International Information and Engineering Technology Association
Advancing the World of Information and Engineering

Traitement du Signal
Vol. 39, No. 3, June, 2022, pp. 991-1003
Journal homepage: http://iieta.org/journals/ts

Android Device Malware Classification Framework Using Multistep Image Feature Extraction and Multihead Deep Neural Ensemble

Hamad Naeem[1*], Amjad Alsirhani[2,3], Mohammed Mujib Alshahrani[4], Abdullah Alomari[5]

[1] School of Computer Science and Technology, Zhoukou Normal University, Zhoukou 466001, China
[2] College of Computer and Information Sciences, Jouf University, Sakaka, Aljouf 72388, Saudi Arabia
[3] Faculty of Computer Science, Dalhousie University, Halifax B3H 4R2, NS, Canada
[4] College of Computing and Information Technology, University of Bisha, Bisha 61361, Kingdom of Saudi Arabia
[5] Department of Computer Science, Albaha University, Albaha 65799, Saudi Arabia

Corresponding Author Email: hamadnaeem@zknu.edu.cn

https://doi.org/10.18280/ts.390326

Received: 28 February 2022
Accepted: 26 May 2022

Keywords:
cybersecurity, android image, multihead ensemble, malware visualization, threat detection

ABSTRACT

The incidence of malicious threats to computer systems has increased with the increasing use of Android devices and high-speed Internet. Malware visualization mechanism can analyze a computer whenever a software or system crash occurs because of malicious activity. This paper presents a new malware classification approach to recognize such Android device malware families by capturing suspicious processes in the form of different size color images. Important local and global characteristics of color images are extracted through a combined local and global feature descriptor (structure based local and statistical based global combined texture analysis) to reduce the training complexity of neural networks. A multihead ensemble of neural networks is proposed to increase network classification performance by merging prediction results from weak learners (convolutional neural network + gated recurrent unit) and using them as learning input to a multi-layer perceptron meta learner. Two public datasets of Android device malware are used to evaluate the classification and detection performance of the proposed approach. A baseline is established to compare the classification performance of the proposed approach with those of state-of-the-art and previous malware detection approaches. The proposed multihead ensemble improved the malware classification performance, with up to 97.8%, accuracy with the R2-D2 dataset and 94.1% accuracy with the MalNet dataset. The overall results show that a multihead ensemble with multi-step feature extraction is a practical approach to classify and detect Android malware.

1. INTRODUCTION

Digital Internet and information technology has progressed swiftly and plays an essential role in daily life and social activities. Alternatively, malicious software has also evolved with such advancements while posing new threats to digital devices. The traditional and state-of-the-art detection approaches are purely based on signature-based methods. Such methods depend on binary sequences generated by malicious activities to identify potential malware [1, 2]. Antimalware programs usually scan and match signatures of malware with other computer files. In general, malware detection using signature matching is effective by generating few false-positive (FP) outcomes. A signature-based malware detection strategy requires a short amount of time to extract potential malware and add its signatures to antimalware software. As a result, a computer system can be exposed to malware threats during extraction [3]. Heuristic algorithms that identify malicious activity have been designed to preserve the characteristics of suspicious programs. An unauthorized packer used to secure malware identify will be flagged as suspicious software aiming to obscure or manipulate original software signatures. Although heuristic approaches can identify emerging malware families, they may generate a significant number of false-positive outcomes. Different strategies have been proposed to overcome these limitations. These strategies can be classified into five categories, namely, static, dynamic, and hybrid, image analysis including memory forensics of malicious activities [4]. Static approaches are fast and clearly distinguishable from other types of malware detection strategies. These approaches generally examine code sequencing, byte structures, bytecode, executable commands, and other key characteristics of suspected applications from portable executables [5]. Such characteristics of a suspected application, often referred as "signatures," are algorithms or distinctive hashes used to differentiate one malware from another as well in malware families. As a result, no actual malware execution or resource consumption is involved throughout the detection process, resulting in the fast detection of malicious applications. However, static approaches have a few drawbacks, such as binary encryption and code obfuscation, which can easily outsmart static malware detection strategies [6]. Therefore, an alternative strategy may be employed to detect obfuscated and encrypted malware with a high true-positive ratio. Therefore, an alternative strategy may be employed to detect obfuscated and encrypted malware with a high true-positive ratio. Dynamic methods can virtually identify dynamic malware activities under a simulated environment. Nonetheless, dynamic methods are computationally expensive and time consuming. For instance, dynamic observations involve several strategies, such as method calls, method workflow tracing, method parameter analysis, and dynamic visualization of software command executions. Various analyzer tools, such as Anubis, TT analyzer, and CW Sandbox, are also available online for dynamic analysis and are widely studied. Although dynamic methods can identify obfuscated and encrypted malware with few false-positive outcomes, they entail more computations than traditional static analyses [7]. A machine or deep learning model captures and trains visual characteristics of malware binaries in the form of local and global features. Scale-invariant feature transform and speeded-up robust features provide the local textural properties of malware binaries. Some researchers have used combined strategies on global and local features to optimize malware detection and classification [8, 9]. Local binary patterns (LBPs) are resilient in capturing grayscale contrast to record fluctuations caused by unknown malware, whereas gray level co-occurrence matrix descriptors employ a degree of correlation among adjacent pixels to estimate separation distance and discrimination. Predictive outcomes of malware classification are effective, but new malware and their subgroups limit the prediction of unknown malware families. Apart from static, dynamic, and predictive hybrid approaches, memory forensics has attracted attention in recent years to overcome the limitations of previous approaches. For instance, instead of the static or dynamic analyses of malware binaries, a volatile memory dump can be generated from malicious processes. A memory dump contains all vital information regarding the structure and execution of malware commands. As a result, abundant discriminating information can be captured to classify malware and benign samples. Memory forensics works in two simple steps. First, the physical memory is converted into memory dump binaries. Second, malicious behavior and anomalies are analyzed in the form of visual images using textual descriptors. Smart approaches based on machine and deep learning models have been applied to classify actual malware families [6]. Key issues which must be solved in machine learning and malware detection areas are as follows. First, convolutional neural networks can extract features from large datasets of high-dimensional images and videos. However, the training process of these networks is complex, time consuming, and resource consuming because of high dimensional input. As a result, the need for these networks must be removed, especially in Internet-of-Things (IoT)-based solutions where the computing resources are less compared with CNN computational complexity [10, 11]. Second, there is no guarantee that the classifier can generalize to new or unknown data. Third, malware samples with obfuscation or encryption are difficult to identify using traditional malware detection methods such as static, dynamic, hybrid, and image analysis. The main contributions of this paper are listed as follows:

(1) A multihead neural ensemble that employs color image representation to store the malicious behavior of Android device malware executables is proposed to replace traditional malware signatures, which can be obfuscated and encrypted to escape detection.
(2) The training complexity of neural networks is reduced by combining deep and handcrafted malware feature extraction. Structure-based local and statistically-based global combined texture analysis is used to extract the handcrafted characteristics from the image.
(3) A multihead neural ensemble is proposed to increase network classification performance. The multi-head base networks (CNN+GRU base learners) embed into an extensive network (multilayer perceptron (MLP) meta learner), which learns from combined predictions from each input base network.
(4) The generalizability of proposed multihead ensemble is tested by adding a verification set through k fold cross validation.
(5) Multihead neural ensembles, CNN models and machine learning approaches are tested against the proposed multihead ensembles in order to verify the aforesaid contributions.

