CNN-Based Malware Variants Detection Method for Internet of Things

Qi Li, Jiaxin Mi, Weishi Li, Junfeng Wang, and Mingyu Cheng

***Abstract*—Malware has become one of the most serious security threats to the Internet of Things (IoT). Detection of malware variants can inhibit the spread of malicious code from the traditional network to the IoT, and can also inhibit the spread of malicious code within the IoT, which is of great significance to the security detection and defense of the IoT. Since the terminals and the operating systems of IoT are very different from the traditional network, when malicious code is transferred from the traditional network to the IoT platform, the characteristics of the variants may change significantly. As a result, malicious code variant detection methods for traditional platforms cannot be directly applied to the IoT. In this article, a malware variant detection method for the IoT is proposed. First, we propose a feature representation method based on RGB image for IoT to solve the problem of representation difficulty caused by platform difference, which pays more attention to the assembly code and developer information of the malware. The generated image has richer texture information, which can dig out the deep association between the IoT variants and the original malicious code. Moreover, this article improves the convolutional neural network model by combining the self-attention mechanism and spatial pyramid pooling to solve the problem of large differences in the size of IoT malware. Experimental results show that our method can be used in cross-platform to detect malware variants in the IoT effectively.**

***Index Terms*—Convolutional neural network (CNN), cross-platform, Internet of Things (IoT), malware variant detection, RGB image.**

I. Introduction

THE Internet of Things (IoT) is a new type of network that interconnects a large number of electronic devices with a variety of applications [1], such as smart appliances, smart houses, smart grid, energy management systems, and so on. It is quickly spreading in all environments, becoming more pervasive. According to a recent Gartner report, the number of connected things worldwide is expected to grow to 20.4 billion by 2020 [2]. The IoT industry is expected to grow in terms of revenue from $892 billion in 2018 to $4 trillion by 2025 [3]. At the same time, various embedded softwares, such as temperature monitoring and device control, are also exploding. Actually, there are mainly three types of operating systems running on the IoT, including Windows, Linux, and Android. Accordingly, there are three types of software running on the physical network platform: 1) APK for Android; 2) binary file for windows; and 3) ELF for Linux. Therefore, the IoT presents the characteristics, such as the diversity of operating systems, the complexity of deployment scenarios [4], and the sophistication of applications. These characteristics make IoT increasingly susceptible to exploitable vulnerabilities and threats [5].

However, as the IoT develops rapidly, the security problem of the IoT has become more and more serious [6]. According to the security report in 2019, IoT devices suffer 5200 attacks per month on average. Mirai attack in the last quarter of 2016 was estimated to infect around 2.5 million devices connected to the Internet and launch a Distributed Denial-of-Service (DDoS) attack. After Mirai, Hajime and Reaper are the other big botnet attacks launched against a large number of IoT devices [7]. In summary, malicious codes to IoT devices have already caused hundred million of economic losses to the world, and become the most serious security issue in the process of using the IoT.

A recent report from Symantec showed that 68 kinds of new malicious code families and more than 10 000 malicious variant codes have been detected till 2018 [8], which presented a great challenge for malicious code detection in IoT [9], [10]. To this end, some researchers have introduced malware variants detection, which is also called malware family classification, to defense the increasing malware. Unlike malware detection, malware variant detection focuses on the similarity between the code to be tested and the existing malware code instead of the malware's behavior itself. When attackers use polymorphism, metamorphic technology, and other means to disguise existing malware to new variants, malware variant detection can detect them in time. Further, it can respond and defend quickly according to the source of the variant based on known knowledge. According to statistics, a large amount of malware on the IoT has appeared on the traditional platform before attacking IoT devices. Some IoT operating systems (such as win10 IoT and WinCE) also run executable files with a similar structure to PE files, which provides more opportunities for malicious code variants. For example, Duqu2.0, discovered by Kaspersky, is a typical example from the traditional platform to the IoT

Manuscript received December 19, 2020; revised March 1, 2021; accepted April 9, 2021. Date of publication April 26, 2021; date of current version November 19, 2021. This work was supported by the National Natural Science Foundation of China (U20B2045, U1836103, U20A20161), the National Key Research and Development Program (2018YFB0804503, 2019QY1400), the Project (C18613), and the Basic Research Program of China (2019-JCJQ-ZD-113). *(Corresponding author: Junfeng Wang.)*

Qi Li, Jiaxin Mi, Weishi Li, and Mingyu Cheng are with the Beijing Key Laboratory of Interconnection and Integration, Beijing University of Posts and Telecommunications, Beijing 100876, China (e-mail: liqi2001@bupt.edu.cn; jxmi@bupt.edu.cn; 2014212998@bupt.edu.cn; chengmingyu@bupt.edu.cn).

Junfeng Wang is with the College of Computer Science, Sichuan University, Chengdu 610065, China (e-mail: wangjf@scu.edu.cn).

Digital Object Identifier 10.1109/JIOT.2021.3075694

platform, and its source, Duqu, is working on the ordinary server.

The malware variants detection based on supervised learning has got notable achievements in the traditional platforms. For example, Firdausi *et al.* [11] extracted the malware behavior features and used the shallow machine learning methods for family classification, including K-nearest neighbors (KNNs), Naive Bayes, J48 decision tree, and support vector machine (SVM). There are also many researchers who use methods, such as PCA, ICA, and LDA, to select the feature with better identification in the process of malware feature extraction. However, it still faces many difficulties in the detection of malicious code variants in the IoT. First, unlike the traditional malicious code variants, malicious code variants on IoT are not quite similar in size and structure of the files. There are many kinds of IoT devices [12], involving multiple operating systems (Windows, WinIoT, WinCE, VxWorks, and so on). When the code with the same function runs on different IoT devices, it needs to be recompiled because of the difference in the platform environment. Even for the same kind of platform, using operating systems of different versions also requires the recompilation of the malicious code. For example, if a malicious code of IoT wants to transfer from the Windows platform to the WinCE platform, it needs to be recompiled in the relevant compilation environment even if the code does not need to be changed. Second, the static features of the malicious code of the IoT will change when it is transferred to other platforms. Due to the mentioned characteristics of the IoT platform, attackers need to recompile and modify the malware to obtain variants instead of compression or obfuscation only. This fact makes the detection method based on static features is difficult to achieve satisfactory performances on IoT. In addition, the dynamic analysis method must maintain a large virtual operating environment library and a complex trigger rules library to debug and analyze malicious code for different IoT platforms, which undoubtedly increases the detection cost. And the dynamic analysis usually needs a higher time consuming to meet the demands of real-time detection on IoT. As analyzed above, malware on IoT varies a lot not only in code features but also in file structure and sizes with the operating system and compilation environment, which makes it difficult to find the representative features of IoT malware variant directly. As a result, machine learning methods based on feature extraction, including static features and dynamic features, are not suitable for malware variant detection on various IoT platforms. With the continuous development of deep learning technology, researchers focus on the detection of malicious code variants based on image and achieve good results [13]–[16]. The image-based representation method has been a research hotspot in recent years. The malicious code is essentially composed of binary files, so different forms of the malicious code (Android, Linux, and Windows) can be represented as images. Compared with machine learning methods, deep learning, such as convolutional neural network (CNN), contains a multilayer linear structure with a powerful expression and modeling capabilities for processing complex tasks. Therefore, the complex feature extraction process of IoT malicious codes can be conducted automatically by algorithms instead of relying on expert knowledge, which avoids the influence of man-made factors. Besides, for IoT malware that mutates via anti-detection techniques, deep learning also shows powerful learning and modeling capabilities. This is because image-based malware variant detection methods can represent the similarity of malicious code on IoT more precisely and show excellent performance in dealing with the adversarial technology used by the attacker. However, most variant detection methods are only for a single IoT operating system, and cannot be applied across platforms, which has difficulty in detection of the heterogeneous malicious code data on multitudinous IoT devices.

Above all, the research of IoT multiplatform variant detection methods faced with some main difficulties so that traditional variant detection methods on PC cannot be used directly. First, malicious codes on different platforms need to be recompiled with the operating system and compilation environment, thus have great differences in binary features with codes on other platforms. Therefore, the features we used in traditional methods cannot be transferred from a single platform to a cross-platform directly. Second, there is a large difference in file size of malware for different platforms especially on IoT devices, which causes the invalidation of the existing models.

To solve these problems, we first proposed a feature representation method based on RGB images for complex feature expression of malicious code on various IoT devices. We extracted features and generated images with more abundant texture information, which can solve the issues of information loss and poor compatibility. Then, we proposed a malware variant detection model for IoT combined with a self-attention mechanism and spatial pyramid pooling. This method uses a spatial pyramid pooling layer to receive the files on IoT with different sizes, which avoids the loss of information caused by data processing methods and consequently improves the detection accuracy and detection efficiency. In general, the image representation method ensures that malicious code with different structures can be analyzed by our method; the introduction of string information and assembly information enriches the visualization features, which effectively deals with the malware representation feature differences of traditional visualization methods in the cross-platform process; the attention mechanism can mine the local feature association between the variant and the origin so as to deal with the changes caused by recompilation when malicious code mutates to the different operating system environment. What is more, when malicious code mutates between different devices, for example, from a server variant with sufficient storage space to a sensor node with limited resources, its file size will change significantly, and the SPP structure is introduced to solve this problem. Therefore, the method can not only detect the variant between different platforms on IoT but also detect the variants transferred from the traditional network to IoT.

The remainder of this article is organized as follows. In Section II, this article introduces the related work of traditional malware variants detection and IoT malware variant detection. Then, this article presents the structure of the detection model in Section III, including the self-attention mechanism and spatial pyramid pooling. The visualization method of malware is

described in Section IV. In Section V, this article uses experiments to evaluate the proposed method and concludes works in Section VI.

