Computers, Materials & Continua

Tech Science Press

DOI: 10.32604/cmc.2023.045512

ARTICLE

Lightweight Malicious Code Classification Method Based on Improved SqueezeNet

Li Li*, Youran Kong and Qing Zhang

School of Computer and Control Engineering, Northeast Forestry University, Harbin, 150040, China

*Corresponding Author: Li Li. Email: lli@nefu.edu.cn

Received: 29 August 2023 Accepted: 15 November 2023 Published: 30 January 2024

ABSTRACT

With the growth of the Internet, more and more business is being done online, for example, online offices, online education and so on. While this makes people's lives more convenient, it also increases the risk of the network being attacked by malicious code. Therefore, it is important to identify malicious codes on computer systems efficiently. However, most of the existing malicious code detection methods have two problems: (1) The ability of the model to extract features is weak, resulting in poor model performance. (2) The large scale of model data leads to difficulties deploying on devices with limited resources. Therefore, this paper proposes a lightweight malicious code identification model Lightweight Malicious Code Classification Method Based on Improved SqueezeNet (LCMISNet). In this paper, the MFire lightweight feature extraction module is constructed by proposing a feature slicing module and a multi-size depthwise separable convolution module. The feature slicing module reduces the number of parameters by grouping features. The multi-size depthwise separable convolution module reduces the number of parameters and enhances the feature extraction capability by replacing the standard convolution with depthwise separable convolution with different convolution kernel sizes. In addition, this paper also proposes a feature splicing module to connect the MFire lightweight feature extraction module based on the feature reuse and constructs the lightweight model LCMISNet. The malicious code recognition accuracy of LCMISNet on the BIG 2015 dataset and the Malimg dataset reaches 98.90% and 99.58%, respectively. It proves that LCMISNet has a powerful malicious code recognition performance. In addition, compared with other network models, LCMISNet has better performance, and a lower number of parameters and computations.

KEYWORDS

Lightweight neural network; malicious code classification; feature slicing; feature splicing; multi-size depthwise separable convolution

1 Introduction

Malicious code is program code that is designed to attack computer systems or networks and is capable of having a significant impact on computer systems or networks, spreading quickly, and having a high replication capability [1]. Malicious code is characterized by the fact that it spreads through various means, has significant destructive potential, is difficult to detect and eliminate, and has a high degree of obfuscation [2]. With the development of the Internet, online offices are becoming

increasingly popular. While this has made people's lives more convenient, it has also led to an increase in security attacks on global networks, resulting in an increasing number of data breaches. According to the China Cybersecurity Report 2022, 73.55 million virus samples with 124 million virus infections and 45.15 million new Trojan viruses were intercepted in 2022. These included 579,200 ransomware samples, 2.61 million mining virus samples, and 1,520,500 samples of cell phone viruses. The results of the Global Cybersecurity Landscape Report 2022 survey revealed that 79% of respondents said they had experienced a ransomware attack; 35% admitted they had been prevented from accessing their data and systems by one or more attacks; 51% of respondents reported attacks on business email, up from 42% the previous year; and 39% of organizations reported internal attacks, up from 27% the previous year. These two studies show that malware attacks are becoming more common these days. Attacks with malicious code not only damage an organization's reputation, but can also lead to unplanned downtime that costs countless amounts of money.

Therefore, it is necessary to identify potential malicious code in the computer system using malicious code detection technologies to prevent malicious code attacks. However, malicious code often uses a variety of obfuscation, encryption and other methods to increase its diversity and obfuscation, which makes malicious code detection even more difficult. Traditional machine learning-based malicious code identification techniques cannot fully extract features and have difficulty identifying malicious codes that are hidden or obfuscated, which can easily lead to missing or false alarms [3,4]. Reference [5] converted malicious code files into grayscale images, extracts local and global texture features, and then uses Support Vector Machine (SVM) to classify the malicious code. However, the classification accuracy based on the BIG 2015 dataset is only 94.50%. Reference [6] extracted the grayscale covariance matrix (GLCM) and second-order statistical texture as features and then uses integrated learning to classify them. However, based on the Malimg dataset, the classification accuracy was only 98.58%. From the above two approaches, it can be seen that although machine learning techniques can successfully categorize malicious code, the machine learning features cannot yet fully and accurately describe the malicious code. In addition, the machine learning features usually have to be extracted manually, which is associated with high costs.

Deep neural networks already have more mature applications in many areas, for example, in medicine [7], robots [8], games [9], stock prediction [10] and image processing [11,12], etc. Without human intervention, neural networks can automatically extract features. Neural networks are not only more cost-effective, but can also extract richer and more systematic features than machine learning. However, as the depth of the network hierarchy increases, the problems of gradient vanishing and gradient explosion gradually appear, which makes it difficult to train the network. The skip connection technique [13] and Batch Normalization [14] alleviate this problem, making deep network models easier and more stable to learn features. These two techniques are often applied to deep neural network models. Malicious code detection by neural networks has gradually become the main direction in malicious code detection [15,16].

Reference [17] converted the malicious code into documents. Then the Word to Vector (Word2Vec) algorithm is used to obtain the word vectors of the assembler instructions, and then each document is converted into a matrix, which is normalized and mapped to a grayscale image. Finally, the classification is performed with Le Net5. Reference [18] used Similarity Hashing (SimHash) to fuse malicious code opcode sequences with predictive coding from Recurrent Neural Network (RNN) to generate feature images. Then, a Convolutional Neural Network (CNN) is used to classify the malicious code images. Reference [19] visualized the malicious code as RGB images and uses the Visual Geometry Group Network (VGGNet) network to identify the malicious code. Reference [20] used a CNN and intelligent algorithms to identify and classify grayscale images converted from executable

files containing malicious code. Reference [21] converted the malicious code into a grayscale map and extracts a fingerprint feature image of the malicious code using a modified grayscale covariance matrix. A CNN model is then used for detection. Reference [22] converted malicious code files into color images and uses a fine-tuned Convolutional Neural Network (CNN) network Image-based malware classification using finetuned convolutional neural network architecture (IMCFN) to classify malicious code images.

The above methods for classifying malicious code suffer from three problems: (1) The methods for processing the features of malicious code in the preprocessing stage of data are complex and costly. (2) Insufficient feature extraction makes it challenging to identify obfuscated malicious code and results in a low accuracy rate. (3) The models used have many parameters and a large amount of data, which requires a large amount of memory and makes it difficult to deploy the model on devices with limited resources. Therefore, the primary goals of this research are to simplify the data pre-processing phase, reduce the number of parameters and the memory footprint of the malicious code identification model, and improve the malicious code identification capabilities of the model.

