scientific reports

Check for updates

OPEN

Mobile malware detection method using improved GhostNetV2 with image enhancement technique

Yao Du[1,2,3], CaiXia Gao[1,3], Xi Chen[1]✉, MengTian Cui[1], LiLi Xu[1] & AoJi Ning[1]

In recent years, image-based feature extraction and deep learning classification methods are widely used in the field of malware detection, which helps improve the efficiency of automatic malicious feature extraction and enhances the overall performance of detection models. However, recent studies reveal that adversarial sample generation techniques pose significant challenges to malware detection models, as their effectiveness significantly declines when identifying adversarial samples. To address this problem, we propose a malware detection method based on an improved GhostNetV2 model, which simultaneously enhances detection performance for both normal malware and adversarial samples. First, Android classes.dex files are converted into RGB images, and image enhancement is performed using the Local Histogram Equalization technique. Subsequently, the Gabor method is employed to transform three-channel images into single-channel images, ensuring consistent detection accuracy for malicious code while reducing training and inference time. Second, we make three improvements to GhostNetV2 to more effectively identify malicious code, including introducing channel shuffling in the Ghost module, replacing the squeeze and excitation mechanism with a more efficient channel attention mechanism, and optimizing the activation function. Finally, extensive experiments are conducted to evaluate the proposed method. Results demonstrate that our model achieves superior performance compared to 20 state-of-the-art deep learning models, attaining detection accuracies of 97.7% for normal malware and 92.0% for adversarial samples.

Keywords Malware detection, Image enhancement, Improved GhostNetV2, Adversarial samples

With the growing popularity of mobile devices in fields such as intelligent agriculture, smart transportation, and business applications, Android is one of the most popular systems due to its openness and ease of scalability, accounting for 72% of the global mobile market in 2024[1]. However, malicious attacks against mobile have become a serious security issue at the same time. According to Kaspersky's report[2], a total of 367,418 Android malware installation packages were detected in the second quarter of 2024. Sophos research[3] shows that ransomware attacks increased by 50% compared to 2023. In addition, the attack methods on Android devices have become more diversified and complex, such as Trojans, ransomware and spyware, etc. These attacks bring great risk to the data security of individuals and companies.

A commonly used detection method involves collecting large amounts of both malicious and benign software, followed by decompiling to extract features. These features are then used to train and test AI-based malware detection models. Although this approach has been widely applied and has led to significant research advancements[4–7], it still has two main limitations: (1) Hackers may use techniques such as shell programming, anti-debugging, and code obfuscation, making reverse engineering and analysis more difficult. (2) Decompiling certain complex applications requires substantial computational resources and time. As a result, researchers have started to explore image-based malware detection techniques to address these issues. This method transforms executable files of malicious code into images, then applies image processing and deep learning techniques to extract features and perform detection. As compare to feature analysis based on decompilation, image feature-based techniques offer several unique advantages: (1) Image feature extraction can capture global statistical properties of binary code, without the need for in-depth analysis of individual syntax features (e.g., permissions, APIs) or local logical structures (e.g., control flow, function calls), which facilitates quicker development of malware detection models. (2) A wide range of mature image processing algorithms exist for efficient feature extraction and classification. (3) Decompilation-based feature extraction methods depend on the internal logical structure of the code, while obfuscation techniques may disrupt or hide these structures, reducing detection

[1]College of Computer Science and Artificial Intelligence, Southwest Minzu University, Chengdu, China. [2]Institute of Qinghai-Tibetan Plateau, Chengdu, China. [3]Yao Du and CaiXia Gao have contributed equally to this work. ✉email: cx@swun.edu.cn

effectiveness. In contrast, image feature extraction methods are more robust against code obfuscation techniques (e.g., packing, encryption, instruction substitution). For instance, packing techniques might add extra decryption code to the binary, but this usually does not significantly alter the byte distribution pattern of the entire binary file. Image feature extraction methods can ignore such local variations and focus on global features.

In recent years, many image-based malware detection methods have been proposed[8–11]. Some of these employ traditional image processing techniques, such as edge detection and texture analysis, to extract visual features from images, while others use deep learning techniques, particularly Convolutional Neural Networks (CNN), to automatically extract features and classify images. However, these methods are limited in that they typically detect only regular malware samples and do not consider how to enhance the model's resilience against adversarial attacks. Adversarial attacks involve inserting carefully crafted perturbation data into the malware, designed to deceive deep learning models and cause incorrect classification results. In this study, we generated 8246 adversarial malware samples using Generative Adversarial Network (GAN), and classified the malware sample images using 20 common neural network models. The experimental results show that adversarial samples generated using this technique can bypass image-based malware detection, thereby reducing detection efficiency.

To solve these problems, this paper proposes a novel malware detection algorithm based on image enhancement and an improved neural network model. It aims to enable the detection model to effectively identify regular malware and improve its performance in detecting adversarial samples. First, Android applications are converted into RGB images, then image enhancement is performed using Local Histogram Equalization (LHE) and Gabor algorithms. Next, the performance of the detection model is further improved using the enhanced GhostNetV2 algorithm. Experimental results indicate that the proposed algorithm significantly improves both detection accuracy and resistance to adversarial samples. Overall, the main contributions of this paper are as follows:

- A novel application image generation method is proposed. Initially, the application is converted into an RGB image, followed by image enhancement using LHE technique and the Gabor algorithm. The primary objective of this method is to extract more effective image features that can be used for the simultaneous identification of both conventional malware and adversarial samples.
- A classification algorithm of GhostnetV2 is implemented for efficiently identifying malware and adversarial samples. It can achieve high detection accuracy and operational efficiency by improving the lightweight neural network.
- Several experiments are designed to evaluate our detection method. Experimental results show that our method has better detection performance on normal malware and adversarial samples as compared to the unenhanced method. The highest detection accuracy reaches 97.7% and 92.0%, respectively.

The structure of the paper is as follows. Section "Related Work" reviews recent relevant studies. Section "The proposed method" details the proposed approach, including image generation, image enhancement, and model improvement. Section "Experimental Results" presents a series of experiments to assess the detection and calculation performance of our method. Finally, Section "Conclusion and Future Work" summarizes the key findings and suggests potential research directions for future work.

Related works

With the rapid development of machine vision technology, researchers continue to apply image analysis techniques to the field of malware detection. At the same time, continuous progress has been made in optimizing detection methods, particularly in feature extraction algorithms and detection model architectures, leading to a large number of valuable research results. The application images used in these researches mainly include two categories: grayscale images and color images. A summary and comparison of the relevant literature on image-based detection of malicious code over the past five years is presented in Table 1.

Reference	Model	Features	Image	Datasets	Accuracy (%)
Ding et al.	CNN	classes.dex	grayscale	Drebin	95.6
Singh et al.	Fusion-SVM	classes.dex	grayscale	Drebin	93.24
Tang et al.	ResNet	classes.dex	grayscale	Drebin	96.35
Zhang et al.	TCN	classes.dex,.xml	grayscale	Drebin	95.44
Wang et al.	CNN	classes.dex,.xml	RGB	Drebin	99.8
Yadav et al.	EfficientNetb4	classes.dex	RGB	R2-D2	95.7
Ye et al.	CNN	classes.dex	RGB	CIC-AndMal-2017/2020	95.98
Ksibi et al	CNN	classes.dex	RGB	CICInvesAndMal2019	95.24
Zhu et al	EfficientNet	classes.dex	RGB	VirusShare	96.9
Propsed	GhostNetV2	classes.dex	RGB	CICAndMal2017/2020, VirusShare, Drebin	97.7

Table 1. Existing Android malware detection methods based on image-based deep learning algorithms.

Graryscale image

A grayscale image can be generated from an application's binary file, where the grayscale value of each pixel corresponds to a specific byte value in the executable file. This transformation enables artificial intelligence algorithms to effectively detect malicious code through image-based analysis. Due to their computational simplicity and efficiency, grayscale images have been widely used in malware detection. For example, Ding et al.[12] proposed a static detection method based on deep learning, which converted class.dex files into grayscale images and trained a Convolutional Neural Network (CNN) for malware classification. This method achieved an accuracy of 95.6%, demonstrating its effectiveness in malware detection. Singh et al.[13] proposed a framework that visualizes Android malware as grayscale images and employed techniques like Gray Level Co-occurrence Matrix (GLCM), Global Image Descriptors (GIST), and Local Binary Pattern (LBP) to extract features for classification. Their results showed that the Feature Fusion Support Vector Machine (SVM) model achieved the highest performance, with an accuracy of 93.24% in identifying and classifying Android malware. Tang et al.[11] proposed efficient Android malware detection system that extracts opcode features at various granularities, used the TFIDF algorithm for weighting, and visualizes features as grayscale images. Experiments showed detection accuracies of 96.35% for unobfuscated samples and 94.55% for obfuscated samples. Zhang et al.[8] proposed a new Android malware detection model that combines XML and DEX file features, converting them into grayscale images for detection using a temporal convolution network (TCN). This model achieves an accuracy of 95.44%.

RGB image

As compared to grayscale images, RGB images incorporate three distinct color channels (red, green, and blue), enabling the independent encoding of diverse categories of code information. Although the computational complexity is higher, the rich feature representation capability of RGB images helps to improve the performance of detection models. Wang et al.[14] proposed a multi-class classification method for Android malware families using multi-class feature files and RGB images. Their method extracted DEX and XML files from APK packages without decompilation and converts them into RGB images. Experimental results showed that this method achieves a high accuracy of 99.84% in multi-class classification. Yadav et al.[10] mapped the bytecode in Android classes.dex files to RGB images and proposed a CNN model based on EfficientNet-B4[15] for malware detection, achieving an accuracy of 95.7%. Ye et al.[16] transformed Android malware classes.dex, AndroidManifest.xml, and resource.arsc into RGB images and used a lightweight convolutional neural network to automatically extract the features of the RGB images. The experimental results of this study indicated that the method performs well in terms of precision and speed of detection. Ksibi et al.[17] converted the binary code of Android APK files into RGB images, using pre-trained models such as DenseNet169[18], InceptionV3[19], ResNet50[20], and VGG16[21] for feature extraction. The experimental results showed that the classification accuracies for DenseNet169, InceptionV3, and VGG16 reached 95.24%, 95.24%, and 95.83%, respectively. Zhu et al.[22] transformed executable files into RGB images and utilized a new variant of CNN known as MADRF-CNN. The experimental findings revealed that this approach attained a malware detection rate of 96.9%.