II. Related Work

A. Traditional Malware Variants Detection Method

Traditional static analysis usually extracts dynamic link library, import table, and export table from PE header, and then uses rules or blacklist to match these features, by which the maliciousness of the file can be proved. Another way is to disassemble the malware and extract the information, such as opcode and register, from the assembly code [17], [18]. For example, Kan *et al.* [19] extracted the opcodes from the malware and classified the malware based on opcodes of the similar function. However, with the advent of obfuscation and anti-detection technology, the effectiveness of static detection methods has been greatly decreased [20].

The dynamic analysis of malware is a technology based on the analysis of malicious behaviors, which needs to actually run the malware in a sandbox and monitor the operation of the program to record execution information [13], [21]. Ye *et al.* [14] proposed a malware detection method based on malicious API sequence mining. This method extracted the dynamic API call sequences by simulating the real runtime environment for the malware and then mined out the pattern of API sequence as features, so as to classify the malware variants.

As the types and quantity of malware characteristics increase steadily, more and more feature selection, optimization, and extraction methods are combined with traditional machine learning methods, such as PCA, ICA, LDA, and RCA [22]. The introduction of these methods can effectively filter the characteristics of the malicious code, and further improve the detection efficiency. However, in the detection of malware variants, attackers will disguise the malware during the process of delivering them by ways, such as shelling, obfuscation, and hiding entry points. Since traditional malware variant detection methods based on feature engineering and shallow machine learning are restricted by the manual feature extraction and expert knowledge, the final results of model efficiency and accuracy are directly affected.

With the development of artificial intelligence technology, detection methods that combine sample visualization and deep learning have received much attention [23]. Compared with methods with traditional features, the image of the malicious code in the visualization method depends on the formats, structure, and size of the source code. That is, the texture information of the image is closely related to the structure of the malicious code and the images corresponding to the malicious code with a similar structure also has an apparent correlation. Compared with machine learning methods, deep learning does not need manual feature extraction, and thus it can save the cost of static feature extraction and expert knowledge. What is more, the visualization method can represent the similarity in malicious codes while resisting anti-detection techniques, such as shelling and obfuscation, which avoid the negative impact of anti-detection technology and can effectively detect malicious code variants. In consequence, several detection methods of malware variants that combine malware visualization and deep learning have been proposed. Nataraj *et al.* [24] proposed a simple and fast method for visualizing malware binary files as grayscale images, and then they used the KNN to classify malware. Shaid and Maarof [25] used transfer learning technology to migrate the Resnet neural network that trained on the ImageNet data set to malware images, which greatly improved the accuracy. Cui *et al.* [26] proposed a variant detection algorithm based on the deep learning method, they first used a BAT algorithm to equalize data and then utilized a CNN-based model to detect the malware variants. Han *et al.* [15] proposed a malware analysis method that uses visualized images and entropy graphs to detect and classify new malware and malware variants. Experimental results showed that the proposed method could effectively classify malware families on Windows PE and was not affected by obfuscation techniques. Vu *et al.* [16] converted malware binaries to color images and classified them into benign or malicious categories by the convolutional transformation network they developed, which conducted the feature representation of malware binary semantics and recognized malware variants automatically. However, these models all fill or truncate the images when processing the images of different sizes for training, resulting in the loss of malicious code structure information.

B. IoT Malware Variants Detection Method

Due to the particular characteristics of the IoT, the traditional PC or Android variant detection methods cannot be directly used on IoT. Thus, researchers began to pay more attention to the malware variants detection on IoT. Lei *et al.* [27] proposed a method that did not use API sequences as model inputs directly, but clustered them on the semantic level. And then, it used the semantic information to classify malware. This method can significantly improve accuracy, but it is susceptible to the effects of shells, confusions, etc. Wei and Qiu [28] used a sandbox to analyze IoT malicious codes and extracted runtime features, and then they built a detection system to detect and classify malware for IoT devices. Hossain *et al.* [29] proposed an IoT authentication system to investigate malicious behaviors of malware, which used distributed digital ledgers. In order to overcome the spread of malware and protect the multilevel signaling privacy on the IoT, Shen *et al.* [30] proposed an intrusion detection system on cloud and fog computing. Wu *et al.* [31] proposed a novel similarity algorithm to classify fake IoT apps. Their method achieved an accuracy of 97.34% in the wild, which can effectively classify and detect the fake app of the IoT.

In addition, the image-based method is also introduced into the IoT malicious code variants detection. This is because when solving the problems in cross-platform detection, the image representation of the malicious code can effectively reflect its difference in the file structure and software form, which is shown as the difference of image texture information. This strong superiority makes the image-based method helpful for malicious code variant detection. And as a result, the

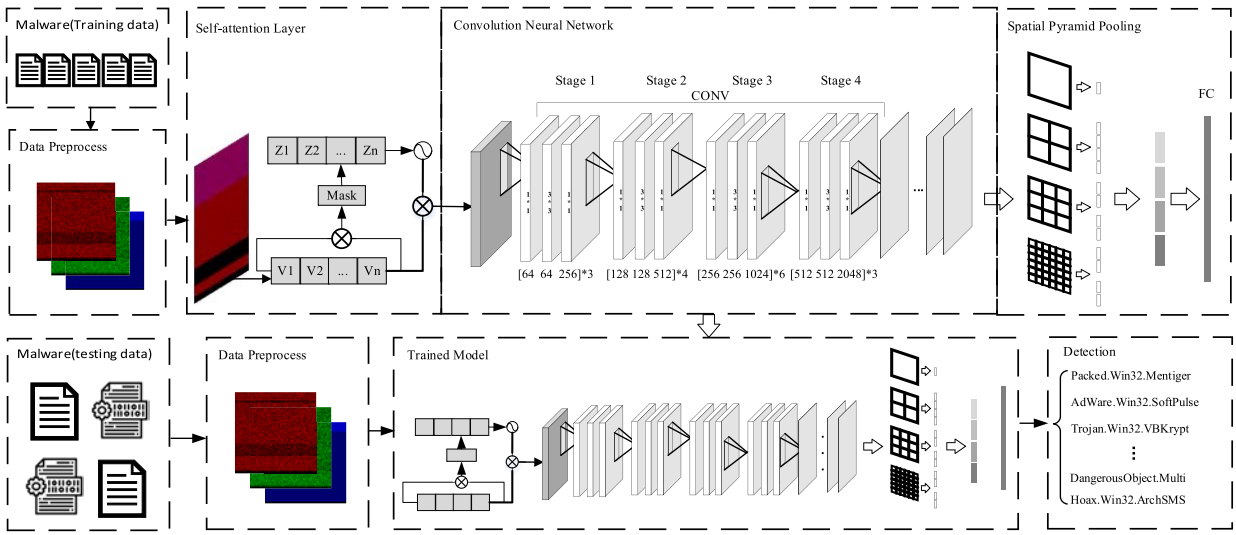


Fig. 1. Architecture of the malware variants detection system for IoT.

visualization method plays an essential role in the variant detection and cross-platform detection of malicious code. Alasmary *et al.* [32] proposed a variant detection method based on the control flow graph, which can classify Android and ELF malware on IoT platform. When Android malware spreads from the traditional platform to the IoT platform, its running environment, including device type and operating system, has not changed, which means that the form of malware remains the same. In other words, the traditional Android variant detection method can be directly applied to that on the IoT platform. However, different from Android and Linux, malware on Windows must be modified and recompiled according to the operating system environment when it spreads from the traditional platform to the IoT platform. Su *et al.* [1] proposed a lightweight malware variants detection method based on images, they converted the binary file malware into grayscale images, and then used deep learning to classify variants and capture DDos attacks on IoT devices. However, in the IoT platform, with the diversification of the platform and the increase of variants method, it is not enough to use binary files individually for variants classification. Other researches show that the introduction of assembly codes can help analyze the similarity of malware variants in terms of distinguishing operating system and compilation environment. For example, Azmoodeh *et al.* [33] proposed a graph generation method based on opcodes in the assembly code, which can resist common obfuscated techniques to a certain extent. However, when malicious code variants are derived in different platforms, the appearance of the assembly code will change greatly. That is, the same function of different variants may be decompiled into completely different assembly statements.

III. Framework Overview

Above all, for binary malware, the variant detection method designed for the traditional platform focuses more on the change of malware due to anti-detection techniques, such as obfuscation and shelling. But for the variant detection method designed for IoT platforms, in addition to the problem on the traditional platform, it is much more needed to consider the change of software form and file structure caused by platform transformation. Therefore, the existing variant detection methods designed for the traditional platform cannot be directly applied to the variant detection of the IoT platform. To solve the problems in existing variant detection methods, we propose a novel malware visualization method and a cross-platform variants detection method based on CNNs to detect and classify malware variants for IoT. The proposed malware variant detection system for IoT is shown in Fig. 1. The training phase mainly includes the data preprocessing module, self-attention module, CNN module, and spatial pyramid pooling module. In the data preprocessing module, we disassembled the malware and transferred them to RGB images. The self-attention module made full use of the context learning ability in the attention mechanism to integrate the whole pixel of the image during the training process, that is, it allows each pixel to attend to all pixels in the image. The CNN module utilizes multiple convolutional layers to extract the high-dimensional features of images, and then the extracted feature map will be input to the spatial pyramid module. The spatial pyramid module is used to transform inputs or feature maps with inconsistent dimensions into the same dimensions. Finally, the optimizer and loss function are selected and set at the end of the training phase. In the online detection phase, the system uses the trained model to predict the variation source of unknown malware on IoT. The symbols used in this paper are defined in Table I.