The main contributions are listed below:

1. A lightweight malicious code identification model (LCMISNet) is proposed.

2. A simple method for visualizing malicious code is proposed in the preprocessing phase of data.

3. A lightweight MFire module for feature extraction is proposed. The MFire module consists of a feature slicing module and a multi-size depthwise separable convolution module to reduce the number of parameters and improve the feature extraction capability.

4. A feature splicing module based on feature reuse is proposed. This module is responsible for concatenating the lightweight feature extraction module MFire to create the lightweight model (LCMISNet).

5. Based on two different datasets, comparative experiments are conducted with other malicious code detection methods and deep neural networks commonly used for image classification. The results show that the LCMISNet model proposed in this paper has obvious advantages in accuracy and model size.

Structure of the paper: Section 2 introduces the current lightweight malicious code detection methods and the techniques on which the model of this paper is based. Section 3 describes the LCMISNet model proposed in this study in detail. Section 4 describes each experiment in detail, and analyzes the results. Section 5 summarizes the content of this paper.

2 Related Work

2.1 Lightweight Methods for Categorizing Malicious Code

With the development of deep learning and visualization techniques, image features of malicious codes extracted by deep neural networks have gradually become the main method for identifying malicious codes. It can be seen from the introduction that most current malicious code identification methods involve complex, multi-parameter network models with high requirements on the operating environment. Therefore, it is necessary to investigate lightweight approaches for malicious code classification. This study mainly focuses on malicious code detection using lightweight neural networks based on image features of malicious code. The relevant studies are listed in Table 1.

Although the above lightweight methods for classifying malicious code reduce the number of model parameters to a certain extent, there is still much room for improvement. The lightweight model

reduces the number of parameters at the expense of model accuracy, which requires that the model has a small number of parameters but still has a strong feature extraction capability. Most of the above malicious code classification methods still have a large number of parameters and a single range of extracted features, which makes it difficult to detect obfuscated malicious code effectively.

Table 1: Lightweight methods for categorizing malicious code

Method	The lightweight approach
Reference [23]	Replacing the standard convolution in Densely Connected Convolutional Networks with a lightweight Ghost module
Reference [24]	Improving the lightweight model MobileNetV2
Reference [25]	Concatenating three lightweight neural networks
Reference [26]	Creating a depthwise separable convolutional network
Reference [27]	Using local response normalization to reduce the complexity of the AlexNet model

2.2 Technical Support for the Model in This Paper

Lightweight methods have been developed to reduce the number of model parameters. DenseNet [28] connects each layer of features to all features of the previous layer in the channel dimension by reusing features. By adding shortcuts, the number of channels is quickly inflated by using a smaller number of convolution kernels in the latter layer. This allows DenseNet to achieve the same performance as other deep models, but has fewer parameters. ShuffleNet [29] drastically reduces the parameters while maintaining accuracy by grouping features and using pointwise group convolution and channel shuffling operations. MobileNet [30] greatly reduces the number of parameters of the model by decomposing the standard convolution into depthwise convolution and pointwise convolution through depthwise separable convolution. Considering that the features extracted by smaller convolution kernels have a smaller range and are generally localized features, while the features extracted by larger convolution kernels are generally features with a larger range, Inception [31] performs convolution by using a variety of convolution kernels with different sizes to extract information with different resolutions. The results show that using convolution kernels of different sizes can effectively improve the classification accuracy of the model.

To further reduce the number of parameters of the malicious code identification model and improve the anti-confusion ability of the model, a lightweight malicious code classification model (LCMISNet) is proposed by improving SqueezeNet based on the above techniques in this study. Table 2 shows the specific applications of the above techniques in the model of this paper.

Table 2: Technical support used for LCMISNet

Technology	Appliance
Group convolution	The feature slicing module
Depthwise separable convolution	The multi-size depthwise separable convolution module
Multi-scale feature fusion	The multi-size depthwise separable convolution module
Feature reuse	The feature splicing module

3 Methodology

3.1 LCMISNet Architecture

The LCMISNet model extracts features through eight lightweight MFire modules. The MFire module consists of a feature slicing module (Feature slice) and a multi-size depthwise separable convolution module (MDSC) for lower parameter count and complex feature extraction. In the MFire module, the group convolution is first carried out by the feature slicing module to reduce the number of convolution parameters. Secondly, different scale features are fused by the multi-size depthwise separable convolution module to improve the feature extraction capability of the model. Finally, the eight lightweight MFire modules are connected by the feature splicing module (Feature splice) to create the lightweight model (LCMISNet). In addition, LCMISNet introduces the skip connections in modules with the same number of input and output channels to enhance feature fusion and introduces a Batch Normalization (BN) layer after each convolutional layer to make the data distribution equal. The structure of the LCMISNet model is shown in Fig. 1. Next, we will introduce the feature slicing module, the multi-size depthwise separable convolution module (MDSC), and the feature splicing module in detail.

