distinction [28]. The t-SNE method reduces high dimensionality in 2 steps. In the first step, high-dimensional data points are assigned to similar objects with a higher probability of selection. In the second step, t-SNE minimizes divergence in low-dimensional space by adopting a uniform probability distribution. The t-SNE visualization is highly popular due to its potential to scale high-dimensional data into low-dimensional space. The t-SNE algorithm begins processing by employing stochastic neighbor embedding (SNE) on the data points and then transforms the high-dimensional distance between elements into the probability of similarities. The similarity between two data points from  $x_j$  to  $x_i$  is represented in terms of conditional probability  $p_{j|i}$  using equation (2).

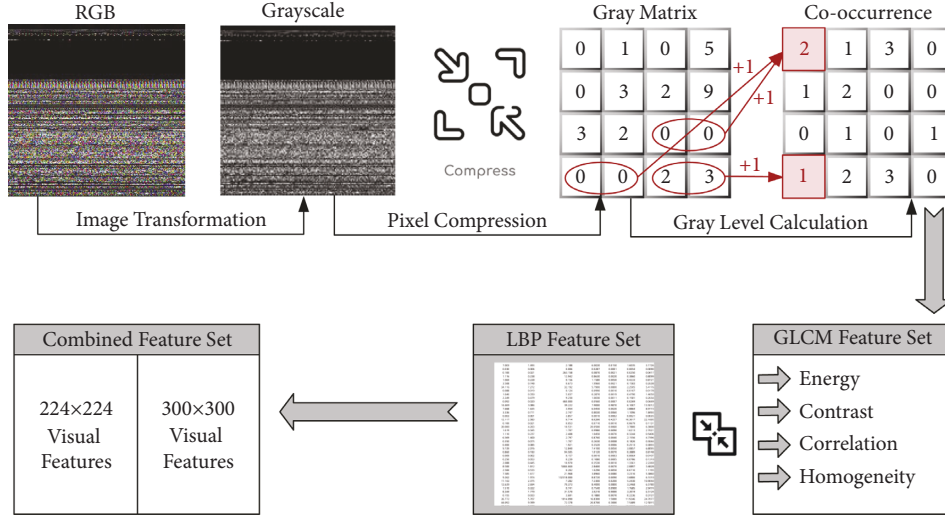


FIGURE 3: GLCM co-occurrence estimation using four properties from RGB malware image.

$$P_{j|i} = \frac{\exp\left(-\|x_i - x_j\|^2 / 2\sigma_i^2\right)}{\sum_{k \neq i} \exp\left(-\|x_i - x_k\|^2 / 2\sigma_i^2\right)}. \quad (2)$$

The probability of similarity in the original feature space is statistically determined using equation (3).

$$P_{i,j} = \frac{P_{i|j} + P_{j|i}}{2n}. \quad (3)$$

Here  $n$  represents the length of a data set. The t-SNE method requires an input parameter termed as “perplexity,” which can be interpreted as an uniform measurement of an effective number of clusters [29]. Mathematically, perplexity can be expressed as

$$\text{Prep}(P_i) = 2^{H(P_i)}. \quad (4)$$

Depending on the pairwise distances between data points, t-SNE method automatically determines the variance  $\sigma_i$ , such that the effective number of clusters corresponds to the user-defined perplexity value. To minimize congestion between data points, t-SNE adopts the student t-distribution based on single degree of freedom. The probability at low dimension  $q_{ij}$  is estimated using the matching distribution, as shown in equation (5).

$$q_{ij} = \frac{\left(1 + \|y_i - y_j\|^2\right)^{-1}}{\sum_{k \neq l} \left(1 + \|y_k - y_l\|^2\right)^{-1}}. \quad (5)$$

Many studies [30] stated that t-SNE distribution plots can be viewed as visual clusters of common distributions. These visual clusters can be improved further by adjusting parameters such as perplexity value and number of iterations. To construct optimal visual clusters, one must grasp the t-SNE parameters and information provided. Moreover, exploratory analysis and supplemental data can facilitate in

the selection of appropriate parameters and the verification of the outcomes. The structure of visual clusters formed by t-SNE is more isolated resulting in a more reliable and observable shape. However, effectiveness of t-SNE algorithm is always influenced by input data. t-SNE reduces dimensionality by utilizing local data properties, which may fail if the data has an exceptionally high-dimensional structure. In addition, several optimization parameters may be necessary to locate the constructed solution. In this study, we integrated both descriptors into a single vector for each malware image and used t-SNE to capture discriminating behavior of both descriptors into three dimensions. We then fed the t-SNE output into a deep learning model to evaluate performance in terms of accuracy and training time.

**3.5. Memory Dumps of Malicious Processes.** Malware detection via memory forensics has recently gained more popularity since process information is stored in volatile memory throughout its execution. As a result, suspicious activity can be retained from volatile memory in the form of memory dumps. One significant advantage of memory forensics is their resistance to obfuscation and packing as the runtime behavior of process remains unchanged. Memory dumps are system core dumps that are frequently used for troubleshooting particularly in application development process. When the system dumps a process, it preserves all of its processing data including thread stacks, data segments, heap sectors, and the calling sequence in the form of raw binaries. We can further utilize memory dumps to extract potential abnormal behavioral information from physical memory. Figure 4 presents the memory forensics lifecycle of a malicious process that begins with runtime behavioral analysis and terminated with RGB image transformation. In this example, a cryptojacking malware `i9prlkgopr.exe` is retrieved via `JOESandbox` (<https://www.joesandbox.com/>) that infects the victim device using `XMRig` (<https://xmrig.com/>) miner. This malware performs two malicious activities