A. Data Preprocessing Module

First, we used an image to represent malicious code uniformly. Malware variant detection in IoT requires more comprehensive features and consideration of complex changes of compilation environment, which is the new challenge on IoT compared with traditional malware variant detection. In the data preprocessing module, we disassembled the malware with IDA Pro, and then extracted the static features of the malware.

TABLE I
DEFINITION OF SYMBOLS

Symbol	Definition
Q	Image pixel matrix
d_k	Vector dimension of embedding
X	Input matrix of image
ω	Weight matrix
b	Bias matrix
$v(i,j)$	Value of feature map at position (i, j)
W	Wight of image or feature map
H	Height of image or feature map
w	Wight of convolution kernel
h	Height of convolution kernel
p	Padding of image
S_{stride}/S_{window}	Size of stride/window
s_ω/s_b	Cumulative gradient momentum
a	the size of feature map after convolution.
n	the size of feature map after pooling.
α	Learning rate
β	Gradient index
ε	Smoothing index, prevents zero denominator from zero

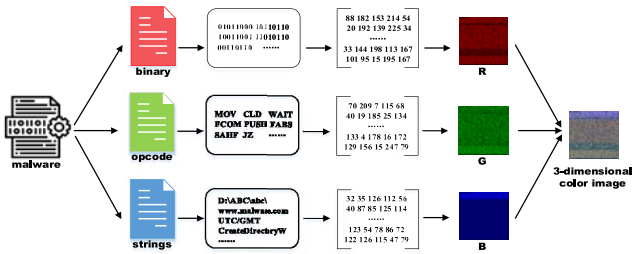


Fig. 2. Illustration of malware imagination.

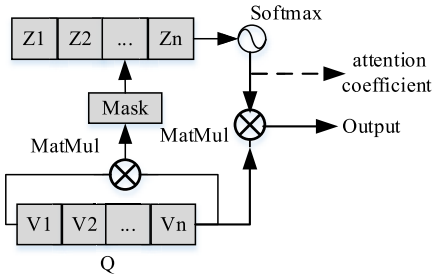


Fig. 3. Structure of self-attention.

Finally, we generated RGB images. The process is shown in Fig. 2. We will describe the method in detail in Section IV.

B. Self-Attention Module

The feature of malware will change significantly during the process of mutation. For example, the malicious code needs recompilation and will cause an image change when mutates to a new operating system environment. This is another important issue of this article. In a malware or other executable file, there is a calling relationship between different code segments. However, the calling relationship presented more obvious in the assembly code than in binary data. In the traditional CNN, the convolution unit only pays attention to the area near the convolution kernel at a time. In spite of the deepening of convolutional layers and the expansion of the receptive field, the convolution unit can only calculate values in a limited area. Therefore, using traditional convolution methods in IoT malware variant detection will ignore the impact of the global code on the target code segment.

To solve the above problems, we introduced a self-attention mechanism [34] in the CNN. The self-attention mechanism is used to calculate the weights between the target pixel and the global pixels, which represents the weights between the current code segment and global code. In this way, the mechanism can remain the logic and correlation among the codes, which is significant to the representation of IoT malware. The structure of self-attention is shown in Fig. 3.

In self-attention, we first calculated the dot product of the malware image pixel matrix Q and its transposed matrix Q^T to get the covariance matrix. The value of the covariance matrix can reflect the relationship between current code segments and global codes. To prevent the value of the covariance matrix from growing too large in magnitude, we scaled the result of dot product by $(1/\sqrt{d_k})$, where d_k represents a vector dimension of embedding. Then, we used the SoftMax function to normalize the result to a probability with the range from 0 to 1, which is the self-attention coefficient we need. Finally, a weighted image matrix can be obtained by calculating the dot product of the weight coefficient and the original image pixel matrix. The expression is shown as follows:

$$\text{Attention}(Q) = \text{softmax}\left(\frac{QQ^T}{\sqrt{d_k}}\right)Q. \tag{1}$$

In the experiment, we set the attention layer before the CNN, and performed self-attention processing on the image before it was input to the convolutional layer.

TABLE II
STRUCTURE OF CNN

Layer Position	Layer Details
Last layer	Attention layer
Stage1	Conv layer $\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix} \times 3$
Stage2	Conv layer $\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix} \times 4$
Stage3	Conv layer $\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix} \times 6$
Stage4	Conv layer $\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix} \times 3$
Next layer	Spatial Pyramid Pooling (1*1, 2*2, 3*3, 6*6)
	FC, SoftMax

C. Convolutional Neural Network Module

After the feature representation of the malicious code, the choice of the model is quite important. In this article, we chose CNN to solve this problem. The main work of CNNs is to abstract and extract the high-dimensional features of images by using multiple convolutional layers. In this article, the CNN module mainly includes two layers: 1) convolutional layer and 2) activation layer.

Convolutional layer carries on the operations for the input images, such as feature extraction, feature mapping, weight sharing, and local connection. The convolution layer can reduce the size of images and further reduce the computational cost of subsequent operations. The formulation of the convolution operation is shown as follows:

$$v(i,j) = (X * \omega)(i,j) + b = \sum_{k=1}^{n} (X_k * \omega_k)(i,j) + b) \quad (2)$$

where n is the number of input matrices, X_k is the kth input matrix, and ω_k is the kth subconvolution kernel matrix of the convolution kernel.

Activation layer performs a nonlinear mapping on the output of the convolution layer. The formulation of the ReLU activation function is shown as follows:

$$\mathrm{ReLU}(x) = \begin{cases} x, & \text{if } x > 0 \\ 0, & \text{if } x \le 0. \end{cases} \quad (3)$$

The CNN module used in this article consists of four stages, and the detail is shown in Table II.

Stage 1: The first stage consists of three sets of $1*1, 3*3, 1*1$ convolution blocks. Each convolution block contains three different sizes of convolution kernels, whose numbers are 64, 64, and 256. The input is an image processed by the attention layer, and the output is a feature map.

Stage 2: The second stage consists of four groups of $1*1, 3*3, 1*1$ convolution blocks. Each convolution block contains three different sizes of convolution kernels, whose numbers are 128, 128, and 512. The input is the feature map processed by the first stage, and the output is a new feature map.

Stage 3: The third stage consists of six groups of $1*1$, $3*3, 1*1$ convolution blocks. Each convolution block contains three different sizes of convolution kernels, whose numbers are 256, 256, and 1024, respectively.

Stage 4: The last stage consists of three groups of $1*1$, $3*3$, $1*1$ convolution blocks, and each convolution block contains three different sizes of convolution kernels, whose numbers are 512, 512, and 2048, respectively.

The feature map processed by the CNN will be input into the spatial pyramid pooling layer.

D. Structure of Spatial Pyramid Pooling Module

So far, we have visualized the malware as RGB images and introduced the CNN with an attention mechanism for image recognition and variants detection. However, there are still several problems when detecting the IoT variants with different file sizes. Once the malware has been visualized as RGB images, the traditional neural-network-based methods for image recognition, such as CNN, can be introduced in variants detection. These traditional neural-network-based approaches, however, have great limitations when detecting the IoT variants with different file sizes and structures. Most neural networks require input data, including training data and test data, to have a fixed size. For example, LeNet requires the size of input images to be $32*32$, and ImageNet requires the size of input images to be $224*224$. In practice, the file size of the malware on different platforms is quite different because of the diversity of IoT devices, and it is difficult to define a general image size for training. Some researchers transfer the malware image into a fixed size by inserting supplementary characters or cutting the image. However, the scaling method would lead to the destruction of the code structure and the cutting method may cause the loss of information.

To solve the above problems, we introduced the pyramid pooling layer in SPP-Net into our model. SPP-Net is a network structure that can receive different image sizes and output a fixed dimensional vector. With the theory of SPP-Net, malware images of different sizes can be directly inputted during the training phase and detection phase, which can prevent the problem of overfitting effectively.

In the traditional neural network model, the convolutional kernel has no limitation of the image size. When the input image size is $W*H$, the convolution kernel size is $w*h$, the stride size is S, and the padding is p, the size of the feature map can be calculated as follows:

$$W^l = \frac{\left(W^{l-1} - w + 2*p\right)}{S_{\text{stride}}} + 1 \quad (4)$$

$$H^l = \frac{\left(H^{l-1} - h + 2*p\right)}{S_{\text{stride}}} + 1. \quad (5)$$

However, because of the limitation of the fully connected layer after convolution, the size of the feature map must match

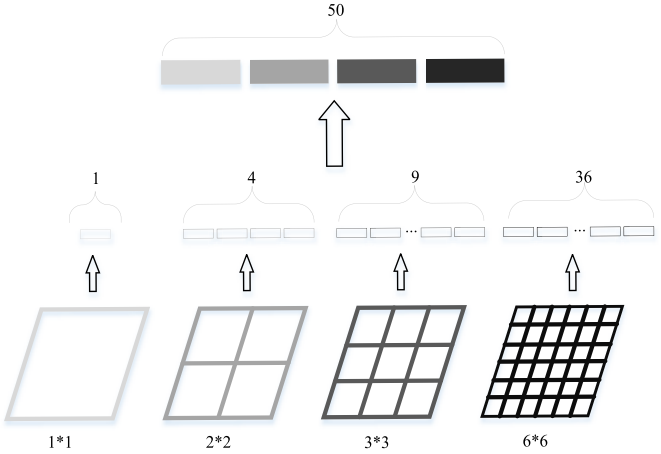


Fig. 4. Structure of the SPP layer.

the number of neurons in the fully connected layer, or vectors of different sizes cannot be transferred to the next layer. The principle of spatial pyramid pooling is to use different pooling layers to process the feature map in the sequence, and then splice these processing results into vectors with the same dimension as the number of neurons in the full connection layer, so that the output dimension can be consistent without scaling the image. In the model we proposed, the spatial pyramid pooling module is set after the CNN module, which includes a spatial pyramid pooling layer and a fully connected layer. The pyramid pooling layer structure used in this article is shown in Fig. 4.

