International Journal of Computational Intelligence Systems (2024) 17:30
https://doi.org/10.1007/s44196-023-00400-9

RESEARCH ARTICLE

# A Lightweight Model for Malicious Code Classification Based on Structural Reparameterisation and Large Convolutional Kernels

**Sicong Li[1] · Jian Wang[1] · Yafei Song[1] · Shuo Wang[2] · Yanan Wang[1]**

Received: 10 December 2023 / Accepted: 27 December 2023


**Abstract**
With the advancement of adversarial techniques for malicious code, malevolent attackers have propagated numerous malicious code variants through shell coding and code obfuscation. Addressing the current issues of insufficient accuracy and efficiency in malicious code classification methods based on deep learning, this paper introduces a detection strategy for malicious code, uniting Convolutional Neural Networks (CNNs) and Transformers. This approach utilizes deep neural architecture, incorporating a novel fusion module to reparametrize the structure, which mitigates memory access costs by eliminating residual connections within the network. Simultaneously, overparametrization during linear training time and significant kernel convolution techniques are employed to enhance network precision. In the data preprocessing stage, a pixel-based image size normalization algorithm and data augmentation techniques are utilized to remedy the loss of texture information in the malicious code image scaling process and class imbalance in the dataset, thereby enhancing essential feature expression and alleviating model overfitting. Empirical evidence substantiates this method has improved accuracy and the most recent malicious code detection technologies.

**Keywords** Malware variant detection · Convolutional neural network · Structurral reparameterisation · Large kernel convolution · Image size normalisation

## 1 Introduction

Malicious code is software designed to carry out malicious activities or launch attacks. According to the "Internet Security Situation Analysis Report for the First Half of 2021" published by the China National Internet Emergency Center (CNCERT/CC) [1], approximately 23.07 million malicious software samples were captured in the first half of 2021, with a daily distribution frequency amounting to 5.82 million instances. This involved about 208,000 malicious code families, infecting approximately 4.46 million computer terminals. Concurrently, the "China Internet Security Report 2022" released by Rising [2] indicated that 73.55 million malicious codes were caught by the Rising "Cloud Security" system in 2022, with virus infections occurring 124 million times, including devices from individuals, companies, and government agencies. The rapid propagation and variance of the malicious code have severely impacted users' everyday lives, jeopardizing our national cybersecurity and hindering the development of a communal digital future. Hence, accurately and efficiently detecting and categorizing malicious software and its varieties has become a focal point in this field.

Traditional malicious software identification methodologies hinge on the match of a signature-based model. This necessitates researchers manually extracting the signature of the malicious software using expert knowledge and then comparing these signatures with known ones stored in a database. However, numerous variations of malicious software have been generated with the evolution of obfuscation and wrapping techniques. This circumstance renders traditional detection methods less efficient and ineffective at detecting and recognizing variations of malicious software. To tackle the challenges faced by static analysis-based malicious code detection methodologies, visualization-based detection and classification techniques for malicious code have emerged [3, 4]. These methods map malicious code as

✉ Yafei Song
yafei_song@163.com

[1] Air and Missile Defense College, Air Force Engineering University, Xi'an 710051, People's Republic of China

[2] Unit of 95285, Chinese People's Liberation Army (PLA), Guilin 541000, People's Republic of China

Published online: 14 February 2024

images based on the distinct texture features within the same malicious code family and differing texture characteristics between various code families. They extract texture features of the malicious code image and use these elements to detect and categorize malicious code samples. Visualization-based malicious code analysis methodologies neither rely on expert knowledge nor static decompilation procedures and have been proven capable of detecting malicious code variations effectively.

Since the method was proposed, a large number of experts and scholars have researched on it and some progress has been made. The main focus is on using machine learning and deep learning techniques to improve detection accuracy and efficiency. Nataraj et al. [5] fused image and signal features to describe the malicious code and used KNN (K-Nearest Neighbor) as a classifier to identify the malicious code. Kancherla et al. [6] to enhance the diversity of the features incorporated Gabor features, Wavelet features and intensity features are fused as total features and SVM (Support Vector Machines) classifier is trained to achieve malicious code classification. Yashu Liu et al. [7] constructed anti-confusion features by fusing GIST features and LBP (Local Binary Pattern) features of malicious images to solve the problem of degradation of classification performance of the model in similar malicious images. The above studies applied machine learning to visualisation-based malicious code detection methods, and although there is some progress, they usually require manual extraction of features from the data, and the detection efficiency is low.

With the advent of deep learning, Naeem et al. [8] proposed a malware variant classification method. They first converted malware files into grey-scale images and then used global malicious and local collective mechanisms to identify malware variants. Their paper describes the proposed method in detail. The only shortcoming may be that comprehensive empirical psychological results are not provided to demonstrate the performance of variant classification. Mathew et al. [9] devised a method to classify malware variants. Their method converts malware files into colour images and introduces a local pyramid pool to handle various input image sizes. Although in practice, their paper also did not address variant classification performance, especially the results of experiments on variant classification for various levels of variability.

In recent years, with the advancement of computer vision and deep learning technologies, deep learning algorithms, represented by Convolutional Neural Networks (CNNs), have made breakthrough progress in image classification and feature recognition, becoming the mainstream architecture for visual models. They have gradually been applied to malicious code detection and classification fields. The classic Convolutional Neural Network (ConvNet) [10], composed of Conv, ReLU, and pooling, has achieved significant success in image recognition. The advent of Inception [11], ResNet [12], and DenseNet [13] have shifted a vast amount of research interest towards intricately designed architectures, escalating the model complexity. Recent architectures are based on automatic [14] or manual [15] architectural searches or searched compound scaling strategies [16]. Although many complex CNNs have improved in accuracy, their disadvantages are significant: (1) Complex multi-branch designs, for instance, residual additions in ResNet and branch connections in Inception, make the model difficult to implement and customize, leading to slow inference speed and low memory utilization. (2) Certain components, such as depthwise convolution in Xception and MobileNets [17, 18] and channel shuffling in ShuffleNets [19], increase memory access overhead and lack support for various devices.

With the success of data-driven models in image classification, detection and segmentation tasks [20, 21], a range of hybrid visual transformer models have emerged [22–25]. Different from convolution layers, the self-attention mechanism of Vision Transformers offers a global context by modeling long-distance dependencies. However, achieving this global view often incurs high computational costs [26] and increases memory access overhead [27], thus resulting in significant latency overhead. To alleviate this challenge, some studies [26, 28, 29] focus on mitigating the computational burden associated with self-attention layers. Design approaches include replacing Patchify Stem with convolution layers [30], introducing early convolution stages [31], or employing window attention [32] to implement implicit hybrid models. The latest research has established explicit hybrid structures that better facilitate information exchange among tokens (or patches) [33–35]. In most hybrid structures, token mixing primarily depends on self-attention.

Inspired by recent work [27] utilizing reparametrized skip-connections to reduce memory access costs, this study introduces an architectural component called the Rep Mixer into the model. This operator is a fully reparametrized token mixer, combining the advantages of convolution architectures and Transformers, supplanting the self-attention layer to achieve computational latency reduction. The jump connections are eliminated through structural reparametrization. In addition, Rep Mixer also employs deep convolution to carry out operations similar to the information space mixing of ConvMixer. To enhance performance, various studies [17, 18, 36] have incorporated deep convolution or group convolution, subsequently resorting to $1 \times 1$ point convolution to factor the $k \times k$ convolution. Despite this technique effectively bolstering the model's overall operational efficiency, the parameter decrease may result in a drop in model capacity. To further ameliorate latency, the number of floating-point operations (FLOPs), and parameter count, more recent

research [27, 37, 38] employs linear training time over-parametrization to increase such a model's capacity. This paper replaces all the dense $k \times k$ convolutions with their decomposition version, that is, depthwise convolution followed by pointwise convolution, and uses the linear training time over-parametrization proposed in [27] to enhance the capacity of these layers. These additional branches are only introduced during training and are reparametrized during inference.

Additionally, in the design of the MDC-RepNet (Structural Reparameterization and Multi-scale Deep Convolutional Classifier Network, MDC-RepNet), we have adopted large kernel convolution to replace the early-stage Self-Attention method. Although the Vision Transformer based on self-attention demonstrates high accuracy, it is inefficient in handling latency [26]. As a result, we introduce large kernel convolutions in the Feed Forward Network (FFN) layer and the patch embedding layer. Compared with other Vision Transformer architectures, our MDC-RepNet has a more negligible impact on overall latency while improving performance.

Aiming at the problem of insufficient extraction accuracy and low efficiency of current deep learning-based malicious code classification methods, this paper proposes a malicious code detection method combining CNN and Transformer, compared with other deep learning-based malicious code detection methods, the MDC-RepNet proposed in this paper has the following advantages.

1. For the problem of data image texture information loss, In the data preprocessing stage, pixel-filling based image size normalization algorithm and data enhancement techniques are used to improve the image texture information loss and dataset category imbalance problem during malicious code image size deflation, respectively, and to enhance the expression of key features and alleviate the overfitting phenomenon of the model.
2. For the problem of slow detection speed, A deep neural network is adopted as the framework, and the fusion module is introduced to make its structure reparameterised, which effectively reduces the memory access cost by eliminating the jump connections in the network.
3. For the problem of poor classification accuracy, linear training time over-parameterisation and large kernel convolution technique are used to improve the network accuracy.
4. Through the final experiments, it is proved that the method in this paper has a stable improvement in both accuracy and operation efficiency, which is better than the latest malicious code detection technology.

In conclusion, MDC-RepNet is based on the Vision Transformers architecture, leveraging structural reparametrization to achieve lower memory access costs and higher efficiency, realizing superior accuracy-latency balance.

## 2 Malicious Code Classification Method Based on MDC-RepNet

Our proposed malicious code detection scheme consists of two core components: data preprocessing and the construction of the MDC-RepNet. During the data preprocessing stage, it includes visualizing malicious code, normalizing image size, and data augmentation techniques. The construction of the MDC-RepNet stage introduces the Rep Mixer token-mixing operator. It employs a structural reparametrization strategy to eliminate skip connections within the network to reduce memory access costs. Simultaneously, it utilizes over-parametrization during training time and extensive kernel convolution techniques to enhance the model's accuracy. The complete architecture is shown in Fig. 1.

### 2.1 Data Preprocessing

#### 2.1.1 Malware Visualization

Malicious Code Visualisation is the conversion of malicious code executables into greyscale images. Malware visualisation does not require any feature engineering or domain expert knowledge and is a simple and easy-to-use method for malicious code analysis. The visualisation-based malicious code analysis method can present the static structural information of the malicious code through images, which can quickly process a huge number of samples. And its ability to capture small changes between malicious code variants while preserving the global structure will help in analysing malicious code.