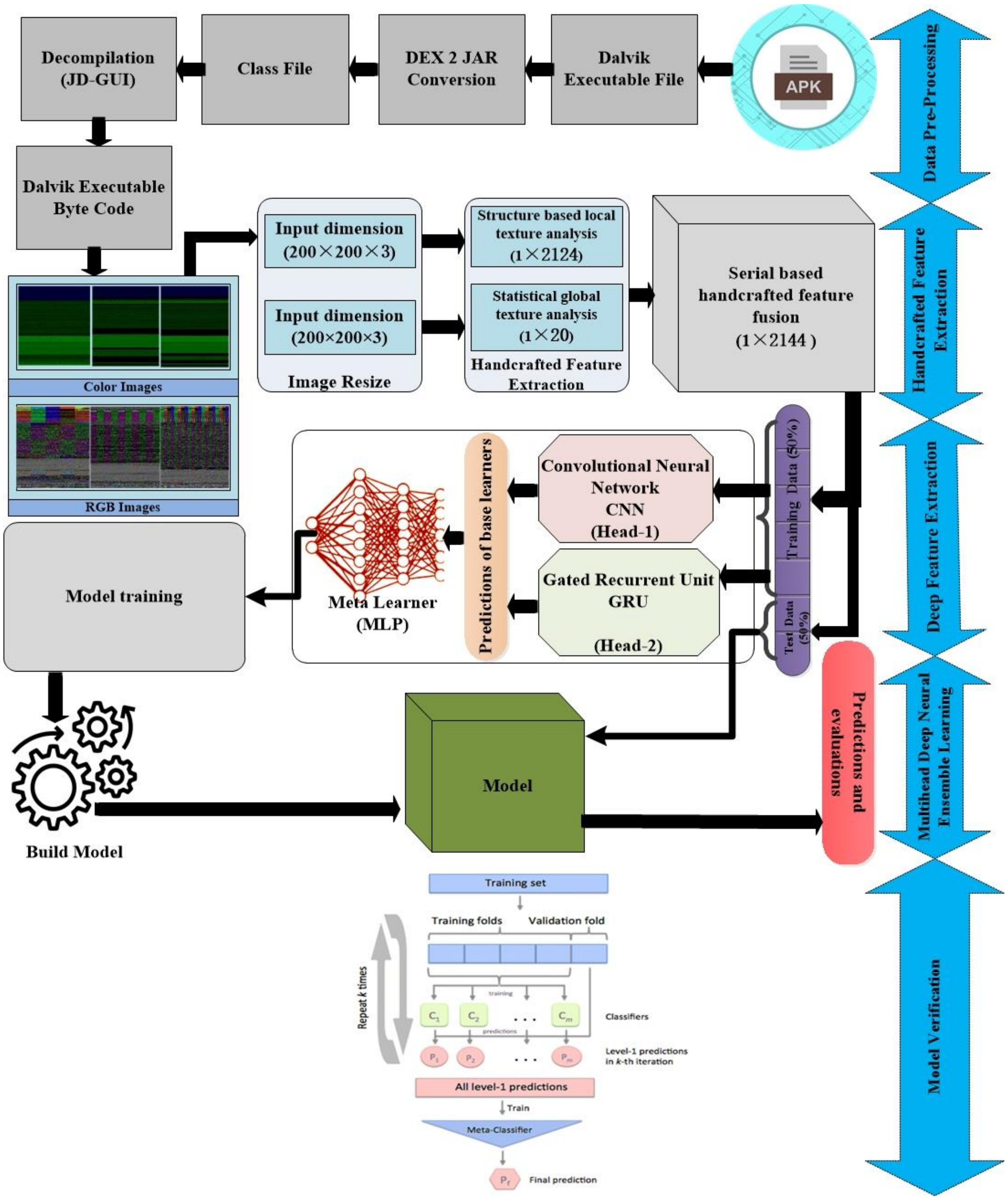
This study is structured as follows. The literature review is given in Section 2, the proposed malware classification approach is explained in Section 3, results and discussions are presented in Section 4, and the conclusion is provided in Section 5.

2. LITERATURE REVIEW

Malware detection studies used various methods to identify different categories and families of malicious software. Malware classification based on image processing is more effective than traditional methods. Image-based malware detection first applies image conversion to transform malware binaries into grayscale or color format. Then, the image features are extracted to train predictive models and detect malware and benign samples. Some advantages and disadvantages of image processing-based malware detection approaches are as follows.

Natraj et al. [12] were the first to apply computer vision to detect malware categories and their families. First, they transformed malware samples into grayscale visual images and designed a dataset of 9339 malicious samples into 25 malware families. A GIST descriptor was applied on the malware dataset to extract LBPs from grayscale images. Empirical evaluation showed that the predictive models achieved an accuracy of 97% on their dataset. Kalash et al. [13] further experimented and evaluated the dataset of Natraj et al. on the optimized CNN predative model. They randomly selected 10% test samples for different malware families and improved the malware detection accuracy by 98.52%. Han et al. [14] proposed a new strategy in order to transform malware binaries into grayscale images. Instead of using traditional transformation, a graph theory based on information entropy was used to find associations and malicious characteristics of malware samples. The empirical evaluation of 1000 malware samples and their families achieved 97.9% accuracy on supplied dataset. Few studies successfully applied feature reduction techniques to improve classification accuracy. Machine learning algorithms can produce equivalent results, but they are restricted by data and feasibility. Researchers have taken use of neural network models for various predictive applications. Ullah et al. [15] studied IoT-oriented industrial malware. The raw binaries of IoT malware were converted to colored images and trained using a CNN model to detect and classify malware samples. The empirical evaluation of IoT-oriented industrial malware achieved an accuracy of 97.46% on multiple malware families. Vasan et al. [16] further used a fine-tuned CNN model to optimize classification accuracy. The empirical evaluation found that fine-tuned CNNs can improve accuracy between 98% to 99% for packed and unpacked malware families. Latest malware studies investigated the concept of memory forensics to identify potential malicious software. Many malwares can remove their traces after successful execution of malicious codes. As a result, traditional malware detection strategies fail to detect such malware in a specified time. However, malware traces remain in volatile memory until the end process execution. Dai et al. [17] used volatile memory data from memory dumps generated from malicious codes to detect such malware and their families. They generated grayscale visual images of each malware and resized images based on bi-cubic interpolation. The empirical evaluation of their own dataset produces a malware detection accuracy of 96.7%. Bozkir et al. [18] further exploited memory forensics for malware detection by generating colored images instead of grayscale. They prepared a public dataset of 4294 malware and benign samples based on memory dumps. Feature descriptors GIST and HOG were used to extract textual features from malware images, and dimensionality reduction was applied on extracted features by using uniform manifold approximation and projection. The overall malware detection accuracy improved by 96.39%, whereas the detection time also improved by 3.56 s.

3. ARCHITECTURE OF ANDROID DEVICE MALWARE CLASSIFICATION FRAMEWORK



Decompilation (JD-GUI)
Class File
DEX 2 JAR Conversion
Dalvik Executable File
APK
Data Pre-Processing
Dalvik Executable Byte Code
Input dimension (200×200×3)
Input dimension (200×200×3)
Image Resize
Structure based local texture analysis (1×2124)
Statistical global texture analysis (1×20)
Handcrafted Feature Extraction
Serial based handcrafted feature fusion (1×2144)
Handcrafted Feature Extraction
Color Images
RGB Images
Training Data (50%)
Test Data (50%)
Convolutional Neural Network CNN (Head-1)
Gated Recurrent Unit GRU (Head-2)
Predictions of base learners
Meta Learner (MLP)
Model training
Deep Feature Extraction
Predictions and evaluations
Multihead Deep Neural Ensemble Learning
Build Model
Model
Training set
Training folds
Validation fold
Repeat k times
training
C1
C2
Cm
Classifiers
predictions
P1
P2
Pm
Level-1 predictions in k-th iteration
All level-1 predictions
Train
Meta-Classifier
Pf
Final prediction
Model Verification

Figure 1. Architecture of Android device malware classification framework

The proposed malware classification consists of three main parts: (a) Android device malware data acquisition with image visualization (b) manual feature extraction (c) generalized multihead neural ensemble. Figure 1 displays the workflow of the suggested malware detection approach. The suggested method employs a multihead neural network ensemble along a manual feature extraction strategy for malware classification. At first step, malware files are pre-processed and visualized as images. At second step, structure based local and statistical based global combined texture analysis is performed by using conventional image descriptors, such as LBP and GLCM. At end, the multihead neural network ensemble utilizes the manual features as input for malware classification. Comprehensive details on each step are provided below.

3.1 Data pre-processing

Each apk file containing malware is unpacked so that it can be examined. Apk file also contains the Dalvik executable (DEX). The byte code is deduced from the apk file in three stages. The apk file is first decompressed to determine the class.DEX file. The class. DEX file is then transformed into a Java.class file by using the dex2jar tool. Finally, the JD-GUI decompiler is utilized to obtain the byte code from the Java. Class file, as shown in Figure 2.

The DEX (bytecode) of each Android app must be gathered prior to developing a feature representation of the applications and labels. A 1D array of 8-bit unsigned integers was created using the DEX file. Each item value in array ranges between 0 to 255, and it represents black and white pixels. One dimensional array is converted into a two-dimensional feature extraction format, data is encoded into RGB pixels, and pictures are scaled to the appropriate size; these are the three steps that are performed on every binary file. The 1D byte array is transformed into a 2D array by using conventional sequential charting techniques [10]. Nonetheless, the image's height can change based on the amount of information being displayed. To generate 256×256 and 200×200 pictures, we employ the Pillow library and a standard Lanczos filtration technique.