Adversarial attack

While these methods have demonstrated success in detecting malware, their detection performance can be significantly compromised by adversarial samples. These adversarial samples evade the detection mechanism by injecting well-designed perturbations in the executable of the malware. Hu et al.[23] were the first to apply Generative Adversarial Networks (GAN) to malicious code, proposing the MalGAN algorithm. This algorithm can generate adversarial malware samples that successfully bypass black-box machine learning detection models. It adapts to the black-box system using alternative detectors and trains the generative network to reduce the probability of being detected as malicious. They also combined Recurrent Neural Networks (RNN) with GAN to generate sequential adversarial samples aimed at attacking malware detection systems based on RNN[24]. Experimental results show that these RNN-based detection algorithms cannot identify most of the generated malicious adversarial samples. Building on this foundation, Wang et al.[25] combined CNN and GAN to design an efficient malware detection method. They implemented a code visualization technique and utilized GAN to generate more samples of malicious code variants for data augmentation. Finally, they used the lightweight AlexNet for malware classification, and the experimental results showed that the model achieved a classification accuracy of 97.78%. Additionally, Li et al.[26] proposed an Android malware classification model based on CTGAN-SVM, combining GAN with Support Vector Machines (SVM) to generate adversarial samples. Through KS-CIR testing and a random forest classifier, SynDroid achieved a 12% increase in accuracy on the CCCS-CICAndMal2020 dataset, effectively mitigating the issue of imbalanced data. Most recently, Gao et al.[27] introduced an innovative adversarial malware generation model named Mal-WGANGP. This model can automatically produce a substantial number of adversarial samples, thereby enhancing the detection capability of the model while also expanding the dataset.

To the best of our knowledge, the majority of existing studies have predominantly focused on normal malware detection, while research on enhancing detection models to identify both malware and adversarial samples remains notably limited. In particularly, there is a lack of researches that use real adversarial sample datasets for testing. This is the main motivation for the method proposed in this paper.

The proposed method

In this section, we propose a novel malware detection method based on an improved version of GhostNetV2. First, we convert the classes.dex files of Android applications into RGB images. We then apply the LHE method for image enhancement, followed by the Gabor transform to improve texture features and reduce the three-channel image to a single channel. Finally, the images are fed into the enhanced GhostNetV2 model for classification. The architecture of our detection model is shown in Fig. 1.

Image generation

The file directory of an Android application is shown in Fig. 2. As the core file, "classes.dex" is the executable file running on the Dalvik virtual machine. It contains the running code and variable space allocation of the entire application. According to related studies, as shown in Table 1, all the studies utilized the classes.dex file. Therefore, we convert the .dex file into an image as an input to the malicious code detection algorithm. It proceeds as follows: first, the APK file is decompiled to get a binary .dex file. After that, the data sequences in the binary file are read in groups of every 8 bits and converted to decimal unsigned integers. These integers are treated as level values of gray scale values, which range from 0 to 255. Finally, a colour mapping mechanism is designed and implemented. Its principle is to dynamically map grayscale values to different RGB colour spaces according to their interval distribution. For example, grayscale values between 0 and 63 are mapped to cyan, 64–127 to green, 128–191 to yellow, and 192–255 to red." The image generation is shown in Algorithm 1.

Require: DEX file
Ensure: RGB image

```
len ← getsize(classes.dex)
rem ← len % width
a ← len - rem
data ← reshape(len(a) // width, width)
data ← uint8(data)
row, col ← img_gray.shape[:]
for i = 0 to row - 1 do
    for j = 0 to col - 1 do
        if img_gray[i, j] < 255 / 4 then
            b[i, j] ← 255
            g[i, j] ← 4 × img_gray[i, j]
            while g[i, j] > 255 do
                g[i, j] ← g[i, j] - 255
            end while
            r[i, j] ← 0
        else if img_gray[i, j] < 255 / 2 then
            b[i, j] ← -4 × img_gray[i, j]
            while b[i, j] < 0 do
                b[i, j] ← b[i, j] + 255
            end while
            g[i, j] ← 255
            r[i, j] ← 0
        else if img_gray[i, j] < 3 × 255 / 4 then
            b[i, j] ← 0
            g[i, j] ← 255
            r[i, j] ← 4 × img_gray[i, j] - 255 × 2
            while r[i, j] > 255 do
                r[i, j] ← r[i, j] - 255
            end while
        else
            b[i, j] ← 0
            g[i, j] ← -4 × img_gray[i, j]
            while g[i, j] < 0 do
                g[i, j] ← g[i, j] + 255
            end while
            r[i, j] ← 255
        end if
    end for
end for
```

Algorithm 1. Android RGB image generation process

LHE processing

The differences between malicious code, benign samples, and adversarial samples at the pixel level may be minimal, but there are significant distinctions in the local image textures. By enhancing the image contrast, we can not only highlight the local texture differences between malicious samples and benign samples but also

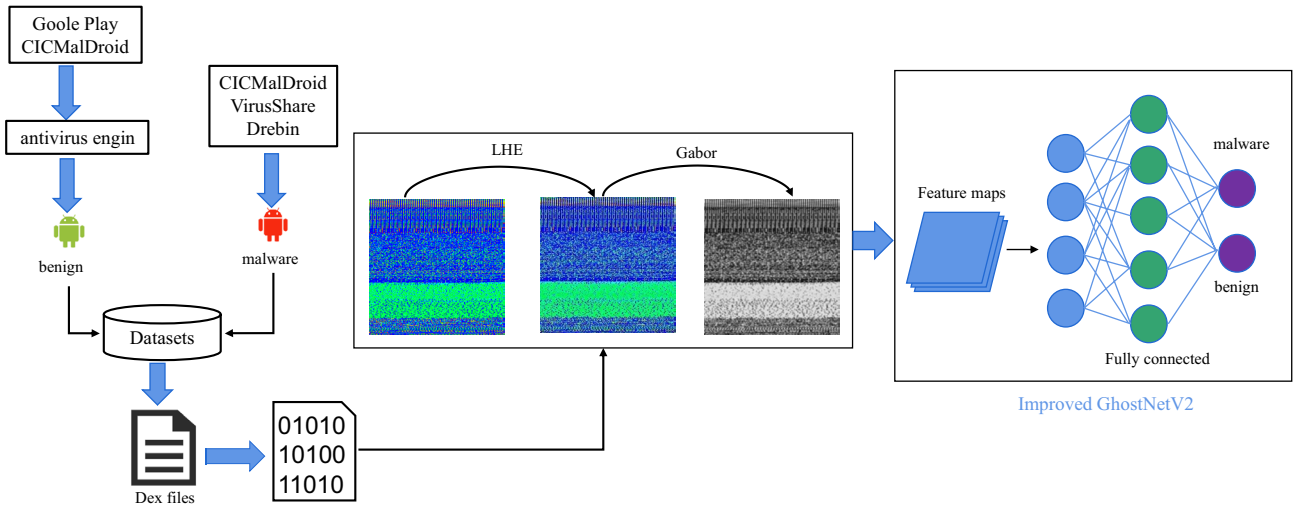


Fig. 1. The architecture of our malware detection model.

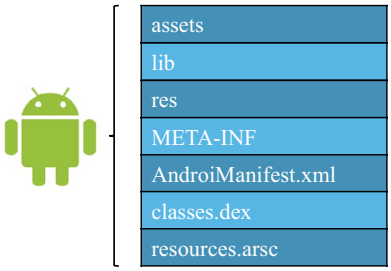


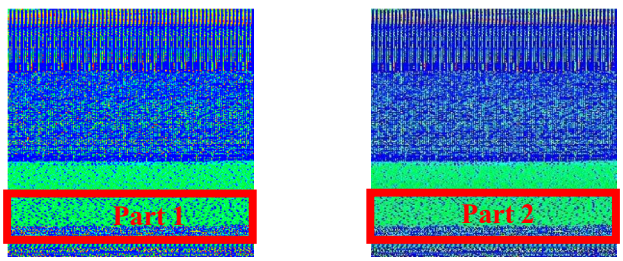
Fig. 2. Directory of APK files.

amplify the texture differences between malicious samples and adversarial samples. This approach aids classifiers in better identifying and analyzing these texture differences, thereby improving the accuracy of malware detection.

Local Histogram Equalization (LHE)[28] is widely used in the field of image enhancement. It can effectively improve the local contrast of an image. Different from global histogram equalization, local histogram equalization divides the image into multiple small regions and performs independent histogram equalization for each small region. Thus, the detailed features of each region are better displayed to optimize the quality of the whole image.

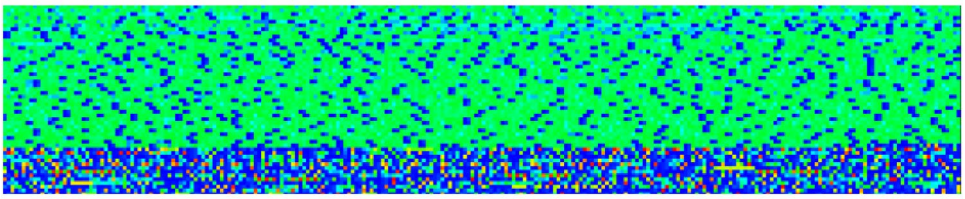
The main steps in generating a local histogram equilibrium image are as follows:

(1) Generate localised image regions: an 8×8 sliding window is used to divide the malicious image into multiple overlapping small regions, each of which is called a local window.
(2) Counting pixels: calculates the number of pixels in different gray levels within each window.

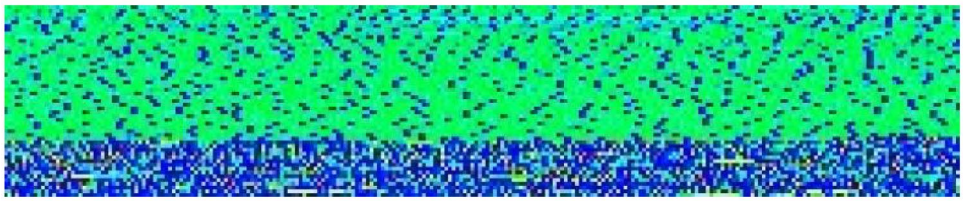


(a) without LHE (b) with LHE

Fig. 3. The RGB image of 1b3372d4243776ef09d50761aa53aa2fe486d468ff2fa31b46363a0c96929eaa.apk before and after using LHE.

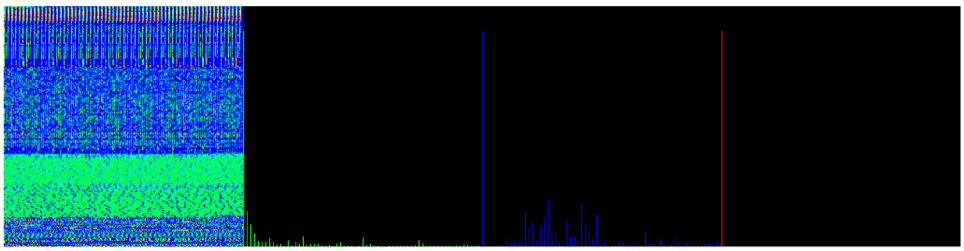


(a) part 1

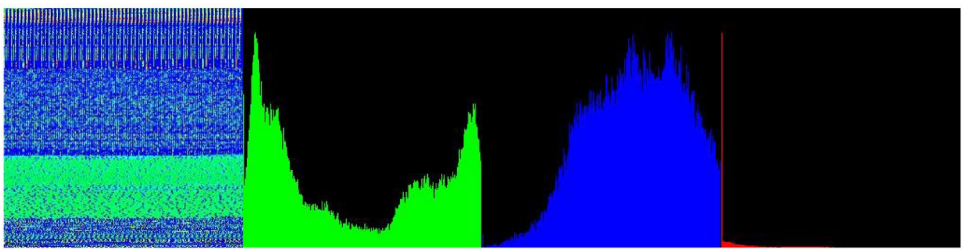


(b) part 2

Fig. 4. Detailed image comparison of the area marked in red in Fig. 3.