The process of malicious code visualization entails transforming malicious code binary files into grayscale images, as illustrated in Fig. 2. First, given a malicious code, a binary file is read in groups of 8-bit unsigned integers. Each group of binary numbers is then converted into a decimal integer. Subsequently, the row width is determined according to the PE file size and transformed into a two-dimensional array, and the correspondence between the row width and the file size is shown in Table 1. Finally, each element in the two-dimensional array is considered as the grayscale value of the image, mapping the two-dimensional array onto a grayscale image—the partial conversion of malicious family samples as depicted in Fig. 3.

The correspondence of the different parts of the malware code binary file mapped to a grey scale image is shown in Fig. 4. In Fig. 4, the text contains not only malware executable code but also black blocks filled with zeros. The data section contains information about initialised and

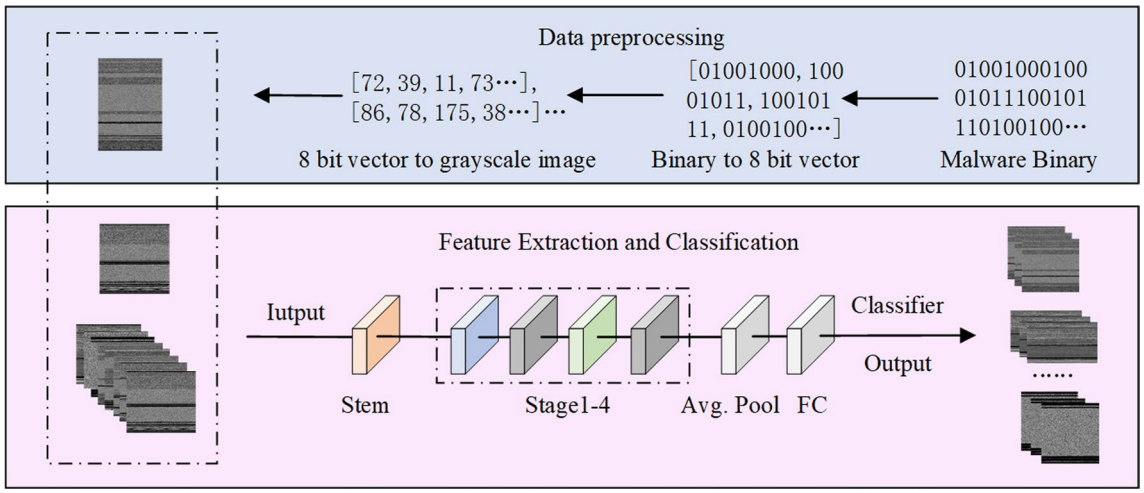



**Fig. 1** Schematic diagram of the model structure

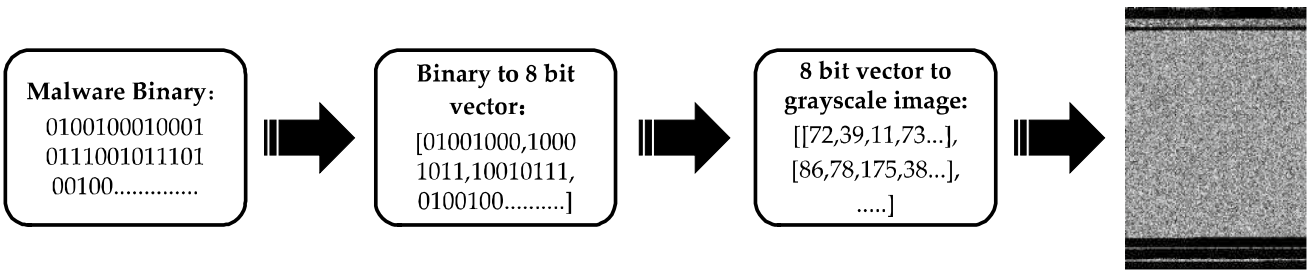



**Fig. 2** Illustration of malware images visualization

**Table 1** Image width for various file sizes

| | File size range | Image width | | File size range | Image width |
|---|---|---|---|---|---|
| 1 | < 10 KB | 32 | 5 | 100–200 KB | 384 |
| 2 | 10–30 KB | 64 | 6 | 200–500 KB | 512 |
| 3 | 30–60 KB | 128 | 7 | 500–1000 KB | 768 |
| 4 | 60–100 KB | 256 | 8 | > 1000 KB | 1024 |

uninitialised variables. The final rsrc section contains all kinds of compiled and generated malicious code resources, including the program's icons and so on.

### 2.1.2 Malware Image Size Normalization

In convolutional neural networks, the size of the weight matrix in the fully connected layer is fixed, meaning the feature size input into the fully connected layer must remain consistent. If the input image sizes vary, the feature sizes following convolution and pooling operations will also differ, leading to disparate feature sizes input into the fully connected layer and rendering the fully connected layer ineffective. Thus, images input into the convolutional neural network must be the same size. However, the sizes of visuals created after visualizing malicious images are all different. Consequently, it is necessary to normalize the size of the malicious images after visualization.

We adopt the bilinear interpolation algorithm for image size normalization to maintain the original texture features of the malicious images after normalization as much as possible. This algorithm first selects four-pixel points directly adjacent to the interpolation point of the malicious image, then performs linear interpolation calculations twice in the $x$ direction, and finally performs linear interpolation in the $y$ direction to obtain the pixels of the interpolation point:

$$f(x, y_1) = \frac{x_2 - x}{x_2 - x_1} f(x_1, y_1) + \frac{x - x_1}{x_2 - x_1} f(x_2, y_1) \tag{1}$$

$$f(x, y_2) = \frac{x_2 - x}{x_2 - x_1} f(x_1, y_2) + \frac{x - x_1}{x_2 - x_1} f(x_2, y_2) \tag{2}$$

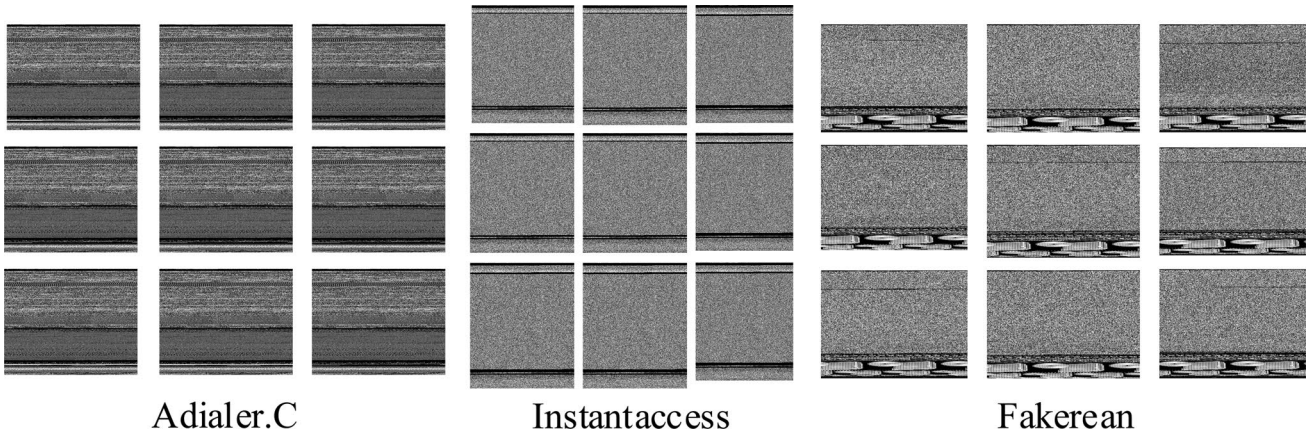



**Fig. 3** Samples of different malware family grayscale images

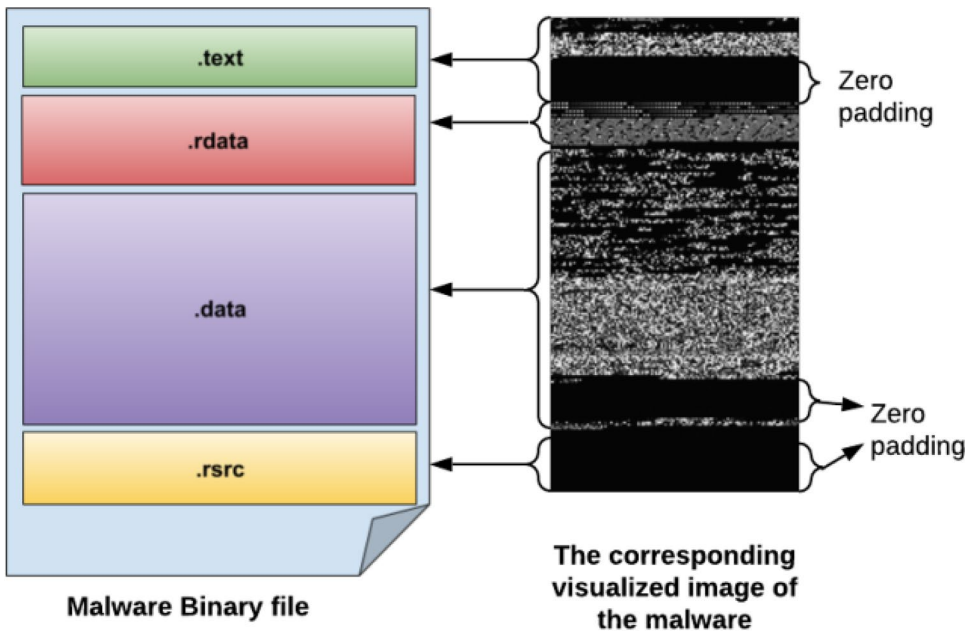



**Fig. 4** Malicious code PE file section and its corresponding visualised image fragment information

$$f(x,y) = \frac{y_2 - y}{y_2 - y_1} f(x, y_1) + \frac{y - y_1}{y_2 - y_1} f(x_2, y_2) \tag{3}$$

where $f(x, y)$ is the pixel value of the interpolation point in the malware image. $(x_i, y_j)\,(i, j = 1, 2)$ are the four pixels near the interpolation point in the malware image. Figure 5 shows the malicious image of a sample in the Allaple.A family after normalisation, by observation it can be seen that the basic texture features of the malicious image after the bilinear interpolation algorithm are well preserved.