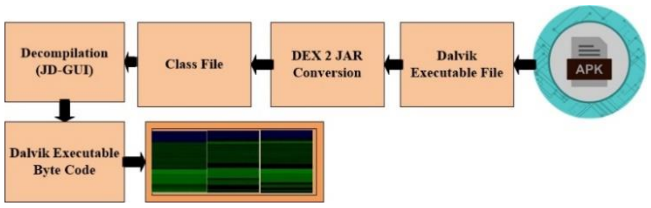


Figure 2. Android malware color image visualization

3.2 Malware sample acquisition

3.2.1 R2-D2 IoT device dataset [19]

It comprises RGB color images translated from the DEX files retrieved by decompressing approximately 2 million benign and malicious Android applications. Leopard Mobile Inc.'s original back-end detection system was used to gather these apps between January 2017 and August 2017. The infected programs belong to different malware types, including Trojans and Ad-Ware as well as Clickers and SMS Spyware. The Android color image dataset enables to save important information about Android apps with 16777216 colors per image compared with 8-bit grayscale images. The size of all images is reduced to 299 × 299. Mini-batch learning is facilitated by resizing pictures to 299 × 299, which allows computer vision models to be trained faster while meeting the computational limits. The image sizes are approximately 10–50 KB.

RGB Image Representation: Android APK is first decompressed to acquire the classes.dex file. This file's bytecode and the RGB color coding are recorded in hexadecimal. Hexadecimal from the DEX files are converted to RGB color coding, such that three-digit numbers are separated in left to right order. As a result, each of these integers is transformed into a decimal form and assigned to a specific code (R, G, or B). For example, 646778 is split into 64, 67, and 78, which are then converted into their decimal forms and assigned as (R:100, G:103; B:120). Finally, RGB images of Android devices are obtained and fed into CNNs for the malware detection of Android devices. Figure 3 shows the chunks of malware images from the R2-D2 IoT device dataset.

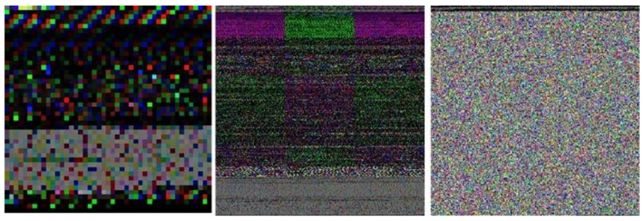


Figure 3. R2-D2 Android device dataset

3.2.2 MalNet IoT device dataset [20]

MalNet has provided 8633 IoT device malware samples of 19 families. The sample distribution of each training malware dataset family is given: Addisplay (1022), Addisplay++Adware (59), Adload (67), Adsware (530), adware++adware (501), adware++grayware++virus (167), adware++virus (55), Backdoor (121), banker++trojan (221), Adwareare (46), clicker (53), 'click (22), clicker++trojan (573), clickfraud++riskware (74), exploit (1116), fakeangry (42), fakeapp (85), fakeapp++trojan (51), and fakeinst++trojan (143), respectively.

Color Image Representation: Embedding semantic features is a complex process. Semantics can be extremely helpful in analyzing the bytecode of an application. For instance, a randomly selected byte could represent an ascii character, an opcode, or a portion of a pointer address. Through the use of color to differentiate each byte according to its function, the image gains an additional layer of semantic information on top of the raw bytecode. Even though various techniques can be used to encode semantic information into an image, a universally accepted standard technique is lacking. The contextual features are encoded by allocating each byte to a specific RGB color channel based on its position in the DEX file structure, (i) header, (ii) signifiers and class interpretations, and (iii) data, and by encoding the spatial meaning in binary form (Figure 4). The first phase in creating a feature representation of the Android applications and labels is to retrieve the DEX (bytecode) out of each application. A 1D array of 8-bit unsigned integers is then created from the DEX document. Each item value in array ranges between 0 to 255, and it represents black and white pixels. Then, for each binary file, 1D array is transformed to 2D feature extraction, information content is encoded into the RGB channels, and images are scaled to an appropriate dimension.

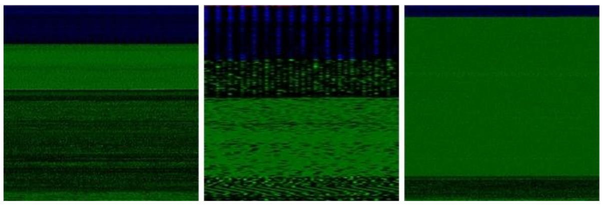


Figure 4. MalNet Android device dataset

3.3 Handcrafted feature extraction

The structure-based repetitive LBPs are retrieved by employing an LBP descriptor based on the textural properties of malware image. In addition, the GLCM feature descriptor is used to perform statistical global texture analysis for malware images. Below is a detailed overview of structure-based local texture analysis in conjunction with statistical global texture analysis. Manual features are extracted by using MATLAB. The feature analysis is performed on a machine with 16 GB of RAM and a 6 GB NVIDIA GeForce RTX 2060 GPU. A local feature set with 2124 dimensions is extracted through the LBP descriptor, and a global feature set with 20 dimensions is extracted through the GLCM descriptor. The combined feature vector contains 2144-dimensional local and global features.

3.3.1 Structure-based local texture analysis

Local features are extracted from malware images using an LBP textural descriptor. LBP description not only accurately recognizes micro-grayscale patterns in textual pictures, but it is also highly effective in extracting LBPs [21]. The binary patterns are textual properties that can be used to detect variations of visual patterns from color images. Textual properties, such as pixel direction, smoothness, surface roughness, and softness, are used to indicate variations among different textual images. Malware textual images consist of unstable patterns that require a strong feature descriptor to identify distinctive characteristics. As a result, an LBP textual descriptor is adopted to extract LBPs of malware images. To illustrate the internal structure of the LBP descriptor, Figure 5 is presented with a 3 × 3-pixel block to measure the intensity of adjacent pixels. The threshold value is assessed using the LBP descriptor to measure adjacent pixel intensity. The computational execution of the LBP descriptor is detailed in the form of a four-step procedure:

(1) Select surrounding pixels *P* within a given radius R for each pixel on the *x*- and *y*-axes.
(2) Calculate the *x*- and *y*-axis intensity difference between the current pixel and the surrounding pixels *P*.
(3) Select a threshold value for the surrounding pixels P and use the difference in intensity to assign 0 and 1 as single-bit values for the pixels.
(4) Replace the original intensity value of the current pixel with the decimal value derived from the bit sequence of the surrounding pixels *P*.

Decimal values for individual pixels are calculated using the following LBP equation.

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

where, *P* represents a set of adjacent pixels from a specified radius R and $g_p - g_c$ represents the intensity variation between the present pixel and its neighbors.

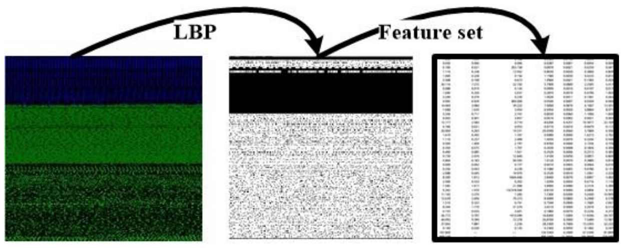


Figure 5. Visual representation of the LBP descriptor

3.3.2 Statistical global texture analysis

Global features are extracted from malware images using an Global textural descriptor. The GLCM descriptor exploits the spatial distribution of each pixel from textual images [22]. GLCM uses random position in textual images and examines the probability distribution of two grayscale pixels located apart at a certain distance. The GLCM descriptor uses three aspects to extract global characteristics: direction of pixel, variation in amplitude, and the integration of information at the grayscale level. The direction of pixels is the angle of change in grayscale from 0° to 135°, where pixel offset values and gray level orientation are both 1, which shows the grayscale contraction. Collectively, grayscale compression, co-occurrence matrix analysis, and final feature extraction make up the GLCM descriptor's global feature extraction. Figure 6 shows the extraction strategy of the GLCM descriptor. First, the colored image is transformed to grayscale and then subjected to pixel contraction to generate a 4-dimensional grayscale array. Second, the similarity of patterns is determined by looking for pixel pairs that appear together. Final set of features is extracted based on different properties, such as correlation, contrast, homogeneity, and energy, for both malware datasets.

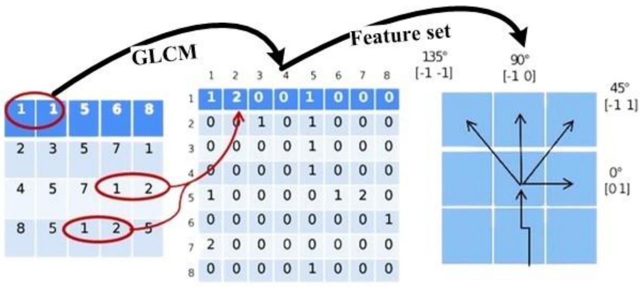


Figure 6. Visual representation of the GLCM descriptor

$$energy=\sum_{i,j} p(i,j)^2 \tag{2}$$

$$contrast=correlation=\sum_{i,j} |i-j|^2\, p(i,j) \tag{3}$$

$$homogeneity=\sum_{i,j} \frac{p(i,j)}{1+|i-j|} \tag{4}$$

Finally, the proposed malware classification approach selected serial-based feature fusion for concatenating the features of both LBP and GLCM descriptors. The resulting fused feature set provides a 2144-dimensional feature vector.