(a) before



(b) after

Fig. 5. Histogram of Android malicious images before and after using LHE.

(3) Define the cumulative distribution function: based on step 2, the CDF is defined for each grey level in the local window, as shown in Eq. (1).

$$CDF(i) = \sum_{0 \sim i} P(r) \tag{1}$$

where i represents the gray level and P(r) represents the probability of the pixel value.

(4) Define the mapping function: according to the CDF, compute the mapping function for each grayscale level to map the original pixel values to new y values, as shown in Eq. (2).

$$y = \text{round}((L - 1) * \text{CDF}(i)) \tag{2}$$

where L is the number of gray levels.

(5) Calculate the mapping value of the pixels in the local windows: according to the mapping function in step 4, calculate the mapping value of each pixel in the local windows.
(6) Repeat steps (1)–(5) to get the enhanced image.

Figure 3a is overall brighter than Fig. 3b, which may lead to detail loss or overexposure. Therefore, we adjust the brightness appropriately to assist the model in learning. To illustrate the effect more intuitively, we provide detailed images of the red-marked areas in Fig. 3, as shown in Fig. 4.

Figure 5 compares the histograms of the RGB channels in the malware images. Figure 5a shows that when local histogram equalization is not used, the distribution of pixel values in the RGB channels is more concentrated, especially in the R channel. Figure 5b demonstrates that after applying local histogram equalization, the distribution of pixel values becomes more uniform and the image has a wider range of pixel values.

Gabor filter processing

Complex textures and adversarial perturbations in images are high frequency information[29]. Although traditional high-pass filters can effectively filter out low-frequency background noise and highlight highfrequency features, they are limited in the high-frequency domain where complex textures and adversarial perturbations coexist. In contrast, band-pass filters can precisely control the perturbation in a specific frequency range and retain the key information of the image well. In contrast, the band-pass filter can precisely control the perturbation in a specific frequency range and retain the key information of the image, which is more suitable for the task of this paper.

A Gabor filter is used in this study. It is good at extracting texture features from images, particularly in terms of frequency and orientation. However, it's sensitive to image contrast. When contrast is insufficient, Gabor filter may fail to extract key texture information effectively. After enhancing contrast with LHE technique, Gabor filter can more efficiently extract texture features, significantly improving the accuracy and effectiveness of texture extraction. Which is defined as follows:

$$g(x, y; \sigma; \theta; \lambda; \gamma; \psi) = \exp\left[-\frac{x^2 + \gamma^2 y^2}{2\sigma^2}\right] \cdot \exp\left[\mathrm{i}\left(2\pi\frac{x}{\lambda} + \psi\right)\right] \tag{3}$$

where (x, y) is the Gabor filter convolution kernel size; σ is the standard deviation, which is used to control the degree of smoothing of the filter; the orientation parameter θ determines the direction of the Gabor filter; the wavelength parameter λ defines the period of the sinusoidal component of the Gabor filter; the aspect ratio γ describes the degree of stretching of the Gabor filter's elliptical shape; and the phase offset ψ can be used to

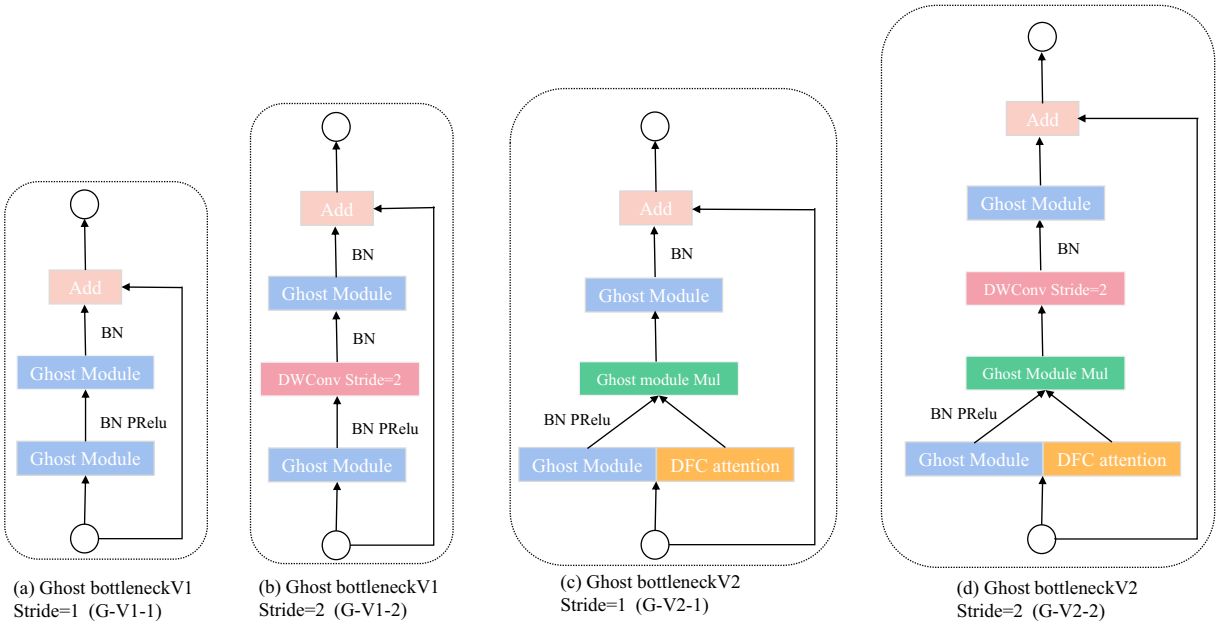
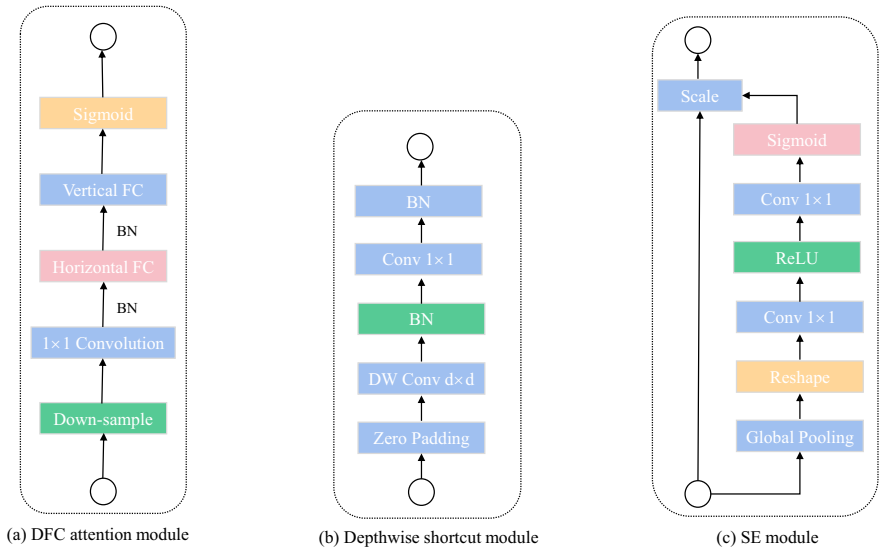


Fig. 6. GhostNetV1 and GhostNetV2 bottleneck.



(a) DFC attention module (b) Depthwise shortcut module (c) SE module

Fig. 7. The structure of the DFC attention module, Depthwise shortcut module and SE module.

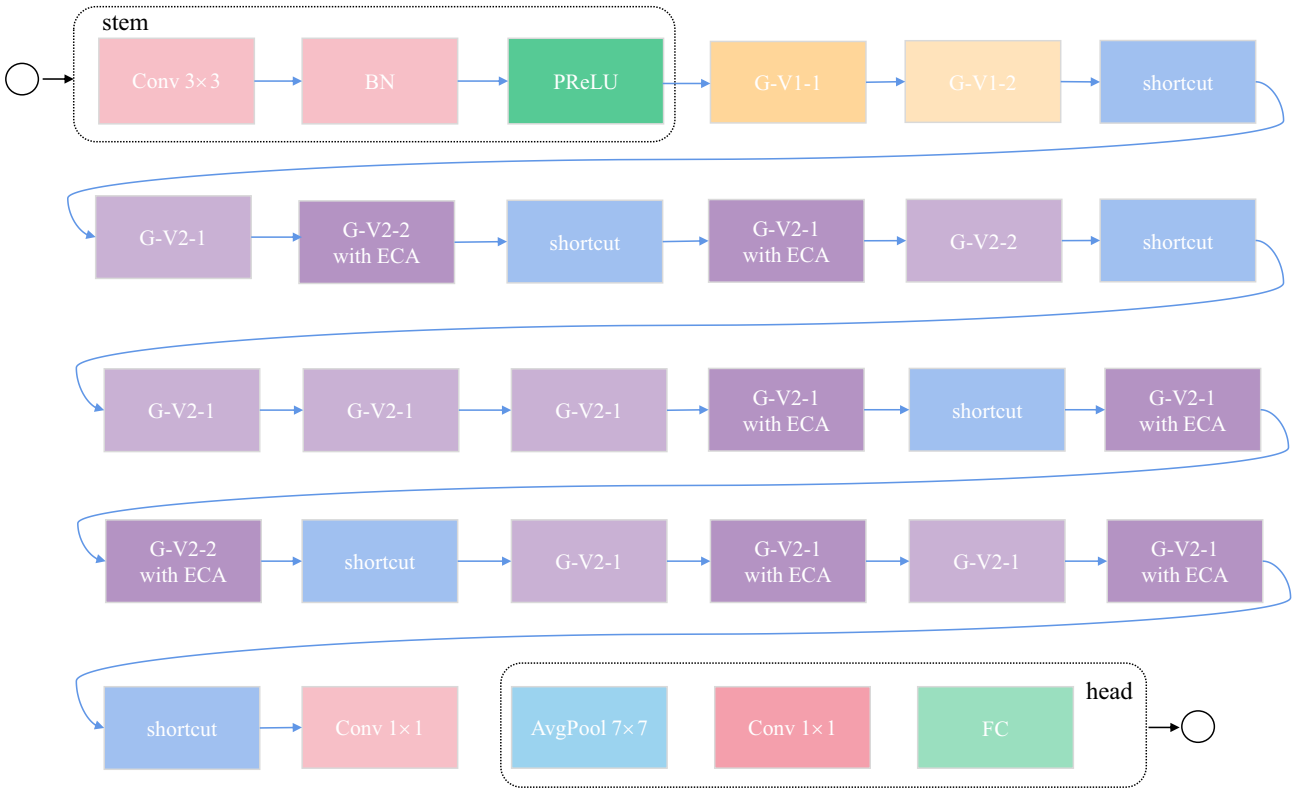


Fig. 8. Architecture of improved GhostNetV2.

adjust the the response of the filter to specific phase features in the image. Here, we set (x, y)=(3,3) , σ=3, θ=180, λ=180, γ=0.5, ψ=0.