### 2.1.3 Data Enhancement Techniques

In deep learning models, the effect of classification is closely related to the quality of the dataset, and an adequate and balanced dataset can not only improve the classification accuracy of the model but also avoid the overfitting phenomenon to a certain extent. When the number of samples in the dataset is small or the number of samples in each category is unbalanced, data enhancement techniques can be used to increase the number of samples in a few categories, so as to suppress the impact of unbalanced samples on the model and improve the robustness of the model. The common image data enhancement is to generate new data by transforming the original image data, such as: scaling, flipping, shifting, etc. To solve the problem of an unbalanced number of samples of various categories in the malicious code dataset, this paper uses the image data augmentation technique function in python to expand the samples of the dataset, and the parameter

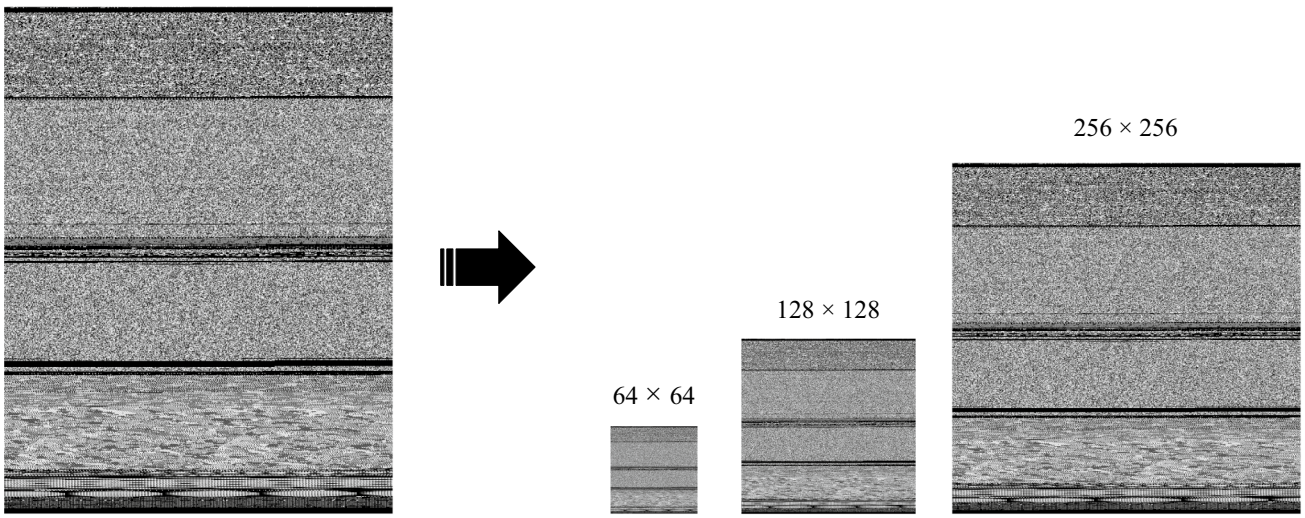



**Fig. 5** Bilinear interpolation method to deflate the malicious code image

**Table 2** The parameter settings of data augmentation

| | Method | Setting | | Method | Setting |
|---|---|---|---|---|---|
| 1 | Rescale | 1/255 | 5 | Shear range | 0.0 |
| 2 | Width shift | 0.0 | 6 | Zoom range | 0.0 |
| 3 | Height shift | 0.0 | 7 | Horizontal flip | False |
| 4 | Rotation range | 0.0 | 8 | Fill mode | None |

settings of the data augmentation technique used in the experiment are given in Table 2.

## 2.2 Feature Extraction and Classification

The design inspiration of MDC-RepNet comes from the combination of CNN and Transformer. CNN is good at extracting local features from images, while Transformer can globally capture sequence information. MDC-RepNet attempts to combine the advantages of both to achieve stronger feature extraction and model representation capabilities. The overall architecture of the MDC-RepNet is shown in Fig. 6.

The starting point of MDC-RepNet is Stem, which uses convolutional structures for feature extraction. During the inference stage, the structure consists of $3\times3$ convolution, $3\times3$ depth convolution, and $1\times1$ convolution to extract multi-scale features from the original image. To achieve structural reparameterization, additional $1\times1$ convolution or Identity branches are introduced during the training stage, providing greater flexibility to the model and helping to optimize its representation ability.

MDC-RepNet is divided into four stages, each of which halves the resolution of the feature map and doubles the number of channels. The first three stages use the same internal structure, using the Rep Mixer in Fig. 6d for token mixing. This structure aims to achieve feature reuse across stages and dimensions, and improves the representational power of the model by reparameterizing the skip connections.

The internal structure of the fourth stage is shown in Fig. 6a, using attention as a token mixer. This design sacrifices inference speed to ensure higher accuracy. The attention mechanism allows the model to focus on key information within the global scope, further improving the quality of feature representation.

The ConvFFN architecture is used in each stage of MDC-RepNet, which is different from traditional FFN. ConvFFN combines deep separable convolutions ($7\times7$) and feedforward networks to achieve more efficient feature extraction and model representation. Deep separable convolutions allow the model to learn more complex spatial features while reducing computation, which helps improve model inference speed and accuracy.

To achieve structural re-parameterization, MDC-RepNet introduces a novel fusion module. This module aims to fuse different levels of features from CNN and Transformer, leveraging the advantages of both. During training, the fusion module allows the model to adaptively adjust the feature fusion method according to task requirements, optimizing the model's performance. This design provides greater flexibility for the model, enabling it to better adapt to various visual tasks.

In summary, MDC-RepNet achieves powerful feature extraction and model representation capabilities by cleverly combining a CNN and a Transformer. the CNN is responsible for extracting local features from an image, while the Transformer captures global sequence information using a self-attentive mechanism. This integration approach enables

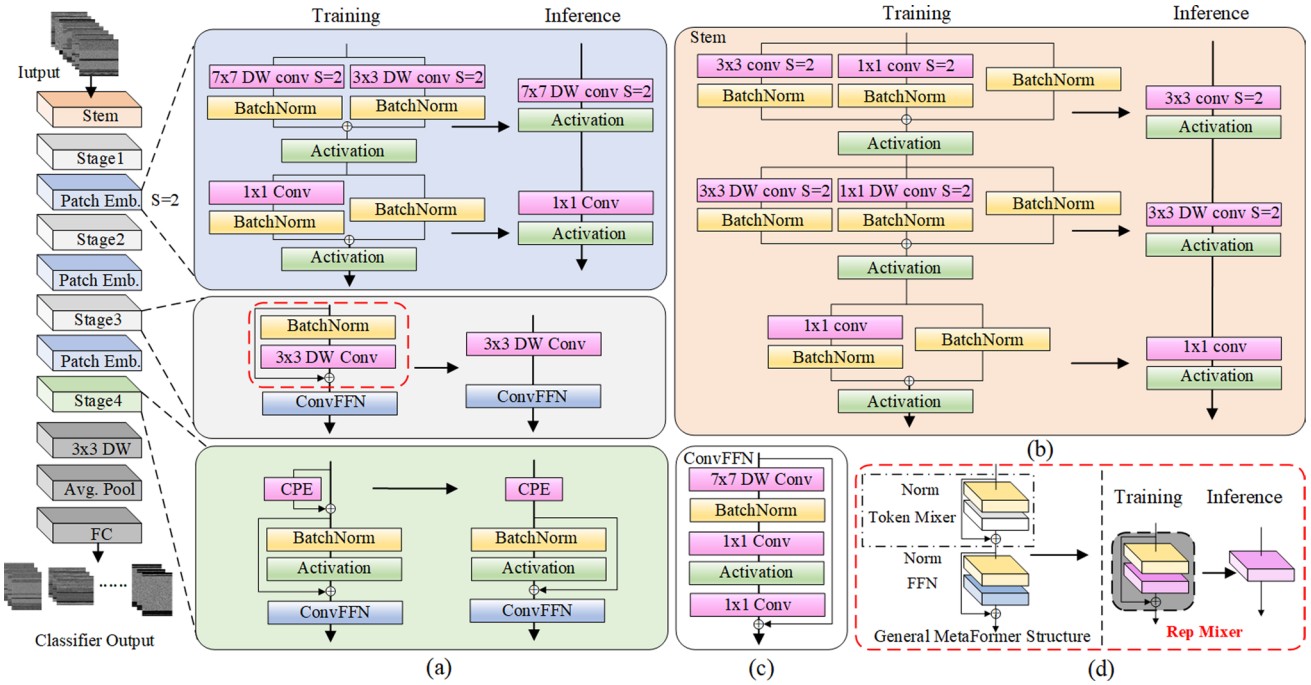



**Fig. 6** Overall architecture of the MDC-RepNet

MDC-RepNet to simultaneously process both spatial and sequence information of an image, resulting in excellent performance in a variety of visual tasks. In addition, by introducing the novel fusion module and ConvFFN architecture, MDC-RepNet further improves its feature extraction and model representation capabilities.

### 2.2.1 Structural Reparameterization

Multi-branch network structures, boasting receptive fields of varying scales, increase the network's width and parameter amount compared to tiling network structures. This is conducive to enhancing network performance. However, as the network becomes more branched, the memory consumption during training and inference speed are significantly affected. Therefore, Ding et al. [39] propose the notion of structural reparametrization, which equivalently converts complex multi-branch structures into a single-branch structure. In this way, a network with a multi-branch structure can be selected during training to enhance network performance. After training, the network can be converted into a single-branch structure for inference. The converted single-branch structure can maintain the original network's performance while improving the network's running speed and reducing memory consumption and the parameter count.

Figure 7 illustrates the transformation of the multi-branch structure during training into a single-branch structure for inference. Figure 7a presents a multi-branch structure known as a basic block. Apart from the bottommost branch in the basic block, each branch comprises a convolution layer and a Batch Normalization (BN) layer. In contrast, the bottommost branch consists solely of a BN layer. Fusing all branches in the basic block into a single branch entails multiple fusion steps, including the fusion of the BN layer with the convolution layer and the fusion of convolution layers of different sizes, etc.

Fusion of the BN layer with the convolution layer. The Conv. change and BN operation are represented as follows:

$$y_{\text{conv}} = \omega \cdot x + b \tag{4}$$

$$BN_{\gamma,\beta}(y_{\text{conv}}) = \gamma \frac{y_{\text{conv}} - \mu_B}{\sqrt{\sigma_B^2 + \varepsilon}} + \beta \tag{5}$$

In the equations, $x$ represents the input. Hence, after passing through the convolution layer and BN layer, the input $x$ can be expressed as:

$$\text{BN}_{\gamma,\beta}(x) = \frac{\gamma\omega}{\sqrt{\sigma_B}^2 + \varepsilon} x + \frac{\gamma}{\sqrt{\sigma_B}^2 + \varepsilon}(b - \mu_B) + \beta \tag{6}$$

In the equations, $\omega$ and $b$ are the weights and bias before merging, $\gamma$ and $\beta$ are the translation and scaling parameters obtained after training, $\mu_B$ and $\sigma_B$ are the means and variances of all training data, respectively, $\varepsilon$ is a very small constant to avoid division by zero.