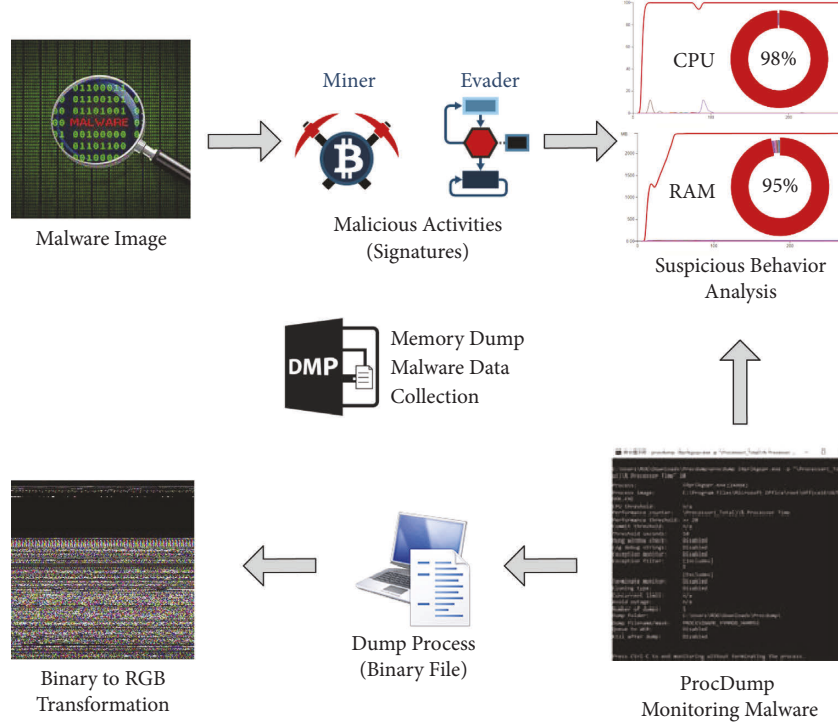


FIGURE 4: Memory forensics lifecycle for visual malware behavioral analysis.

to affect the victim computer. First, malware evades operating system protection using obfuscation and nonstandard tools. Second, it installs a miner script on the victim device in order to mine cryptocurrencies using the host CPU/GPU power. The miner script implemented by this malware can consume up to 98% CPU and 95% RAM depending on the configuration of the victim system. The malware is operated in a sandbox environment, while *ProcDump* is used to monitor suspicious activities. The *ProcDump* utility terminates the process and generates a raw memory dump file of the terminated process on detection of suspicious activity. Lastly, the raw dump binaries are transformed into RGB visual images to extract LBP and GLCM features for malware detection.

### 3.6. Optimized Convolutional Neural Network (O-CNN).

The following section provides the overview of O-CNN model used to train deep learning models for malware detection and variant classification. Four parameter tuning components have been selected to construct the O-CNN model. The first component is the input layer used to initiate training process. The second component is a convolutional layer used to reduce noise and improve image characteristics. To optimize the learning performance, we additionally employed convolutional kernel width and learning rate. Next, a pooling layer and a dense layer have been utilized to transform two-dimensional image properties into a one-dimensional feature set. Figure 5 shows the internal structure of proposed O-CNN model including neural network layers. The brief overview of each layer and its fine-tuned functions is given below:

**3.6.1. Convolutional Layer.** The CNN layer collects crucial characteristics of input images such as interpretation, rotation, and scaling of invariance to reduce training parameters. It significantly decreases overfitting and increases the generalization capability of the proposed DCNN model. The input value of CNN layer consists of several building blocks [31]. The output mapping variable is computed based on total addition to input building block of CNN layer. The CNN mapping for random input  $x_j^l$  is shown in equation (6).

$$x_j^l = f \left( \sum_{i \in M_j} x_j^{l-1} \times k_{ij}^l + b_j^l \right), \quad (6)$$

where  $M_j$  is the building block for CNN input;  $k_{ij}^l$  is a convolutional kernel that is combined with the  $i^{th}$  input and  $j^{th}$  output features; bias for the  $i^{th}$  input feature is represented by  $b_j^l$ ; and  $f$  is an activation function associated to the corresponding CNN layer.

$$\delta_j^l = \delta_j^{l+1} W_j^{l+1} \times f'(u^l) = \beta_j^{l+1} \text{up}(\delta_j^{l+1}) \times f'(u^l), \quad (7)$$

where  $l+1$  signifies a pooling layer,  $W$  denotes the current convolution kernel, and  $\beta_j^{l+1} \text{up}(\delta_j^{l+1})$  specifies Upsampling for minor class. The partial derivative  $\partial$  and error cost function of convolution kernel are computed as shown below:

$$\frac{\partial E}{\partial b_j} = \sum_{s,t} (\delta_j^l) u, v, \quad (8)$$

where  $(\delta_j^l) u, v$  represents a patch value for each convolution kernel in a stack.



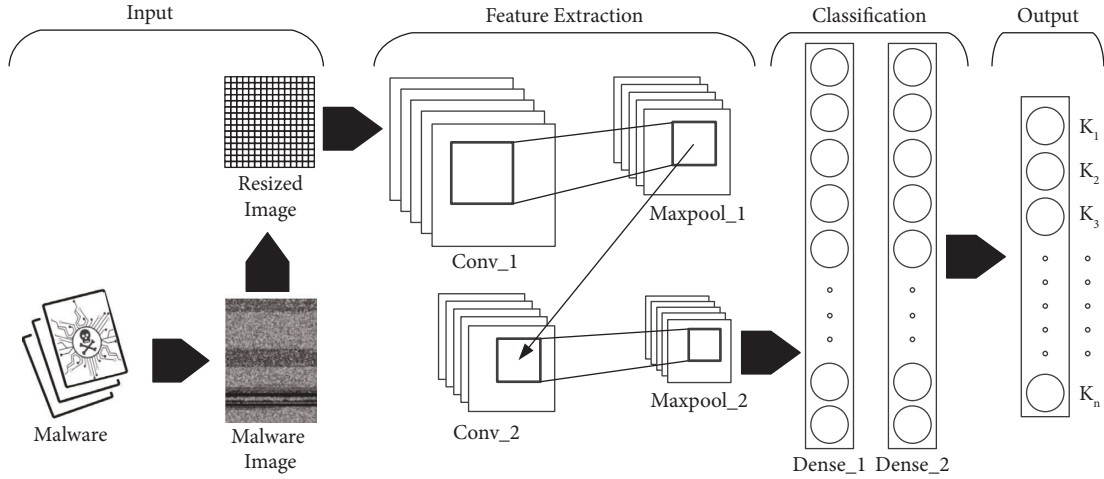


FIGURE 5: Internal structure of O-CNN model depicting convolutional, pooling, dense and output layers.