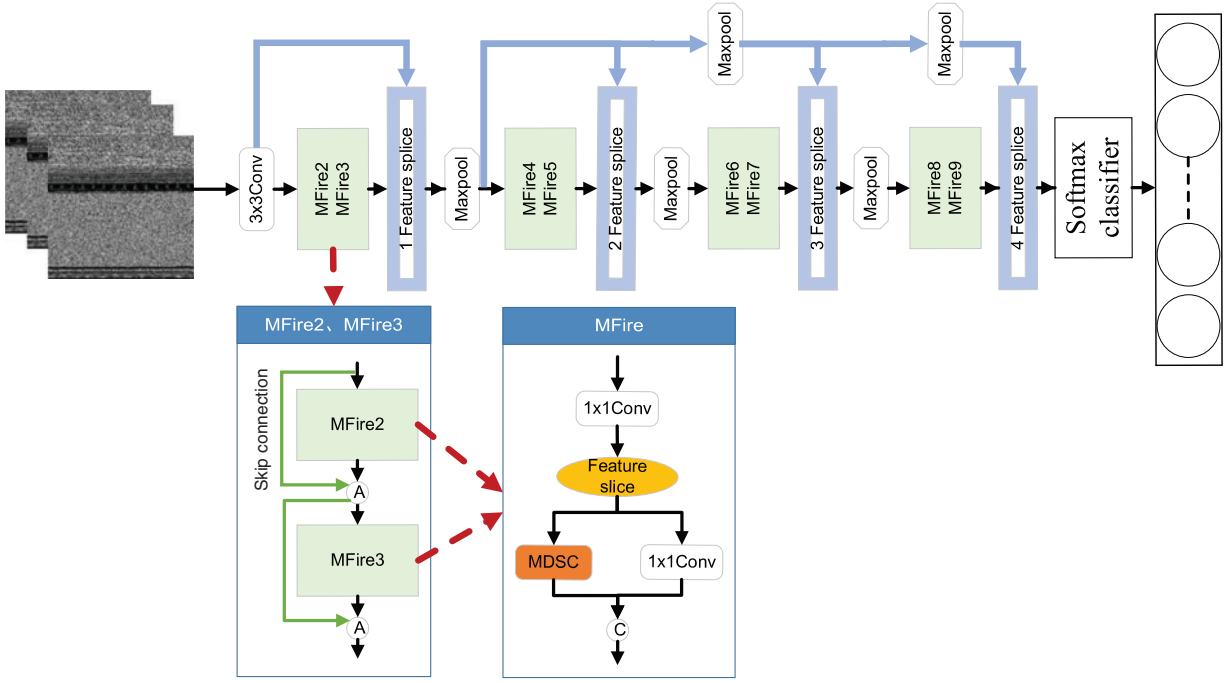


Figure 1: LCMISNet architecture

3.2 Feature Slicing Module

In this paper, the features are extracted using the feature extraction structure of the Fire module of the SqueezeNet model. The Fire module, consisting of two sections, Squeeze and Expand, is primarily responsible for feature extraction. Its structure is shown in Fig. 2, where C represents the feature concatenation.

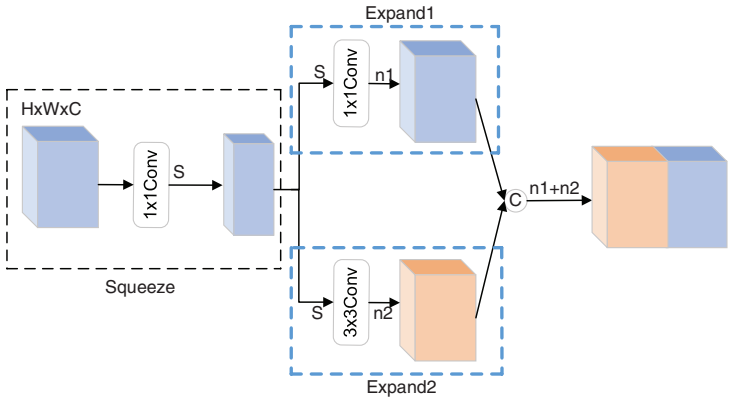


Figure 2: Fire module

As shown in Fig. 2, the number of feature map channels in the Squeeze layer is compressed from C to S by convolving the feature maps of C input channels with S 1×1 convolution kernels. In the Expand layer, the 1×1 convolution kernels and 3×3 convolution kernels are then convolved with the S feature maps from the Squeeze layer. Finally, the feature maps of channels n1 and n2 are merged to obtain the final feature maps.

Based on the realization that the group convolution can reduce the number of parameters, two feature slicing operations are introduced before the Expand layer, as shown in Fig. 3, where C represents the feature concatenation.

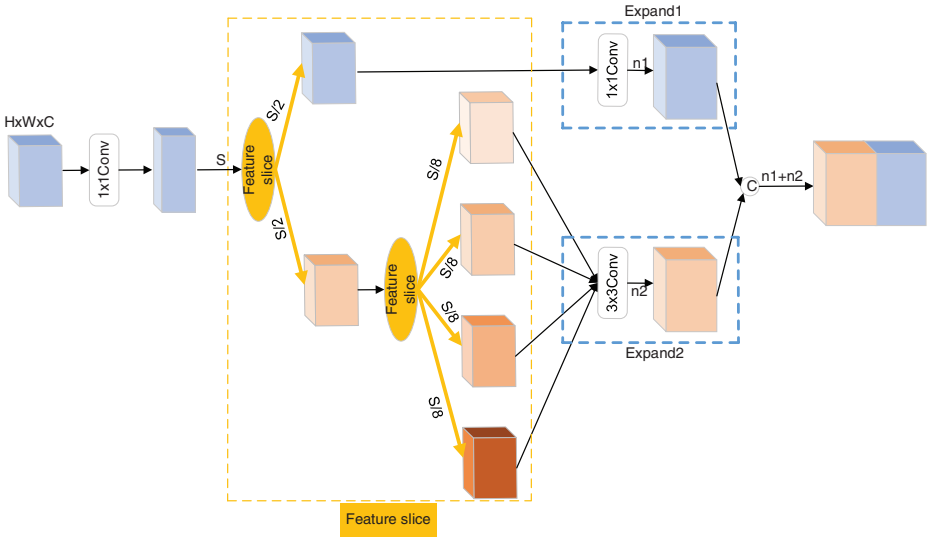


Figure 3: Feature slicing module

During the first feature slicing, the entered S feature maps are divided into two groups, each with the number of S/2 feature maps. One group is sent to the Expand1 layer for convolution, and the other group is divided into four groups by the second feature slicing operation, resulting in four groups with S/8 feature maps each. Following the second feature slicing operation, the four groups of feature maps are fed into the Expand2 layer. Grouping feature maps into N groups, where N is the number of groups, can reduce the number of parameters for a standard convolution to 1/N. The first feature

slicing reduces the number of parameters by a factor of one compared to the original Fire module. The second feature slicing reduces the number of parameters by a factor of eight.

3.3 Multi-Size Depthwise Separable Convolution Module

From the structure diagram of the Fire module in Fig. 2, it can be seen that the Fire module only has two types of convolution kernels, 1×1 and 3×3, which leads to a limited range of extracted features, that are not conducive to training the model. What's more, the 3×3 convolution in the Expand2 layer of the Fire module generates a large number of parameters. MobileNet has shown that replacing the standard convolution with a depthwise separable convolution effectively reduces the number of parameters. Inception has shown that the classification performance of the model can be successfully improved by using convolution kernels of different sizes. Therefore, a multi-size depthwise separable convolution module is developed, replacing the 3×3 convolution of the original Fire module. Fig. 4 shows the structure of the multi-size depthwise separable convolution, where C represents the feature concatenation.