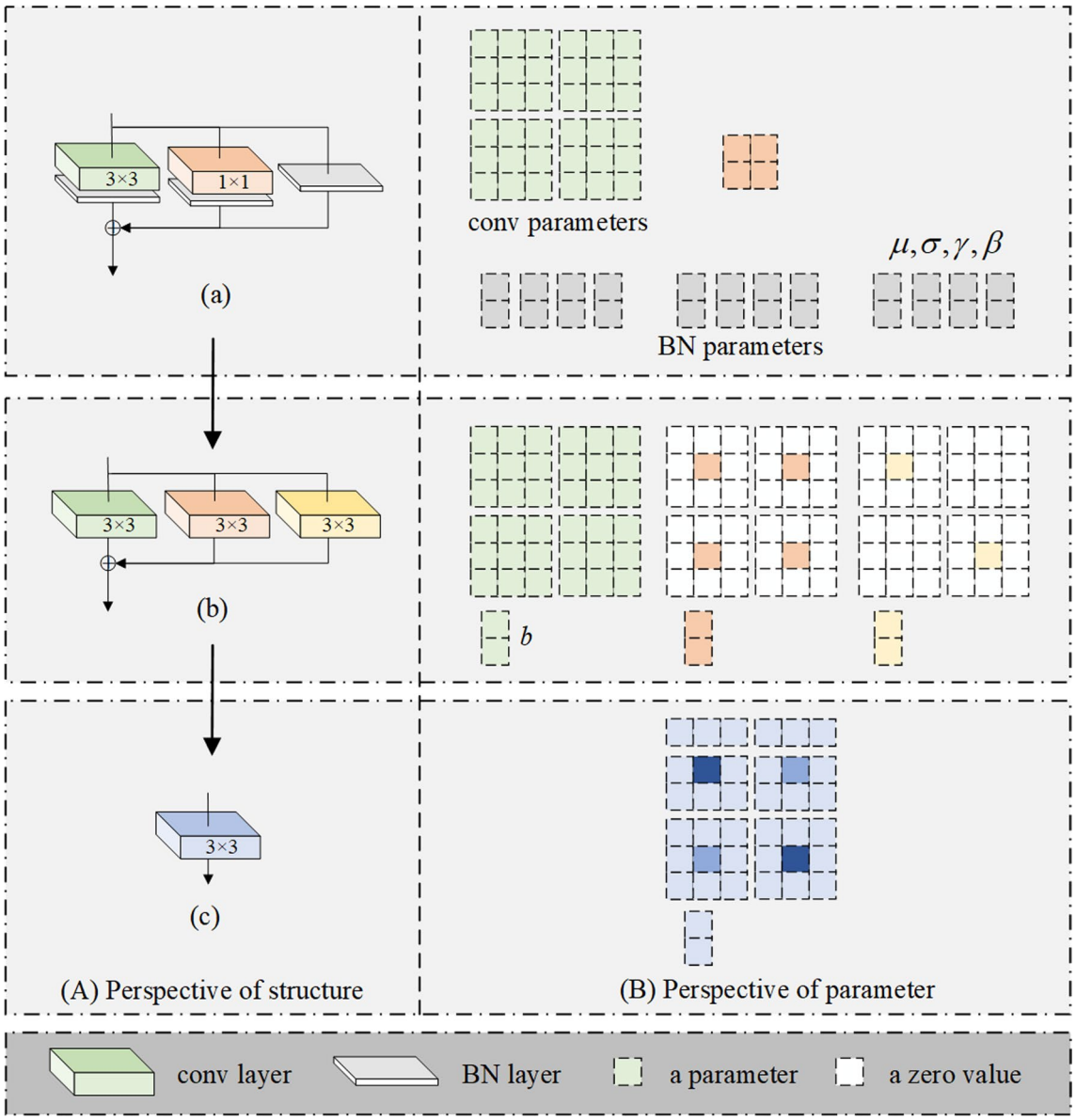



**Fig. 7** Processing flow for transforming a multi-branch structure into a single-branch structure

$$\hat{\omega} = \frac{\gamma \omega}{\sqrt{\sigma_B^2 + \varepsilon}} \tag{7}$$

$$\hat{b} = \frac{\gamma}{\sqrt{\sigma_B^2 + \varepsilon}}(b - \mu_B) + \beta. \tag{8}$$

From Eqs. (7) and (8):

$$\mathrm{BN}_{\gamma,\beta}(x) = \hat{\omega} \cdot x + \hat{b} \tag{9}$$

where $\hat{\omega}$ and $\hat{b}$ are the weights and bias of the fused convolutional kernel, respectively. The convolutional layer can be fused with the BN layer by the above transformations to reduce the computation amount during inference. The specific fusion method is:

(1) Fusion of $1\times1$ convolution with $3\times3$ convolution. The $1\times1$ convolution is filled to $3\times3$ size using 0. Utilizing the additivity of convolution, the convolution obtained from the filling is added to the original $3\times3$ convolution to obtain the fused $3\times3$ convolution.

(2) Fusion of $3\times3$ convolution and $7\times7$ convolution. Similar to the fusion of $1\times1$ convolution and $3\times3$ convolution, the $3\times3$ convolution is filled with zeros to a convolution of size $7\times7$ and then added to the original $7\times7$ convolution to obtain the fused $7\times7$ convolution.

(3) Fusion of residual branches with BN layer. Residual branching can be considered a convolution operation of the original input data with a convolution kernel with unique parameters. Using $1\times1$ convolution and making the parameter of the convolution kernel corresponding to the current number of channels one and the parameter corresponding to the rest of the channels 0, the output can be obtained as the same as the input. This $1\times1$ convolution is fused with the BN layer and then filled to a $5\times5$ convolution using zeros.

After fusing the convolutional layer with the BN layer using the fusion method described above, the transformation of Fig. 7a to Fig. 7b can be realized, and the additivity of the convolution can be utilized to transform Fig. 7b, c. Up to this point, a multi-branch basic block during training can be equivalently transformed into a single convolution during inference.

#### 2.2.2 Linear Train-Time Overparameterization

To further improve the efficiency of parameter counts, FLOPs, and delays, all the model's dense $k\times k$ convolutions are replaced with their factorized versions, i.e., $k\times k$ depth convolutions followed by $1\times1$ point convolutions. However, the lower number of parameters resulting from the factorization reduces the model's capacity. To increase the capacity of the decomposition layer, a linear training time over-parameterization is used, which means that the model is trained using more parameters than are required. Adding more parameters during the training process helps to fit the training data better and thus improves the model's performance. However, due to the increased computational overhead of branching, training time overparameterization leads to an increase in training time.

Over-parameterizing the model with additional parameters on a linear scale. The advantage is that although the complexity of the model increases, the training time does not increase dramatically because the scale of the model is linear, so the model can still be trained in a reasonable amount of time. MDC-RepNet replaces only dense $k\times k$ convolutions with their decomposed forms and over parameters as described above. These layers are in the convolutional backbone, block embedding, and fully connected layers. Since the computational cost of these layers is lower than the rest of the network, overparameterizing these layers does not result in a significant increase in training time.

#### 2.2.3 Large Kernel Convolution

Compared with the sensory field of Self-Attention in Vision Transformer architecture, the sensory field of Rep Mixer is localized. However, the Vision Transformer based on the Self-Attention mechanism has a higher computational overhead. Introducing deep large kernel convolution in FFN and patch embedding layers is a computationally efficient way to improve the sensory field in the early stages without using Self-Attention by combining deep large kernel convolution. MDC-RepNet introduces large kernel convolution at the Patch Embedding layer and FFN. The experimental model parameter settings are shown in Table 3.

The architecture of the FFN and patch embedding layer is shown in Fig. 5c. The FFN block has a similar structure to the ConvNet block with some key differences. The MDC-RepNet architecture uses Batch Normalization instead of Layer Normalization in the ConvNet block. The BN layers can be fused with the previous layer during inference, and, similar to the implementation of the ConvNet block, no additional reshaping operations are required to obtain a suitable tensor layout for the Layer Norm.

Conv. FFN blocks are usually more robust than vanilla-FFN blocks [40], as the sensory field increases, a sizeable convolutional kernel helps to improve the robustness of the model. Therefore, combining large convolutional kernels is an effective way to improve model performance and robustness.

**Table 3** MDC-RepNet parameter settings

| Stage | Tokens | Layer spec | | Settings |
|---|---|---|---|---|
| Stem | $H\times W$ | Conv | | $3\times3$, stride $=2$ |
| | | | | $3\times3$, stride $=2$ |
| | | | | 48 |
| 1 | $\frac{H}{4}\times\frac{W}{4}$ | Patch embed | | $7\times7$, stride $=2$ |
| | | | | 48 |
| | | MDC-Rep block | Mixer | Rep mixer |
| | | | Exp | 3 |
| | | | Blocks | 2 |
| 2 | $\frac{H}{8}\times\frac{W}{8}$ | Patch embed | | $7\times7$, stride $=2$ |
| | | | | 96 |
| | | MDC-Rep block | Mixer | Rep mixer |
| | | | Exp | 3 |
| | | | Blocks | 2 |
| 3 | $\frac{H}{16}\times\frac{W}{16}$ | Patch embed | | $7\times7$, stride $=2$ |
| | | | | 192 |
| | | MDC-Rep block | Mixer | Rep mixer |
| | | | Exp | 3 |
| | | | Blocks | 4 |
| 4 | $\frac{H}{32}\times\frac{W}{32}$ | Patch embed | | $7\times7$, stride $=2$ |
| | | | | 384 |
| | | MDC-Rep block | Mixer | Rep mixer |
| | | | Exp | 3 |
| | | | Blocks | 2 |

## 3 Experiments and analyses

### 3.1 Data Set and Experimental Environment

This experiment uses the Malimg dataset published by the University of California Vision Research Laboratory for model training and classification. The dataset contains a total of 9435 malicious code sample data from 25 malicious families, and the operating system in the model training is Ubuntu version 22.04.2, running in CUDA 11.8 environment, using Pytorch1.11.0 deep learning framework, and hardware configuration of Nvidia GeForce RTX 3080Ti.

The Malimg dataset is divided into two parts, the training set and the validation set, in a ratio of 9:1. Among them, the training set is used for model training, and the validation set is used for observing and evaluating the model's performance. The name of each family and the distribution of the number of samples per family are shown in Table 4.