3.4 Deep feature extraction

Ensemble methods combine several different learning algorithms to improve overall prediction precision. These ensemble classifiers combine many classification models to reduce the risk of overfitting in the training results. Consequently, generalization efficiency can be enhanced because of the increased utilization of training data. Although various ensemble classification models have already been created, researchers can still increase sample classification accuracy, which is helpful for malware detection. Therefore, a multihead neural network ensemble is proposed to strengthen network classification performance. The formation of an ensemble of base classifiers is accomplished by first reshuffling training dataset, and then adding a base classifier to each reshuffled training dataset. An ensemble model is created by combining the predictions of base classifiers and learned from combined predictions of base classifiers. The proposed ensemble comprises two levels of learners, such as weak 1D CNN and GRU learners, and MLP meta learner. The proposed multihead neural ensemble consists of two base networks (CNN and GRU weak learners) and one meta classifier (MLP strong learner), which learns from combined predictions from each input base network.

3.4.1 Base networks

The proposed multihead neural ensemble selects two base networks namely 1D CNN and GRU for weak learning. The detail workflow of each base network is given below:

1D-CNN is selected for weak learning at this level. The input 2144-dimensional combined local–global feature set substantially assists in decreasing the training complexity of the CNN. The CNN captures more significant features with a subsequent reduction in the dimension. The 1D CNN contains two convolution layers, two pooling layers, one flattened, one dropout, and one fully connected layer, as shown in Figure 7. After sliding over the combined local-global feature set, the convolution layer filters retrieve the best deep features. A new feature set is generated as a consequence of applying each filter, which is denoted as a “feature map.” The max-pooling layer reduces spatial and feature size as well as computational complexity. The resulting feature set is further compressed using the flattened layer. The proposed CNN also includes a fully connected classification layer. In order to prevent overfitting in the suggested CNN, we employ the SoftMax activation function and a dropout layer. Eq. (5) is used to express the output of the proposed 1D CNN.

$$o_k^l = f\left(c_k^l + \sum_{i=1}^{N_{l-1}} Conv1D(X_{ik}^{l-1}, t_i^{l-1})\right) \qquad (5)$$

where, c_k^l represents the k^{th} neuron's scalar bias in the first layer, t_i^{l-1}denotes the output of ith neuron at layer l-1, $X_{ik}^{l-1}$is the kernel weight from the ith neuron at layer l-1 to kth neurons at layer l, and f denotes the activation function. The CNN and GRU networks are trained with 50% data.

GRU Network is also selected for weak learning at this step. The 2144-dimensional combined local-global feature set is used as input. GRU is a recurrent neural network that is highly similar to the long short-term memory (LSTM) but simpler in design. Instead of the three gates utilized by LSTM, the GRU uses two gates: update and reset gate. It also does not have a separate cell state or memory. Instead, it uses the hidden state for transferring information. The update gate serves the functions of the forget and input gates in that it decides what new information to consider and what information to forget. The reset gate is used for controlling the amount of past information to forget.

$$\begin{aligned} z_t &= \sigma_g\,(W_{zx_t} + U_z h_{t-1} + b_z) \\ r_t &= \sigma_g\,(W_{rx_t} + U_r h_{t-1} + b_r) \\ \hat{h}_t &= \phi_h(W_{hx_t} + U_h(r_t \otimes h_{t_1}) + b_h) \\ h_t &= ((1-z_t)\otimes h_{t-1}) + (z_t \otimes \hat{h}_t) \end{aligned} \qquad (6)$$

where, σ_g is the sigmoid function, ϕ_h is the hyperbolic tangent, x_t is the input vector, h_t is the output vector, $\hat{h}_t$ is the candidate activation vector, z_t is the update gate vector, r_t is the reset gate vector, W and U are the parameter matrices, and b is the bias vector.

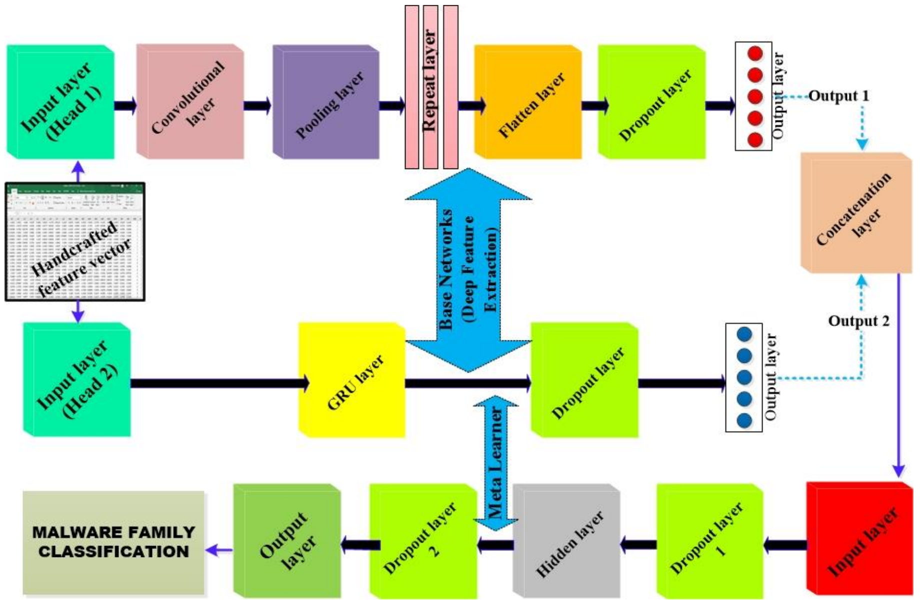


Figure 7. Multihead neural ensemble for malware classification

3.4.2 Meta learning

An MLP neural network performs the meta-learning process. The weak combined prediction results of base networks are used as the learning input of the MLP meta learner. A feedforward neural network with one input layer, one hidden layer, and one output layer is used in the proposed multihead ensemble, as shown in Figure 7. The first and last layers in the MLP model are used as input and output, respectively. However, the hidden layer uses a weighted input set for model learning and an activation function for output production. The computation of hidden layer H_i can be mathematically formulated as follows.

$$H_i(x) = f(w_i^T x + b_i) \tag{7}$$

where, $H_I : R^{d_{i-1}} \rightarrow R^{d_i}, f: R \rightarrow R, w_i \in R^{d \times d_{i-1}}, b_i \in R^{d_i}$. where, f is the element wise nonlinear function. The SoftMax activation function is selected with the output layer that displays the nonlinear neural network form.

4. RESULTS AND DISCUSSIONS

A confusion matrix was used to evaluate the proposed and other models. Confusion matrices were used to summarize the predictions of classifiers (also known as an error matrix). The confusion matrix included FP, false-negative (FN), and true-positive (TP) outcomes. For each false sample of malware, FP was counted, and the total number of false samples was reported as FN. These values were used to compute the following metrics:

$$\begin{gathered}\text{Precision} = \frac{\text{TP}}{\text{TP + FP}}, \text{Recall} = \frac{\text{FP}}{\text{FP + TN}}, \\ \text{F1-Score} = \frac{2 \times \text{TP}}{2 \times \text{TP + FP + FN}}, \text{Accuracy} = \frac{\text{TP+TN}}{\text{TP + TN + FP + FN}}\end{gathered} \tag{8}$$

Experiments were carried out to (1) validate the impact of the handcrafted feature set over different malware datasets, (2) evaluate the impact of layer numbers and tuning settings on the proposed multihead ensemble's training performance, (3) and assess the suggested method's accuracy in classifying data compared to other published studies. This experiment has 200 epochs, with a batch size of 32 and a learning rate of 0.01.

4.1 Impact of hand-crafted feature set over different malware datasets

The impact of the hand-crafted feature set on the classification performance of the proposed multihead ensemble approach was investigated. Table 1 displays the classification performance of individual feature and combined features across different malware datasets (R2-D2 and MalNet, respectively). The overall classification accuracy of the combined handcrafted feature set was above 50% for both malware datasets. The local binary feature set obtained classification accuracy (0.98 and 0.933, respectively) with two different malware datasets. The global statistical feature set obtained classification accuracy (0.917 and 0.521, respectively) with two different malware datasets. The combined feature set obtained classification accuracy (0.978 and 0.941, respectively) with two different malware datasets. The global statistical feature set and the local binary feature set obtained the lowest classification accuracy with the MalNet malware dataset. The combined extraction of LBPs and global statistical features from malware images significantly increased the classification performance. Hence, the combined set of features achieved better classification accuracy than the individual feature set across a variety of malware datasets, as shown in Table 1.