Malware detection based on the improved GhostNetV2 model

GhostNetV2

In this paper, we use the improved GhostNetV2 framework to detect malware. GhostNet[30] is a lightweight convolutional neural network designed for mobile devices with excellent performance in image classification tasks. The key component of GhostNet is the Ghost module, an efficient plug-and-play module that can generate more feature maps with fewer parameters. It can be implemented in the following way: for a given input feature

$X \in R^{H \times W \times C}$, (H, W, C are the height, width and number of channels of the feature map respectively), the Ghost module splits the output channel into two parts. The first part is the regular convolution as shown in Eq. (4):

$$Y' = X * F_{1\times1} \tag{4}$$

where $*$ is the convolution operation, $F_{1\times1}$ is point-wise convolution and $Y' \in R^{H\times W\times C'_{\text{out}}}$ is a partial output feature.

The second part is the additional feature maps generated by cheap operations (for example, simple linear transformations). After that, the results of these two parts are merged by the concat function to get the final output, as shown in equation (5):

$$Y = \text{Concat}\left(\left[Y', Y' * F_{dp}\right]\right) \tag{5}$$

where F_{dp} is depth-wise convolution and $Y \in R^{H\times W\times C_{\text{out}}}$ is the final output feature.

Although the Ghost module has significantly reduced the number of parameters, its ability to capture spatial information has also been reduced. To address this problem, GhostNetV2[31] adds the DFC (Decoupled Fully Connected) module to capture long distance spatial positional dependencies and to improve inference speeds. The calculation process of DFC is as follows.

Consider a given input feature layer $X \in R^{H\times W\times C}$ as $H * W$ feature tokens, $z_i \in R^C$ and $Z = \{z_{11}, z_{12}, \ldots, z_{HW}\}$ aggregating features along the horizontal and vertical directions, respectively. The computational procedure can be defined as:

$$a'_{hw} = \sum_{h'=1}^{H} F^H_{h,h'w} \odot z_{h'w'} h = 1, 2, \cdots, H, w = 1, 2, \cdots, W \tag{6}$$

$$a_{hw} = \sum_{w'=1}^{W} F^W_{w,hw'} \odot a'_{hw'}, h = 1, 2, \cdots, H, w = 1, 2, \cdots, W \tag{7}$$

where F^H and F^W are the weights, $\odot$ represents element-wise multiplication, and $A = \{a_{11}, a_{12}, \ldots, a_{HW}\}$ is the obtained attention map.

As shown in Fig. 6c and d, the GhostNetV2 bottleneck consists of two modules: the DFC and the Ghost. The structure of the DFC is shown in Fig. 7a, where BN refers to Batch Normalization. The input features are processed by the Ghost module to generate the feature *Y*, while the attention matrix *A* is computed by the DFC module. Then, Y is dot-multiplied with *A* to obtain *O*, which is passed as input to the subsequent Ghost module ($\mathcal{V}$), as shown in Eq. (8).

$$O = \text{Sigmoid}(A) \odot \mathcal{V}(X) \tag{8}$$

Model optimization

In order to improve the effectiveness of malware detection, this paper makes three improvements to the GhostNetV2 model. First, the activation function of GhostNetV2 is replaced from ReLU to PReLU to reduce the model generalization error. Second, channel blending is introduced in the second Ghost Module of GhostNetV2 bottleneck to enhance the information exchange between features and improve the network performance. Third, the ECA module is used to replace the SE module. It can reduces the network parameters and computation while maintaining high detection accuracy. The overall architecture of the improved GhostNetV2 is shown in Fig. 8. G-V1-1, G-V1-2, G-V2-1 and G-V2-2 represent the Ghost bottleneck V1 with stride=1, Ghost bottleneck V1 with stride=2, Ghost bottleneck V2 with stride=1, and Ghost bottleneck V2 with stride=2, respectively. The detailed structures are shown in Fig. 6a, b, c and d. The shortcut is a depthwise (DW) shortcut, as illustrated in Fig. 7b.

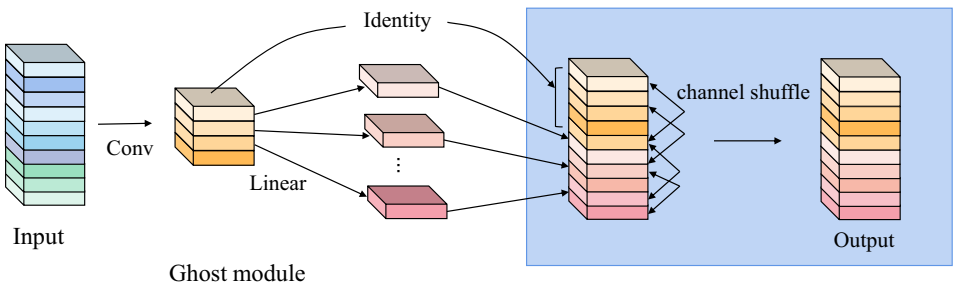


Fig. 9. Ghost module with channel shuffle.

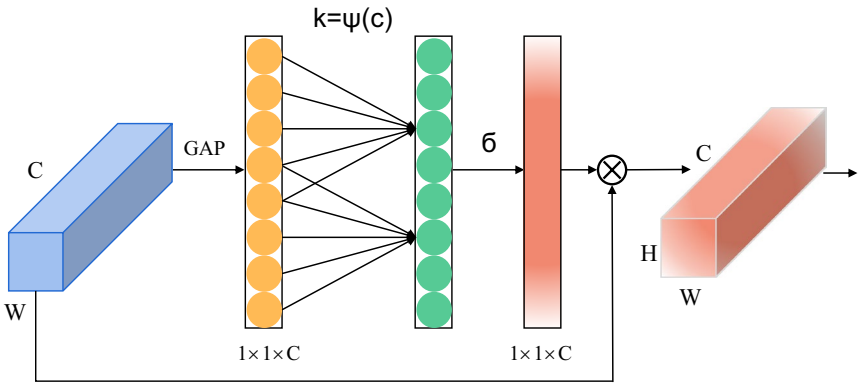


Fig. 10. ECA module.

APK type	Quantity	Family list
Malware	11552	Dowgin, Ewind, Feiwo, Gooligan, Kemoge, Mobidash, Jisut, Pletor, PornDroid, VirusShield, SMSsniffer, FakeMart, BeanBot, AndroidSpy, Iconosys, FakeApp, Plankton, koodous, GinMaster, DroidDream, Glodream, ExploitLinuxLotoor, DroidKungFu, RansomBO, FakeNotify, etc.
Benign	10060	Tools, Family, Game, Business, Medical, Shopping, Social, Dating, Education, Lifestyle, Sports, Entertainment, Video Players, Music, Photography, Health, Weather, etc.

Table 2. Information of datasets.

Environment	Configure
CPU	Intel(R) Xeon(R) Platinum 8481C
Memory	80G
OS	Ubuntu 20.04
GPU	GeForce RTX 4090 D
Video Memory	24G

Table 3. Information of hardware.

a. *Replacement of ReLU* GhostNetV2 uses the ReLU activation function. This function limits the network's ability to handle nonlinear problems by using only non-negative activation values. To solve this problem, the PReLU activation function is chosen. PReLU (Parametric Rectified Linear Unit)[32] is designed based on ReLU, which introduces learnable parameters that allow negative activation values. PReLU improves the network's ability to learn complex nonlinear functions, which enables it to achieve better performance in image recognition. PReLU is defined as follows:

$$f(x) = \max(0, \mathrm{x}) + \alpha \min(0, \mathrm{x}) \tag{9}$$

where $\alpha = 0.25$.

b. *Channel shuffle* To reduce the computation, GhostNetV2 bottleneck parallelized only the first Ghost module with the DFC. In the 2nd Ghost module, GhostNetV2 first generates the first set of features by standard convolution and then performs lightweight operations on the first set of features to obtain the second set of features. However, this leads to a lack of effective mapping links between the two sets of features, which degrades the model performance. Therefore, this paper innovatively introduces channel shufflle (CS)[33] to enhance the data interaction between the two sets of feature maps, thus improving the model performance.

As shown in Fig. 9, channel shuffling is realized by matrix reshaping, transposition and splicing, with much lower computational consumption than convolutional operations. It is executed in the following steps:

(1) Divide the feature layer output from the Ghost module into g groups (g=4), each containing n channels. In total, $g \times n$ output channels are generated.
(2) Adjust the output matrix to (g, n) shape by reshape operation and subsequently transpose it to (n, g) shape.

(3) Flatten the matrix obtained in step 2 into a one-dimensional vector. After that, repartition the one-dimensional vectors into g groups of n channels each;

This makes the use of channel shuffle in the model not only improve the model performance, but also avoid a significant increase in computational cost.

c. *The introduction of ECA* It is found that the SE (Squeeze-and-Excitation)[34] module in the original GhostNetV2 model fails to sufficiently focus on the key malicious features of the malware images. SENet optimizes the local features by dynamically adjusting the channel weights. Its architecture is shown in Fig. 7c. However, the global pooling mechanism employed in SENet tends to adjust feature weights at the overall level while ignoring locally important features. In addition, the dimensionality reduction strategy in SENet may reduce the performance of the channel attention mechanism.

Therefore, we use ECA (Efficient Channel Attention)[35] instead of SE, as shown in Fig. 10. ECA achieves local cross-channel interaction through one-dimensional convolution. For the ECA module's 1D convolution, the adaptive kernel size k was determined by:

$$k = \psi(C) = \frac{\log_2(C)}{\gamma} + \frac{\beta}{\gamma} \tag{10}$$

where C is the channel dimension, and $\gamma = 2$, $\beta = 1$. This adaptively balances local and global attention. ECA discards the dimensionality reduction and global pooling operations, which significantly reduces the number of parameters and computational cost. The module can flexibly adjust the convolutional kernel size to adapt to different feature scales, which improves the performance of deep convolutional neural network (DCNN) and simplifies the model complexity at the same time. ECA can effectively enhance the model performance in tasks such as image classification and target detection, etc.