**3.6.2. Pooling Layer.** The DCNN model employs two types of pooling: maximal and average pooling. It has no effect on backward propagation but minimizes the effects of image deformation during DCNN training phase. The pooling layer further improves model performance while reducing the size of the visual input feature set:

$$x_j^l = f(\text{do wn}(x_j^{l-1}) + b_j^l), \quad (9)$$

where  $\text{do wn}(x_j^{l-1})$  is a pooling task and  $b$  is the bias value. The overall sensitivity can be computed using following formula:

$$\delta_j^l = \delta_j^{l+1} W_j^{l+1} \times f'(u^l). \quad (10)$$

**3.6.3. Dense Layer.** In TensorFlow Keras, the output of the pooling layer is further characterized based on the dense layer. The neurons inside the dense layer are all interconnected to the neurons in the pooling layer. It flattened the two-dimensional feature vector into a one-dimensional feature space prior transferring it to the output layer of proposed DCNN model.

**3.6.4. Output Layer.** In the output layer, test samples of memory dump images were labeled as malware or benign files. To validate the performance of the train and test models, we employ the SoftMax-Cross Entropy loss function throughout the DCNN model. Equation (9) can be used to estimate the training and testing data loss.

$$\text{Loss} = -\log\left(\frac{\exp(f_{zt})}{\sum_k \exp(f_{zt})}\right), \quad (11)$$

where  $f_{zt}$  is the rank of the  $k^{\text{th}}$  class label. The Adam optimizer is also used to learn DCNN model parameters that minimize training data loss.

**3.7. Evaluation Matrices.** We selected several evaluation metrics also used in previous studies to measure the performance of malware detection. The independent variables

of the study are true positives (TP), false positives (FP), true negatives (TN), and false negatives (FN). TP refers to malware samples that have been correctly labeled, whereas FP refers to benign samples that have been mistakenly labeled as malware. Similarly, TN refers to benign samples that have been correctly labeled, whereas FN refers to malware samples that have been mistakenly labeled as benign. In addition to the base definitions, we utilized accuracy, precision, recall, and  $F_1$ -score as dependent variables to assess the overall predictive performance of the O-CNN and state-of-the-art detection models.

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

$$\text{Precision} = \frac{TP}{TP + FP}$$

$$\text{Recall} = \frac{TP}{TP + FN}$$

(12)

$$F_1 - \text{Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

## 4. Empirical Evaluation and Discussion

This section examines the impact of feature extraction descriptors on predictive models. First, the proposed O-CNN model is tested on both  $224 \times 224$  and  $300 \times 300$ -dimensional images. Second, the optimal feature set is applied to several predictive models for state-of-the-art comparison. The best outcome is further compared to existing studies. Finally, the selected features are visualized using t-SNE and overall performance on both image dimensions is evaluated.

**4.1. Malware Detection via Proposed O-CNN Model.** We examine the impact of  $224 \times 224$  and  $300 \times 300$ -dimensional malware images on malware detection. Table 1 compares performance of three feature sets on O-CNN model. The

TABLE 1: Comparison of detection performance of DCNN model on three types of feature sets.

Feature	Dimension	Accuracy	Sample	Precision	Recall	F <sub>1</sub> -score
<b>LBP</b>	224 × 224	0.9297	Malware	0.90	0.61	0.73
			Benign	0.93	0.99	0.96
	300 × 300	0.9399	Malware	0.97	0.96	0.96
			Benign	0.80	0.82	0.81
<b>GLCM</b>	224 × 224	0.8873	Malware	0.95	0.28	0.43
			Benign	0.88	1.00	0.94
	300 × 300	0.8929	Malware	0.87	0.36	0.50
			Benign	0.89	0.99	0.94
<b>LBP + GLCM</b>	224 × 224	0.9780	Malware	0.97	0.98	0.98
			Benign	0.98	0.97	0.98
	300 × 300	0.9807	Malware	0.98	0.99	0.98
			Benign	0.99	0.98	0.98

accuracy of all three feature sets is above 88% on both dimensions. The lowest accuracy is achieved by GLCM feature set on 224 × 224, while the highest accuracy is achieved by combining LBP + GLCM feature set on 300 × 300 dimensional images. The combined approach outperforms LBP and GLCM descriptors by achieving above 97% accuracy on both image dimensions. The main purpose of this study is to categorize malware and benign samples; therefore, we distinctly measure the precision, recall, and f1-score of malware and benign categories. GLCM recall value is pretty low compared to other descriptors, which is 0.28 on 224 × 224-dimensional images and 0.36 on 300 × 300-dimensional images. The observation shows that a large number of benign files were incorrectly classified as malware samples, even though the precision and recall of their benign samples are above 88%. LBP recall is 0.61 on 224 × 224-dimensional images, while the recall on 300 × 300-dimensional images is 0.96, respectively. From this observation, we conclude that local features are more effective in detecting malware patterns compared to global textural features. However, obfuscation and encryption strategies can evade detection. As a result, some textural global features can facilitate predictive model to effectively detect obfuscated malware samples.

The fine-tuned parameters are used to optimize the predictive performance of deep learning models. As a result, the model accuracy and model loss are observed on number of epochs to visualize potential overfitting or underfitting of deep learning models. We observed train and test accuracy of all three feature sets on 200 epochs for 224 × 224 and 300 × 300-dimensional images as shown in Figure 6. In comparison to LBP and GLCM feature sets, combine LBP + GLCM strategy fits train and test accuracy more effectively on proposed O-CNN model. In Figure 6(a), the LBP train and test accuracy begins at 0.86 and remains between 0.92 and 0.99 after 20 epochs, whereas the model accuracy of train data is 5% higher than model accuracy of test data. In Figure 6(b), the model accuracy of GLCM feature set is between 0.85 and 0.91, respectively. The overall model accuracy of LBP has shown to be better than GLCM. In Figure 6(c), the accuracy of combined LBP + GLCM feature set has shown to be most effective than all. The train and test model perfectly fit on each other without any visible differences. This observation showed that the fine-tuned

parameters of O-CNN model can detect more unknown malware samples when local and global features of LBP and GLCM are deployed together.