We determined the train and test performance (accuracy, loss and roc curve) of three different handcrafted features over the R2-D2 and MalNet malware datasets, as shown in Figures 8 and 9. The combined set of features better matched the train and test accuracy on the proposed multi head ensemble than the individual feature set. As shown in Figure 8, the training and testing accuracy curves of the local handcrafted features started at the 0-graph scale and increased between the 0.90 and 0.98 graph scales after the 14th epoch. Similarly, the training and testing loss curves of the local handcrafted features started at the 0.4 graph scale and decreased between the 0.088 and 0.0050 graph scales after the 40th epoch. The training and testing accuracy curves of the global statistical features started at the 0.7 graph scale and increased between the 0.79 and 0.91 graph scales after the 12th epoch. Similarly, the training and testing loss curves of the global statistical features started at the 0.45 graph scale and decreased between the 0.39 and 0.2 graph scales after the 9th epoch. The training and testing accuracy curves of the combined features started at the 0.7 graph scale and increased between the 0.79 and 0.99 graph scales after the 7th epoch. Similarly, the training and testing loss curves of the combined features started at the 0.47 graph scale and decreased between the 0.24 and 0.003 graph scales after the 9th epoch.

As shown in Figure 9, the training and testing curves of the local handcrafted features started at the 0.1 graph scale and increased between the 0.32 and 0.99 graph scales after the 9th epoch. Similarly, the training and testing loss curves of the local handcrafted features started at the 2.67 graph scale and decreased between the 2.3 and 0.02 graph scales after the 4th epoch. The training and testing accuracy curves of the global statistical features started at the 0.04 graph scale and increased between the 0.10 and 0.52 graph scales after the 3rd epoch. Similarly, the training and testing loss curves of the global statistical features started at the 2.94 graph scale and decreased between the 2.82 and 1.54 graph scales after the 2nd epoch. After the second epoch, the combined features' training and testing accuracy curves both begin at the 0.1 graph scale and increase to the 0.14 to 0.99 graph scale. Similarly, both the train and test loss curves for the combined features begin at a graph scale of 2.71 and fall between 2.4 to 0.01 scales after the second epoch. Table 2 shows the training and testing performance of the hand-crafted features with the different malware datasets.

The accuracy of each instance's prediction is shown in two normalized confusion matrices. The prediction accuracies of the instances are provided in two normalized confusion matrices, as shown in Figure 10. The confusion matrix of the R2-D2 dataset showed a malware detection rate of 0.96, which is close to the benign detection rate (0.99). Thus, none of the sample distributions in the two classes were biased. Compared with other malware families, the Addisplay and exploit families had the lowest classification rates (0.69 and 0.87, respectively) in the confusion matrix of the MalNet dataset.

Generalization is a key advantage of neural networks. It is the model's capacity to apply what it has learned to new and unknown data. The generalizability of proposed multihead ensemble was tested by adding a validation set and evaluating the model's performance on this additional data. For verifying the generalization of the proposed multihead ensemble, the R2-D2 dataset was separated into three subsets using threefold cross validation: train set (75%), the test set (25%), and validation set (10%). The generalizability of the proposed multihead ensemble was tested using threefold cross validation with a 10% validation set during the training phase. Table 3 shows the detection performance of the proposed multihead ensemble with three distinct subsets. The proposed multihead ensemble had a detection accuracy of 99.61% on the training set, 98.34% on the testing set, and 98.28% on the validation set. As shown in Table 3, the proposed multihead ensemble generalized not only the known and unseen sets well, but also the new set extremely effectively. In addition, the dynamic graphs of accuracy and loss through threefold cross validation demonstrate the generalizability of the proposed multihead ensemble, as shown in Figure 11.

Table 1. Impact of hand-crafted features on classification performance with different malware datasets

Feature	Dataset	Feature Type	Accuracy	Precision	Recall	F1-Score
Local	R2-D2	Binary Features	0.98	0.98	0.98	0.98
	MALNET		0.933	0.93	0.93	0.93
Global	R2-D2	Statistical Features	0.917	0.92	0.92	0.92
	MALNET		0.521	0.49	0.52	0.48
Local-Global	R2-D2	Binary+ Statistical Features	**0.978**	**0.98**	**0.98**	**0.98**
	MALNET		**0.941**	**0.94**	**0.94**	**0.94**

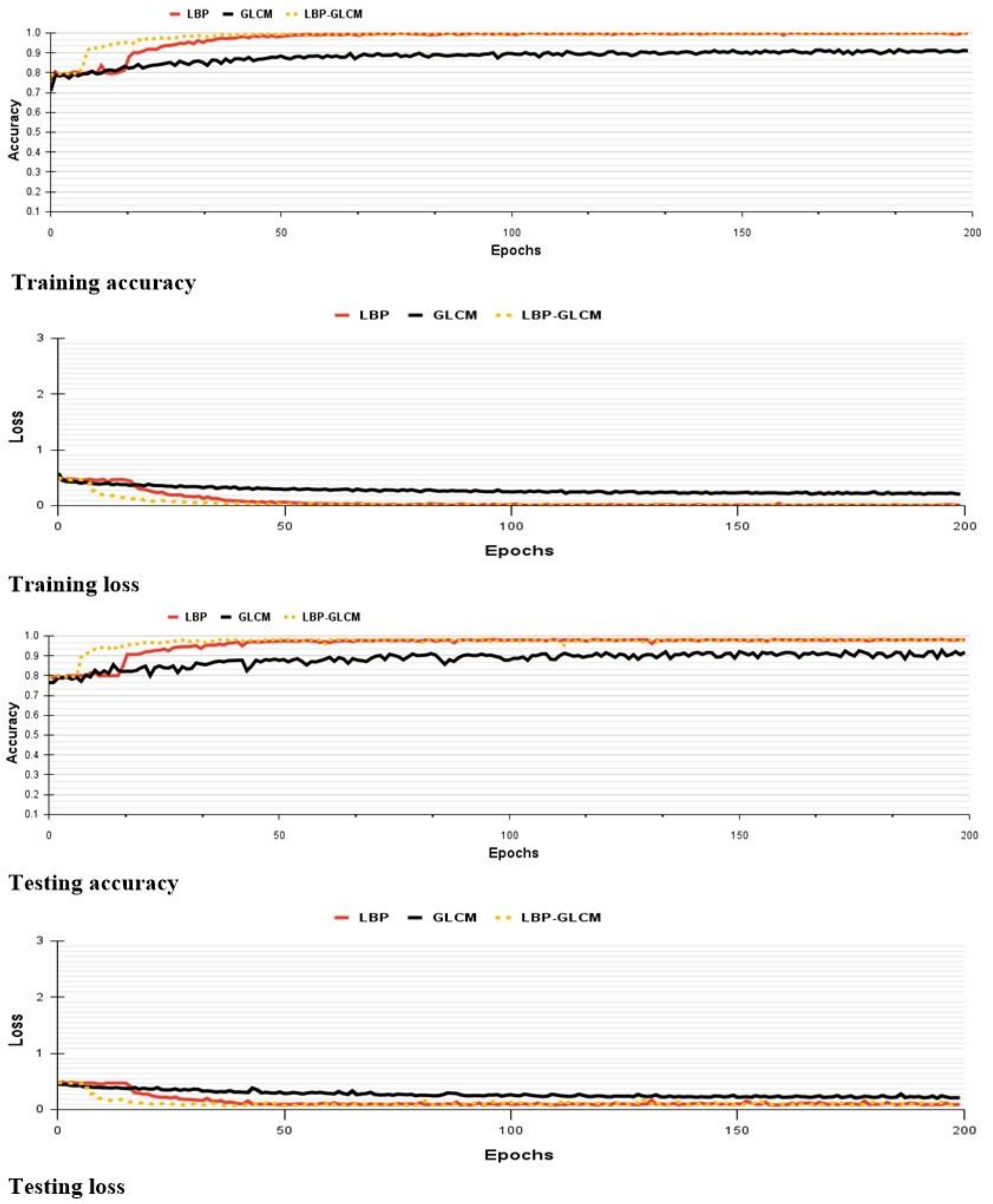
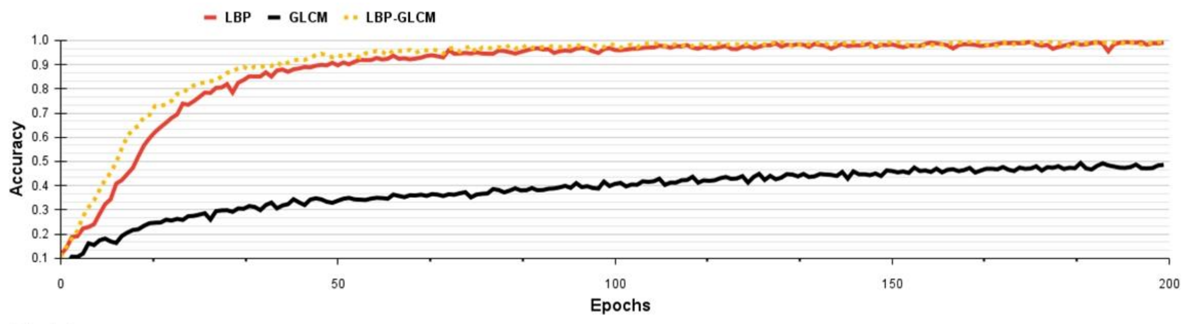
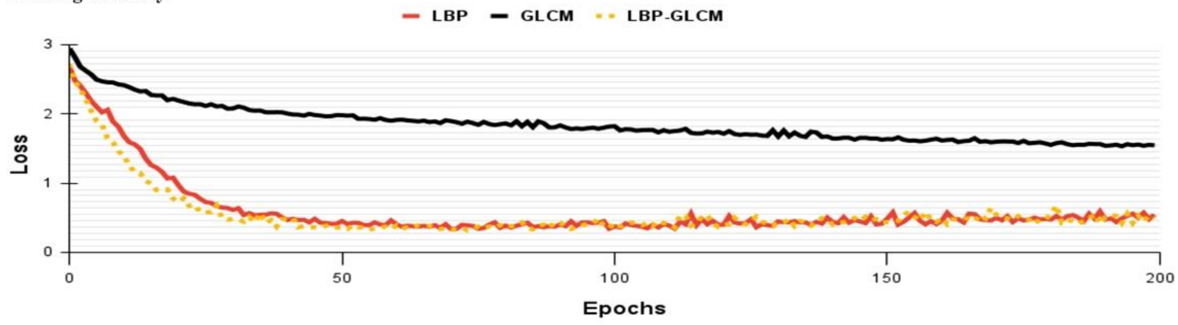