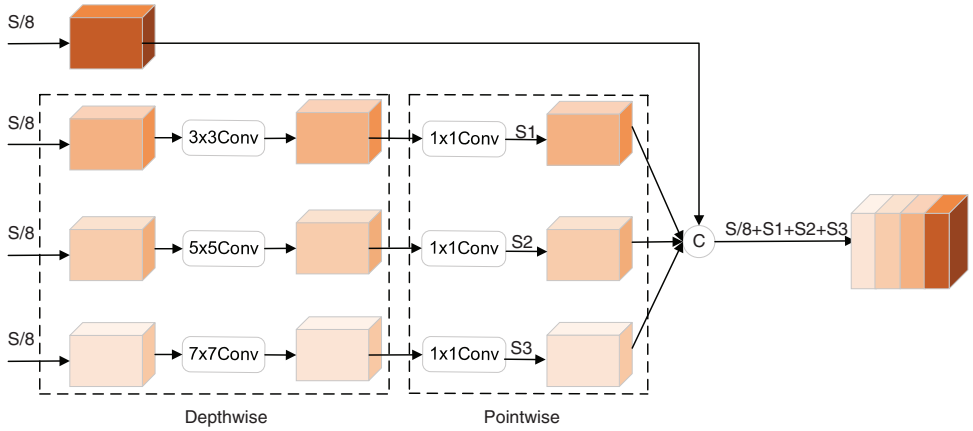


Figure 4: Multi-size depthwise separable convolution module

In the multi-size depthwise separable convolution module, depthwise separable convolution is applied to the three sets of feature maps using convolution kernels of sizes 3×3, 5×5, and 7×7 to extract feature information at different resolutions. Finally, the fourth set of feature maps is concatenated with the three sets of feature maps that have undergone multi-size depthwise separable convolution. One advantage is that the fourth group of feature maps is not convolved, which would prevent the generation of the parameters. Another advantage is that features from different receptive fields can be mixed to improve the ability of the model to extract features.

3.4 MFire Module

The lightweight MFire feature extraction module consists of two parts: the feature slicing module (Feature slice) and the multi-size depthwise separable convolution module (MDSC). Its structure is shown in Fig. 5. The input feature maps are first convolved by the 1×1 convolution, which compresses the number of channels from C to S. After the first feature slicing, the feature maps with S channels are divided into two groups. The feature maps of the S/2 channels are entered into the Expand1 layer for the 1×1 convolution. The remaining half of the feature maps are divided into four groups of the same number by the second feature slicing and fed into the Expand2 layer for a multi-size

depthwise separable convolution. Finally, the output feature maps of the Expand1 and Expand2 layers are merged.

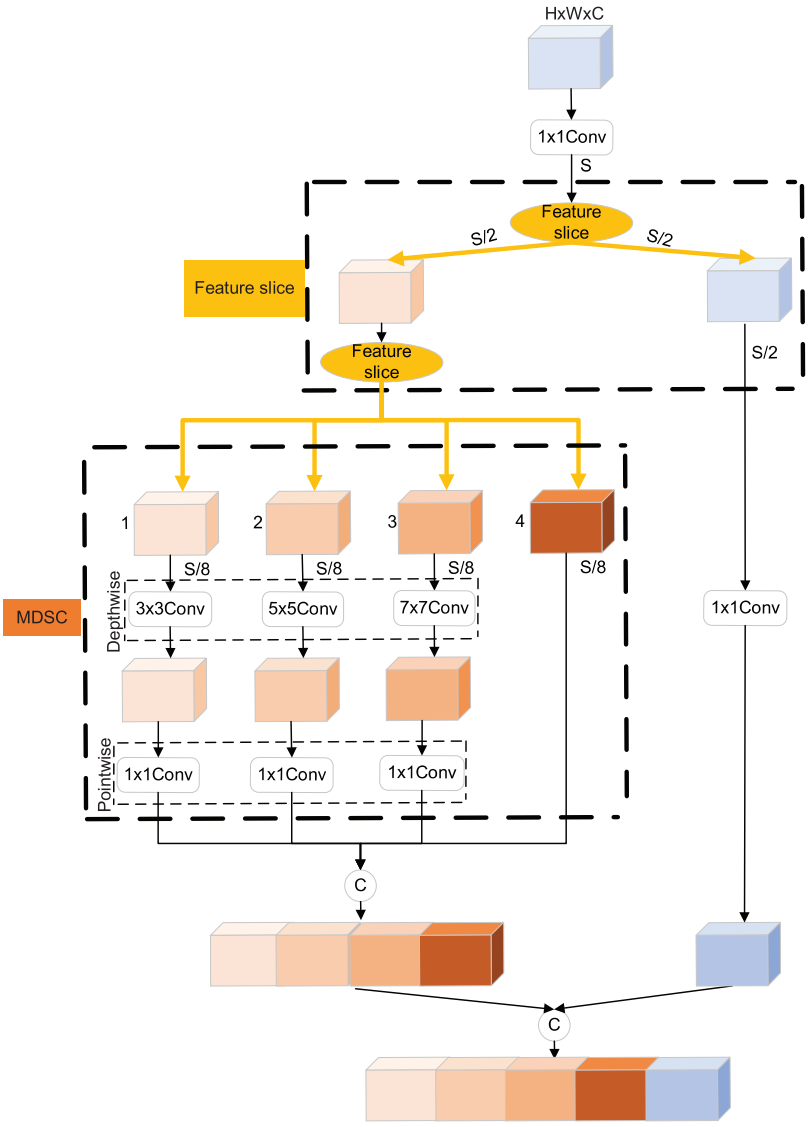


Figure 5: MFire module

3.5 Feature Splicing Module

We developed the feature splicing module (Feature splice) based on the feature reuse technique. It connects the different MFire modules to form the LCMISNet model. Using the outputs of trained submodules or layers as inputs to other modules or layers to build more sophisticated network architectures is called feature reusing. DenseNet has shown that it is possible to reduce the number of parameters of the model while accelerating its convergence by reusing the previous features. Feature reuse allows the model to quickly increase the number of feature channels with fewer convolution kernels. Fig. 6 shows the feature splicing module.

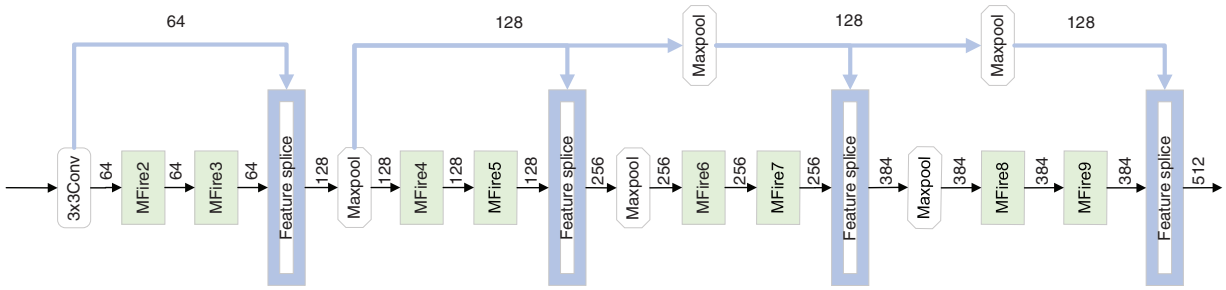


Figure 6: Feature splicing module

LCMISNet has four feature splicing operations. The first feature splicing operation concatenates the output feature maps of MFire3 with the input feature maps of the MFire2 module. The second feature splicing operation merges the output feature map of MFire5 with the input features of MFire4. The first and second feature splicing both link the output feature maps of an MFire module with the input feature maps of the previous MFire module. The third feature splicing is to stitch the output feature maps of MFire7 with the input feature maps of MFire4. The fourth feature splicing is to stitch the output feature maps of MFire9 with the input feature maps of MFire4. Taking the third feature splicing as an example: If the output feature maps of MFire7 are directly spliced with the input feature maps of MFire6, the number of input channels for MFire8 will increase from 384 to 512, which significantly increases the number of parameters for MFire8. Splicing the output feature maps of MFire7 with the feature maps that have fewer channels can reduce the number of input channels and parameters of the deep MFire module. The principles of the fourth and third feature splicing processes are identical.