Experimental results

Data preparation

The experimental dataset we use contains 11552 malware samples, of which 5978 are from CICMalDroid[36], 2453 from VirusShare[37], and 3121 from Drebin[38]. These malware samples employ various obfuscation techniques, including code restructuring, renaming of functions and variables, insertion of junk code, and encryption and decryption of code. At the same time, the dataset includes 10060 benign samples, with 4185 from Google[39] Play and 5875 from CICMalDroid, as detailed in Table 2. All downloaded benign samples have been scanned for security using VirusTotal[40] and Kaspersky (the version is standard 21.17)[41]. The dataset is split with 80% used for

Model	Typical variants	Para(M)	ImageNet Top-1 Acc (%)	Key features
ResNet[20]	ResNet-34	21.8	74.60	Medium-depth residual network with 4 stacked residual blocks
	ResNet-50	25.6	76.15	Mainstream benchmark model using Bottleneck architecture to reduce computation
	ResNet-101	44.5	77.37	Deep network with 33 residual blocks, suitable for high-accuracy scenarios
	ResNet-152	60.2	78.31	Deepest standard ResNet variant, primarily used in research
MobileNet	MobileNetV2[42]	3.4	72.00	Inverted residual structure with linear Bottleneck design for reduced memory consumption
	MobileNetV3[43]	5.5	75.20	NAS-optimized architecture with h-swish activation and SE modules, 20% faster inference on mobile devices
DenseNet[18]	DenseNet-121	7.9	74.70	Dense cross-layer connections where each layer receives features from all preceding layers
	DenseNet-169	14.2	76.20	Balanced parameters and performance, uses transition layers for feature dimension compression
	DenseNet-201	20	77.30	Deep dense connections (201 layers) for smoother gradient flow
ShuffleNet	ShuffleNetV2[44]	1.3	69.70	Channel shuffle + grouped convolution, optimized memory access efficiency (120 FPS on mobile)
ESPNet	ESPNetv2[45]	3.5	72.10	Dynamic dilated convolution pyramid, multi-branch feature fusion, 35% lower FLOPs than MobileNetV2
EfficientNet[15]	EfficientNet-B0	5.3	77.30	Baseline compound scaling model ($\phi = 1.0$), balances depth/width/resolution
	EfficientNet-B1	7.8	79.10	$\phi = 1.1$ scaling with 240x240 input resolution (+1.8% accuracy)
	EfficientNet-B2	9.1	80.40	$\phi = 1.2$ scaling with 15% increased channels, suitable for moderate compute resources
	EfficientNet-B3	12	81.60	Medium scaling ($\phi = 1.3$), significant accuracy improvement at 1.8 GMACs
	EfficientNet-B4	19	82.90	Large scaling ($\phi = 1.4$), optimal server-side balance (V100 inference: 45ms)
	EfficientNet-B5	30	83.60	Deep scaling ($\phi = 1.6$), extreme accuracy optimization (9.9 GMACs)
GhostNet	GhostNet[30]	7.3	75.70	Ghost feature generation via cheap operations, 40% lower FLOPs than MobileNetV3
	GhostNetV2[31]	8.9	76.90	Cross-stage attention mechanism for dynamic feature enhancement (+15% mobile inference speed)
Proposed		4.6	78.70	Further lightweight design and accuracy optimization based on GhostNetV2

Table 4. Comparison of deep learning models for image classification performance and key features.

training and 20% for testing. To ensure the fairness and credibility of the experiment, we ensure that all models are trained and evaluated under the same experimental conditions. After multiple rounds of hyperparameter tuning, all models reached a balanced state. We set the following training parameters for each model: number of epochs = 26, batch size = 32, learning rate = 0.01. Due to the large sample size, we conduct the following experiments on a high-performance computing platform. The hardware information is shown in Table 3.

Model introduction

To more comprehensively evaluate the performance of different deep learning models in malware detection, we compare several popular network models and analyze their features and advantages in the context of malware detection. Below are the characteristics of several common deep learning models and their relevance to malware detection.

ResNet ResNet introduces residual connections, successfully addressing the issues of gradient vanishing and explosion in deep neural network training, allowing the network to be deeper and more effectively optimized. In malware detection, ResNet can automatically learn complex feature representations of malware, from low-level bytecode to high-level semantic features. It is particularly skilled at capturing subtle patterns in binary files and effectively distinguishing malicious code from legitimate software.

MobileNet MobileNet uses depthwise separable convolutions to reduce computational load and model size, making it particularly suitable for mobile devices and edge computing environments. In real-time malware detection, MobileNet, with its lightweight design, can perform efficient low-latency detection tasks on resource-constrained devices, making it ideal for applications in embedded systems.

DenseNet DenseNet promotes feature reuse through dense connections, enhancing the model's ability to capture fine-grained malicious behavior. For subtle malware behavior patterns, such as API call sequence analysis, DenseNet can effectively extract valuable features without overfitting, making it a powerful model for malware detection.

ShuffleNet ShuffleNet introduces channel shuffling operations to break the information isolation between grouped convolutions, enhancing information flow. It maintains high performance while reducing computational burden. In malware detection, ShuffleNet helps integrate different types of features through channel shuffling, improving the model's adaptability to the diversity of malicious code. Additionally, its lightweight design accelerates detection speed, meeting the real-time detection requirements.

ESPNet ESPNet utilizes efficient spatial pyramid modules and multi-scale branch structures to expand the receptive field, making it particularly suited for handling diverse malware data. Its low parameter count enables outstanding real-time processing capabilities in embedded or edge computing scenarios, making it well-suited for malware detection tasks that require efficient resource utilization.

EfficientNet EfficientNet balances the depth, width, and resolution of the network using a compound scaling method, allowing it to efficiently handle large-scale malware datasets even with limited resources. Its application in cloud server environments supports rapid model training and iteration, making it an ideal choice for large-scale malware detection tasks.

As shown in Table 4, GhostNetV2 demonstrates significant advantages in balancing accuracy-efficiency trade-offs and hardware compatibility, and was ultimately selected as the foundational model for this research.

Model	para	Training Time(s)	Test Time(s)	Accuracy (%)
ReNet34	21,797,672	2277.08	164.02	97.10
ResNet50	25,557,032	2315.64	141.10	97.00
ResNet101	44,549,160	2373.34	154.94	97.00
ResNet152	60,192,808	2371.84	139.78	97.10
MobileNetV2	3,504,872	2263.82	129.62	96.50
MobileNetV3	5,483,032	2264.24	139.10	94.10
DenseNet121	7,978,856	2206.52	144.10	97.10
DenseNet169	14,149,480	2248.00	134.94	97.10
DenseNet201	20,013,928	2367.04	154.30	97.20
ShuffleNetV2	2,278,604	2412.46	151.98	96.60
ESPNetV2	1,712,759	2422.36	130.88	96.70
EfficieNetV2	54,139,356	2418.96	134.70	97.00
EfficienNet_b0	5,288,548	2299.58	148.32	96.00
EfficienNet_b1	7,794,184	2351.98	161.96	96.30
EfficienNet_b2	9,109,994	2327.22	159.36	96.60
EfficienNet_b3	12,233,232	2310.06	164.36	96.60
EfficienNet_b4	19,341,616	2491.20	133.82	96.50
EfficienNet_b5	30,389,784	2368.82	143.42	95.80
GhostNetV2	6,156,908	2261.72	139.38	95.90
Proposed	4,653,105	2250.60	125.12	96.90

Table 5. Performance comparison of 20 models on Android malware detection.

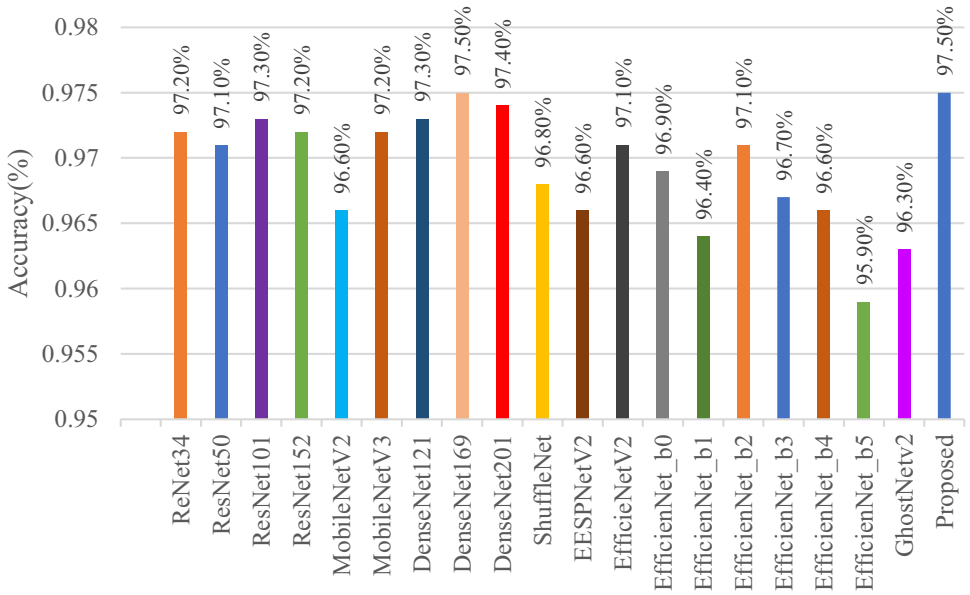


Fig. 11. Comparison of the detection accuracy of 20 models on RGB images using the LHE method.

Evaluation metrics

The performance of the detection models is evaluated by four metrics. They are precision, recall, F1-Score, accuracy and confusion matrix. Precision reflects the classification ability of the detection model, focusing specifically on its ability in predicting malware rather than all correctly classified samples. Recall is a measurement of the ability of a detection model to predict the proportion of malware in actual malware. F1-score is a metric that combines the harmonic mean of recall and precision to assess a model's ability in predicting malware. Accuracy indicates the overall performance of the detection model in classifying applications as malware or benign. The confusion matrix visualizes a classification model's predictions by comparing true labels with predicted labels. The definitions of the aforementioned evaluation metrics are as follows:

$$\text{Precision} = \frac{\text{TP}}{\text{FP} + \text{TP}} \tag{11}$$

$$\text{Recall/TPR} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{12}$$

$$\text{FPR} = \frac{\text{FP}}{\text{FP} + \text{TN}} \tag{13}$$

$$\text{F1} - \text{score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{14}$$

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{FP} + \text{TN} + \text{FN}} \tag{15}$$

where *TP* is the True Positive, *TN* is the True Negative, *FP* is the False Positive, *FN* is the False Negative.

Evaluation of images enhancement methods

Experiment on ordinary images

Table 5 evaluates 20 state-of-the-art deep learning models for Android malware detection using ordinary RGB images. Experimental results indicate that the DenseNet201 model achieved the highest detection accuracy, reaching 97.2%. In contrast, MobileNetV2 has the fewest parameters, followed closely by our proposed method. Notably, as compared to GhostNetV2, our optimized model significantly reduced the parameter size to 4.6M, a decrease of approximately 1.5M. This improvement also enhanced training and testing efficiency, reducing training time to 2250.60s and testing time to 125.12s. Despite the decrease in parameters and computational costs, our model's accuracy improved by 1%, coming within 0.2% of the top-performing DenseNet201.

Experiment on LHE

To further improve the detection accuracy of malware, Fig. 11 shows 20 state-of-the-art deep learning models for Android malware detection using RGB images with LHE. It can be observed that the use of the LHE method improves the detection accuracy of all models, ranging from about 0.1–3.1%. Our method and DenseNet169 had the highest accuracy rates, both at 97.5%.