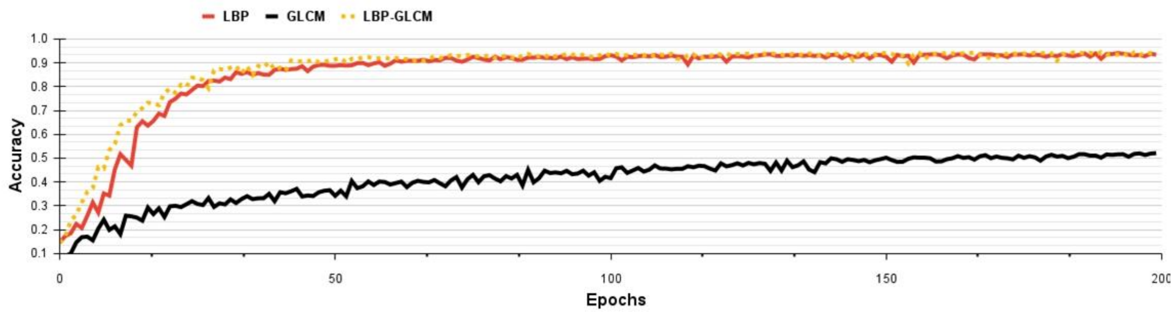
Figure 8. Training and testing dynamic plots of three different handcrafted features with the R2-D2 malware dataset



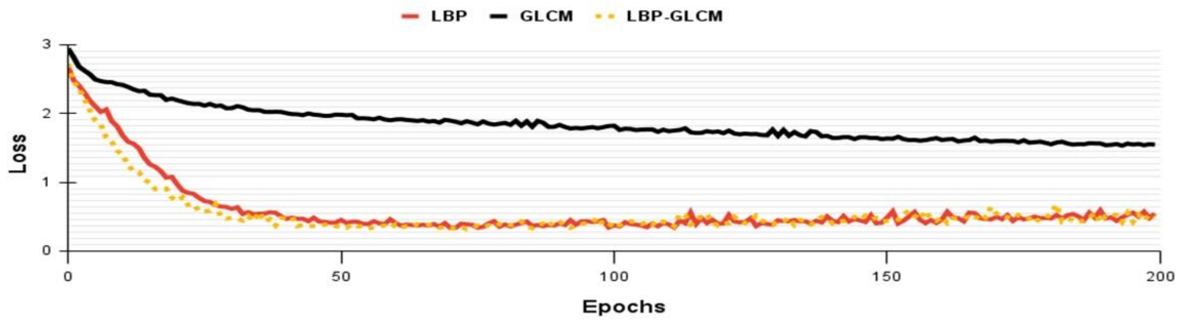
Training accuracy



Training loss



Testing accuracy



Testing loss

Figure 9. Training and testing dynamic plots of three different handcrafted features with the MalNet malware dataset

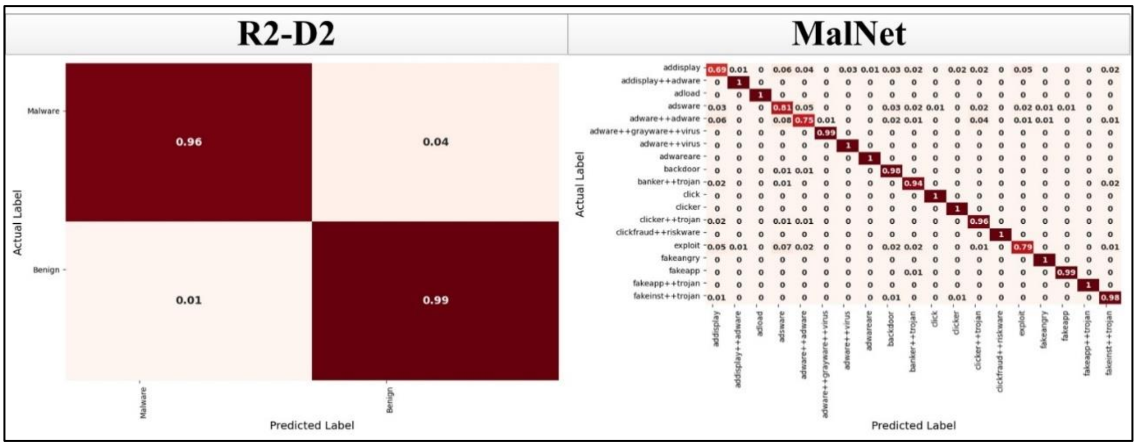


Figure 10. Confusion matrices for the proposed multihead ensemble over malware datasets

Table 2. Training and testing performance of hand-crafted features with different malware datasets

Feature	Dataset	Train Accuracy	Test Accuracy	Train Loss	Test Loss
Local	R2-D2	0.999	0.980	0.0050	0.091
	MALNET	0.9908	0.9332	0.02834	0.544
Global	R2-D2	0.9103	0.9176	0.2050	0.2084
	MALNET	0.4861	0.5210	1.6126	1.5450
Local-Global	R2-D2	**0.9993**	**0.97814**	**0.00362**	**0.10546**
	MALNET	**0.9943**	**0.9412**	**0.0185**	**0.49242**

Table 3. Classification performance of the proposed multihead ensemble over three data subsets

Performance with Training Set			
Families	**Precision**	**Recall**	**F1-Score**
Benign	1	1	1
Malware	1	1	1
Accuracy: 99.61%		**Loss: 0.02**	
Performance with Test Set			
Benign	0.99	0.98	0.98
Malware	0.98	0.99	0.98
Accuracy: 98.34%		**Loss: 0.07**	
Performance with Validation Set			
Benign	0.99	0.97	0.98
Malware	0.97	0.99	0.98
Accuracy: 98.28%		**Loss: 0.10**	