**Table 4** Details of malimg dataset

| ID | Family name | Type | Number of malignant samples |
|---|---|---|---|
| 1 | Adialer.C | Dialer | 122 |
| 2 | Agent.FYI | Backdoor | 116 |
| 3 | Allaple.A | Worm | 2949 |
| 4 | Allaple.L | Worm | 1591 |
| 5 | Alueron.gen!J | Trojan | 198 |
| 6 | Autorun.K | Worm:AutoIT | 106 |
| 7 | C2LOP.gen!g | Trojan | 200 |
| 8 | C2LOP.P | Trojan | 146 |
| 9 | Dialplatform.B | Dialer | 177 |
| 10 | Dontovo.A | TrojanDownloader | 162 |
| 11 | Fakerean | Rogue | 381 |
| 12 | Instantaccess | Dialer | 431 |
| 13 | Lolyda.AA1 | PWS | 213 |
| 14 | Lolyda.AA2 | PWS | 184 |
| 15 | Lolyda.AA3 | PWS | 123 |
| 16 | Lolyda.AT | PWS | 159 |
| 17 | Malex.gen!J | Trojan | 136 |
| 18 | Obfuscator.AD | TrojanDownloader | 142 |
| 1 | Rbot!gen | Backdoor | 158 |
| 20 | Skintrim.N | Trojan | 80 |
| 21 | Swizzor.gen!E | TrojanDownloader | 128 |
| 22 | Swizzor.gen!I | TrojanDownloader | 132 |
| 23 | VB.AT | Worm | 408 |
| 24 | Wintrim.BX | TrojanDownloader | 97 |
| 25 | Yuner.A | Worm | 800 |

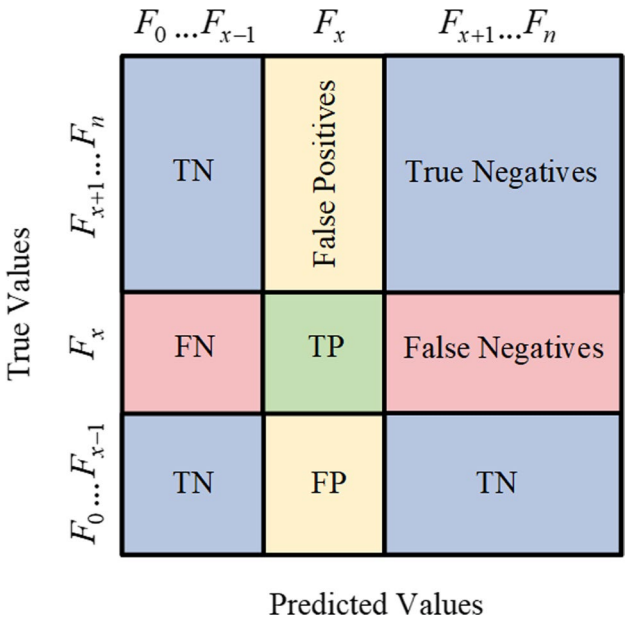



**Fig. 8** Confusion matrix for multi-categorization problems

### 3.2 Evaluation Indicators

#### 3.2.1 Model Training Performance Evaluation Metrics

Four common rubrics in the field of malicious code classification: Accuracy, Precision, Recall, and F1-score are selected to evaluate the model classification, which has been widely used in related research [41–43]. The formulas are as follows:

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} \tag{10}$$

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

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

$$F1 - \text{score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{13}$$

where TP is the actual class (meaning that malware is correctly categorized as malware), FN is the false negative class (meaning that malicious code is incorrectly categorized as regular code), FP is the false positive class (meaning that regular code is incorrectly categorized as malicious code), and TN is the actual negative class (meaning that regular code is correctly categorized as standard code).

The performance of the model is presented using a visual representation of the confusion matrix, and the values of

TP, FP, TN, and FN for the multiclassification problem are shown in Fig. 8 where Fi ($i=0,1,2\ldots n$) denotes the malicious code family category.

#### 3.2.2 Indicators for Evaluating the Speed of Model Inference

The MDC-RepNet uses structural reparameterization to optimize the rate of model operation in the inference phase. Using a divided test set (10%) of about 944 images as experimental data, the inference speed of MDC-RepNet versus classical network models for predicting an unknown sample, i.e., the time overhead incurred in categorizing each image of malicious code, is calculated as a metric for determining the inference speed of the model.

### 3.3 Analysis of Experimental Results

#### 3.3.1 Hyperparameter Comparison Experiment

Limited by the characteristics of fully connected layers in CNNs, the size size of the malignant code image input to the model must be deterministic. In addition, the size of the image put into the CNN will not only affect the size of the model but also change the effectiveness of the model. In order to obtain the most suitable input image size for the model, this paper applies bilinear interpolation to normalize the malicious code images to $32\times32$, $64\times64$, $128\times128$, $256\times256$, and $512\times512$. The performance of the model is experimentally tested by inputting the malicious images in the Malimg dataset into the model, and it can be analyzed in Table 5 to know that, after the size of the malicious code image is increased from $32\times32$ to $256\times256$, is an increase in accuracy from 86.65 to 99.57%, however, when the image size continues to increase from $256\times256$ to $512\times512$, the accuracy decreases from 99.57 to 98.92%, which indicates that the model is overfitting. Moreover, as the image size increases, the number of parameters increases gradually. This is because the more significant the image size, the more convolution operations and parameters are involved, which further increases the consumption of computer resources and leads to longer model training time. Then, after a comprehensive trade-off between the malicious code classification accuracy and the number of parameters, a $256\times256$ malicious code image is finally chosen as the input to the model.

In deep learning, optimizers are algorithms that update and find the optimal parameters of a model. For the purpose of optimizing model parameters, in this paper, we conduct comparative experiments on some optimizers Adagrad, Adamax, Adam, NAdam, and AdamW that perform well in classification tasks. Table 6 lists the performance graphs of the above optimizers concerning the relevant metrics. The experimental results show that AdamW outperforms other optimizers regarding accuracy, precision, recall, and F1-score. Therefore, in this paper, AdamW is selected as the optimizer of MDC-RepNet for performing the malicious code family classification task.

#### 3.3.2 Experiments to Verify the Validity of Structural Reparameterization

A structural reparameterization technique is used to decouple and separate the training process and inference phase of the MDC-RepNet. The multi-branch model is first trained, then equivalently transformed into a one-way model, and finally the model is deployed. In the prediction phase, the actual model that has been transformed is run. To verify the effectiveness of structural reparameterization, experiments are set up to analyze the training and inference phases quantitatively, and to evaluate the accuracy, FLOPs, inference speed, the number of parameters, and the model size of the MDC-RepNet before and after the transformation, to verify the actual effect of structural reparameterization. In the model training, the number of samples selected for each training batch size is set to 32, the number of Epoch is set to 30, the initial learning rate is set to 0.0001 using the AdamW optimizer, the weight decay is 0.05, the peak learning rate is $10^{-3}$, and the cosine scheduling is used to attenuate the learning rate, and the loss function is the cross-entropy loss function, the results are shown in Table 7.

The MDC-RepNet has the same recognition accuracy before and after the conversion, the FLOPs, the number of parameters, and the model volume are reduced by about 10%, while the inference speed is reduced by about

**Table 5** Experimental results on the effect of input image size on the model

| Input | Accuracy (%) | Recall (%) | F1-score (%) | Params (M) |
|---|---|---|---|---|
| $32\times32$ | 86.65 | 86.65 | 86.64 | 0.31 |
| $64\times64$ | 95.32 | 95.32 | 95.30 | 0.35 |
| $128\times128$ | 98.47 | 98.47 | 98.47 | 0.50 |
| $256\times256$ | 99.57 | 99.58 | 99.57 | 1.11 |
| $512\times512$ | 98.92 | 98.90 | 98.91 | 2.58 |

**Table 6** Comparative experimental results of different optimizers

| Optimizer | Accuracy (%) | Precision (%) | Recall (%) | F1-score (%) |
|---|---|---|---|---|
| Adagrad | 97.20 | 96.26 | 97.20 | 96.72 |
| Adamax | 97.57 | 96.71 | 97.57 | 97.63 |
| Adam | 98.23 | 98.20 | 98.23 | 98.21 |
| NAdam | 99.10 | 99.18 | 99.10 | 99.13 |
| AdamW | 99.57 | 99.56 | 99.58 | 99.57 |

**Table 7** Validation experiments on the validity of structural reparameterization

| Mould | Accuracy (%) | FLOPs (G) | Pt (ms) | Params (M) | Model Volume (M) |
|---|---|---|---|---|---|
| MDC-Net | 99.57 | 1.52 | 124 | 7.65 | 60.4 |
| MDC-RepNet | 99.57 | 1.36 | 104 | 4.29 | 46.1 |

*Pt* prediction time

15–20%. The experimental results show that with the increase of the model depth and width, the inference speed of the model is reduced, and the recognition efficiency is significantly improved.

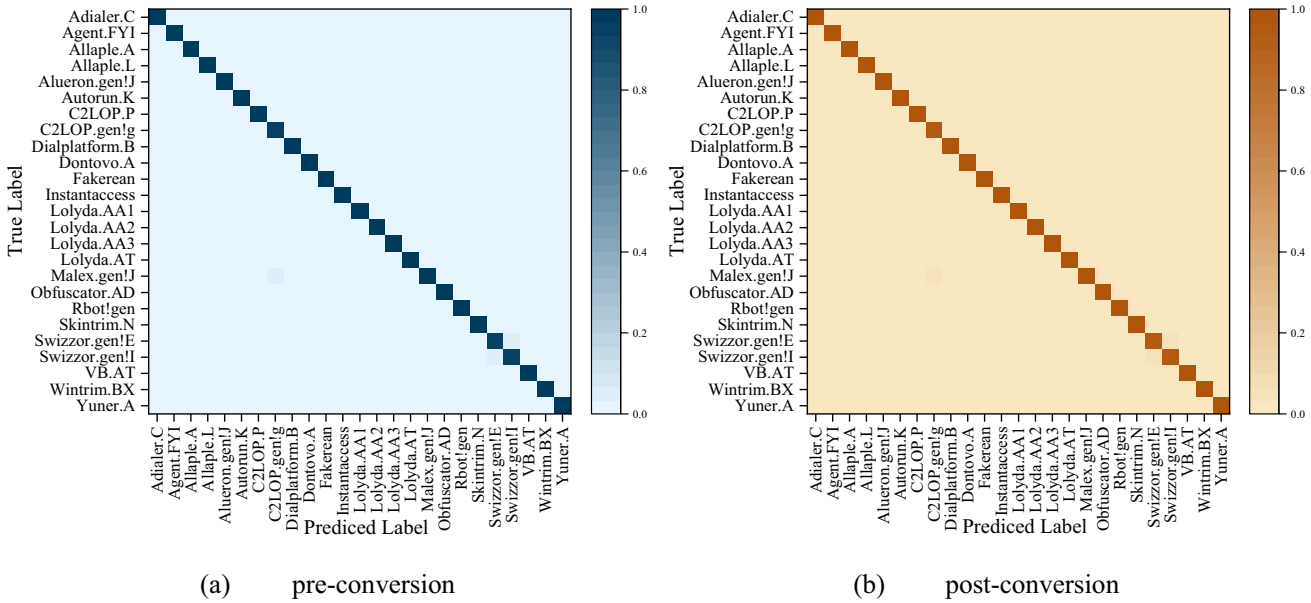
To further verify the effectiveness of structural reparameterization, the confusion matrices of the MDC-RepNet before and after conversion are plotted to verify its effectiveness more intuitively. The confusion matrices before and after structural reparameterization are shown in Fig. 9. Observing that the confusion matrices before and after structural reparameterization are identical, it can be learned that the recognition accuracy before and after the conversion is also identical, and the network can achieve a high recognition accuracy by training 30 Epochs. The results show that the MDC-RepNet before and after structural reparameterization does not reduce the recognition accuracy and affects the performance of the model before conversion. Decoupling the training and inference phases reduces the inference speed of the model and reduces the time for malicious code image classification and recognition.