Model	Average type	Precision	Recall	F1-score
ReNet34	Macro	0.9719	0.9721	0.9719
	Weighted	0.9725	0.9720	0.9720
ResNet50	Macro	0.9711	0.9711	0.9711
	Weighted	0.9711	0.9711	0.9711
ResNet101	Macro	0.9734	0.9734	0.9734
	Weighted	0.9734	0.9734	0.9734
ReNet152	Macro	0.9719	0.9719	0.9719
	Weighted	0.9720	0.9720	0.9720
MobileNetV2	Macro	0.9656	0.9655	0.9655
	Weighted	0.9656	0.9656	0.9655
MobileNetV3	Macro	0.9726	0.9721	0.9723
	Weighted	0.9725	0.9723	0.9723
DenseNet121	Macro	0.9754	0.9750	0.9752
	Weighted	0.9753	0.9752	0.9752
DenseNet169	Macro	0.9758	0.9757	0.9757
	Weighted	0.9757	0.9757	0.9757
DenseNet201	Macro	0.9748	0.9747	0.9748
	Weighted	0.9748	0.9748	0.9748
ShuffleNetV2	Macro	0.9623	0.9620	0.9621
	Weighted	0.9622	0.9622	0.9621
ESPNetV2	Macro	0.9663	0.9658	0.9660
	Weighted	0.9661	0.9660	0.9660
EfficienNetV2	Macro	0.9702	0.9703	0.9703
	Weighted	0.9703	0.9704	0.9704
EfficienNet_b0	Macro	0.9692	0.9687	0.9689
	Weighted	0.9690	0.9689	0.9689
EfficienNet_b1	Macro	0.9597	0.9597	0.9597
	Weighted	0.9597	0.9597	0.9597
EfficienNet_b2	Macro	0.9705	0.9703	0.9704
	Weighted	0.9704	0.9704	0.9704
EfficienNet_b3	Macro	0.9622	0.9621	0.9621
	Weighted	0.9622	0.9622	0.9622
EfficienNet_b4	Macro	0.9627	0.9618	0.9621
	Weighted	0.9624	0.9622	0.9621
EfficienNet_b5	Macro	0.9087	0.9088	0.9087
	Weighted	0.9088	0.9088	0.9088
GhostNetV2	Macro	0.9631	0.9623	0.9626
	Weighted	0.9629	0.9626	0.9626
Proposed	Macro	0.9766	0.9761	0.9763
	Weighted	0.9765	0.9763	0.9763

Table 6. Detection Performance comparison of 20 state-of-the-art deep learning models on RGB images using the LHE method.

Table 6 presents the macro-average and weighted-average precision, recall, and F1-score evaluation values for 20 state-of-the-art deep learning models used in malware detection on RGB images using the LHE method. Our model achieved the highest values in precision, recall, and F1-score, each approximately equal to 0.976. The higher precision and recall values indicate that the the model performs well in detecting malware.

Figure 12 shows the confusion matrices for the detection of RGB images with LHE using 20 models. Experimental results show that our method accurately classifies 98% of benign images and only 2% of images are misidentified as malware. Furthermore, the method successfully identified 97% of malware images, with only 3% misclassified as benign.

Experiment on LHE_Gabor

In Table 7, we evaluate the detection performance of 20 state-of-the-art deep learning models for Android malware detection using RGB images with LHE_Gabor. The results show a significant reduction in both training and testing times. This improvement is due to the fact that RGB images are transformed into single-channel

(a) ReNet34 (b) ResNet50 (c) ResNet101 (d) ReNet152

(e) MobileNetV2 (f) MobileNetV3 (g) DenseNet121 (h) DenseNet169

(i) DenseNet201 (j) ShuffleNetV2 (k) ESPNetV2 (l) EfficienNetV2

(m) EfficienNet_b0 (n) EfficienNet_b1 (o) EfficienNet_b2 (p) EfficienNet_b3

(q) EfficienNet_b4 (r) EfficienNet_b5 (s) GhostNetV2 (t) Proposed

Fig. 12. Confusion matrices obtained from the RGB images with LHE using 20 models.

grayscale images through Gabor processing. Deep learning models can train and infer more quickly on these single-channel images.

It should be noted that despite the reduced number of data channels in the application image, the detection accuracy of almost every model has improved. This indicates that using single-channel grayscale images does not necessarily reduce the models' learning capabilities. Adding Gabor processing to LHE not only saves time in training and testing, but also further improves detection accuracy.

Model	Training time(s)	Test time(s)	Accuracy (%)
ReNet34	1042.96	54.68	97.30
ResNet50	1031.82	53.90	97.20
ResNet101	1034.38	60.98	97.30
ResNet152	1074.22	62.92	97.30
MobileNetV2	1026.04	54.22	96.80
MobileNetV3	1022.40	54.30	97.30
DenseNet121	1021.30	58.82	97.50
DenseNet169	1027.30	64.16	97.60
DenseNet201	1030.16	66.82	97.50
ShuffleNetV2	1009.46	54.10	97.00
ESPNetV2	1017.56	59.48	96.80
EfficieNetV2	1052.74	71.70	97.30
EfficienNet_b0	1021.48	60.96	97.00
EfficienNet_b1	1023.67	67.92	96.70
EfficienNet_b2	1018.26	64.82	97.20
EfficienNet_b3	1004.32	63.42	96.90
EfficienNet_b4	1016.08	75.64	96.90
EfficienNet_b5	1025.64	66.72	96.20
GhostNetV2	1011.88	59.40	96.50
Proposed	1012.42	53.40	97.70

Table 7. Detection Performance comparison of 20 state-of-the-art deep learning models on images using LHE_Gabor.

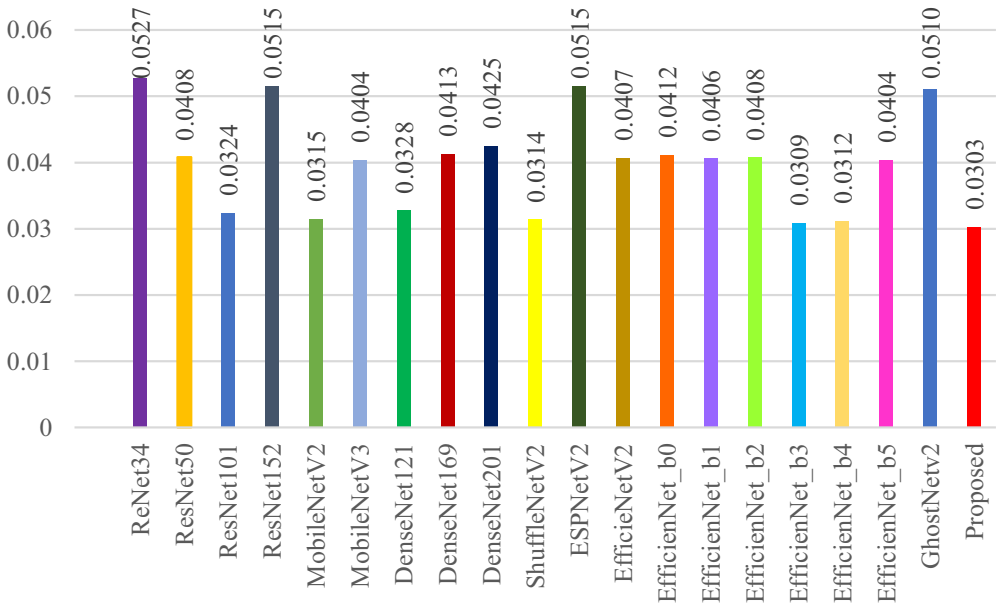


Fig. 13. FPR comparison for different models using LHE_Gabor.

Furthermore, Fig. 13 shows the FPR values of 20 models after applying the LHE_Gabor method. A lower FPR indicates better detection performance, and our method achieves the lowest FPR.

The ROC curve for the proposed method in distinguishing between malware and benign samples is presented in Fig. 14. The curve's proximity to the top-left corner indicates strong model performance, with an AUC score of 0.977, suggesting that the feature extraction process from malware images by the proposed model is highly effective.

From Fig. 15, it is evident that the improved GhostNetV2 model demonstrates a rapid convergence during training. Both the training and validation losses remain below 0.1 within 26 epochs, while the validation accuracy stabilizes above 99.7%, with the accuracy curve approaching 1.0. Furthermore, the training and validation curves

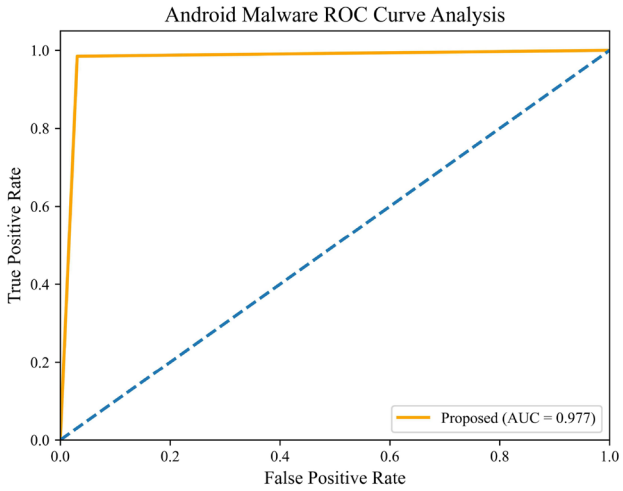


Fig. 14. ROC curve analysis for the proposed method on Android malware detection.

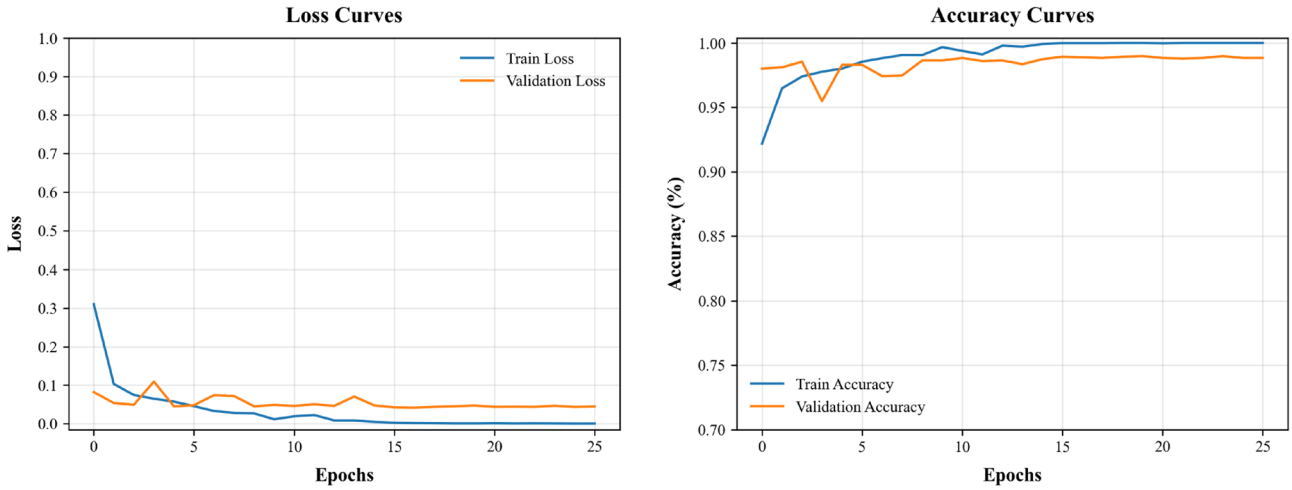


Fig. 15. Learning curve during the training process of our proposed model.

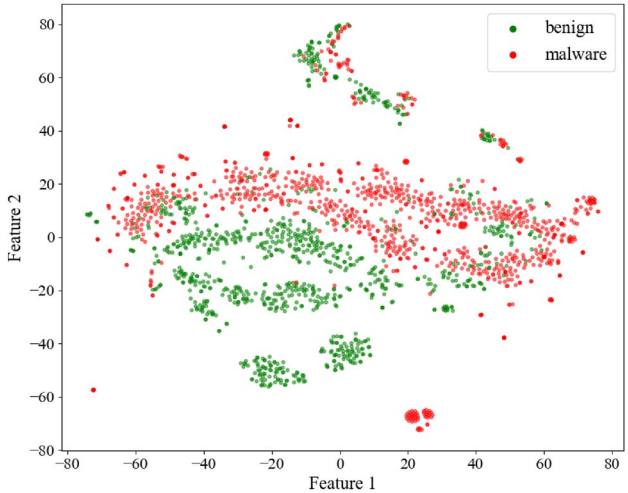


Fig. 16. Visualization of the proposed model on the malware detection.