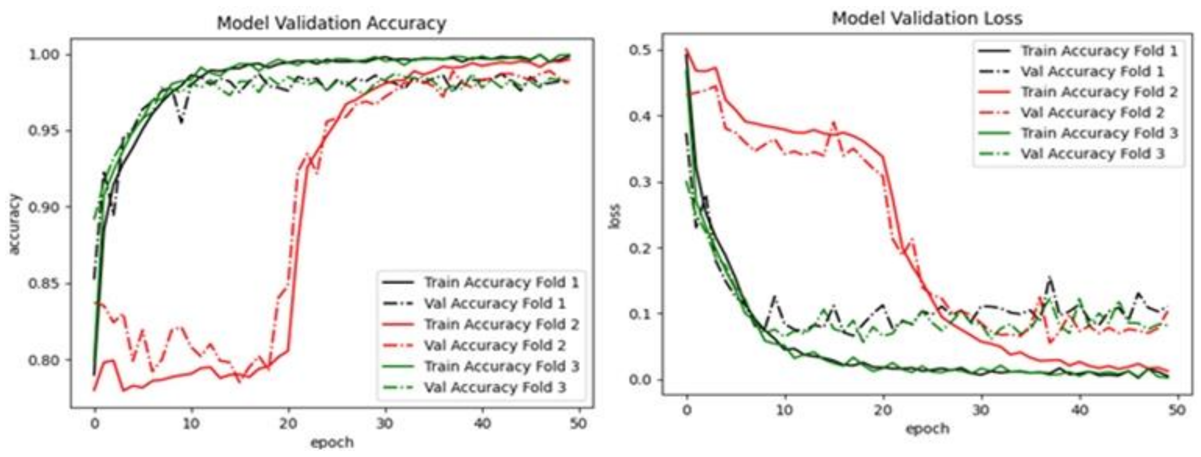


Figure 11. Dynamic plots of validation accuracy and loss through threefold cross validation

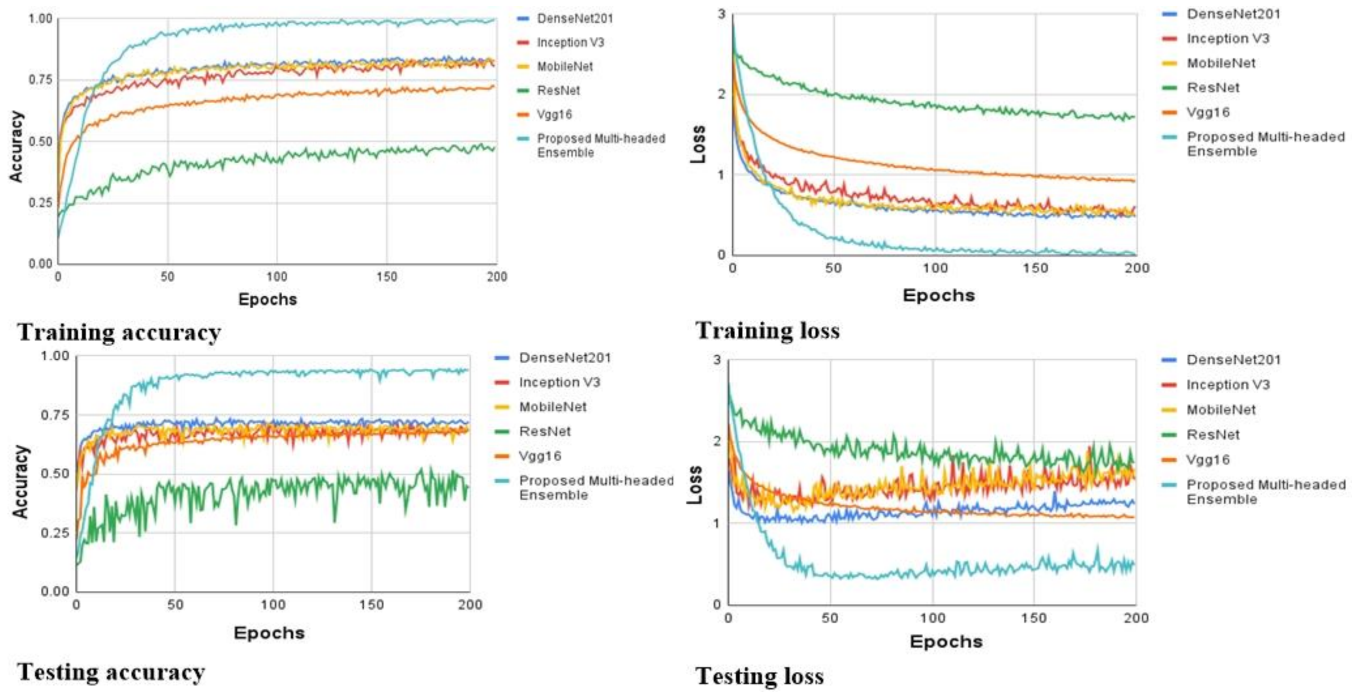


Figure 12. Comparison between dynamic plots of state-of-the-art CNN pre-trained models and the proposed multihead ensemble

4.2 Effect of combined handcrafted and deep feature extraction on the training performance of the proposed multihead ensemble

Table 4 evaluates the performance of the suggested multihead ensemble with those of four machine learning classifiers and different combinations of multihead neural ensembles. Different machine learning classifiers, such as decision tree (DT), logistic regression (LR), K-nearest neighbor (KNN), and random forest (RF), were used to evaluate the prediction performance of the proposed multihead ensemble. The predictive accuracy, precision, recall, and f1-score of the RF classifier were 0.70, 0.71, 0.70, and 0.67, respectively, whereas those of the DT classifier were 0.567, 0.56, 0.56, and 0.56; KNN classifier, 0.711, 0.70, 0.71, and 0.70; and LR classifier, 0.662, 0.62, 0.66, and 0.63. The overall accuracy of the proposed multihead ensemble was above 0.941. The lowest accuracy (0.567) was achieved by the DT classifier. The proposed multihead ensemble outperformed the other classifiers on all performance indicators.

Table 4. Comparison of classification performance with different learning methods

Machine Learning Method				
Model	**Accuracy**	**Precision**	**Recall**	**F1-Score**
Logistic Reg.	0.662	0.62	0.66	0.63
K-Near Neighbor	0.711	0.70	0.71	0.70
Decision Tree	0.567	0.56	0.56	0.56
Random Forest	0.700	0.71	0.70	0.67
Multi head Neural Ensemble				
CNN-Bidirectional GRU+MLP	0.939	0.94	0.94	0.94
CNN-Bidirectional LSTM+MLP	0.938	0.94	0.94	0.94
CNN-LSTM+MLP	0.931	0.93	0.93	0.93
CNN-GRU+MLP	**0.941**	**0.94**	**0.94**	**0.94**
Conventional CNN Pre-Trained Models				
	Pre-Trained	**Train Parameters**	**Layers**	**Accuracy**
ResNET50	Yes	38,931	175	0.441
InceptionV3	Yes	38,931	311	0.687
DenseNET201	Yes	36,499	707	0.721
Mobile NET	Yes	19,475	87	0.681
VGG16	Yes	9,747	19	0.685
Proposed Multi-head Ensemble	**No**	**678,489**	**18**	**0.941**

The performance of the multihead neural ensemble was also compared with those of several combinations of multihead neural ensembles, such as CNN–Bidirectional GRU, CNN–Bidirectional LSTM, and CNN–LSTM. The predictive accuracy, precision, recall, and f1-score of CNN–Bidirectional GRU were 0.939, 0.94, 0.94, and 0.94, respectively, whereas those of CNN-Bidirectional LSTM were 0.938, 0.94, 0.94, and 0.94, respectively, and those of CNN-LSTM were 0.931, 0.93, 0.93, and 0.93, respectively. The proposed multihead ensemble had an overall accuracy of 0.941. On all performance measures, the proposed multihead ensemble outperformed the other multihead ensembles.

The proposed multihead ensemble was compared with traditional pre-trained ResNET50, Inception V3, DenseNET201, Mobile NET, and VGG16 models to demonstrate how different layer and parameter sizes affect network training. To train the network, the proposed multihead ensemble uses fewer layers than traditional pre-trained CNN models, which need entire images to be fed into the model. The total detection accuracy of the proposed multihead ensemble was 0.941. In this regard, pre-trained models such as ResNET50, Inception V3, DenseNet201, Mobile NET, and VGG16 all require additional layers to learn the whole set of images. ResNET50, Inception V3, DenseNET201, MobileNET, and VGG16 models were pre-trained using the ImageNet dataset. Therefore, fine-tuning the CNN network will be more difficult if the training and testing scenes are used for different purposes. Although the pre-trained ResNET50, Inception V3, DenseNET201, MobileNET, and VGG16 models required less parameters for the training phase, their detection accuracies were 0.441, 0.687, 0.721, 0.681, and 0.685, respectively. These pre-trained CNN models require further weight fine tuning to obtain satisfactory results.

The proposed multi-head ensemble requires a fewer number of training layers than the pre-trained CNN models, and it delivers superior detection results. Figure 12 compares the dynamic graphs of state-of-the-art CNN pre-trained models with that of the proposed multihead ensemble.