4 Experiment

4.1 Experimental Environment

The network models in this study were all created using the Tensorflow 2.3 framework in the Python 3.7 environment and trained on the NVIDIA GeForce RTX 2080 Ti for 35 epochs. The batch size is set to 16 based on GPU memory and the size of the dataset. Adamax is used to optimize the model parameters and a callback function adjusts the model's learning rate adaptively. The learning rate is initially set to 0.01. If the loss value of the validation set does not decrease after two consecutive epochs, it is reduced by a factor of 10 until it reaches the lowest value of 0.000001.

4.2 Datasets

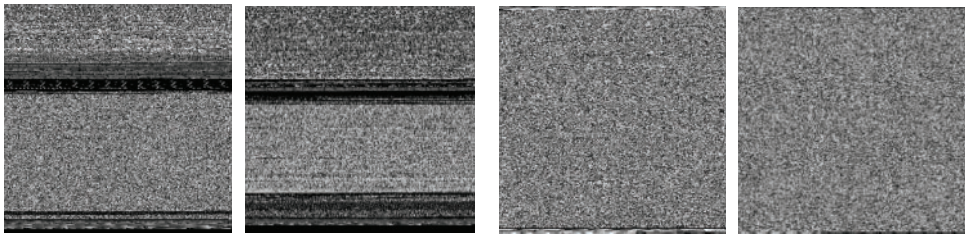
In this paper, experiments are conducted with two datasets: the BIG 2015 dataset and the Malimg dataset.

(1) BIG 2015 dataset. The dataset provided by the Microsoft Malware Classification Challenge is referred to as BIG 2015. BIG 2015 comprises a total of 10,868 samples, which are hexadecimal Portable Executable files labeled with the suffix .byte. These samples come from 9 families of malware, including Gatak, Simda, Vundo, etc.

(2) Malimg dataset. This malicious code dataset is often used in malicious code detection work. In this study, the Malimg grayscale image dataset is downloaded from Kaggle, which contains 9,339 instances of malicious code from 25 different families.

We present a simple malicious code visualization method that converts malicious code files from the BIG 2015 dataset into grayscale images. First, the malicious code executable file is processed by converting every two hexadecimal digits to a decimal digit (in the range of [0–255]) to obtain a one-dimensional integer vector where each decimal digit represents a pixel dot, where 0 represents black and 255 represents white. Second, determine the width and height of the image based on the size of the byte file of the malicious code. Next, fill the one-dimensional numeric vector with 0 and convert it to a two-dimensional array corresponding to the width and height. Finally, the two-dimensional array matrix is converted into a grayscale image of the malicious code.

The grayscale images in Fig. 7 are two different malicious code families from the BIG 2015 dataset. As shown in Fig. 7, the grayscale images of different malware families have different texture structures, while the images visualized by the same malware family are visually similar. The grayscale images of the BIG 2015 dataset were all resized to 320 × 320. All grayscale maps of the Malimg dataset were resized to 256 × 256.



Ramnit family Kelihos_ver3 family

Figure 7: Grayscale images of the BIG 2015 dataset

4.3 Evaluation Indicators

In this paper, four metrics are used to evaluate the classification effect of the models: accuracy (*Acc*), precision (Pr *e*), recall (*Recall*), and F1 score ($F1-score$).

Accuracy is the ratio between the number of correctly classified samples and the total number of samples and is calculated as follows:

$$Acc = \frac{TP + TN}{TP + TN + FP + FN}$$

Precision is the proportion of data correctly predicted as positive in relation to the data predicted to be positive and is calculated as follows:

$$\Pr e = \frac{TP}{TP + FP}$$

Recall is the proportion of correctly predicted positive data relative to the actual positive data and the formula is:

$$\text{Re}call = \frac{TP}{TP + FN}$$

The F1 score is the harmonic mean of precision and recall and its formula is:

$$F1 - score = 2 \times \frac{\Pr e \times Recall}{\Pr e + Recall}$$

where *True Negative* represents the number of negative class samples predicted to be negative, *True Positive* represents the number of positive class samples predicted as positive class, *False Positive* represents the number of negative class samples predicted to be positive, and *False Negative* represents the number of positive class samples predicted as negative class.

In addition to the four commonly used neural network evaluation metrics listed above, two new evaluation metrics, Parameter and FLOPs, are introduced in this study.

The Parameter measures the size of the model's parameter count. A smaller Parameter means that the model requires less memory. Its formula is:

$$Parameter = (C_{in} \times (K \times K) + 1) \times C_{out}$$

FLOPs measures the size of the total calculations of the model. The fewer FLOPs, the lower the computational complexity of the model. Its formula is:

$$FLOPs = 2 \times H \times W \times (C_{in} \times (K \times K) + 1) \times C_{out}$$

where C_{in} represents the number of input channels, K represents the size of the convolution kernel, C_{out} represents the number of output channels, H represents the height of the feature map, and W represents the width of the feature map.

4.4 Experimental Results

In this section, we evaluate the LCMISNet model. The experiment consists of three parts: (1) Comparison experiments between LCMISNet and the original model. (2) Comparison experiments with other network models commonly used in image detection. (3) Comparison with existing malicious code classification methods.

4.4.1 Comparison Experiments between LCMISNet and the Original Model

To verify the effectiveness of the modules proposed in this paper in improving model performance and reducing the number of model parameters, we conducted experiments comparing the model of this paper with the original SqueezeNet model without improvements based on the Malimg dataset and the BIG 2015 dataset. Table 3 shows the comparison of accuracy (Acc), precision (Pre), recall (Rec), F1 score (F1), number of parameters (Parameter), and FLOPs of LCMISNet with the original SqueezeNet model based on the two datasets. Fig. 8 shows the variation curve of the accuracy of LCMISNet compared to the original model based on the BIG 2015 dataset. Fig. 9 shows the variation curve of the accuracy of LCMISNet compared to the original model based on the Malimg dataset.