In the first layer, we utilize a $1 * 1$ max pooling layer to obtain a 1-D output; in the second layer, the feature map is divided into four blocks, and each block is pooled to obtain a 4-D output; the third layer is a $3 * 3$ pooling layer, so the feature map is divided into nine blocks and the max pooling is performed to obtain a 9-D output. Similarly, the fourth layer can obtain 36-D data. Finally, the data obtained from each pooling layer are patched to obtain 50-D features and input to the fully connected layer. In the spatial pyramid pooling structure, when the pyramid parameter of one layer is n, the pooling parameter of the layer is shown as follows:

$$S_{\text{window}} = \text{ceil}\left\lceil \frac{a}{n} \right\rceil \tag{6}$$

$$S_{\text{stride}} = \text{floor}\left\lfloor \frac{a}{n} \right\rfloor \tag{7}$$

where $a * a$ is the size of feature map after convolution.

As known to all, the CNN-based variants detection method needs a lot of samples, while in the IoT platform, we cannot collect enough malware in the same family for training. From the above description, we can see that by using SPP, training samples with different sizes can be input into CNN without information loss. That is, we can use the malware in the traditional platform to train the model for IoT malware variants detection, which makes full use of more samples to improve the detection accuracy.

E. Training Phase

In the training phase, the preprocessed malware image is input into the model, and the probabilities of the categories can be obtained by the model. And then, we use the loss function to calculate the errors between output and the ground truth. Finally, the weights of the model are updated by back-propagating the errors. We use the loss function of the model as the objective function for model optimization. The goal of the optimization task is to find an optimal value to reduce the error of loss function.

In our work, we use the RMSprop optimizer to minimize the cross-entropy loss function.

Cross-Entropy Loss Function:

$$L = -\sum_{i=1}^{N} y_i \log \widehat{y_i} + (1 - y_i)\log(1 - \widehat{y_i}). \tag{8}$$

RMSProp Optimizer:

$$s_\omega = \beta s_\omega + (1 - \beta)\omega^2 \tag{9}$$

$$s_b = \beta s_b + (1 - \beta)b^2 \tag{10}$$

$$\omega = \omega - \alpha \frac{\omega}{\sqrt{s_\omega} + \varepsilon} \tag{11}$$

$$b = b - \alpha \frac{b}{\sqrt{s_b} + \varepsilon} \tag{12}$$

where α is the learning rate, and β is an index of gradient accumulation.

The steps of the core process in this article are shown in Algorithm 1.

Algorithm 1 Core Algorithm

Input: $Dataset = \{M_1, M_2, \ldots, M_k\}$
Output: Trained model M
1. Begin
2. For i=1 to k //k is the number of malwares in dataset
3. {//Q_i is the malware pixel matrix of sample M_i
4. $Q_i = \{B_i, A_i, S_i\}_{\text{r,g,b}}$
5. }
6. While t < maxIteration do:
7. for each image $q \epsilon Q$ in batchset:
8. {
9. Attention$(q) \leftarrow softmax\left(\frac{qq^T}{\sqrt{d_k}}\right)$q
10. $q_{t+1}(i,j) \leftarrow \text{Conv}q_{t+1}(i,j)$
11. $q_t(i,j) \leftarrow q_{t+1}(i,j)$
12. $q_{t+1}(i,j) \leftarrow \text{pooling}(q_{t+1}(i,j))$
13. $q_t(i,j) \leftarrow q_{t+1}(i,j)$
14. $q_t^*(\text{i,j}) \leftarrow \text{SPP}(q_t(i,j))$
15. $\widehat{y_i} \leftarrow \text{Dense}(q_t^*(\text{i,j}))$
16. }
17. $L \leftarrow -\sum_{i=1}^{N} y_i log\widehat{y_i} + (1 - y_i)\log(1 - \widehat{y_i})$
18. $X_{k+1}, W_{k+1} \leftarrow update(X_k, W_k)$
19. $accuracy^{t+1} \leftarrow trainModel(batchset)$
20. $t = t + 1$
21. End while
22. Return M

TABLE III
SETTING OF IMAGE SIZE

File size	width	File size	width
<10kb	32	100kb-200kb	384
10kb-30kb	64	200kb-500kb	512
30kb-60kb	128	500kb-1000kb	768
60kb-100kb	256	>1000kb	1024

F. Detection Phase

The trained model is finally used for the detection of testing samples. In the detection phase, the detection data set is preprocessed as the training data set to extract the features of the malware, and an RGB image is generated to input to the trained detection model to judge the results.

IV. Visualization Method for IoT Malware

In this section, we proposed a method to convert IoT malware into RGB images. Our method combines binary file features, assembly code features, and developer information together to generate RGB images, which can involve information from different perspectives and reflect more spatial characteristics of the malicious code than utilizing information from these perspectives individually. In that way, the generated images will have richer texture information and can better reflect the correlation between samples in IoT.

A. Binary Malware Visualization

In this article, we adopt the method in [26], which can directly retain the information reflected by the binary code. The binary bitstream of malware can be divided into groups of 8 bits, each byte can be seen as a pixel. The value of pixels can be calculated as follows:

$$P_{\text{bin}} = b_7 * 2^7 + b_6 * 2^6 + b_5 * 2^5 + b_4 * 2^4 + b_3 * 2^3 + b_2 * 2^2 + b_1 * 2^1 + b_0 * 2^0 \quad (13)$$

$$P_{\text{hex}} = h_1 * 16^1 + h_0 * 16^0 \quad (14)$$

where $(b_7, b_6, b_5, b_4, b_3, b_2, b_1, b_0)$ is a binary sequence, and (h_1, h_0) is a hexadecimal sequence.

Due to the obvious different file sizes from different malware families or different IoT platforms, it is unreasonable to generate them into images of the same size. Therefore, we applied the method used in [24] to define the width of the image (see Table III).

As shown in Fig. 5, due to the difference in compiling environment and operating system, the size and texture of IoT malware variant images have changed significantly, which affects the detection effect of the model. However, when we enlarge the image textures, we can find that the malware variants still retain a similar texture. That is, although the overall similarity of variant malware images changes, the local similarity still exists. On the one hand, we need to supply relevant features into the binary image to improve the robustness of the detection algorithm. On the other hand, to better take advantage of this local similarity on IoT malware variants, we have implemented a detection model to extract the local correlation in images (Section III).

B. Assemble Code Visualization

The process of program development and compilation depends on various factors, such as machine instructions, operating systems, and compilation environments. Thus, there will be large differences between the same high-level language compiled in different operating systems. The analysis

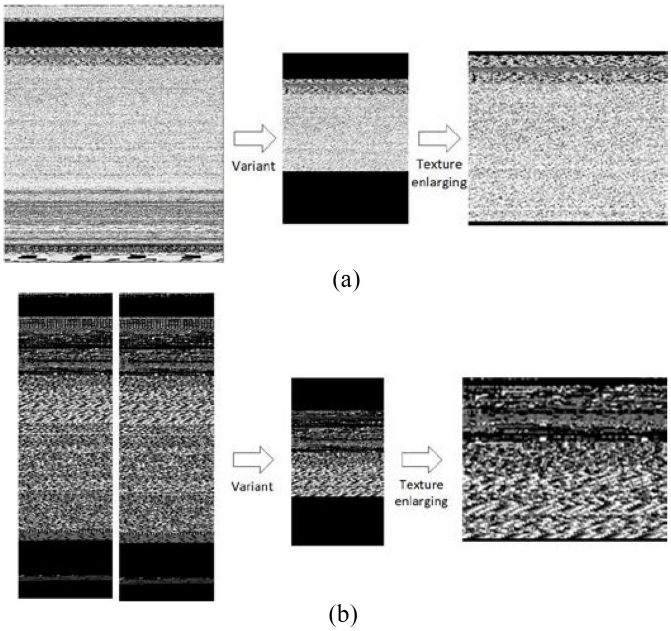


Fig. 5. Variants images in different platforms. (a) Trojan.Win32.Vucha and WinCE/InfoJack. (b) Trojan.Win32.VBKrypt and WinCE/Mepos.

of assembly language cannot only show the function of malware, but also reflect the compilation environment used by the developer. Fig. 6 shows the binary file and assembly file of the malicious code on Windows PC, as well as the assembly code of the WinCE malicious code after variant. We can find that there is a state switch instruction in WinCE malware, but the opcode sequence after the state switch can be matched in the original malware. In the process of extracting opcode, we have ignored the instructions such as state switch. The process of opcode extraction is shown as follows.