A model loss is an outcome of a wrong prediction by a deep learning model. We use loss function to indicate how bad a model predicts on each test sample. Higher loss indicates worst prediction model for given samples. Figure 7 shows the train and test loss generated by proposed O-CNN model for three types of feature sets. In case of LBP feature set, the model loss of train and test samples have a difference of 30%. The lowest loss generated by train samples is 0.03, while the lowest loss generated by test samples is 0.30 on 200 epochs. In case of GLCM feature set, the difference in model loss of train and test samples is constant throughout epoch iterations that is around 9%. The combined LBP + GLCM approach proved to be more effective by outperforming single feature descriptors. The train loss of combined LBP + GLCM feature set is less than 0.10, while the test loss of combined LBP + GLCM feature set is less 0.20 only.

A confusion matrix visualizes correct and incorrect samples predicted by deep learning models. Figure 8 shows a normalized confusion matrix of our best feature set combined LBP + GLCM on both 224 × 224 and 300 × 300-dimensional images using proposed O-CNN model. In the case of 224 × 224-dimensional images, the misclassification of benign samples is higher than malware samples. In confusion matrix, 3% benign samples were misclassified as malware, while 2% malware samples were misclassified as benign. For 300 × 300-dimensional images, the performance is further improved by achieving only 2% misclassification on benign samples and 1% misclassification on malware samples. From the observations of Figures 6–8, we conclude that the combined LBP + GLCM on 300 × 300-dimensional images is an optimal choice.

*4.2. Performance Comparison with Related Works.* We implemented optimal feature set on state-of-the-art machine learning and deep learning classifiers for comparative evaluation of proposed DCNN model. Table 2 presents the performance of optimal feature set on five machine learning and four deep learning algorithms. The overall accuracy of combined LBP + GLCM feature set is above 80%. The lowest

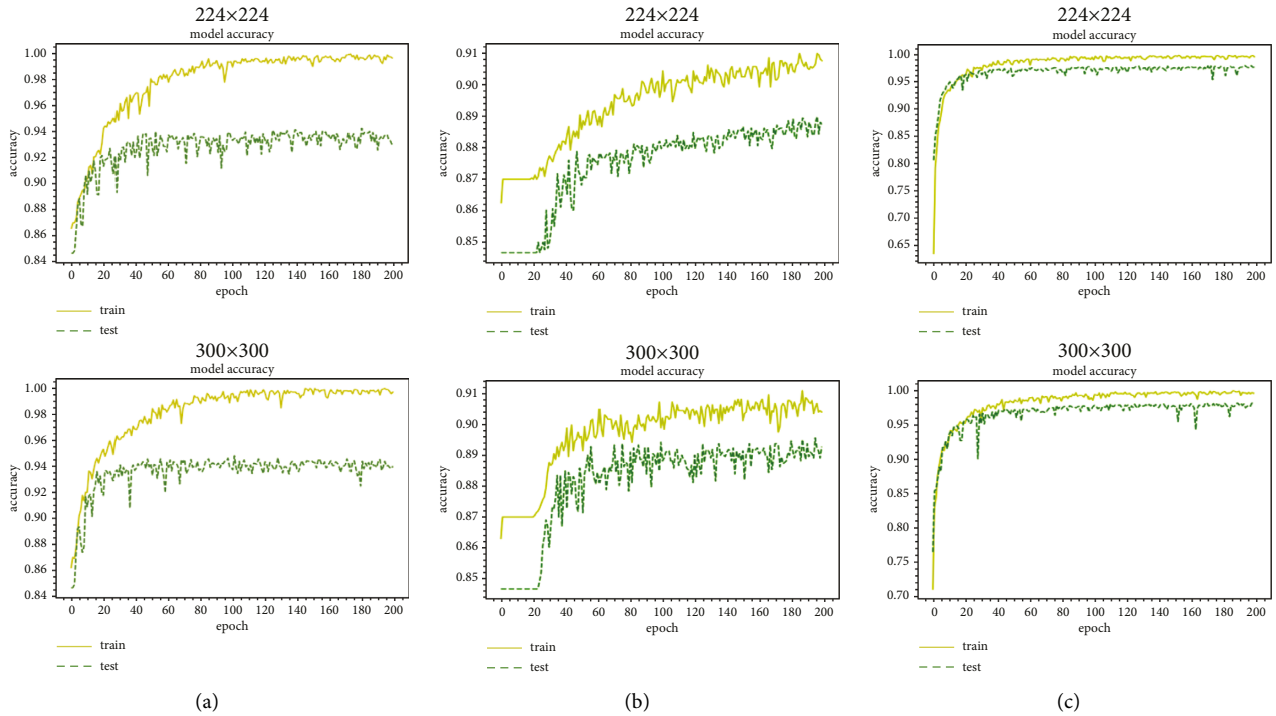


FIGURE 6: Train and test accuracy of three feature sets on proposed DCNN model. (a) LBP. (b) GLCM. (c) LBP + GLCM.