First, as shown in Table 3, the FLOPs and Parameter of the LCMISNet model based on the BIG 2015 dataset are reduced by 33.99 billion and 594060, respectively, compared to the original model without lightweight improvement. Based on the Malimg dataset, the FLOPs and Parameter of the LCMISNet model are reduced by 21.47 billion and 593996, respectively, compared to the original model without lightweight improvement. Since the group convolution and depthwise separable convolution are used in the lightweight feature extraction module MFire, compared to the standard convolution of the original model, the parameters and computational cost of the model can be effectively reduced. In addition, compared to the simple sequential connection of the original model, LCMISNet also uses a feature splicing module to connect MFire, which makes LCMISNet much lighter.

Table 3: Comparison of LCMISNet and the original model based on different datasets

Dataset	Method	Acc (%)	Pre (%)	Rec (%)	F1 (%)	Parameter	FLOPs (billion)
BIG 2015	Original model	94.78	89.43	92.11	90.48	730633	42.32
	LCMISNet	**98.90**	**98.18**	**98.83**	**98.45**	**136573**	**8.33**
Malimg	Original model	90.80	64.46	72.29	67.44	738841	26.87
	LCMISNet	**99.58**	**98.92**	**98.86**	**98.85**	**144845**	**5.40**

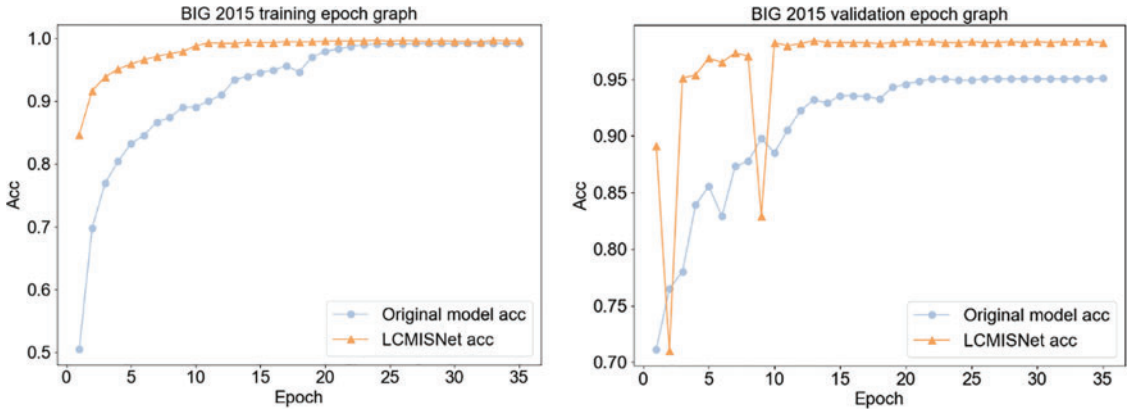


Figure 8: Accuracy curves on the BIG 2015 training set (left) and the BIG 2015 validation set (right)

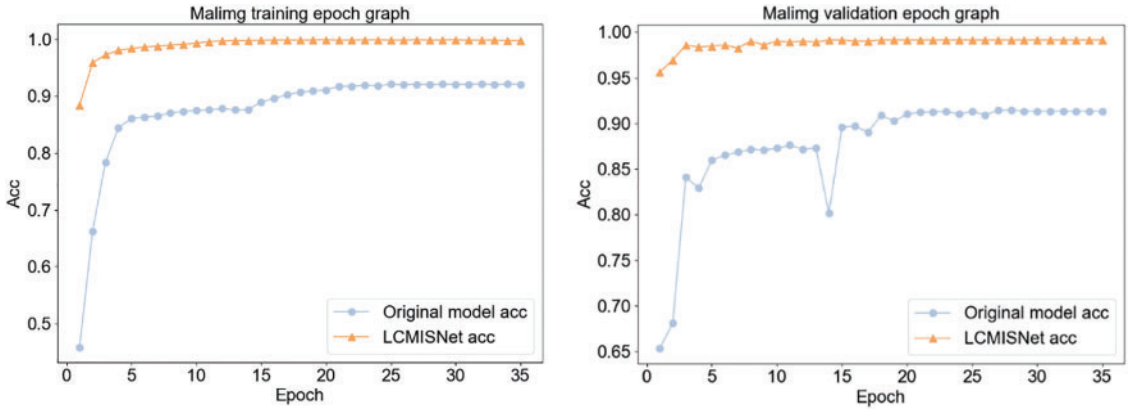


Figure 9: Accuracy curves on the Malimg training set (left) and the Malimg validation set (right)

Second, as shown in Table 3, the LCMISNet model has a significant improvement in accuracy, precision, recall, and F1 score on two different datasets compared to the original model. LCMISNet introduces the skip connection, and thus strengthens the feature fusion for different feature extraction modules. In addition, LCMISNet uses multiple convolution kernels of different sizes to extract different ranges of features. Compared to the original model with a single convolution kernel size, LCMISNet can extract more complex features.

Finally, Figs. 8 and 9 show that LCMISNet has improved the accuracy and convergence speed on two different datasets compared to the original model. This proves that LCMISNet is easier to train and has better recognition results. We use Batch Normalization layers in LCMISNet, which makes the model more stable. Moreover, the skip connection and feature splicing module have a shortcut structure, enabling the model to converge quickly.

To summarize, the LCMISNet model has better recognition, faster convergence, and a smaller number of parameters and computations than the original model, which proves the effectiveness of the module proposed in this paper.

4.4.2 Comparative Experiments with Other Network Models Commonly Used in Image Recognition

We compare LCMISNet with other lightweight and non-lightweight neural network models commonly used in image recognition. Table 4 shows the experimental results of accuracy (Acc), precision (Pre), recall (Rec), F1 score (F1), parameters (Parameter), and FLOPs on the BIG 2015 test set and the Malimg test set. Fig. 10 shows the Parameter and FLOPs for each model based on two different datasets.

Table 4: Comparison of different network models on two datasets

Dataset	Model	Acc (%)	Pre (%)	Rec (%)	F1 (%)	Parameter (M)	FLOPs (billion)
BIG 2015	ResNet50	97.35	96.59	97.36	96.94	24.114569	253.11
	Xception	94.24	89.18	92.21	90.16	21.388337	299.09
	DenseNet121	98.26	96.27	98.16	97.09	7.302217	186.14
	MobileNet	97.99	97.35	97.59	97.46	3.493577	37.41
	MobileNetV2	98.17	97.84	98.32	98.05	2.588233	20.01
	LCMISNet	**98.90**	**98.18**	**98.83**	**98.45**	**0.136573**	**8.33**
Malimg	ResNet50	97.38	91.94	92.36	92.11	24.118681	162.00
	Xception	96.24	89.26	89.97	89.51	21.392449	191.12
	DenseNet121	99.05	97.64	97.43	97.34	7.306329	119.13
	MobileNet	99.16	97.86	97.76	97.78	3.497689	23.95
	MobileNetV2	98.74	97.03	96.84	96.81	2.592345	12.81
	LCMISNet	**99.58**	**98.92**	**98.86**	**98.85**	**0.144845**	**5.40**

As shown in Fig. 10, LCMISNet has a smaller number of parameters and FLOPs compared to the other models. In addition, as shown in Table 4, LCMISNet is better at detecting malicious code compared to other models. LCMISNet introduces a feature slicing module and a multi-size depthwise separable convolution module in the MFire feature extraction module, which makes the feature extraction structure of the model lighter and extracts more complex features. In addition, LCMISNet introduces a feature splicing module when connecting the lightweight MFire feature extraction modules, which makes LCMISNet lighter compared to other model structures. In ResNet, the skip connection technique is used, and in Xception, convolution kernels of different sizes are used, which enhances feature fusion. However, both models have a large number of parameters and FLOPs. DenseNet121, MobileNet and MobileNetV2 reduce the number of parameters to a certain extent, but still need to improve feature extraction.