1) Use relative virtual address (RVA) to locate the assembly instruction corresponding to the binary code and extract the opcode sequence. Both hex files and asm files are marked with relative virtual addresses. As is shown in Fig. 6, the RVA 401000 on the left indicates the RVA of the instruction data in memory, that is, the offset address relative to the program's initial address in memory (Image Base) when loading the operating system loader. "Push ebp" on the right is the related readable assembly instruction. An assembly instruction consists of two parts: a) opcode and b) operand. The opcode indicates the operation to be performed by the instruction, such as MOV for moving data. The operand indicates the information used by the instruction, including the immediate type, register, and memory address. Because opcode can reflect the behavior of the program, the malware from the same family will have similar opcode sequences to some extent. Therefore, we extract opcode in the corresponding position and fill the remaining positions with 0.
2) Encode opcode according to its function. The functions of opcodes can be divided into six categories: a) data transfer instructions (MOV, PUSH, MVN, etc.); b) arithmetic operations instructions (ADD, INC, SUB, etc.); c) logic operations and shift instructions (AND, SAL, etc.);

TABLE IV
ILLUSTRATION OF ENCODED OPCODE

RVA	00401000	00401001	00401002	00401003	00401004	00401005	00401006
opcode	push	null	null	sub	null	null	cmp
encoded	4	0	0	42	0	0	46
RVA	00401007	00401008	00401009	0040100A	0040100B	0040100C	…
opcode	null	null	null	jz	null	null	…
encoded	0	0	0	106	0	0	…

RVA	Hex View
401000	55 8B EC 83 EC 5C 83 7D 0C 0F 74 2B 83 7D 0C 46
401010	8B 45 14 75 0D 83 48 18 10 8B 0D 04 24 7A 00 89
401020	48 04 50 FF 75 10 FF 75 0C FF 75 08 FF 15 54 72
…	…

(a)

RVA	opcode	register
401000	push	ebp
401001	mov	ebp, esp
401003	sub	esp, 5Ch
401006	cmp	[ebp+Msg], 0Fh
40100A	jz	short loc_401037
40100C	cmp	[ebp+Msg], 46h
…	…	

(b)

RVA	opcode	register
01463C	LDR	R0,=Label+1
014640	BX	R0
014644	PUSH	R4, LR
014648	MOV	R2,R1
01464C	SUB	esp, 5Ch
014650	CMP	R0,#0
…	…	

(c)

Fig. 6. Illustration of RVA to locate the assembly instruction. (a) Binary file of malware. (b) Assembly code of the original malware. (c) Assembly code of variant on the WinCE platform.

RVA	Hex View	Dec View
401000	55 8B EC 83 EC 5C 83 7D	85 139 236 131 236 92 131 125
	0C 0F 74 2B 83 7D 0C 46	12 15 116 43 131 125 12 70
401010	8B 45 14 75 0D 83 48 18	139 69 20 117 13 131 72 24
	10 8B 0D 04 24 7A 00 89	16 139 13 4 36 122 0 137
401020	48 04 50 FF 75 10 FF 75	72 4 80 255 117 16 255 117
	53 6C 65 65 70 15 54 72	83 108 101 101 112 21 84 114
…	…	…

RVA	Encoded View (ChannelG)	Encoded View (ChannelG)
401000	85 0 0 0 0 92 0 125	U \ }
	0 0 116 43 0 125 0 70	t + } F
401010	0 69 0 117 0 0 72 0	E u H
	0 0 0 0 36 122 0 0	$ z
401020	72 0 80 0 117 0 255 117	H P u 255 u
	83 108 101 101 112 0 84 0	S l e e p T
…	…	…

Fig. 7. Illustration of characters encoding.

d) string operation instructions (MOVS, CMPS, etc.); e) control branch instructions (JMP, CALL, RET, etc.); and f) processor control instructions (NOP, etc.).

Although the opcodes in the same category have similar functions, we use similar values instead of the same values to encode opcode from the same category. Meanwhile, in order to distinguish the different opcodes, even for the same type of opcodes, we still encode them differently since the types and lengths of operands they process are not the same. For example, we distinguish MOV, MOVQ, and MOVD as three types of opcodes, though they are all used for the general data transfer function. In the experiment, we collected a total of 247 different opcodes with a coding range of 1–247. Because the coding range proposed in this article is smaller than the gray-level pixel value range 0–255, the coding result of opcode can be directly used as the pixel value for generating images. The operator extracted after step 1 is shown in the second line in Table IV, the encoded result corresponds to the RVA as shown in the third line.

C. Developer Feature Visualization

In general, IoT malware inevitably contains the information of compile environment, time, and path used by developers, which present as visible strings. The visible characters in the malware text section and data section can reflect the functions of malware, such as connecting to the server and hidden path. The information mentioned above is often presented in the form of the strings of malware. Although these strings are included in the binary code, if we do not single them out, these strings' semantic information will be lost, which is ignored by the traditional visualization method. In Table V, we analyzed the developer information of the malicious code and its variant on IoT platforms. We can find when a malicious code is transferred from the personal computer to the IoT device, its operating system will change from Windows 7 to WinCE, but some characters still remained, which provides an important basis for IoT variants detection.

TABLE V
CATEGORIES OF EXTRACTED INFORMATION

Type	Windows	WinCE
URL	http://nsis. sf.net/	http://nsis. sf.net/
API	GetCurrentProcess	GetCurrentProcess
Timestamp	1361665851	1408966994
Path	C:\xyc\document\msv.zip	\windows\xyc\msv.zip
Others	$z.sD..i.....|	$z.sD..i.....|

Visible strings are usually represented in the file as letters (A–Z, a–z), numbers (0–9), and symbols (!, #, and %), whose ASCII value is between 32 and 126. While using the convolution kernel in a CNN to extract local features, the invisible string will cause noise and may have a bad effect on visible strings. Therefore, as is shown in Fig. 7, we encode the visible characters as 32–126 and the invisible characters as 0.

TABLE VI
SOURCE OF THE DATA SETS

Source ID	Project	Description
1	Malware capture facility project [35]	a project captures malware and normal data
2	VX Heavens [36]	a set of known malware files
3	Virusshare [37]	a repository of malware samples on Windows OS

TABLE VII
DETAILS OF DATASET1

Families	Max size	Min size	Ave size	samples	IoT	Not IoT
Packed.Win32.Mentiger	812KB	157KB	523KB	315	315	0
AdWare.Win32.SoftPulse	327KB	78.8KB	146KB	53	53	0
AdWare.Win32.Amonetize	164KB	69.1KB	108KB	224	224	0
Trojan.Win32.VBKrypt	298KB	32KB	192KB	247	247	0
AdWare.Win32.DealPly	499KB	277KB	311KB	122	122	0
Trojan.Win32.Vucha	317KB	87.7KB	264KB	107	107	0
Trojan.Downloader.JS.Small	478KB	105KB	354KB	143	143	0
AdWare.Win32.OutBrowse	322KB	73.3KB	206KB	879	879	0
AdWare.Win32.Wajagan	581KB	187KB	319KB	89	89	0
AdWare.Win32.Wajam	327KB	100KB	191KB	130	130	0
Trojan.Win32.Yakes	1310KB	1220KB	1300KB	150	150	0
AdWare.Win32.Amonetize	1022KB	920KB	989KB	106	106	0
Trojan.Win32.Patched	327KB	15.4KB	107KB	198	198	0
DangerousObject.Multi	589KB	160KB	431KB	542	542	0
Trojan.Win32.Pakes	524KB	105KB	407KB	302	302	0
Trojan.Win32.Reconyc	122KB	66.6KB	79KB	77	77	0
AdWare.Win32.4Shared	897KB	588KB	712KB	53	53	0
AdWare.Win32.Agent	914KB	573KB	627KB	139	139	0
AdWare.Win32.InstallMonster	2150KB	1620KB	1774KB	110	110	0
Hoax.Win32.ArchSMS	778KB	769KB	774KB	42	42	0
AdWare.Win32.SwiftBrowse	3280KB	2840KB	3070KB	398	398	0
Trojan.Boot.Cidox	1577KB	104KB	622KB	492	492	0
Trojan.Win32.Generic!B	932KB	653KB	715KB	298	298	0
Trojan.Win32.Runner	544KB	318KB	421KB	322	322	0
Adware.Win32.Qjwmonkey	927KB	273KB	492KB	241	241	0

V. EXPERIMENTAL EVALUATION

A. Data Set and Experiment Setting

In this section, we presented the evaluation of the effectiveness of our method. The samples used for the experiment contain most major categories, such as Trojans, worms, and backdoors from 25 families, with different attack scenarios (such as traditional servers, PCs, and IoT devices). The sizes of these samples range from 30 to 5000 kB. Even in the same family, the sample size is greatly different. In the experiment, the testing data are all collected from the IoT platforms, the training data can be split into two parts: 1) Dataset1 and 2) Dataset2. Samples in Dataset1 are all IoT malware, while samples in Dataset2 include malware from both IoT platforms and traditional platforms. The malware samples used in the experiment are mainly collected from open-source data sets, such as samples in the Malware capture facility project, VX Heavens and Virus Share, which are shown in Table VI. These data sets are widely used in works for malware detection. For example, VxHeaven is used for malware variant detection in [15]. Naeem *et al.* [38] used Virus Share and other data sets for malware homology analysis on Windows and Android platforms. Details of samples in different families are shown in Tables VII and VIII.

All experiments were carried out in PC with an Intel Core i5-8400 CPU (2.8 GHz), GPU is TITAN(X) (Pascal). We use python 3.5.4 and Scikit-learn 0.20.0 library to build a machine learning model, Keras 2.1.2 to build a neural network, and use Tensorflow 1.4.1 as a backend computing framework.