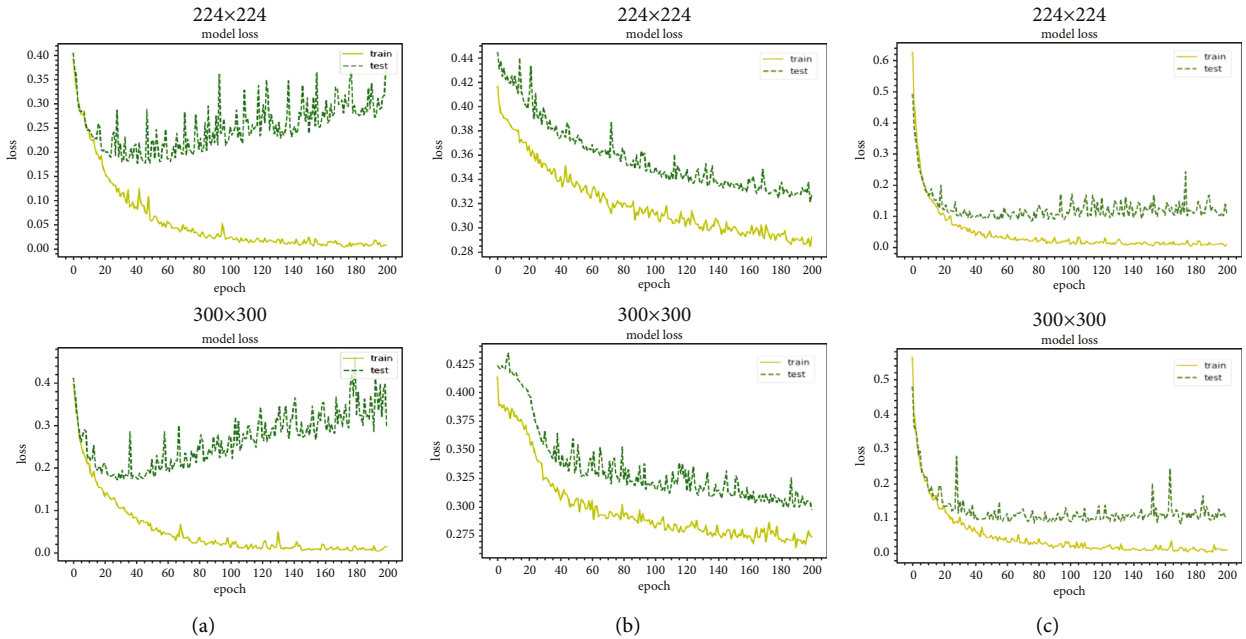


FIGURE 7: Train and test loss of three feature sets on proposed DCNN model. (a) LBP. (b) GLCM. (c) LBP + GLCM.

accuracy is achieved by naïve bayes classifier that is also above 80.4%, whereas the highest accuracy is achieved by the proposed O-CNN model. The proposed model outperforms other classifiers on all performance indicators. The average precision, recall, and  $f_1$ -score is 98% and model loss is 10%, respectively. In this observation, we conclude that the optimal feature not only performs efficiently on proposed model but also flexible to other predictive classifiers.

The optimal outcomes generated by proposed O-CNN model are compared to similar studies directed on various image dimensions. For instance, Nataraj et al. [16] used multidimensional images spanning between  $32 \times 32$  to  $1024 \times 1024$ , and Dai et al. [10] used  $2048 \times 2048$  and  $4096 \times 4096$ -dimensional images to implement their predictive models. Since the large images generate more features, therefore, enhancing the predictive performance of

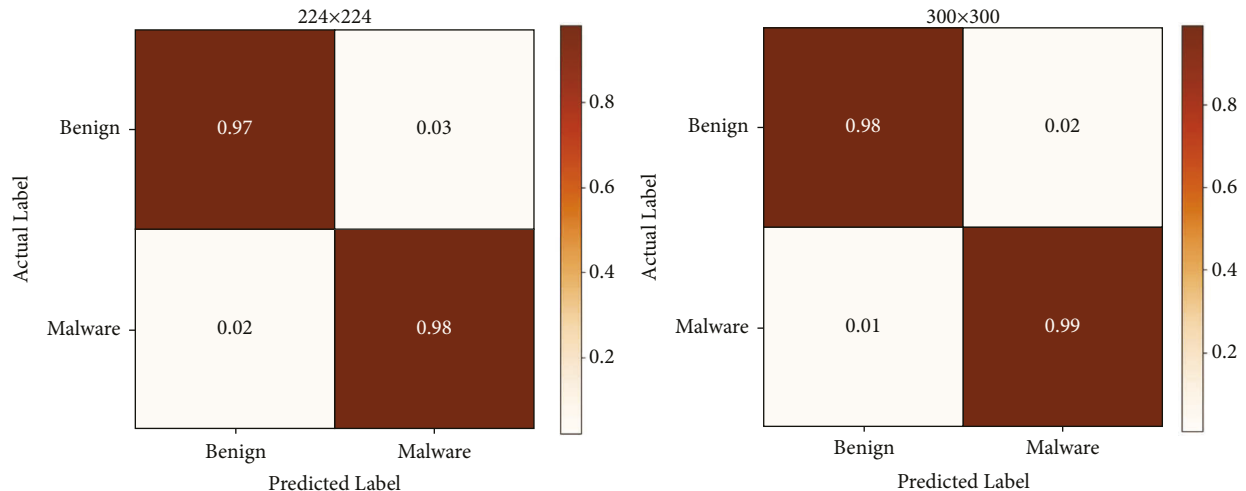


FIGURE 8: Confusion matrix of LBP + GLCM for  $224 \times 224$  and  $300 \times 300$ -dimensional images.

TABLE 2: Comparison of state-of-the-art predictive models on optimal feature set

Classifier	Accuracy (%)	Precision (%)	Recall (%)	F <sub>1</sub> -score (%)	Loss (%)
Logistic reg.	89.4	89.8	89.4	87.1	N/A
Naïve bayes	80.4	85.4	80.4	82.2	N/A
K-near neighbor	94.8	94.7	94.8	94.5	N/A
Decision tree	87.1	86.8	87.1	86.9	N/A
Random forest	91.5	91.4	91.5	90.4	N/A
DNN	84.7	72.0	85.0	78.0	43.0
GRU	94.7	95.0	95.0	95.0	20.0
RNN	85.9	91.0	86.0	87.0	36.0
LSTM	94.5	94.0	95.0	94.0	22.0
Proposed model	<b>98.1</b>	<b>98.0</b>	<b>98.0</b>	<b>98.0</b>	<b>10.0</b>

TABLE 3: Comparison of proposed DCNN model with other reference studies.

Study	Dimension	Accuracy (%)	Precision (%)	Recall (%)	F <sub>1</sub> -score (%)
Nataraj et al. [16] (2011)	$32 \times 32$ $1024 \times 1024$	91.4	91.5	91.4	91.5
Dai et al. [10] (2018)	$2048 \times 2048$ $4096 \times 4096$	94.5	94.6	94.5	94.5
Rezende et al. [32] (2018)	$224 \times 224$	96.9	97.0	96.9	96.9
Bozkir et al. [25] (2021)	$224 \times 224$ $300 \times 300$	96.3	96.4	96.4	96.4
Our method (optimal)	<b><math>224 \times 224</math></b> <b><math>300 \times 300</math></b>	<b>98.1</b>	<b>98.0</b>	<b>98.0</b>	<b>98.0</b>

deep learning models. Unfortunately, the larger images required more computational time and resources to extract all features. Rezende et al. [32] and Bozkir et al. [25] used  $224 \times 224$  and  $300 \times 300$  and generate more than 96% accuracy on four performance indicators. Table 3 shows the performance of proposed model compared to reference studies. In comparison to related studies, our combined LBP + GLCM strategy outperforms them by achieving more than 98% accuracy on same performance indicators.