Method	PRelu	CS	ECA	ACC(%)	F1-score(%)	Time(s)	Para	Flops(G)
GhostNetV2	×	×	×	96.3	96.29	69.69	6,156,908	3.461
New_GhostNetV2_1	✓	×	×	96.5	96.53	65.18	6,156,942	3.461
New_GhostNetV2_2	✓	✓	×	96.9	96.94	63.37	6,156,942	3.461
Proposed	✓	✓	✓	97.5	97.63	62.56	4,653,105	3.460

Table 8. Performance comparison of GhostNetV2 in ablation study.

	RGB image		RGB image with LHE_Gabor	
Detection model	MalGAN (%)	DCGAN (%)	MalGAN (%)	DCGAN (%)
ReNet34	82.10	80.00	90.30	89.80
ResNet50	87.20	86.50	91.40	90.90
ResNet101	86.10	85.50	89.50	89.00
ResNet152	80.40	79.80	91.60	91.00
MobileNetV2	78.60	77.50	83.80	82.30
MobileNetV3	80.10	79.80	90.10	89.70
DenseNet121	85.20	84.50	89.20	88.60
DenseNet169	89.60	88.90	90.70	90.40
DenseNet201	87.40	86.90	90.30	89.80
ShuffleNetV2	70.40	69.50	84.60	83.90
ESPNetV2	72.50	71.80	84.10	83.60
EfficieNetV2	87.30	86.80	90.80	90.40
EfficienNetb0	77.20	76.00	91.20	90.70
EfficienNetb1	85.60	84.80	91.60	91.10
EfficienNetb2	85.80	85.10	86.50	86.00
EfficienNetb3	85.90	85.30	91.40	90.80
EfficienNetb4	86.20	85.70	90.30	90.10
EfficienNetb5	85.90	85.00	89.30	88.90
GhostNetV2	83.10	82.40	89.20	88.70
Proposed	85.30	84.80	92.00	91.60

Table 9. Accuracy comparison of different models with and without LHE_Gabor, using MalGAN and DCGAN configurations.

are highly synchronized, with no apparent signs of overfitting. These results indicate that the model exhibits favorable convergence, high accuracy, and strong generalization ability during the training process.

To evaluate the performance of our model in the classification task of malware and benign applications, we utilize tdistributed Stochastic Neighbor Embedding (t-SNE) for visualizing the features extracted from the GAP layer. t-SNE is an effective dimensionality reduction technique that maps high-dimensional data into a lower-dimensional space while preserving the relative distances and local structures between data points as much as possible. In this study, we set the t-SNE learning rate to 200.0, the early exaggeration parameter to 12.0, the perplexity to 30.0, and the number of iterations to 1000. The visualization results in Fig. 16 clearly demonstrate a distinct separation between the malware and benign applications, with only a small degree of overlap. This indicates that our detection methods exhibit strong discriminatory ability in classifying these two categories of samples.

Evaluation of improved model

To further validate the effectiveness of the improvements made to the GhostNetV2 model, we conduct a series of ablation experiments. Based on the original GhostNetV2, New_GhostNetV2_1 incorporates only PRelu, while New_GhostNetV2_2 includes both CS and PReLU. Additionally, the model improvement methods we proposed introduces ECA alongside the previous components. We subsequently compared the four models across multiple metrics, including accuracy, precision, F1-score, testing time, parameter, and flops.

Table 8 shows that adding PReLU increases model parameters by 0.034k, but it outperforms GhostNetV2 in accuracy and F1-Score. This improvement is due to PReLU's ability to enhance the model's non-linear expression, allowing it to capture more complex patterns and avoid the "dying ReLU" issue, thus improving classification accuracy.

When we add CS, parameters remain unchanged, while both accuracy and F1-Score increase, test time decreases, and flops stay stable. CS helps reduce redundant feature maps and retains more meaningful information without increasing computational cost.

Detection model	RGB image		RGB image with LHE_Gabor	
	MalGAN	DCGAN	MalGAN	DCGAN
ReNet34	0.8196	0.8000	0.9031	0.8981
ResNet50	0.8717	0.8651	0.9142	0.9094
ResNet101	0.8609	0.8554	0.8951	0.8864
ResNet152	0.8041	0.7984	0.9164	0.9089
MobileNetV2	0.7858	0.7751	0.8382	0.8251
MobileNetV3	0.8096	0.7982	0.9014	0.8969
DenseNet121	0.8517	0.8451	0.9021	0.8868
DenseNet169	0.8668	0.8893	0.9172	0.9039
DenseNet201	0.8638	0.8692	0.9030	0.8981
ShuffleNetV2	0.7036	0.6951	0.8460	0.8391
ESPNetV2	0.7314	0.7180	0.8412	0.8361
EfficieNetV2	0.8534	0.8680	0.9082	0.9042
EfficienNetb0	0.7731	0.7600	0.9124	0.9072
EfficienNetb1	0.8563	0.8482	0.9162	0.9114
EfficienNetb2	0.8484	0.8414	0.8851	0.8600
EfficienNetb3	0.8591	0.8535	0.9142	0.9082
EfficienNetb4	0.8621	0.8572	0.9030	0.9014
EfficienNetb5	0.8591	0.8500	0.8931	0.8890
GhostNetV2	0.8314	0.8241	0.8921	0.8875
Proposed	0.8537	0.8482	0.9200	0.9164

Table 10. TPR comparison of different models with and without LHE_Gabor, using malGAN and DCGAN configurations.

Detection model	RGB image		RGB image with LHE_Gabor	
	MalGAN	DCGAN	MalGAN	DCGAN
ReNet34	0.1912	0.2015	0.1221	0.1196
ResNet50	0.1323	0.1432	0.0928	0.1056
ResNet101	0.1429	0.1549	0.1143	0.1124
ResNet152	0.2011	0.2198	0.0942	0.0932
MobileNetV2	0.2357	0.2346	0.1811	0.1834
MobileNetV3	0.2002	0.2104	0.1032	0.1145
DenseNet121	0.1515	0.1619	0.1145	0.1256
DenseNet169	0.1104	0.1206	0.1076	0.1075
DenseNet201	0.1312	0.1534	0.1079	0.1089
ShuffleNetV2	0.3012	0.3189	0.1689	0.1754
ESPNetV2	0.2805	0.2916	0.1667	0.1758
EfficieNetV2	0.1323	0.1469	0.1006	0.1042
EfficienNetb0	0.4123	0.4424	0.2901	0.2916
EfficienNetb1	0.2303	0.2404	0.0925	0.0932
EfficienNetb2	0.1501	0.1584	0.1401	0.1421
EfficienNetb3	0.1502	0.1545	0.0925	0.1001
EfficienNetb4	0.1412	0.1594	0.1079	0.1054
EfficienNetb5	0.1516	0.1585	0.1012	0.1213
GhostNetV2	0.1745	0.1815	0.1213	0.1294
Proposed	0.1524	0.1635	0.0824	0.0903

Table 11. FPR comparison of different models with and without LHE_Gabor, using malGAN and DCGAN configurations.

Finally, the introduction of ECA reduces FLOPS slightly and dramatically lowers parameters to about 1.5 M. This reduction is due to the efficiency of ECA, which uses fewer parameters but improves the attention mechanism. At this stage, accuracy and F1 scores reach their highest, and test time is minimized, showing that ECA optimizes the model's performance, making it more accurate and computationally efficient.

Therefore, the model improvement method proposed in this study is considered to be the optimal solution when considering these important of performance changes.

Detection of adversarial samples

The aim of the experiments in this subsection is to further evaluate the performance of our method in detecting unknown malware. Considering the emergence of new adversarial malware samples that can significantly reduce the classification ability of neural network models, 8246 adversarial malware samples generated by MalGAN and DCGAN[46], and 7124 benign samples are used as the dataset for the following experiments.

Table 9 compares the accuracy of 20 different models in detecting images of adversarial samples. We detected both RGB images of the applications, and RGB images with LHE_Gabor to evaluate the effectiveness of our image enhancement method at the same time.

The experimental results show that all 20 models are significantly reduced in accuracy when detecting RGB images of adversarial samples. However, the image enhancement method proposed we proposed can effectively improve their detection accuracy. Meanwhile, our detection model has the highest accuracy of 92.0% and 91.6% in detecting these unknown adversarial malware samples.

Table 10 presents the TPR of 20 models in detecting adversarial samples. A higher TPR indicates better classification performance. The table includes results for two adversarial attacks, MalGAN and DCGAN. The results show that, without the LHE_gabor method, the average TPR of the models is approximately 0.82, indicating relatively weak detection performance. However, after applying the LHE_gabor method, the detection capability of the models significantly improves to around 0.90, with our method achieving the best TPR in detecting adversarial samples.

Table 11 shows the performance of these models in terms of FPR. A lower FPR indicates better performance, with a reduced probability of misclassification. The results demonstrate that, when facing adversarial attacks, the models using the LHE_gabor method exhibit a significant reduction in FPR, with all models showing smaller FPR values compared to when the LHE_gabor method was not applied. This suggests that our LHE_gabor method effectively reduces false positives, enhancing the stability and reliability of the models.

Overall, whether considering accuracy, TPR, FPR, or other metrics, our method demonstrates clear advantages and effectively improves model performance in adversarial sample detection tasks.

Discussion

Image enhancement The application of LHE significantly enhanced the texture feature differentiation, leading to a notable improvement in model performance. Our model, in combination with LHE and Gabor filtering, demonstrated an increase in accuracy by 0.1–3.1%, achieving a top accuracy of 97.7%. This improvement is further evidenced by the macro-average F1-score of 0.9763, outperforming other models in the comparison. Notably, our approach had a misclassification rate of only 2% for benign samples, which indicates a very low FPR, while the 97% recall rate for malicious samples reflects the model's strong ability to correctly identify threats. The combination of LHE and Gabor filtering also led to a significant reduction in both training and testing times, decreasing them by 50–60%, with training time reduced to just 1021.30 seconds. Furthermore, the model achieved an impressive AUC of 0.977 and the lowest FPR of 0.0303. The feature visualization using t-SNE confirmed that the enhanced features were well-separated, suggesting that the enhancement strategy is not only effective but also robust.

Network architecture improvement In the ablation experiments, several modifications to the network architecture proved to be beneficial. The introduction of parametric PReLU activation resulted in a 1.6% improvement in the F1-score, demonstrating its positive impact on the model's ability to balance precision and recall. The incorporation of CS further boosted the TPR by 3.1%, highlighting its effectiveness in improving the model's sensitivity to malicious samples. Additionally, replacing the SE block with ECA reduced the model's parameters by 24.5%, from 6.15 to 4.65M, demonstrating significant computational efficiency improvements. These architectural enhancements contributed to a highly optimized model, achieving 97.7% accuracy with only 23.2% of the parameters of DenseNet201. Compared to 20 mainstream models, our approach outperformed GhostNetV2 by 1.8%, making it both efficient and competitive in terms of performance.