### 3.3.3 MDC-RepNet Ablation Synthesis Experiment

The MDC-RepNet introduces a large kernel convolution at two locations, the patch embedding layer, and the FFN, to improve the sensory field in the early stages without using self-attention by combining the deep large kernel convolution. To verify the role of the large kernel convolution for the whole network. Setting up the ablation experiment using the large kernel convolution to replace the self-attention module layer by layer, the experimental results are shown in Table 8.

RM indicates that the current stage uses RepMixer-FFN blocks. SA indicates that the current stage uses Self-Attention-FFN blocks. The standard setup uses $3\times3$ factorization convolution in the block embedding and backbone layers, and $1\times1$ convolution in the FFN. In variants V4 and V5, large kernel convolution ($7\times7$) is used for the patch embedding and FFN layers.

Comparison shows that V5 has an 11.2% increase in model size, a 0.4% gain in accuracy, and a $2.3\times$ increase in inference speed with V3. V2 is 20% larger than V4 and achieves similar accuracy, while inference speed is 7.1% higher than V4. Overall, large kernel convolution provides




(a) pre-conversion (b) post-conversion

**Fig. 9** Confusion matrix before and after structural reparameterization

**Table 8** Large kernel convolutional ablation experiments

| Variant | Stages | | | | Params (M) | Accuracy (%) | Pt (ms) |
|---|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | | | |
| Standard setting | | | | | | | |
| V1 | RM | RM | RM | RM | 6.37 | 97.89 | 52 |
| V2 | RM | RM | RM | SA | 10.82 | 98.92 | 60 |
| V3 | RM | RM | SA | SA | 10.41 | 99.66 | 122 |
| Large kernel convolutions (7×7) | | | | | | | |
| V4 | RM | RM | RM | RM | 6.81 | 98.48 | 56 |
| V5 | M | RM | RM | SA | 8.92 | 99.57 | 67 |

**Table 9** MDC-RepNet ablation synthesis experiment

| 1×1 Shortcut Branch | BN Shortcut Branch | MDC Branch | Accuracy (%) | Pt (ms) |
|---|---|---|---|---|
| | | | 90.41 (9.16↓) | 163 |
| √ | | | 92.74 (6.83↓) | 103 |
| | √ | | 94.28 (5.29↓) | 138 |
| √ | √ | | 96.47 (3.10↓) | 87 |
| √ | √ | √ | 99.57 | 86 |

"90.41(9.16↓)" means that the accuracy is 90.41% without any component, which is a decrease of 9.16% compared to MDC-RepNet, "92.74(6.83↓)" means that the accuracy is 92.74% with only 1×1 shortcut branches only, the accuracy is 92.74%, which is 6.83% lower than that of MDC-RepNet, and so on

a 0.9% accuracy gain on MDC-RepNet. Experimental results show that using deep large-kernel convolution has similar effects as using the self-attention mechanism while inducing a slight increase in inference speed.

To further verify the enhancement effect of the multi-branch structure and the introduction of the large kernel convolution on the network representation ability, several components of MDC-RepNet are removed by the network ablation method, and the role of the components for the whole network is analyzed. The two shortcut branches and the large kernel convolutional block of MDC-RepNet are used as components to test the recognition accuracy of the model after combining with the main branch, respectively. Ensuring that other parameters remain unchanged, the results of the ablation analysis experiments are shown in Table 9.

As can be seen from Table 8, when not including any of the components, the accuracy decreases by 9.16% compared to MDC-RepNet, but the inference speed increases dramatically. When only one component is used, the accuracy rate also decreases to different degrees compared to MDC-RepNet, and the inference speed increases accordingly. The results show that the multi-branch structure can increase the network's representation ability and improve the model's recognition accuracy. After the introduction of large kernel convolution, MDC-RepNet increased the accuracy by 3.1% compared to the original network, and the inference speed was the same, indicating that the introduction of large kernel convolution did not increase any computational resources for the inference stage.

### 3.3.4 Comparison with Other Deep Learning Models in Image Classification Tasks

The study expects to improve the inference speed of the model as much as possible and reduce the recognition time of the image while ensuring higher recognition accuracy. In this section, the model is trained and tested against classical networks (AlexNet, VGG series, and ResNet series) and lightweight networks (DenseNet, MobileNetV2, and ShuffleNetV2), which are deep neural network models with excellent performance in image classification tasks based on the Malimg dataset. The results of the test of the aforementioned network models are compared to the data shown in Table 10.

As can be seen from Table 9, the MDC-RepNet possesses fewer FLOPs and faster inference speed with higher accuracy than the VGG16 network, the number of parameters is about 5% of the latter, and the model volume is about 22% of the latter. Through structural reparameterization, the MDC-RepNet adopts the one-way model architecture of the VGG16-style network in the inference stage, and the inference speed is about twice as fast as that of the latter. The MDC-RepNet adopts the multi-branch structure as the backbone network and discards the fully connected layer of the VGG16-style network. The FLOPs are about 10% of that of the latter, and the complexity of the model is reduced dramatically.

Compared with the multi-branch structure ResNet series network, MDC-RepNet has more evident advantages in FLOPs, inference speed, number of parameters, and model volume over the former group compared with the ResNet50 network. The MDC-RepNet borrows the residual module from the ResNet series network, making it possible to train the network deeper and making the network more complex in terms of similar recognition results. The MDC-RepNet borrows the residual module from the ResNet series network, making it possible to train the network deeper and

**Table 10** Comparison of experimental results of different network models

| Model | Accuracy (%) | FLOPs (G) | Pt (ms) | Params (M) | Model volume (M) |
|---|---|---|---|---|---|
| AlexNet8 | 95.01 | 22.14 | 65 | 276.49 | 513.19 |
| VGG16 | 96.91 | 15.91 | 51 | 131.37 | 214.00 |
| ResNet50 | 97.94 | 6.52 | 75 | 32.75 | 96.07 |
| DenseNet | 99.07 | 3.16 | 118 | 28.44 | 126.30 |
| MobileNetV2 | 97.63 | 0.31 | 124 | 2.51 | 9.07 |
| ShuffleNet | 98.41 | 0.15 | 131 | 1.48 | 5.23 |
| MDC-RepNet | 99.57 | 1.56 | 86 | 7.79 | 47.10 |

reduce the model complexity in the case of a similar recognition effect with a lower number of parameters and faster inference.

MobileNet and ShuffleNet networks are network architectures designed for devices with limited computing resources, which significantly reduce the computational overhead while retaining a certain level of model accuracy, significantly increase the inference speed, and greatly reduce the FlOPs, the number of parameters, and the size of the model. The MDC-RepNet is a high-efficiency model designed for dedicated hardware or GPU, which pursues faster inference speed and more memory-saving. Faster and more memory-efficient, with less attention to the number of parameters and FLOPs. MDC-RepNet improves the recognition accuracy by 1–2% compared to MobileNetV2 and ShuffleNetV2, but the inference speed is not as fast as the two, and the number of parameters and FLOPs are higher than the two. However, it can be observed that compared with MobileNetV2, the FLOPs are five times higher than the latter. However, the inference speed is 71% of the latter, which shows that the FLOPs do not entirely determine the inference speed, indicating that the computational density of the MDC-RepNet is higher.

To observe the combined classification accuracy and detection time more intuitively, the accuracy and detection time results of each model are plotted in Fig. 9. The Fig. shows that the MDC-RepNet has the highest accuracy rate, and it only takes 219 ms to predict an unknown sample, which better realizes the balance between accuracy and delay. In summary, the MDC-RepNet has a high accuracy rate, a low number of parameters, and a high computational density. This is because the MDC-RepNet efficiently captures both local and global information. The structural reparameterization module obtains lower memory access costs and higher efficiency by eliminating jump connections in the network. At the same time, techniques such as large kernel convolution are used to improve accuracy. This ultimately reduces the number of parameters in the model and decreases the amount of floating-point operations, improving the model's speed.

To further observe and analyze the classification performance of the models, Fig. 10 plots the classification details