**4.3. t-SNE Visualization and Performance of Dimensionality Reduction.** The t-SNE algorithm is an alternative way for cross-validation that does not require any data training like

supervised algorithms. As an unsupervised algorithm, t-SNE does not use labels to classify but only reveals the structure of feature set and similarity between data points. t-SNE use perplexity value to balance local and global aspects of data on resulting plot. In general, it is observed that lower the perplexity value, the more local structure is preserved whereas with higher perplexity more global structure is preserved. We applied t-SNE visualization on both LBP, GLCM, and combined LBP + GLCM feature sets. We visualize the best separation of malware and benign classes by providing best perplexity values such as 10, 30, 70, and 100, respectively. In Figure 9, the t-SNE plot shows only two clusters for malware and benign samples, which indicates low-dimensional t-SNE data have the ability to categorize



FIGURE 9: t-SNE visualization and dimensional reduction using optimal perplexity values on LBP, GLCM and combine LBP + GLCM feature sets.

TABLE 4: Comparison of proposed O-CNN model with other reference studies.

MalwareVariants	Before t-SNE reduction				After t-SNE reduction			
	224 × 224		300 × 300		224 × 224		300 × 300	
	Correct	Incorrect	Correct	Incorrect	Correct	Incorrect	Correct	Incorrect
Adposhel	0.99	<b>0.01</b>	1.00	0.00	1.00	0.00	1.00	0.00
Allapple	0.95	<b>0.05</b>	0.94	<b>0.06</b>	1.00	0.00	1.00	0.00
Amonetize	0.97	<b>0.03</b>	0.96	<b>0.04</b>	0.98	<b>0.02</b>	0.98	<b>0.02</b>
Autorun	0.73	<b>0.27</b>	0.75	<b>0.25</b>	0.96	<b>0.04</b>	0.96	<b>0.04</b>
Browsefox	0.91	<b>0.09</b>	0.85	<b>0.15</b>	0.99	<b>0.01</b>	1.00	0.00
Dinwod	0.71	<b>0.29</b>	0.89	<b>0.11</b>	1.00	0.00	1.00	0.00
Installcore	0.99	<b>0.01</b>	0.99	<b>0.01</b>	1.00	0.00	1.00	0.00
Multiplug	0.84	<b>0.16</b>	0.84	<b>0.16</b>	1.00	0.00	1.00	0.00
Vba	1.00	0.00	1.00	0.00	0.99	<b>0.01</b>	1.00	0.00
Vinsel	1.00	0.00	1.00	0.00	1.00	0.00	1.00	0.00
Average (%)	<b>0.90</b>	<b>0.09</b>	<b>0.92</b>	<b>0.08</b>	<b>0.99</b>	<b>0.01</b>	<b>0.99</b>	<b>0.01</b>
Time (s)	117s		119s		30 seconds (74.3x)		32 seconds (74.1x)	