4.3 Comparison of the classification performance of the proposed approach with those of previously published works

We compared the computational time and resource consumption of the proposed malware classification approach with those of previously published works. For example, Nataraj et al. [12] extracted textural features using the GIST descriptor from 32 × 32-dimensional malware images. Their approach consumes 1.45 s feature extraction time with 91.40% classification accuracy. Fu et al. [23] further extracted textural, local, and color features using SimHash, Color moment, and GLCM descriptor from color malware images. Their approach utilizes 1.17 s feature extraction time with 97.47% classification accuracy.

Table 5. Comparison of the proposed model with previously published works

Study	Feature Selection	Dimension	Accuracy
Nataraj et al. [12]	GIST	32×32	0.914
Fu et al. [23]	Sim Hash-Color Moment	NA	0.974
Naeem et al. [8]	DSIFT-GIST	200×200	0.984
Bozkir et al. [18]	GIST-HOG	4096×4096	0.963
Proposed Approach	**LBP-GLCM**	**299×299**	**0.978**

Naeem et al. [8] proposed a combined DSIFT and GIST feature descriptor for local and global feature extraction from 200 × 200-dimensional malware images. Their strategy attained 98.4% classification accuracy with 9.31 s total time consumption. Bozkir et al. [18] also proposed a combined HOG and GIST feature descriptor for local and global feature extraction from 4096 × 4096-dimensional malware images. Their approach obtained 96.39% classification accuracy with 3.56 s total time consumption. The methods using large images were resource and time consuming with the best classification accuracies. By contrast, the proposed malware classification approach extracted textural and local features through a combined LBP-GLCM descriptor. The proposed approach obtained 97.8% classification accuracy. The detailed results in Table 5 justify our claim.

5. CONCLUSIONS

This paper introduced a new way to classify malware families by capturing malicious behavior from suspect Android files as color images of various sizes. A combined feature descriptor selected relevant local and global characteristics of color images to reduce neural network training complexity. To improve network classification performance, a multihead neural ensemble was developed by integrating prediction results from weak learners (CNN and GRUs) and applying them as learning input to a meta learner (MLP). Performance was assessed using two publicly available android malware datasets. A baseline was established for comparing the malware classification performance of the proposed approach with those of state-of-the-art and previous systems. The proposed multihead ensemble method increased malware classification performance up to 97.85% accuracy with the R2-D2 dataset and 94.1% accuracy with the MalNet dataset. We plan to create a blockchain-based, memory-petite malware categorization methodology in the future to cut down on the time and resources spent processing malware.

ACKNOWLEDGMENTS

This work was supported in part by the Zhoukou Normal University High Level Talent Scientific Research under grant no. ZKNUC2021027.

REFERENCES

[1] Damodaran, A., Troia, F.D., Visaggio, C.A., Austin, T. H., Stamp, M. (2017). A comparison of static, dynamic, and hybrid analysis for malware detection. Journal of Computer Virology and Hacking Techniques, 13(1): 1-12. https://doi.org/10.1007/s11416-015-0261-z

[2] Bin-Salem, A.A., Zubaydi, H.D., Alzubaidi, M., Tariq, Z.U.A., Naeem, H. (2022). A scoping review on COVID-19's early detection using deep learning model and computed tomography and ultrasound. Traitement du Signal, 39(1): 205-219. https://doi.org/10.18280/ts.390121

[3] Ye, Y., Li, T., Adjeroh, D., Iyengar, S.S. (2017). A survey on malware detection using data mining techniques. ACM Computing Surveys (CSUR), 50(3): 1-40. http://dx.doi.org/10.1145/3073559

[4] Sihwail, R., Omar, K., Zainol Ariffin, K.A., Al Afghani, S. (2019). Malware detection approach based on artifacts in memory image and dynamic analysis. Applied Sciences, 9(18): 3680. https://doi.org/3680.10.3390/app9183680

[5] Gibert, D., Mateu, C., Planes, J. (2020). The rise of machine learning for detection and classification of malware: Research developments, trends and challenges. Journal of Network and Computer Applications, 153: 102526. https://doi.org/10.1016/j.jnca.2019.102526

[6] Or-Meir, O., Nissim, N., Elovici, Y., Rokach, L. (2019). Dynamic malware analysis in the modern era—A state of the art survey. ACM Computing Surveys (CSUR), 52(5): 1-48. https://doi.org/10.1145/3329786

[7] Cheng, Y., Fan, W., Huang, W., An, J. (2017). A shellcode detection method based on full native API sequence and support vector machine. In IOP Conference Series: Materials Science and Engineering, 242(1): 012124. https://doi.org/10.1088/1757-899X/242/1/012124

[8] Naeem, H., Guo, B., Ullah, F., Naeem, M.R. (2019). A cross-platform malware variant classification based on image representation. KSII Transactions on Internet and Information Systems (TIIS), 13(7): 37563777. http://doi.org/10.3837/tiis.2019.07.023

[9] Naeem, H., Ullah, F., Naeem, M.R., Khalid, S., Vasan, D., Jabbar, S., Saeed, S. (2020). Malware detection in industrial internet of things based on hybrid image visualization and deep learning model. Ad Hoc Networks, 105: 102154. https://doi.org/10.1016/j.adhoc.2020.102154

[10] Elmoogy, A.M., Dong, X., Lu, T., Westendorp, R., Tarimala, K.R. (2020). Surfcnn: A descriptor accelerated convolutional neural network for image-based indoor localization. IEEE Access, 8: 59750-59759. http://doi.org/10.1109/ACCESS.2020.2981620

[11] Naeem, H., Bin-Salem, A.A. (2021). A CNN-LSTM network with multi-level feature extraction-based approach for automated detection of coronavirus from CT scan and X-ray images. Applied Soft Computing, 113: 107918. http://doi.org/10.1016/j.asoc.2021.107918

[12] Nataraj, L., Karthikeyan, S., Jacob, G., Manjunath, B.S. (2011). Malware images: Visualization and automatic classification. In Proceedings of the 8th International Symposium on Visualization for Cyber Security, 17. https://doi.org/10.1145/2016904.2016908

[13] Kalash, M., Rochan, M., Mohammed, N., Bruce, N.D., Wang, Y., Iqbal, F. (2018). Malware classification with deep convolutional neural networks. In 2018 9th IFIP International Conference on New Technologies, Mobility and Security (NTMS), pp. 1-5. https://doi.org/0.1109/NTMS.2018.8328749

[14] Han, K.S., Lim, J.H., Kang, B., Im, E.G. (2015). Malware analysis using visualized images and entropy graphs. International Journal of Information Security, 14(1): 1-14. https://doi.org/10.1007/s10207-014-0242-0

[15] Ullah, F., Naeem, H., Jabbar, S., Khalid, S., Latif, M.A., Al-Turjman, F., Mostarda, L. (2019). Cyber security threats detection in internet of things using deep learning approach. IEEE Access, 7: 124379-124389. https://doi.org/10.1109/ACCESS.2019.2937347

[16] Vasan, D., Alazab, M., Wassan, S., Naeem, H., Safaei, B., Zheng, Q. (2020). IMCFN: Image-based malware classification using fine-tuned convolutional neural network architecture. Computer Networks, 171: 107138. https://doi.org/10.1016/j.comnet.2020.107138

[17] Dai, Y., Li, H., Qian, Y., Lu, X. (2018). A malware classification method based on memory dump grayscale image. Digital Investigation, 27: 30-37. https://doi.org/10.1016/j.diin.2018.09.006

[18] Bozkir, A.S., Tahillioglu, E., Aydos, M., Kara, I. (2021). Catch them alive: A malware detection approach through memory forensics, manifold learning and computer vision. Computers & Security, 103: 102-166. https://doi.org/10.1016/j.cose.2020.102166

[19] Hsien-De Huang, T., Kao, H.Y. (2018). R2-d2: Color-inspired convolutional neural network (CNN)-based android malware detections. In 2018 IEEE International Conference on Big Data, pp. 2633-2642. https://doi.org/10.1109/BigData.2018.8622324

[20] Freitas, S., Duggal, R., Chau, D.H. (2021). MalNet: A large-scale cybersecurity image database of malicious software. arXiv preprint arXiv:2102.01072.

[21] Ojala, T., Pietikainen, M., Maenpaa, T. (2002). Multiresolution gray-scale and rotation invariant texture classification with local binary patterns. IEEE Transactions on Pattern Analysis and Machine Intelligence, 24(7): 971-987. https://doi.org/10.1109/ACCESS.2018.2842078

[22] Haralick, R.M., Shanmugam, K., Dinstein, I.H. (1973). Textural features for image classification. IEEE Transactions on Systems, Man, and Cybernetics, (6): 610-621. https://doi.org/10.1109/TSMC.1973.4309314

[23] Fu, J., Xue, J., Wang, Y., Liu, Z., Shan, C. (2018). Malware visualization for fine-grained classification. IEEE Access, 6: 14510-14523. https://doi.org/10.1109/ACCESS.2018.2805301