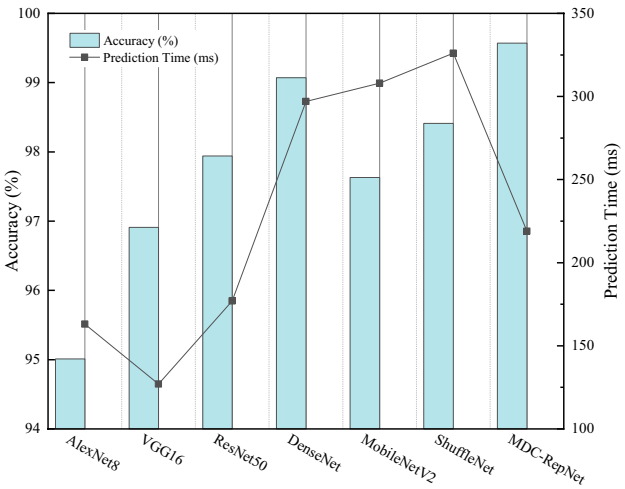
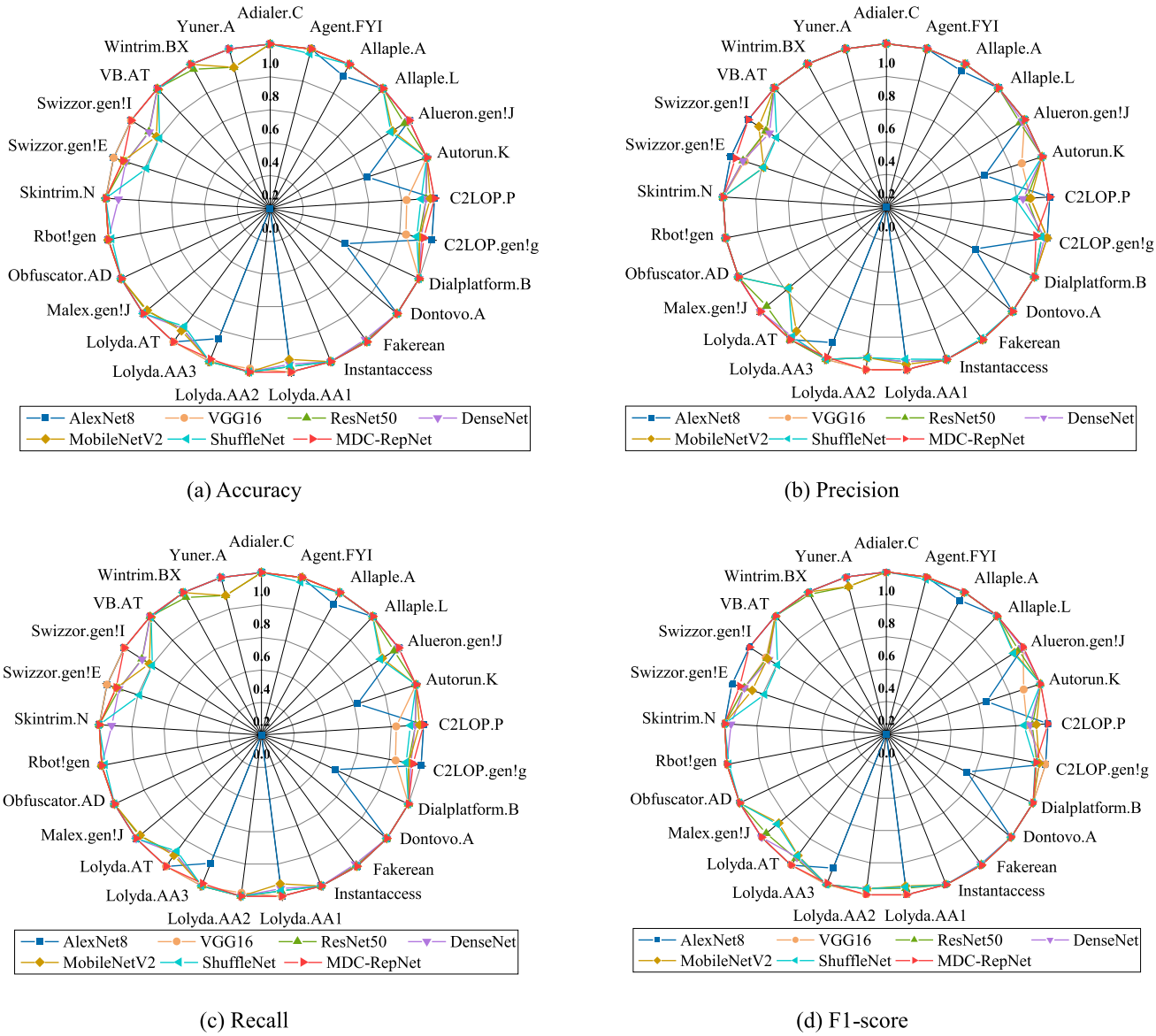



**Fig. 10** Comparison of classification accuracy and detection time of each model

of each model in each malicious family, and the results show that the MDC-RepNet, through time-training over-parametrization and extensive kernel convolution techniques, improves the above classical deep neural network model's insufficient classification accuracy in some confusing malicious families to varying degrees, and then improves the overall classification accuracy, Accurately classify 25 families on the Malimg dataset. Experiments prove that the malicious code classification performance of this paper's model outperforms other models (Fig. 11).

### 3.3.5 Comparison with Other Malicious Code Classification Techniques

To verify that the proposed model can achieve satisfactory performance and to further validate the model's malicious code detection capability, this section compares MDC-RepNet with existing malicious code detection methods based on visual analysis based on the Malimg dataset. Table 11 summarises the performance metrics of each malicious code detection method, from which we can see that MDC-RepNet achieves an accuracy of 99.57% and a prediction time of 67 ms, which is much lower than that of the existing




(a) Accuracy

(b) Precision

(c) Recall

(d) F1-score

**Fig. 11** Classification details of each model in malicious families

state-of-the-art research. Our model achieves better results in terms of accuracy, precision and other evaluation metrics than all existing research techniques.

The reason for this is related to the properties of convolution: in this paper, we use point-by-point convolution after deep convolution, large kernel convolution instead of the self-attention method to improve the model performance in the early stage, and re-parameterisation and jump-over connectivity in inference, which eliminates the time overhead of the extra branches in the inference stage and improves the classification accuracy without introducing the extra inference time to aggravate the computational burden. Thus the model parameters from the training phase can be equivalently used at inference time, resulting in lower memory access costs and higher efficiency, achieving an accuracy-latency balance.

## 4 Conclusion

This paper proposes a malicious code detection method that combines CNNs and Transformers. The method takes deep neural networks as a framework, adopts the modular design idea, and introduces a new Token hybrid operator to make its structure reparameterized, which reduces the memory access cost by eliminating the jump connections in the network. Meanwhile, this paper adopts techniques such as training time over-parameterization and large kernel convolution to

**Table 11** Performance metrics of each malicious code classification method

| Methods [references] | Year | Dataset | Description | Accuracy (%) | Precision (%) | Recall (%) | F1-score (%) | Pt (ms) |
|---|---|---|---|---|---|---|---|---|
| [44] | 2011 | Malimg | GIST + KNN | 97.18 | – | – | – | – |
| [5] | 2016 | Malimg | SPAM | 97.40 | – | – | – | – |
| [45] | 2018 | Malimg | DRBA + CNN | 94.50 | 96.60 | 88.40 | – | – |
| [46] | 2019 | Malimg | LGMP + KNN | 98.40 | – | 98.20 | 97.1 | – |
| [47] | 2019 | Malimg | NSGAII + CNN | 97.60 | 97.60 | 88.40 | – | – |
| [48] | 2019 | Malimg | CNN + LSTM | 96.30 | 96.30 | 96.20 | 96.20 | – |
| [49] | 2019 | Malimg | CNN + BiGRU | 96.30 | 91.80 | 91.50 | 91.60 | – |
| [50] | 2019 | Malimg | CSGM + KNN | 98.40 | – | 98.20 | 97.10 | – |
| [51] | 2020 | Malimg | MxN + GLCM | 98.58 | 98.04 | 98.06 | 98.05 | – |
| [52] | 2020 | Malimg | SWS + RF | 98.65 | 98.86 | 98.63 | 98.74 | – |
| [53] | 2020 | Malimg | DCNN | 98.79 | 98.79 | 98.47 | 98.46 | – |
| [54] | 2020 | Malimg | IMCFN | 98.82 | 98.85 | 98.81 | 98.75 | 810 |
| [55] | 2021 | Malimg | DenseNet201 | 98.97 | – | – | 98.88 | – |
| [55] | 2021 | Malimg | DEAM + Densenet | 98.50 | 96.90 | 96.60 | 96.70 | – |
| [56] | 2021 | Malimg | Two-level ANN | 99.13 | – | – | – | – |
| [57] | 2021 | Malimg | MCFT-CNN | 99.19 | 97.72 | 97.76 | 97.68 | – |
| [58] | 2022 | Malimg | DTMIC | 98.93 | 99.00 | 99.00 | 99.00 | – |
| Ours | 2023 | Malimg | MDC-RepNet | 99.57 | 99.56 | 99.58 | 99.57 | 67 |

improve the accuracy. In the data preprocessing stage, this paper uses pixel-filling-based image size normalization algorithm and data enhancement techniques to improve the loss of image texture information and dataset category imbalance problem during malicious code image size deflation, respectively, and to enhance the expression of critical features to alleviate the overfitting phenomenon of the model. Finally, the deep neural network model is trained to realize the classification of malicious code and its variants. Through experiments, it is proved that the method in this paper has a stable improvement in accuracy and operation efficiency, which is better than the current malicious code detection technology.

In our future research, we will further investigate, design and implement appropriate improvements to enhance the performance and applicability of MDC-RepNet. We will try to address some of the current limitations of MDC-RepNet with respect to family confusion and continue to explore new methods and techniques such as color deconvolution techniques [59], dynamic multi-scale topology techniques [60] and linguistic sequence-based evaluation methods [61, 62], etc., Combining the latest deep learning-based research results [22, 24] to improve the research in the field of malware classification.

**Author Contributions** The authors confirm their contribution to the paper as follows: SL: conceptualization, methodology, writing—original draft, JW: software, validation, resources, YS: funding acquisition, project administration, supervision, SW: writing—review and editing, YW: review and proofreading.

**Funding** This research is supported by the National Natural Science Foundation of China (61806219, 61703426, 61876189), the Natural Science Foundation of Shaanxi Province (2021JM-226), the Young Talent Fund of Association for Science and Technology in Shaanxi, China (20190108, 20220106), and the Innovation Capability Support Program of Shaanxi (2020KJXX-065).

**Availability of Data and Material** The data that support the findings of this study are available from the corresponding author, Yafei Song, upon reasonable request.