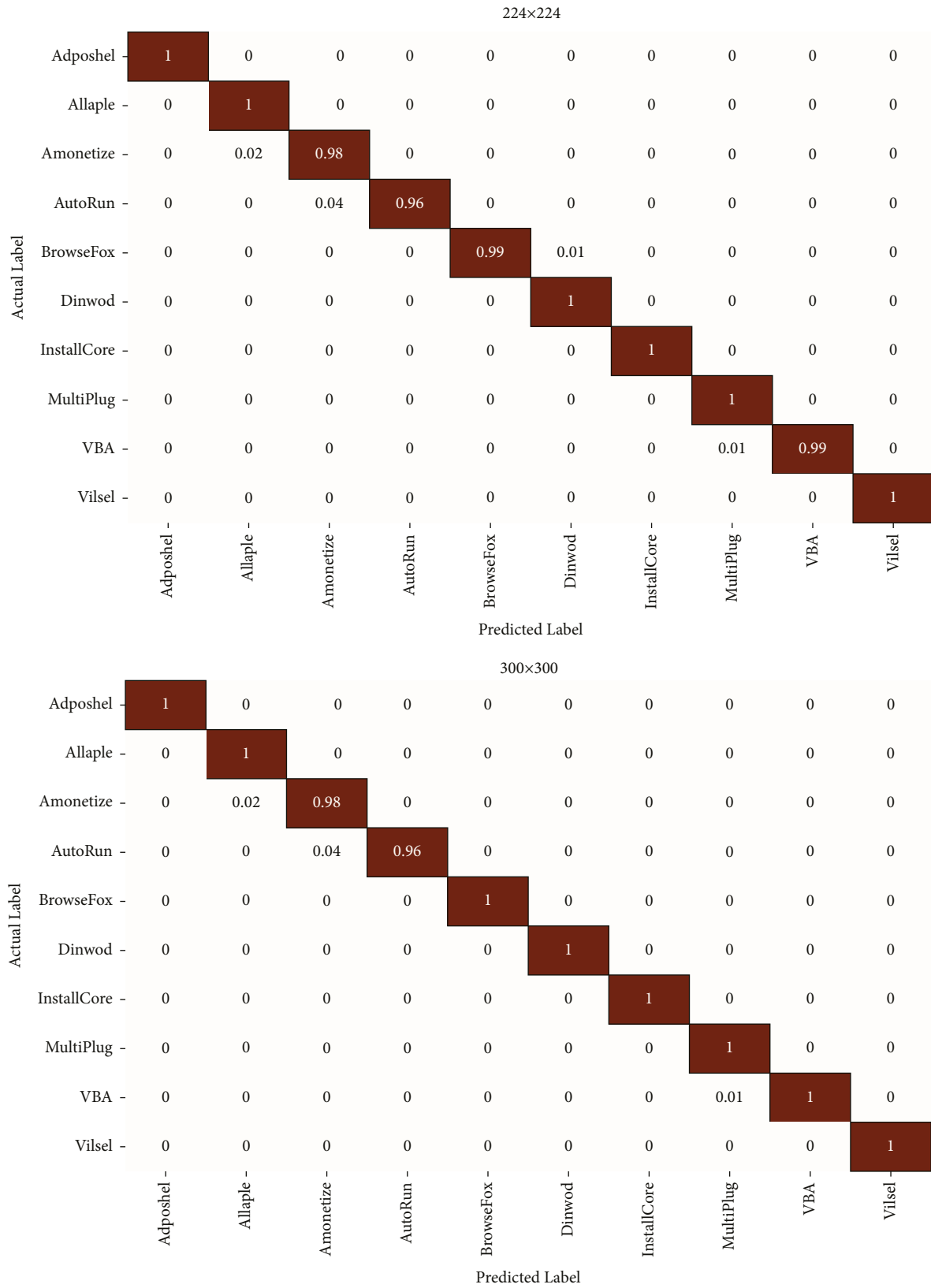


FIGURE 10: Confusion Matrix of proposed O-CNN model for 224 × 224 and 300 × 300-dimensional images.

samples into their respective categories. LBP feature set shows clear separation when perplexity value is 70 and 100. Comparatively, GLCM feature set shows clear separation at perplexity 30. One probable explanation is that the LBP feature set comprises of edges and small visual patches. Therefore, multiple cluster points are visible and separated from each other. Alternatively, GLCM feature set consists of visual content and textural characteristics. The combined LBP + GLCM consists of both edges and textural characteristics. All nonoverlapping clusters are clearly visible with the exception of a few outlier values. In general, t-SNE applies dimensionality reduction on data set and facilitate exploratory analysis for appropriate selection of parameters. The visual clusters formed by t-SNE are more isolated that can improve classification performance.

The performance of t-SNE dimensionality reduction on malware detection and variant classification is further evaluated for proposed O-CNN model. Table 4 presents the classification outcomes before and after using t-SNE reduction on both  $224 \times 224$  and  $300 \times 300$ -dimensional images. Before t-SNE reduction, the average percentage of correct classifications are greater than 0.90 on both dimensional images. However, few samples from some malware variants are also misclassified. For instance, *Autorun* variant has more than 25% misclassified samples. After t-SNE reduction, the classification is improved by producing only 0.01 misclassified variants. In the case of *Autorun* variant, the misclassified samples are also reduced from 25% to 2%. Furthermore, the average detection and variant classification time is also improved by 74.3x and 74.1x on for both dimensional images. From this observation, we conclude that t-SNE dimensionality reduction not only improves the accuracy of malware variant classification but also optimizes the malware detection time which will be the major requirement in windows devices and high availability servers. Lastly, Figure 10 presents the confusion matrices of  $224 \times 224$  and  $300 \times 300$ -dimensional images generated using proposed O-CNN model after t-SNE reduction. The figure shows that the majority of malware variant classes are effectively detected with less than 2% to 4% accuracy loss.

## 5. Conclusion

Malware detection is a critical security issue for Windows users who interact with the Internet on a regular basis. Because of the popularity of 5G Internet devices and the growing number of Internet users, malware attacks have become a big threat for Windows devices. This study has focused on performing memory forensics in order to detect malware attacks and their potential variants. First, we extracted local and global features using LBP and GLCM feature descriptors from textual images of memory dump files. Next, the proposed O-CNN model was fine-tuned on fused features of LBP and GLCM descriptors to detect malware and benign samples from test models. The widely studied performance indicators were selected to compare proposed methods with state-of-the-art models and related studies. The empirical evaluation showed that the proposed O-CNN model achieved above 98% accuracy on individual malware and benign samples.

Furthermore, t-SNE dimensionality reduction is applied on different feature sets to visualize malware and benign samples as isolated visual clusters. t-SNE feature set achieved upto 99% accuracy on malware variant classification as well as improved the training time of O-CNN by 74%. We believe that memory forensics store execution information of malware samples into RAM which significantly facilitate cybersecurity analyst to detect malware presence regardless of obfuscation and encryption. In the future, we planned to develop a combined blockchain and memory less malware detection model to overcome malware processing cost under limited resource capabilities.

## Data Availability

The malware data set investigated during the current study is available in the *Dumpware10* repository (<https://web.cs.hacettepe.edu.tr/~selman/dumpware10/>).

## Conflicts of Interest

The authors declare that they have no conflicts of interest.

## References

- [1] A. Damodaran, F. D. Troia, C. A. Visaggio, T. H. Austin, and M. Stamp, "A comparison of static, dynamic, and hybrid analysis for malware detection," *Journal of Computer Virology and Hacking Techniques*, vol. 13, no. 1, pp. 1–12, 2017.
- [2] Y. Ye, T. Li, D. Adjeroh, and S. S. Iyengar, "A survey on malware detection using data mining techniques," *ACM Computing Surveys*, vol. 50, no. 3, pp. 1–40, 2018.
- [3] R. Sihwail, K. Omar, K. A. Zainol Ariffin, and S. Al Afghani, "Malware detection approach based on artifacts in memory image and dynamic analysis," *Applied Sciences*, vol. 9, no. 18, p. 3680, 2019.
- [4] D. Gibert, C. Mateu, and J. Planes, "The rise of machine learning for detection and classification of malware: research developments, trends and challenges," *Journal of Network and Computer Applications*, vol. 153, Article ID 102526, 2020.
- [5] I. Ullah, M. A. Khan, F. Khan et al., "An efficient and secure multimedias and multireceiver signcryption scheme for edge-enabled internet of vehicles," *IEEE Internet of Things Journal*, vol. 9, no. 4, pp. 2688–2697, 2022.
- [6] Y. Cheng, W. Fan, W. Huang, and J. An, "A shellcode detection method based on full native api sequence and support vector machine IOP Conference Series: materials Science and Engineering," *IOP Conference Series: Materials Science and Engineering*, vol. 242, no. 1, Article ID 012124, 2017.
- [7] B. Alhayani, H. Jasim Mohammed, I. Zeghaiton Chaloob, and J. Saleh Ahmed, "Effectiveness of artificial intelligence techniques against cyber security risks apply of IT industry," *Materials Today Proceedings*, 2021.
- [8] H. Naeem, B. Guo, F. Ullah, and M. R. Naeem, "A cross-platform malware variant classification based on image representation," *KSII Transactions on Internet and Information Systems (TIIS)*, vol. 13, no. 7, pp. 3756–3777, 2019.
- [9] M. A. Khan, H. Shah, S. U. Rehman et al., "Securing internet of drones with identity-based proxy signcryption," *IEEE Access*, vol. 9, pp. 89133–89142, 2021.
- [10] Y. Dai, H. Li, Y. Qian, and X. Lu, "A malware classification method based on memory dump grayscale image," *Digital Investigation*, vol. 27, pp. 30–37, 2018.