In our experiments, the initial learning rate was set to 0.01; the maximum number of iteration times was set to 12 000; the number of epochs was 30. In the SPP layer, the sizes of each layer were set to $1*1$, $2*2$, $3*3$, and $6*6$. In the experiments, the evaluation criteria include accuracy (ACC), precision (P), recall (R), and F1-score

$$\mathrm{ACC} = \left(1 - \frac{\mathrm{sample}_{\mathrm{error}}}{\mathrm{sample}_{\mathrm{sum}}}\right) \times 100\% \tag{15}$$

$$P = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}} \times 100\% \tag{16}$$

TABLE VIII
DETAILS OF DATASET2

Families	Max size	Min size	Ave size	samples	IoT	Not IoT
Packed.Win32.Mentiger	3610KB	157KB	700KB	696	27	669
AdWare.Win32.SoftPulse	786KB	78.8KB	426KB	107	14	93
AdWare.Win32.Amonetize	361KB	69.1KB	448KB	495	31	464
Trojan.Win32.VBKrypt	648KB	32KB	160KB	556	24	532
AdWare.Win32.DealPly	1530KB	277KB	892KB	248	6	242
Trojan.Win32.Vucha	943KB	87.7KB	633KB	248	11	237
Trojan.Downloader.JS.Small	2530KB	105KB	1330KB	324	18	306
AdWare.Win32.OutBrowse	976KB	73.3KB	477KB	2102	40	2062
AdWare.Win32.Wajagan	1320KB	187KB	1021KB	204	15	189
AdWare.Win32.Wajam	1030KB	100KB	420KB	290	12	278
Trojan.Win32.Yakes	1500KB	1220KB	1350KB	328	31	297
AdWare.Win32.Amonetize	1760KB	920KB	1330KB	330	29	301
Trojan.Win32.Patched	550KB	15.4KB	260KB	453	17	436
DangerousObject.Multi	1750KB	160KB	474KB	1109	42	1067
Trojan.Win32.Pakes	1750KB	105KB	649KB	623	16	607
Trojan.Win32.Reconyc	349KB	66.6KB	177KB	182	13	169
AdWare.Win32.4Shared	916KB	588KB	800KB	124	22	102
AdWare.Win32.Agent	3550KB	573KB	687KB	315	12	303
AdWare.Win32.InstallMonster	4170KB	1620KB	3060KB	251	31	220
Hoax.Win32.ArchSMS	788KB	769KB	774KB	105	11	94
AdWare.Win32.SwiftBrowse	5530KB	2840KB	3580KB	843	24	819
Trojan.Boot.Cidox	3550KB	104KB	675KB	1125	51	1074
Trojan.Win32.Generic!B	4170KB	653KB	955KB	617	19	598
Trojan.Win32.Runner	756KB	318KB	451KB	766	21	745
Adware.Win32.Qjwmonkey	3870KB	273KB	892KB	560	9	551

TABLE IX
DETAILS OF VISUALIZATION METHODS IN OTHER PAPERS

Method	Reference	Description
M1	Ref.[26]	rotation, zoom, rescale the image
M2	Ref.[39]	utilize different colors to represent characters
M3	Ref.[15]	extract information entropy
M4	Ref.[40]	multiple channel method with Word2Vec

$$R = \frac{\text{TP}}{\text{TP} + \text{FN}} \times 100\% \tag{17}$$

$$\text{F1-score} = \frac{2 \times P \times R}{P + R \times 100\%}. \tag{18}$$

B. Experimental Results

1) Evaluation of the Malware Visualization Method: In order to prove the effectiveness of the RGB visualization method proposed in this article, we compare it with other visualization methods published in some excellent papers. M1, mentioned in Ref. [26], processed the malware images by rotation, zooming, and other methods to expand the data set; M2, mentioned in Ref. [39], proposed an RGB image generation method, which used different colors to represent characters with different meanings; M3, mentioned in Ref. [15], introduced information entropy on the basis of gray image; and M4, mentioned in Ref. [40], developed a multichannel image method, which used a 256-D vector to represent the characteristics of the assembly code. The details of these methods are shown in Table IX. In order to prove that the proposed visualization method can achieve satisfying performance on detection models, we test its effect on the four most commonly used models in the field of malware variant detection methods, such as LeNet, ResNet, Inception, Xception, and so on. These methods have been implemented in the Keras library.

In the first part of the experiment, we use IoT samples as the training set and test set to verify the effectiveness of the proposed image-based method for malware mutated from the IoT platform. The results are shown in Fig. 8.

From Fig. 8, we can find that when using IoT samples as data set, the visualization method proposed in this article achieves the best detection effect on the four image detection models. Compared with the gray image method in M1 and M3, the RGB image includes more information and

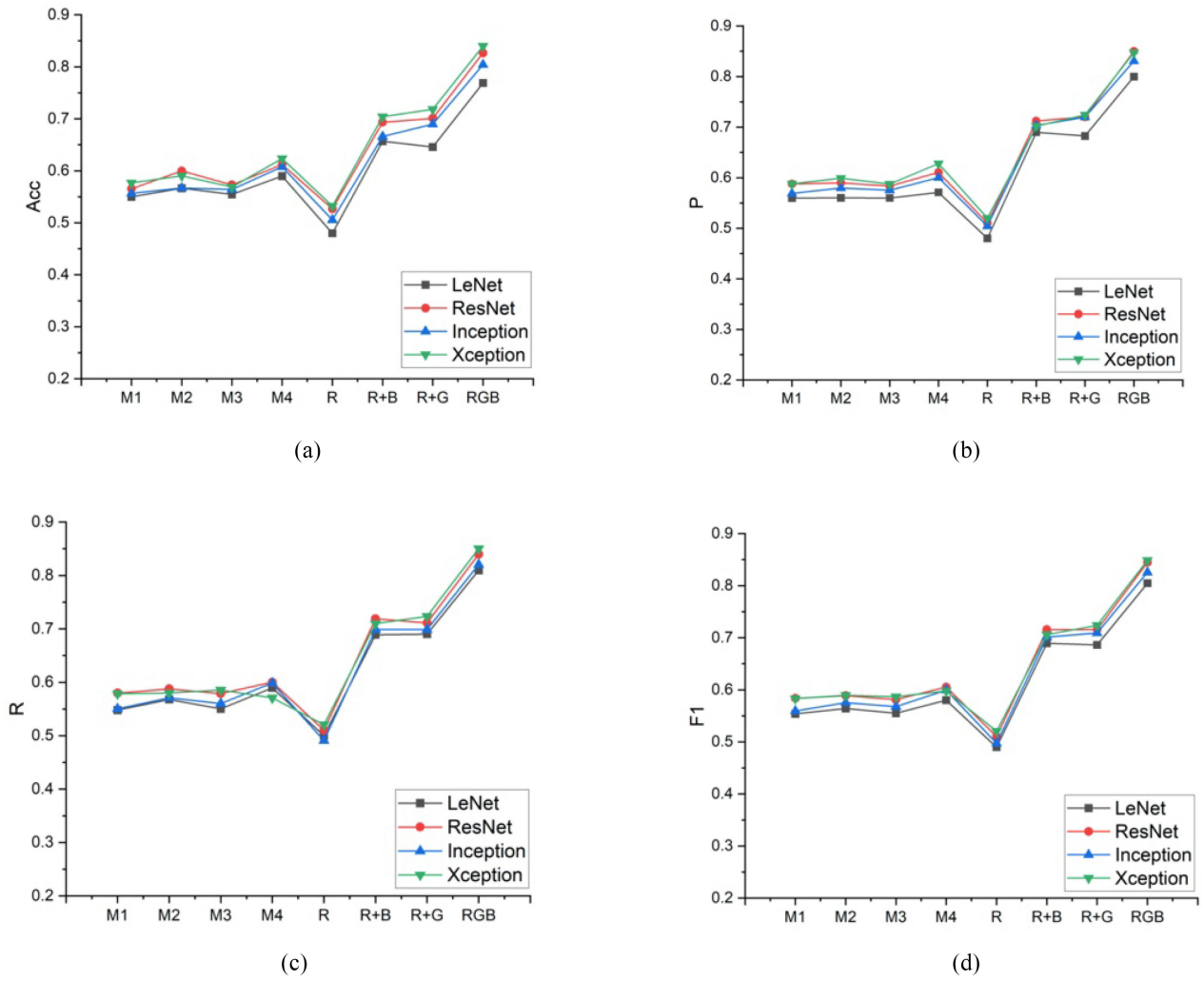


Fig. 8. Performance evaluation of the malware visualization method (both training data and test data are from IoT). (a) Accuracy. (b) Precision. (c) Recall. (d) F1-score.

can enhance the difference of malware characteristics, which effectively avoids the overfitting problem in the training process. Although M2 and M4 also used the RGB image to represent the malware, they simply spliced and represented different features of malware, which ignored the correlations between features. Since the code structure of malware will significantly change after variation due to the diversity of the IoT platforms, it is difficult for the model to converge even though using IoT malware as both the training set and the testing set. Unlike these methods, the visualization method proposed in this article extracts features from three dimensions, including binary code, assemble code, and visible string, and establishes the relationship between features through the RVA in the IoT malicious code. This method selects and represents the local feature of malware much better and thus can achieve excellent detection accuracy specifically for the malware variants on diversified IoT platforms.

Most malware on IoT are mutated from malware on the traditional platform. When a malware variant is transferred from the existing malware family to an IoT platform for the first time, it is very difficult to check it out because of the change of code caused by platform differences. And this is one of the primary problems that we have committed to in IoT malware variant detection. To solve this problem, in the second part of the experiment, we use the malware on the traditional platform as the training set, and then use the malware in IoT as the test set to verify the effect of the model. The experimental results are shown in Fig. 9.

As shown in Fig. 9, compared with training the model with IoT samples individually, introducing the traditional samples into the training data can extend sample space and avoid overfitting, as a result, the detection accuracy of each imagination method has been improved. However, due to the great difference between the IoT variants and their origin in the traditional platform, other imagination methods cannot achieve satisfactory detection results. Our method has considered the internal correlation between the IoT variants and their origin sufficiently, and explored the relationship between malware variants from multiple dimensions, such as the core function, sentence characteristics, and so on, so as to achieve the optimal detection result.