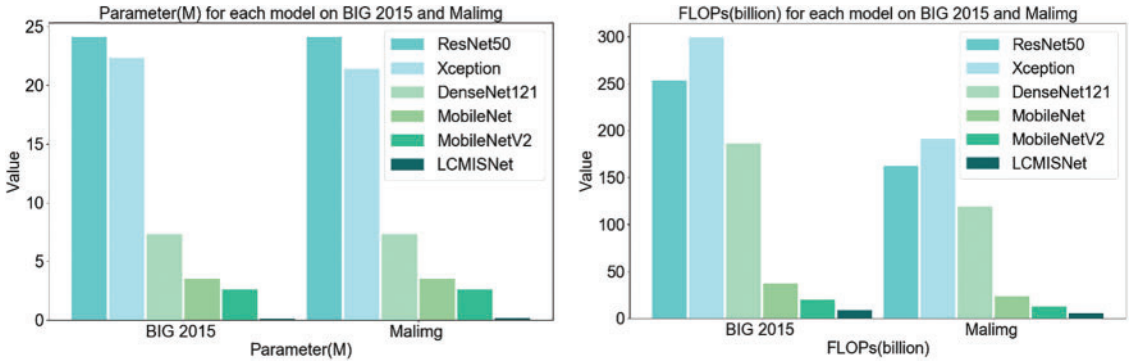


Figure 10: Parameter (left) and FLOPs (right) for each model based on two different datasets

4.4.3 Comparison with Existing Methods for Classifying Malicious Code

In this study, the LCMISNet model is compared with existing malicious code classification methods using two malicious code datasets. Table 5 shows the comparison results based on the Malimg and BIG 2015 datasets.

Table 5: Comparison with other methods on two datasets

Dataset	Methods	Feature	Acc (%)	Parameter (M)
Malimg	CNN [20]	Grayscale image	97.60	–
	AlexNet [27]	RGB image	97.80	–
	Ensemble learning [6]	via first and second-order texture features	98.58	–
	IMCFN [22]	RGB image	98.82	134.36
	MobileNet ShuffleNet Xception [25]	Grayscale image	99.31	17.04
	LCMISNet	**Grayscale image**	**99.58**	**0.14**
BIG 2015	SVM [5]	HOG + Dense SIFT	94.50	–
	CNN [21]	Grayscale image	96.20	–
	RNN + CNN [18]	Grayscale image	98.80	–
	LCMISNet	**Grayscale image**	**98.90**	**0.14**

It can be seen from Table 5 that LCMISNet shows good classification performance on both the Malimg dataset and the BIG 2015 dataset, outperforming existing malicious code classification methods in terms of classification accuracy and number of parameters. References [5,6] used machine learning techniques to detect malicious code, resulting in higher feature extraction costs. References [20,27,21] used neural networks to extract malicious code features automatically, but all have low detection accuracy and use models that are not lightweight. Reference [22] has improved the accuracy of malicious code detection to a certain extent, but the model needs to be lighter. References [25,18] further improved the accuracy of the model, while both methods require training multiple neural network models, which makes model training more complex.

The LCMISNet malicious code detection model proposed in this paper solves all of the above problems: (1) LCMISNet can automatically extract malicious code features, which reduces the cost of feature extraction. (2) LCMISNet has a smaller number of parameters and calculations, which makes deployment easier and reduces memory resource consumption. (3) LCMISNet has a stronger feature extraction function, which is more conducive to identifying malicious code. (4) LCMISNet model training is simpler and more practical for daily use.

5 Conclusion

This paper introduced LCMISNet, a lightweight method for classifying malicious code based on the improvement of SqueezeNet. In this paper, a feature slicing module and a multi-size depthwise separable convolution module are proposed to improve the Fire module at a lightweight level, and a lightweight feature extraction module, MFire, is created. In addition, a feature splicing module based on feature reuse is proposed in this paper to connect the lightweight MFire module. After experimenting with two different malicious code datasets, it has been shown that LCMISNet has a smaller number of parameters and computations, better detection of malicious code, faster convergence, and easier training.

In the following work: (1) There is an imbalance in the number of samples of different classes of malicious code, such as the BIG 2015 dataset and the Malimg dataset, which causes the model to focus on learning the features of the classes of malicious code with a higher number of samples while ignoring the classes with fewer samples. Therefore, the next step is to enlarge the dataset, oversample the categories with smaller sample sizes, and equalize the dataset to solve the class imbalance problem and reduce the impact of the class imbalance problem on the performance of the model. (2) The variants of the same malicious code family have similar characteristics, so it is difficult for the model to distinguish between the malicious code variants of the same family. Next, we consider introducing an attention mechanism into the model to make the model focus more on the features with better learning effects and further improve the performance of the model to increase the ability to recognize similar malicious code classes.

Acknowledgement: First and foremost, I want to express my sincere gratitude to my supervisor for all of her support and encouragement in helping me finish this thesis. Your profound ideas, strict approach, and careful direction have made a significant impact on my life. You patiently supported me and advised me when I ran into problems, which helped me go through a lot of problems. I also want to thank all the lab participants for thinking with me, talking through issues, and offering suggestions. I have been able to learn and develop in this setting, which has a strong academic atmosphere, and I have felt warm thanks to your friendship and support. I also want to express my gratitude to my family and friends, whose unwavering support and tolerance have made it possible for me to concentrate on my studies. I cannot thank you enough for your support and encouragement; it keeps me going. Last but not least, I would like to sincerely thank everyone who has helped and supported me. I could not have finished this thesis without the help and encouragement of you all. Your commitment and assistance are greatly appreciated and will be kept in mind. Thank you again to everyone who helped with my research!

Funding Statement: The source of funding to support this work is the Heilongjiang Provincial Education Science Planning Key Issues, specific grant numbers is GJB1421251, initials of authors is L, and the URLs to sponsors' websites is http://www.hljjyxh.org.cn.