- [11] J. Clement, "Share of global mobile website traffic 2015-2020," *Statista: Mobile Internet Usage Worldwide*, 2020, <https://www.statista.com/statistics/277125/share-of-website-traffic-coming-from-mobile-devices/>.
- [12] K. S. Han, J. H. Lim, B. Kang, and E. G. Im, "Malware analysis using visualized images and entropy graphs," *International Journal of Information Security*, vol. 14, no. 1, pp. 1–14, 2015.
- [13] N. Barath, D. Ouboti, and M. Temesguen, "Pattern recognition algorithms for malware classification," in *Proceedings of the 2016 IEEE Conference of Aerospace and Electronics*, pp. 338–342, Dayton, OH, USA, July 2016.
- [14] B. Xiaofang, C. Li, H. Weihua, and W. Qu, "Malware variant detection using similarity search over content fingerprint," in *Proceedings of the 26th Chinese Control and Decision Conference (2014 CCDC)*, pp. 5334–5339, IEEE, Changsha, China, May 2014.
- [15] A. S. Bozkir, A. O. Cankaya, and M. Aydos, "Utilization and comparison of convolutional neural networks in malware recognition," in *Proceedings of the 2019 27th Signal Processing and Communications Applications Conference (SIU)*, pp. 1–4, IEEE, Sivas, Turkey, April 2019.
- [16] L. Nataraj, S. Karthikeyan, G. Jacob, and B. S. Manjunath, "Malware images: visualization and automatic classification," in *Proceedings of the 8th International Symposium on Visualization for Cyber Security*, pp. 1–7, Pittsburgh, Pennsylvania, USA, July 2011.
- [17] F. Ullah, H. Naeem, S. Jabbar et al., "Cyber security threats detection in internet of things using deep learning approach," *IEEE Access*, vol. 7, pp. 124379–124389, 2019.
- [18] J. Hemalatha, S. A. Roseline, S. Geetha, S. Kadry, and R. Damaševičius, "An efficient DenseNet-based deep learning model for malware detection," *Entropy*, vol. 23, no. 3, p. 344, 2021.
- [19] R. Ronen, M. Radu, C. Feuerstein, E. Yom-Tov, and M. Ahmadi, "Microsoft malware classification challenge," 2018, <https://arxiv.org/abs/1802.10135>.
- [20] A. Nappa, M. Z. Rafique, and J. Caballero, "The MALICIA dataset: identification and analysis of drive-by download operations," *International Journal of Information Security*, vol. 14, no. 1, pp. 15–33, 2015/02/01 2015.
- [21] F. Ullah, M. R. Naeem, H. Naeem, X. Cheng, and M. Alazab, "CroLSSim: cross-language software similarity detector using hybrid approach of LSA-based AST-MDrep features and CNN-LSTM model," *International Journal of Intelligent Systems*, vol. 37, no. 9, pp. 5768–5795, 2022/09/01 2022.
- [22] F. Ullah, S. Ullah, M. R. Naeem, L. Mostarda, S. Rho, and X. Cheng, "Cyber-threat detection system using a hybrid approach of transfer learning and multi-model image representation," *Sensors*, vol. 22, no. 15, p. 5883, 2022.
- [23] V. Rey, P. M. Sánchez Sánchez, A. Huertas Celdrán, and G. Bovet, "Federated learning for malware detection in IoT devices," *Computer Networks*, vol. 204, Article ID 108693, 2022.
- [24] Y. Dai, H. Li, Y. Qian, R. Yang, and M. Zheng, "SMASH: a malware detection method based on multi-feature ensemble learning," *IEEE Access*, vol. 7, pp. 112588–112597, 2019.
- [25] A. S. Bozkir, E. Tahillioglu, M. Aydos, and I. Kara, "Catch them alive: a malware detection approach through memory forensics, manifold learning and computer vision," *Computers & Security*, vol. 103, Article ID 102166, 2021.
- [26] T. Ojala, M. Pietikainen, and T. Maenpaa, "Multiresolution gray-scale and rotation invariant texture classification with local binary patterns," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 7, pp. 971–987, 2002.
- [27] R. M. Haralick, K. Shanmugam, and I. H. Dinstein, "Textural features for image classification," *IEEE Transactions on systems, man, and cybernetics*, vol. 6, pp. 610–621, 1973.
- [28] L. Van der Maaten and G. Hinton, "Visualizing data using t-SNE," *Journal of Machine Learning Research*, vol. 9, no. 11, 2008.
- [29] C. R. García-Alonso, L. M. Pérez-Naranjo, and J. C. Fernández-Caballero, "Multiobjective evolutionary algorithms to identify highly autocorrelated areas: the case of spatial distribution in financially compromised farms," *Annals of Operations Research*, vol. 219, no. 1, pp. 187–202, 2014.
- [30] N. Pezzotti, B. P. F. Lelieveldt, L. v. d. Maaten, T. Höllt, E. Eisemann, and A. Vilanova, "Approximated and user steerable tSNE for progressive visual analytics," *IEEE Transactions on Visualization and Computer Graphics*, vol. 23, no. 7, pp. 1739–1752, 2017.
- [31] J. Bouvrie, *Notes on Convolutional Neural Networks*, Cambridge, MA, USA, 2006.
- [32] E. Rezende, G. Ruppert, T. Carvalho, A. Theophilo, F. Ramos, and P. de Geus, "Malicious software classification using VGG16 deep neural network's bottleneck features," *Information Technology-New Generations*, vol. 738, pp. 51–59, 2018.