2) Evaluation of the Self-Attention Algorithm: CNN and RNN are widely used in malware variant detection. In this section, we will test whether the self-attention mechanism

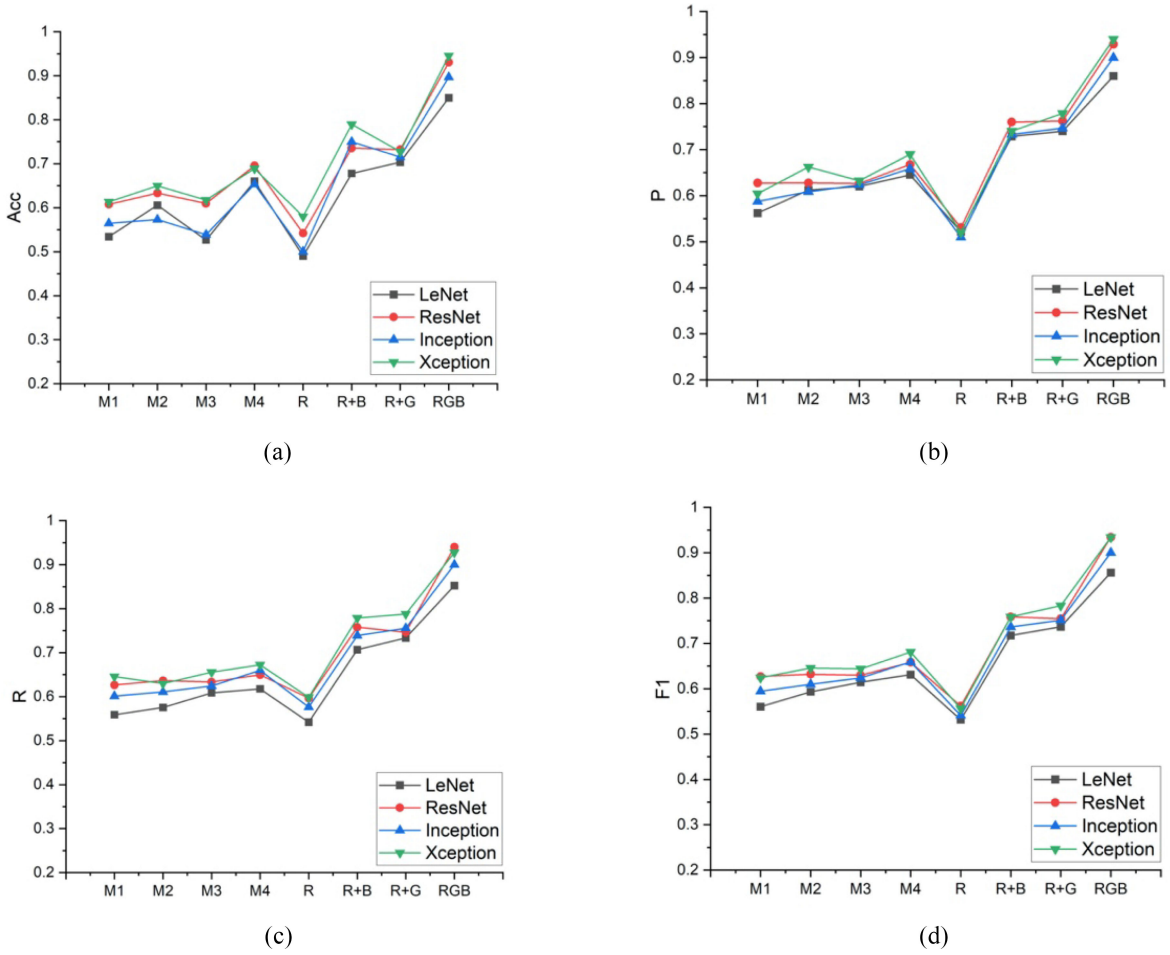


Fig. 9. Performance evaluation of the malware visualization method (training data from the traditional platform, and test data from IoT). (a) Accuracy. (b) Precision. (c) Recall. (d) F1-score.

can improve the detection effect of these two models. In this experiment, we combined Dataset1 and Dataset2 together as the training data.

As is shown in Fig. 10, the ordinate of each point represents the actual family number of the malicious code, and the abscissa represents that the family is judged. In that way, the diagonal of the confusion matrix can indicate the correct prediction result. That is, the fewer discrete points there are, the better detection accuracy it presents.

The self-attention mechanism can consider the global information of the data rather than local features. For malware, the invocation relationship exists in the entire code, not just in the adjacent code segments. Therefore, the accuracy of both CNN and RNN has been significantly improved by introducing the self-attention mechanism. Moreover, since CNN can extract features through the convolutional layer and modify attention direction through the pooling layer, while RNN can only deal with serializable features in a single direction, CNN is more suitable for malware variant detection in IoT.

3) Evaluation of the SPP Algorithm: In the section, on the basis of the CNN model with the self-attention mechanism above, we will continue to evaluate the improvement effect after introducing spatial pyramid pooling to the model.

We can find from Fig. 11, before SPP was introduced, several malware families, such as family 10, family 18, and family 25, had a poor detection accuracy. As shown in Tables VII and VIII, the size of malware from these families has a large span. Taking family 10 as an example, the largest code size in this family is 1000 kB and the smallest is only 80 kB. In this article, our convolutional layer adopted the structure of Resnet50, and when the spatial pyramid pooling layer is not introduced, the input image must be reshaped into the size of 224 $*$ 224 by padding or truncation. For malware whose file size is smaller than the requirement, the zero padding will increase the invalid information in the data; for malware whose file size is larger than the requirement, the truncation will cause the loss of valid information in the data. By introducing SPP, we can ensure the conciseness and completeness of IoT malware information.

4) Comparison With Other Researchers' Methods: To verify the effectiveness of the overall detection system we proposed, we compare our method with seven widely used methods for malware variant detection in well-known papers. In this part, we select methods in [24], [26], [39], and [40]. What is more, considering the differences between the IoT platform and the traditional platform, we evaluated our method from two parts. In the first part, we use the IoT malware as the training set and the test set with the ratio of 8:2 to prove the effectiveness of the proposed method under the condition of sufficient IoT samples. In practical application, most

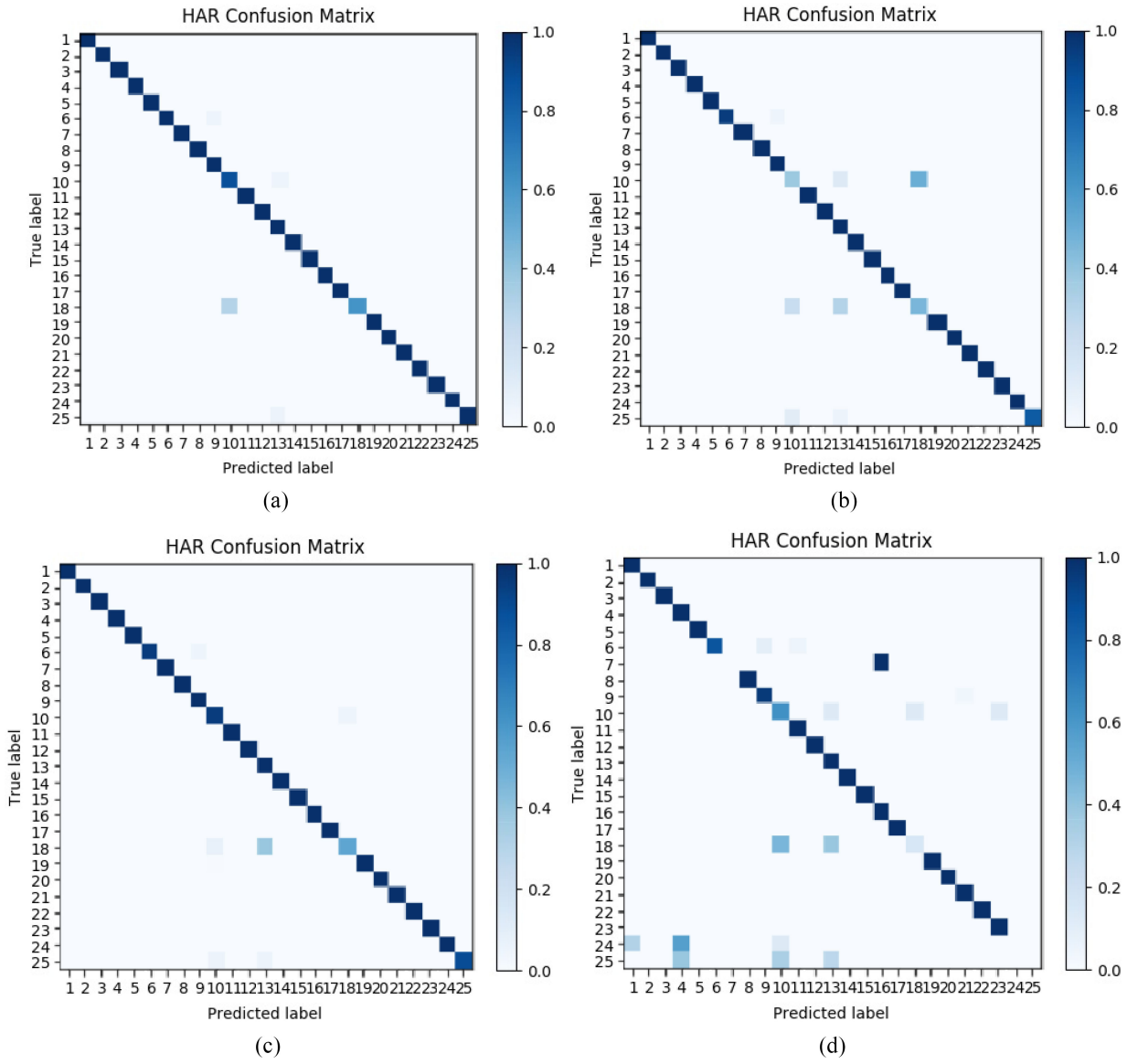


Fig. 10. Effect of self-attention. (a) CNN + self-attention. (b) CNN. (c) RNN + self-attention. (d) RNN.