## Declarations

**Conflict of Interest** There is no conflict of interest in this study.



## References

1. National Internet Emergency Response Center: First half of China's Internet network security detection data analysis report

[EB/OL]. [2021–07–31]. First-half year cybersecurity report 2021.pdf (cert. org.cn) (Chinese) (2021)
2. Rising Star. China Cybersecurity Report [EB/OL]. [2022–02–03]. http://it.rising.com.cn/d/file/it/dongtai/20230203/2022baogao.pdf.(Chinese) (2022)
3. Conti, G., Bratus, S., Shubina, A., et al.: Automated mapping of large binary objects using primitive fragment type classification[J]. Digit. Investig. **7**, S3–S12 (2010)
4. Nataraj, L., Karthiketan, S., Jacob, G., et al. Malware images: visualization and automatic classification [C]. In: Proceedings of the 8th International Symposium on Visualization for Cyber Security. ACM, New York, pp 1–7 (2011)
5. Nataraj, L., Manjunath, B.S.: SPAM: signal processing to analyze malware[J]. IEEE Signal Process. Mag. **33**, 105–117 (2016)
6. Kancherla, K., Mukkamala, S.: Image visualization based malware detection[C]. In: 2013 IEEE Symposium on Computational Intelligence in Cyber Security. Singapore: IEEE pp. 40–44 (2013)
7. Liu, Y.S., Wang, Z.H., Yan, H.B., et al.: Method of anti-confusion texture feature descriptor for malware images[J]. J. Commun. **39**(11), 44–53 (2018). ((**in Chinese**))
8. Naeem, H., Guo, B., Naeem, M.R., et al.: Identification of malicious code variants based on image visualization[J]. Comput. Elect. Eng. **76**, 225–237 (2019)
9. Mathew, A.B., Kurian, S.: Identification of malicious code variants using SPP-net model and color images[C]. In: 2020 IEEE 15th International Conference on Industrial and Information Systems (ICIIS). IEEE, pp. 581–585 (2020)
10. Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., Rabinovich, A.: Going deeper with convolutions. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 1–9 (2015)
11. Szegedy, C., Ioffe, S., Vanhoucke, V., Alemi, A.A.: Inception-v4, inception-resnet and the impact of residual connections on learning. In: Thirty-first AAAI conference on artificial intelligence (2017). https://doi.org/10.1609/aaai.v31i1.11231
12. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 770–778 (2016)
13. Huang, G., Liu, Z., van der Maaten, L., Weinberger, K.Q.: Densely connected convolutional networks. In: 2017 IEEE conference on computer vision and pattern recognition, CVPR 2017, Honolulu, HI, USA, pages 2261–2269. IEEE Computer Society, 2017 (2017)
14. Real, E., Aggarwal, A., Huang, Y., Le, Q.V.: Regularized evolution for image classifier architecture search. Proc Aaai Conf Artif Intell **33**, 4780–4789 (2019)
15. Radosavovic, I., Prateek Kosaraju, R., Girshick, R., He, K., Dollar, P.: Designing network design spaces. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp 1042810436 (2020)
16. Tan, M., Le, Q.: Efficientnet: rethinking model scaling for convolutional neural networks. In: International Conference on Machine Learning, pp 6105–6114 (2019)
17. Howard, A.G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., Adam, H.: Mobilenets: efficient convolutional neural networks for mobile vision applications. arXiv preprint arXiv:1704.04861 (2017)
18. 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, pages 4510–4520 (2018)
19. 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), pages 116–131 (2018)
20. Zhang, J., Lin, M., Pan, Y., Zeshui, Xu.: CRFTL: cache reallocation-based page-level flash translation layer for smartphones. IEEE Trans. Consum. Electron. **69**(3), 671–679 (2023)
21. Chen, Y., Lin, M., He, Z., Polat, K., Alhudhaif, A., Alenezi, F.: Consistency-and dependence-guided knowledge distillation for object detection in remote sensing. Expert Syst. Appl. **229**, 120519 (2023)
22. Xiuqin, Xu., Lin, M., Luo, X., Zeshui, Xu.: HRST-LR: a hessian regularization spatio-temporal low rank algorithm for traffic data imputation. IEEE Trans. Intell. Transp. Syst. **24**(10), 11001–11017 (2023)
23. Pan, Z., Zhuang, B., He, H., Liu, J., Cai, J.: Less is more: pay less attention in vision transformers. In: AAAI (2022)
24. Chen, H., Lin, M., Liu, J., Yang, H., Zhang, C., Zeshui, Xu.: NT-DPTC: a non-negative temporal dimension preserved tensor completion model for missing traffic data imputation. Inf. Sci. **653**, 119797 (2024)
25. Pan, Z., Cai, J., Zhuang, B.: Fast vision transformers with hilo attention. In: Advances in Neural Information Processing Systems (NeurIPS) (2022)
26. Marin, D., Rick Chang J.-H., Ranjan, A., Prabhu, A., Rastegari, M. Tuzel, O.: Token pooling in vision transformers. arXiv preprint arXiv:2110.03860 (2021)
27. Anasosalu Vasu, P. K., Gabriel, J., Zhu, J., Tuzel, O., Ranjan A.: An improved one millisecond mobile backbone. arXiv preprint arXiv:2206.04040 (2022)
28. Wang, S., Li, B.Z., Khabsa, M., Fang, H., Ma, H.: Linformer: self-attention with linear complexity (2020)
29. Kitaev, N, Kaiser, L., Levskaya, A.: Reformer: the efficient transformer. In: International Conference on Learning Representations (2020)
30. Xiao, T., Singh, M., Mintun, E., Darrell, T., Dollar, P., Girshick, R.B.: Early convolutions help transformers see better. CoRR, abs/2106.14881 (2021)
31. Dai, Z., Liu, H., Le, Q.V., Tan, M.: Coatnet: marrying convolution and attention for all data sizes. Adv. Neural Inform. Process. Syst. **34**, 3965–3977 (2021)
32. Chu, X., Tian, Z., Wang, Y., Zhang, B., Ren, H., Xiaolin, W., Xia, H., Shen, C.: Twins: revisiting the design of spatial attention in vision transformers. arXiv preprint arXiv:2104.13840 (2021)
33. Guo, J., Han, K., Wu, H., Xu, C., Tang, Y., Xu, C., Wang, Y.: Cmt: convolutional neural networks meet vision transformers. arXiv preprint arXiv:2107.06263 (2021)
34. d'Ascoli, S., Touvron, H., Leavitt, M., Morcos, A., Biroli, G., Sagun, L.: Convit: improving vision transformers with soft convolutional inductive biases. In: Proceedings of the 38th International Conference on Machine Learning (ICML) (2021)
35. Haiping, W., Bin, X., Noel, C., Mengchen L., Xiyang, D., Lu, Y., Lei, Z.: Cvt: introducing convolutions to vision transformers (2021)
36. Andrew, H, Mark, S., Grace, C., Liang-Chieh, C., Bo, C., Mingxing, T., Weijun, W., Yukun, Z., Ruoming, P., Vijay, V., et al.: Searching for mobilenetv3. In: Proceedings of the IEEE/CVF International Conference on Computer Vision, pages 1314–1324 (2019)
37. Xiaohan, D., Yuchen, G., Guiguang, D., Jungong, H.: Acnet: strengthening the kernel skeletons for powerful cnn via asymmetric convolution blocks. In: Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV) (2019)
38. Xiaohan, D., Xiangyu, Z., Jungong, H., Guiguang, D.: Diverse branch block: building a convolution as an inception-like unit. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (2021)
39. Ding, X., Zhang, X., Ma, N., et al.: Repvgg: making vgg style convnets great again [C]. In: Proceedings of the IEEE/CVF

conference on computer vision and pattern recognition. IEEE, 13733–13742 (2021)
40. Zeyu, W., Yutong, B., Yuyin, Z., Cihang, X.: Can cnns be more robust than transformers? arXiv preprint arXiv:2206.03452 (2022)
41. Li, Q., Mi, J., Li, W., et al.: CNN-based malware variants detection method for internet of things[J]. IEEE Internet Things J. **8**(23), 16946–16962 (2021)
42. Sudhakar, K.S.: MCFT-CNN: malware classification with fine-tune convolution neural networks using traditional and transfer learning in internet of things[J]. Fut. Gener. Comput. Syst. **125**, 334–351 (2021)
43. Danish, V., Mamoun, A., Sobia, W., et al.: Image-based malware classification using ensemble of CNN architectures (IMCEC)[J]. Comput. Secur. **92**, 101748 (2020)
44. Nataraj, L., Karthikeyan, S., Jacob, G., et al.: Malware images: visualization and automatic classification[C]. In: Proceedings of the 8th international symposium on visualization for cyber security, pp. 1–7 (2011)
45. Cui, Z., Fei, X., Xingjuan, C., et al.: Detection of malicious code variants based on deep learning[J]. IEEE Trans. Industr. Inf. **14**(7), 3187–3196 (2018)
46. Naeem, H., Bing, G., Muhammad-Rashid, N., et al.: Identification of malicious code variants based on image visualization[J]. Comput. Elect. Eng. **76**, 225–237 (2019)
47. Cui, Z., Lei, D., Penghong, W., et al.: Malicious code detection based on CNNs and multi-objective algorithm[J]. J Parall Distrib Comput **12**, 950–958 (2019)
48. Vinayakumar, R., Mamoun, A., Soman, K.-P., et al.: Robust intelligent malware detection using deep learning[J]. IEEE Access **74**, 6717–46738 (2019)
49. Sitalakshmi, V., Alazab, M., Vinayakumar, R.: A hybrid deep learning image-based analysis for effective malware detection[J]. J. Inform. Secur. Appl. **47**, 377–389 (2019)
50. Naeem, H., Bing, G., Farhan, U., et al.: A cross-platform malware variant classification based on image representation[J]. KSII Trans. Internet Inform. Syst. **13**, 3756–3777 (2019)
51. Vinita, V., Muttoo, S.K., Singh, V.B.: Multiclass malware classification via first- and second-order texture statistics[J]. Comput. Secur. **97**, 101895 (2020)
52. Roseline, S.A., Geetha, S., Seifedine, K., et al.: Intelligent vision-based malware detection and classification using deep random forest paradigm[J]. IEEE Access **8**, 206303–206324 (2020)
53. Naeem, H., Farhan, U., Muhammad-Rashid, N., et al.: Malware detection in industrial internet of things based on hybrid image visualization and deep learning model[J]. Ad Hoc Netw. **10**, 5102154 (2020)
54. Danish, V., Alazab, M., Wassan, S., et al.: IMCFN: image-based malware classification using fine-tuned convolutional neural network architecture[J]. Comput. Netw. **17**, 1107138 (2020)
55. Anandhi, V., Vinod, P., Varun-G, M.: Malware visualization and detection using DenseNets[J]. Person. Ubiquit. Comput. (2021). https://doi.org/10.1007/s00779-021-01581-w
56. Moussas, V., Antonios, A.: Malware detection based on code visualization and two-level classification[J]. Information **12**(3), 118 (2021)
57. Sudhakar, K.S.: MCFT-CNN: Malware classification with fine-tune convolution neural networks using traditional and transfer learning in Internet of Things[J]. Fut. Gen. Comput. Syst. **12**, 5334–5351 (2021)
58. Kumar, S., Janet, B.: DTMIC: deep transfer learning for malware image classification [J]. J. Inform. Secur. Appl. **64**, 103063 (2022)
59. He, Z., Lin, M., Zeshui, Xu., et al.: Deconv-transformer (DecT): a histopathological image classification model for breast cancer based on color deconvolution and transformer architecture. Inf. Sci. **608**, 1093–1112 (2022)
60. Zhong, M., Lin, M., He, Z.: Dynamic multi-scale topological representation for enhancing network intrusion detection. Comput. Secur. **135**, 103516 (2023)
61. Wei, X., Mingwei, L.: Information security evaluation of industrial control systems using probabilistic linguistic MCDM method. Comput. Mater. Cont. **77**(1), 199–222 (2023)
62. Yong, C., Mingwei, L.: Linguistic knowledge representation in DPoS consensus scheme for blockchain. Comput. Mater. Cont. **77**(1), 845–866 (2023)