Adversarial sample robustness Adversarial robustness is a critical challenge for deep learning models, particularly in security applications. Our method demonstrated exceptional resilience to adversarial samples generated by MalGAN and DCGAN. With the combination of LHE and Gabor filtering, our model achieved detection accuracies of 92.0% and 91.6%, respectively, surpassing other models by 15–30%. TPR for MalGAN and DCGAN adversarial samples reached 0.93 and 0.92, respectively, the highest among the models tested. This robust performance in the presence of adversarial attacks is a testament to the efficacy of our proposed enhancements, ensuring that the model remains effective even under adversarial conditions.

In conclusion, our approach successfully balances detection accuracy, computational efficiency, and robustness to adversarial samples. The combination of advanced image enhancement techniques, architectural improvements, and the ability to withstand adversarial attacks positions our model as a feasible solution for real-time malware detection on mobile devices. This approach not only demonstrates high detection performance but also offers a computationally efficient solution, making it suitable for resource-constrained environments. Future work could further explore the integration of additional robust features to enhance the model's performance in even more challenging scenarios, such as with novel or unseen adversarial attacks.

Conclusion and future work

This paper proposes a novel malware detection method based on an improved GhostNetV2, aimed at enhancing the detection performance for both normal malware and adversarial samples. First, we introduce a technique that applies local histogram equalization and Gabor methods to Android application images. Next, we develop a detection model for malware and adversarial samples using the improved GhostNetV2 algorithm. Finally, in our experiments, we analyze the performance of 20 state-of-the-art network models in detecting malware and adversarial samples, with results demonstrating that our proposed method performs exceptionally well.

However, it is important to note that the method in this paper primarily focuses on identifying novel adversarial attack samples from a detection perspective. Recent research indicates that new attack methods continue to emerge, especially attacks targeting artificial intelligence classifiers. Future research can be developed in several directions: further collection of a large number of new adversarial malicious code samples, conduct more extensive testing and validation of these adversarial samples to analyze malicious attack patterns, incorporate other mechanisms for updating the model as new threats emerge, explore methods to further improve the detection model's computational performance, such as parallel processing techniques or hardware acceleration, and validate their reliability in real-world applications.

Data availability

The datasets generated and analyzed during the current study are available in the virusshare repository, https://virusshare.com/, CICMalDroid repository, https://www.unb.ca/cic/datasets/index.html, GooglePlay repository, https://www.techspot.com/downloads/. Another dataset that support the findings of this study are available from Drebin, but restrictions apply to the availability of these data, which were used under license for the current study and so are not publicly available. Data are however available from the authors upon reasonable request and with permission of the authors of Drebin, https://www.ndss-symposium.org/ndss2014/ndss-2014-programme/drebin-effective-and-explainable-detection-android-malware-your-pocket/.

Received: 26 December 2024; Accepted: 17 June 2025
Published online: 11 July 2025

References

1. StatCounter. Mobile vendor market share worldwide on Oct 2024 (2024). https://gs.statcounter.com/vendor-market-share/mobile/worldwide/2024.
2. Kaspersky. IT threat evolution in Q2 2024. Mobile statistics (2024). https://securelist.com/it-threat-evolution-q2-2024-mobile-statistics/113678/.
3. Sophos. Sophos Threat Report: Ransomware and the rise of the data (2024). https://news.sophos.com/en-us/2024/04/30/the-state-of-ransomware-2024/.
4. Pan, Y., Ge, X., Fang, C. & Fan, Y. A systematic literature review of android malware detection using static analysis. *IEEE Access* **8**, 116363–116379 (2020).
5. Liu, K. et al. A review of android malware detection approaches based on machine learning. *IEEE Access* **8**, 124579–124607 (2020).
6. Sasidharan, S. K. & Thomas, C. Prodroid-an android malware detection framework based on profile hidden Markov model. *Pervasive Mob. Comput.* **72**, 101336 (2021).
7. Qiu, J. et al. A survey of android malware detection with deep neural models. *ACM Comput. Surveys (CSUR)* **53**, 1–36 (2020).
8. Zhang, W., Luktarhan, N., Ding, C. & Lu, B. Android malware detection using TCN with bytecode image. *Symmetry* **13**, 1107 (2021).
9. Shen, L. et al. Self-attention based convolutional-LSTM for android malware detection using network traffics grayscale image. *Appl. Intell.* **53**, 683–705 (2023).
10. Yadav, P., Menon, N., Ravi, V., Vishvanathan, S. & Pham, T. D. Efficientnet convolutional neural networks-based android malware detection. *Comput. & Secur.* **115**, 102622 (2022).
11. Tang, J., Li, R., Jiang, Y., Gu, X. & Li, Y. Android malware obfuscation variants detection method based on multi-granularity opcode features. *Futur. Gener. Comput. Syst.* **129**, 141–151 (2022).
12. Ding, Y., Zhang, X., Hu, J. & Xu, W. Android malware detection method based on bytecode image. *J. Ambient Intell. Hum. Comput.* **14**(5), 6401–6410 (2020).
13. Singh, J. et al. Classification and analysis of android malware images using feature fusion technique. *IEEE Access* **9**, 90102–90117 (2021).
14. Wang, Z., Liu, Q., Wang, Z. & Chi, Y. Deep learning-based multi-classification for malware detection in IoT. *J. Circuits Syst. Comput.* **31**, 2250297 (2022).
15. Tan, M. & Le, Q. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International conference on machine learning*, 6105–6114 (PMLR, 2019).
16. Ye, G., Zhang, J., Li, H., Tang, Z. & Lv, T. Android malware detection technology based on lightweight convolutional neural networks. *Secur. Commun. Netw.* **2022**, 8893764 (2022).
17. Ksibi, A., Zakariah, M., Almuqren, L. & Alluhaidan, A. S. Efficient android malware identification with limited training data utilizing multiple convolution neural network techniques. *Eng. Appl. Artif. Intell.* **127**, 107390 (2024).
18. Huang, G., Liu, Z., Van Der Maaten, L. & Weinberger, K. Q. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 4700–4708 (2017).
19. Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J. & Wojna, Z. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2818–2826 (2016).
20. He, K., Zhang, X., Ren, S. & Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 770–778 (2016).
21. Simonyan, K., Zisserman, A. Very deep convolutional networks for large-scale image recognition. *arXiv preprint* arXiv:1409.1556 (2014).
22. Zhu, H., Wei, H., Wang, L., Xu, Z. & Sheng, V. S. An effective end-to-end android malware detection method. *Expert Syst. Appl.* **218**, 119593 (2023).
23. Hu, W. & Tan, Y. Generating adversarial malware examples for black-box attacks based on gan. In *International Conference on Data Mining and Big Data*, 409–423 (Springer, 2022).
24. Hu, W. & Tan, Y. Black-box attacks against rnn based malware detection algorithms. In *Workshops at the Thirty-Second AAAI Conference on Artificial Intelligence* (2018).

25. Wang, Z. et al. CNN-and GAN-based classification of malicious code families: A code visualization approach. *Int. J. Intell. Syst.* **37**, 12472–12489 (2022).
26. Li, S. et al. Gmadv: An android malware variant generation and classification adversarial training framework. *J. Inf. Secur. Appl.* **84**, 103800 (2024).
27. Gao, C. et al. A new adversarial malware detection method based on enhanced lightweight neural network. *Comput. Secur.* **147**, 104078 (2024).
28. Yin, Z. *et al.* Adversarial examples detection with enhanced image difference features based on local histogram equalization. *arXiv preprint* arXiv:2305.04436 (2023).
29. Duan, R. *et al.* Advdrop: Adversarial attack to dnns by dropping information. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 7506–7515 (2021).
30. Han, K. *et al.* Ghostnet: More features from cheap operations. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 1580–1589 (2020).
31. Tang, Y. et al. Ghostnetv2: Enhance cheap operation with long-range attention. *Adv. Neural. Inf. Process. Syst.* **35**, 9969–9982 (2022).
32. He, K., Zhang, X., Ren, S. & Sun, J. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, 1026–1034 (2015).
33. Zhang, X., Zhou, X., Lin, M. & Sun, J. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 6848–6856 (2018).
34. Hu, J., Shen, L. & Sun, G. Squeeze-and-excitation networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 7132–7141 (2018).
35. Wang, Q. *et al.* Eca-net: Efficient channel attention for deep convolutional neural networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 11534–11542 (2020).
36. Lashkari, A. H., Kadir, A. F. A., Taheri, L. & Ghorbani, A. A. Toward developing a systematic approach to generate benchmark android malware datasets and classification. In *2018 International Carnahan conference on security technology (ICCST)*, 1–7 (IEEE, 2018).
37. VirusShare. A database that provides malicous code. https://virusshare.com/.
38. Arp, D. et al. Drebin: Effective and explainable detection of android malware in your pocket. *In Ndss* **14**, 23–26 (2014).
39. Viennot, N., Garcia, E. & Nieh, J. A measurement study of google play. In *The 2014 ACM international conference on Measurement and modeling of computer systems*, 221–233 (2014).
40. VirusTotal. Virustotal-free online virus, malware and url scanner. https://www.virustotal.com/gui/home/upload.
41. Kaspersky. Antivirus software and the version used in this study is standard 21.17. https://www.kaspersky.com/enterprise-security.
42. Sandler, M., Howard, A., Zhu, M., Zhmoginov, A. & Chen, L.-C. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 4510–4520 (2018).
43. Howard, A. *et al.* Searching for mobilenetv3. In *Proceedings of the IEEE/CVF international conference on computer vision*, 1314–1324 (2019).
44. Ma, N., Zhang, X., Zheng, H. -T. & Sun, J. Shufflenet v2: Practical guidelines for efficient cnn architecture design. In *Proceedings of the European conference on computer vision (ECCV)*, 116–131 (2018).
45. Mehta, S., Rastegari, M., Shapiro, L. & Hajishirzi, H. Espnetv2: A light-weight, power efficient, and general purpose convolutional neural network. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 9190–9200 (2019).
46. Radford, A. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint* arXiv:1511.06434 (2015).

Acknowledgements

This work was supported by the Scientific and Technological Innovation Team for Qinghai-Tibetan Plateau Research in Southwest Minzu University (Grant No. 2024CXTD09). Sichuan Science and Technology Program (2024ZHCG0194) and Sichuan Science and Technology Program (Grant No. 2025YFHZ0178).

Author contributions

Conceptualization: Y.D. and C.X.G.; Data curation: M.T.C., X.C, L.L.X. and A.J.N.; Methodology: Y.D., C.X.G., M.T.C., X.C., L.L.X., and A.J.N.; Writing original draft: Y.D., M.T.C, X.C., and C.X.G.; Visualization: Y.D. and C.X.G.; Supervision: Y.D.

Declarations

Competing interests

The authors declare that they have no competing interests.

Additional information

Correspondence and requests for materials should be addressed to X.C.

Reprints and permissions information is available at www.nature.com/reprints.

Publisher's note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.