Author Contributions: Study conception and design: Y. Kong; data collection: Y. Kong; analysis and interpretation of results: Y. Kong; draft manuscript preparation: Y. Kong; dissertation guidance: L. Li, Q. Zhang. All authors reviewed the results and approved the final version of the manuscript.

Availability of Data and Materials: All data used in the experiment can be downloaded from the Kaggle website. BIG 2015 dataset: https://www.kaggle.com/competitions/malware-classification/data. Malimg dataset: https://www.kaggle.com/datasets/manmandes/malimg/.

Conflicts of Interest: The authors declare that they have no conflicts of interest to report regarding the present study.

References

[1] N. Cheng, "Research on computer network security prevention strategy," *Information Recording Materials*, vol. 24, no. 3, pp. 86–88, 2023.
[2] S. Li, "Research on computer network virus and its defense technology," *Digital Communication World*, vol. 19, no. 4, pp. 41–43, 2023.
[3] K. Chen, "Research on malicious code detection technology based on deep learning," M.S. dissertation, University of Beijing Jiaotong: China, 2020.
[4] Q. Guo, "Summary of research on malicious code detection technology," *Computer Knowledge and Technology*, vol. 19, no. 13, pp. 79–81+93, 2023.
[5] R. Tan, L. Zuo and E. Liu, "Malicious code detection based on image feature fusion," *Netinfo Security*, vol. 21, no. 10, pp. 90–95, 2021.
[6] V. Verma, S. K. Muttoo and V. B. Singh, "Multiclass malware classification via first and second-order texture statistics," *Computers & Security*, vol. 97, pp. 101895, 2020.
[7] S. Chattopadhyay, "A study on various common denoising methods on chest x-ray images," *Artificial Intelligence Evolution*, vol. 3, no. 2, pp. 87–106, 2022.
[8] G. Roy, J. Fiaidhi and S. Mohammed, "Multi-timeframe algorithmic trading bots using thick data heuristics with deep reinforcement learning," *Artificial Intelligence Evolution*, vol. 3, no. 2, pp. 107–159, 2022.
[9] S. Gao and S. Li, "Bloody mahjong playing strategy based on the integration of deep learning and XGBoost," *CAAI Transactions on Intelligence Technology*, vol. 7, no. 1, pp. 95–106, 2022.
[10] K. Yadav, M. Yadav and S. Saini, "Stock values predictions using deep learning based hybrid models," *CAAI Transactions on Intelligence Technology*, vol. 7, no. 1, pp. 107–116, 2022.
[11] M. Zheng, K. Zhi, J. Zeng, C. Tian and L. You, "A hybrid CNN for image denoising," *Journal of Artificial Intelligence and Technology*, vol. 2, no. 3, pp. 93–99, 2022.
[12] J. Meng, Y. Li, H. Liang and Y. Ma, "Single-image dehazing based on two-stream convolutional neural network," *Journal of Artificial Intelligence and Technology*, vol. 2, no. 3, pp. 100–110, 2022.
[13] K. He, X. Zhang, S. Ren and J. Sun, "Deep residual learning for image recognition," in *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition*, Las Vegas, NV, USA, pp. 770–778, 2016.
[14] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Int. Conf. on Machine Learning*, Lille, France, pp. 448–456, 2015.
[15] J. Chen, D. Zhan and S. Xia, "Interpretability of deep learning-based malicious code detection," *Journal of Nanjing University of Science and Technology*, vol. 47, no. 3, pp. 343–351, 2023.
[16] Z. Mei and H. Fang, "Malicious code API classification based on convolutional neural network," *Journal of Chifeng University (Natural Science Edition)*, vol. 39, no. 2, pp. 39–43, 2023.
[17] Y. Qiao, Q. Jiang and L. Gu, "Malware classification method based on word vector of assembly instruction and CNN," *Netinfo Security*, vol. 220, no. 4, pp. 20–28, 2019.
[18] X. Chen, S. Wei and Z. Qin, "Malware family classification based on deep learning visualization," *Computer Engineering and Applications*, vol. 57, no. 22, pp. 131–138, 2021.

[19] B. Wang, H. Cai and Y. Su, "Classification of malicious code variants based on VGGNet," *Journal of Computer Applications*, vol. 40, no. 1, pp. 162–167, 2020.
[20] Z. Cui, L. Du, P. Wang, X. Cai and W. Zhang, "Malicious code detection based on CNNs and multi-objective algorithm," *Journal of Parallel and Distributed Computing*, vol. 129, pp. 50–58, 2019.
[21] Z. Fan, J. Li and Y. Liu, "Classification of malware based on gray texture fingerprint," *Science Technology and Engineering*, vol. 20, no. 29, pp. 12014–12020, 2020.
[22] D. Vasan, M. Alazab, S. Wassan, H. Naeem, B. Safaei *et al.,* "IMCFN: Image-based malware classification using finetuned convolutional neural network architecture," *Computer Networks*, vol. 171, pp. 107138, 2020.
[23] Y. Li and J. Li, "A malicious code detection method based on Ghost-DenseNet-SE," *Journal of Air Force Engineering University*, vol. 22, no. 5, pp. 49–55, 2021.
[24] B. Xuan, J. Li, Y. Song and Z. Ma, "Malicious code classification method based on improved MobileNetV2," *Journal of Computer Applications*, vol. 43, no. 7, pp. 2217–2225, 2023.
[25] R. Jiang and R. Qin, "Multi-neural network malicious code detection model based on depthwise separable convolution," *Journal of Computer Applications*, vol. 43, no. 5, pp. 1527–1533, 2023.
[26] R. Wang, J. Gao and X. Tong, "Research on malicious code family classification combining attention mechanism," *Journal of Frontiers of Computer Science and Technology*, vol. 15, no. 5, pp. 881–892, 2021.
[27] K. Jiang, W. Bai and L. Zhang, "Malicious code detection based on multi-channel image deep learning," *Journal of Computer Applications*, vol. 41, no. 4, pp. 1142–1147, 2021.
[28] G. Huang, Z. Liu and L. van der Maaten, "Densely connected convolutional networks," in *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition*, Honolulu, HI, USA, pp. 4700–4708, 2017.
[29] X. Zhang, X. Zhou, M. Lin and J. Sun, "Shufflenet: An extremely efficient convolutional neural network for mobile devices," in *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition*, Salt Lake City, UT, USA, pp. 6848–6856, 2018.
[30] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang *et al.,* "MobileNets: Efficient convolutional neural networks for mobile vision applications," arXiv preprint arXiv:1704.04861, 2017.
[31] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed *et al.,* "Going deeper with convolutions," in *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition*, Boston, MA, USA, pp. 1–9, 2015.