TABLE X
PROPOSED METHOD COMPARED WITH OTHER METHODS ON DATASET1

Method	Description	Accuracy	Precision	Recall	F1-score
Ref[24]	GIST+KNN+gray	82.55	83.12	82.79	82.95
Ref[24]	GIST+SVM+gray	81.13	81.02	81.41	81.21
Ref[26]	GLCM+KNN+gray	82.15	82.66	82.31	82.48
Ref[26]	GLCM+SVM+gray	83.14	83.43	83.01	83.21
Ref[26]	IDA+BAT+gray	84.22	84.57	84.37	84.47
Ref[39]	RF/SVM/KNN+RGB	87.17	87.88	85.42	86.63
Ref[40]	LeNet5+RGB	91.22	92.75	91.32	92.02
Our method	**Attention+SPP+RGB**	**95.31**	**96.04**	**95.23**	**95.63**

malware in IoT mutates from the malware on a traditional platform. When a certain kind of malware variant that has been developed for a long time in the traditional platform appears in IoT for the first time, it is difficult to check it out directly by existing methods. To solve this problem, our research focuses on how to use existing malware in the traditional platform to detect cross-platform malware variants in IoT. So, in the second part, we use the malware on the traditional platform as the training set and then use the IoT malware as the test set to verify the effectiveness of the proposed method for detecting cross-platform malware variants on the IoT platform. The experimental results are shown in Tables X and XI.

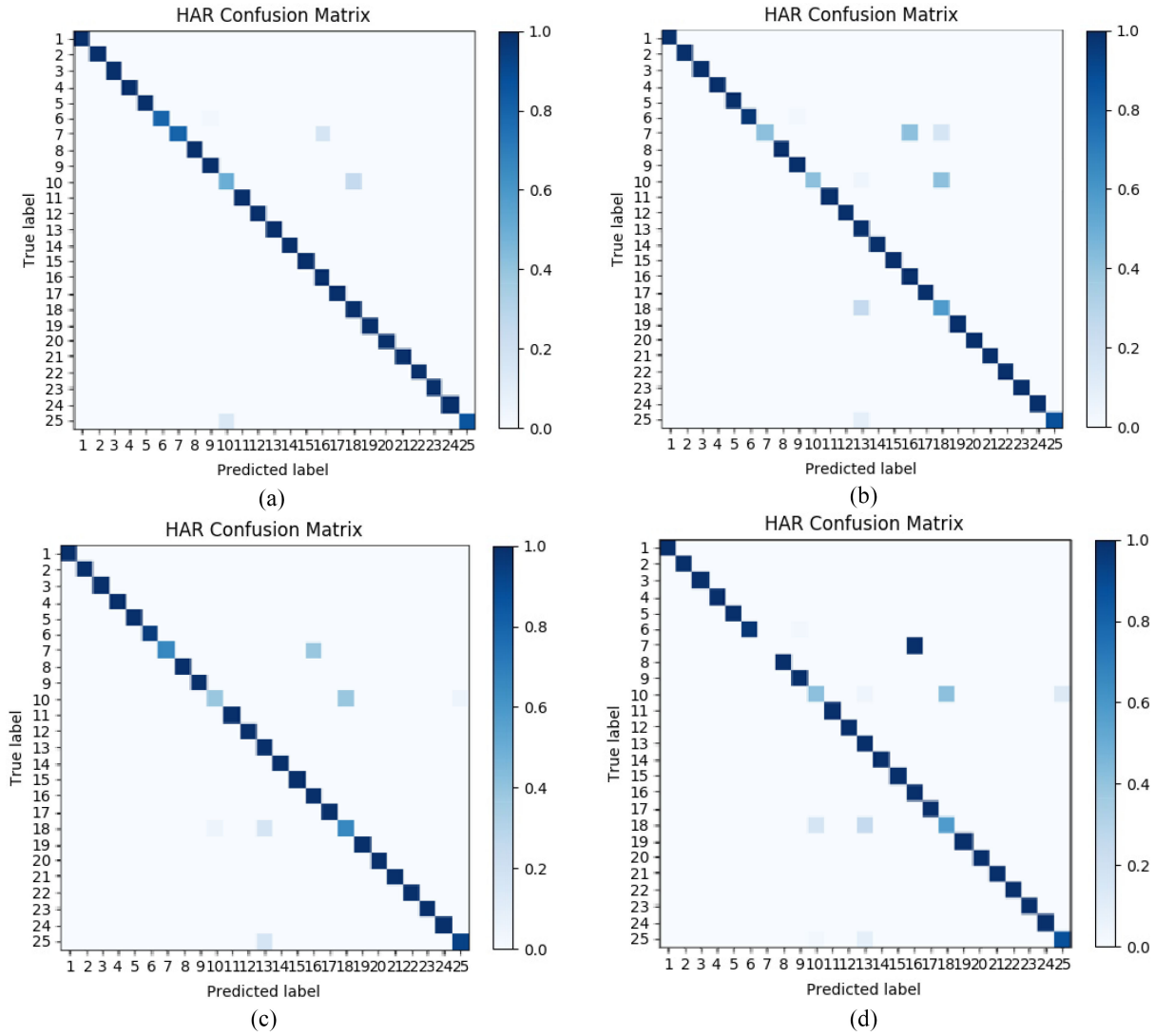


Fig. 11. Effect of the SPP layer. (a) RGB + SPP. (b) RGB. (c) Gray + SPP. (d) Gray.

TABLE XI
PROPOSED METHOD COMPARED WITH OTHER METHODS ON DATASET2

Method	Description	Accuracy	Precision	Recall	F1-score
Ref[24]	GIST+KNN+gray	89.19	89.77	89.62	89.69
Ref[24]	GIST+SVM+gray	89.31	89.02	89.28	89.15
Ref[26]	GLCM+KNN+gray	89.51	89.73	89.31	89.52
Ref[26]	GLCM+SVM+gray	90.24	90.41	90.01	90.16
Ref[26]	IDA+BAT+gray	91.42	91.61	91.55	91.58
Ref[39]	RF/SVM/KNN+RGB	95.22	94.48	94.02	94.25
Ref[40]	LeNet5+RGB	95.57	95.44	95.32	95.38
Our method	**Attention+SPP+RGB**	**98.57**	**98.79**	**98.47**	**98.62**

Table X shows the results of our method compared with other researchers' methods on Dataset1. From Table X, we can find that our method has a higher recall and precision rate than other detection methods. There are three main reasons. First, many researchers ignore the assembly information of IoT malware when they detect malware variants, which can reflect the semantic feature of malware. Compared with their methods, the visualization method we proposed contains more static features that are important for IoT variant detection, which makes the texture of the image clearer. Second, traditional CNN does not consider the global features of malware but only focuses on the relationship around the receptive field. In contrast, the method we use considers the function and logical relationship among malware from the perspective of both visualization method and model design. Finally, the proposed method can input the malware images of different sizes into the

model for training and detection instead of using zero padding or truncation, which avoids the lack of information.

Table XI shows that when using malware on the traditional platform as training data and using malware in IoT as the test data, the detection model proposed in this article can still maintain a high accuracy rate while the accuracy rates of other methods decrease notably. There are two primary reasons. First, the file size and image texture of the IoT malware variants have all changed significantly after the variation from traditional platforms to IoT platforms but the corresponding relationship between the binary code and assembly code of the core functional segment has not changed. Taking this rule into account, our visualization method has constructed a connection between the different features, so the features we use can better represent the characteristics after variation. Second, the model structure proposed in this article takes the differences caused by the variation into consideration, and uses the attention mechanism and the SPP structure to minimize the impact of code differences. In contrast, the methods proposed in other papers have difficulty in achieving good results for data with great differences.

VI. Conclusion

With the rapid development of IoT, it is of great significance to detect the variant of malware in IoT. In order to solve the disadvantages of the existing methods, including complex feature extraction, poor antagonism, detection efficiency, etc. This article proposed an improved malware visualization scheme particularly for malware variants in the IoT. The generated image contains binary information, assembly information, and visible string information of the malware. Then, we used an improved CNN to detect variations of the malware. We compared our method with some state of the art on more than 10 000 samples from 25 malware families, and the experiment results showed that the proposed method achieved the accuracy of 98.57% on IoT malware.

References

[1] J. Su, D. V. Vargas, S. Prasad, D. Sgandurra, Y. Feng, and K. Sakurai, "Lightweight classification of IoT malware based on image recognition," in *Proc. IEEE 42nd Annu. Comput. Softw. Appl. Conf. (COMPSAC)*, Tokyo, Japan, 2018, pp. 664–669.

[2] R. Kandaswamy *et al. Blockchain-Based Transformation*. Accessed: Jun. 5, 2018. [Online]. Available: https://www.gartner.com/en/doc/3869696-blockchain-based-transformation-a-gartner-trend-insightreport/

[3] Gsma. *Safety, Privacy and Security*. Accessed: Jan. 29, 2019. [Online]. Available: https://www.gsma.com/publicpolicy/resources/safetyprivacysecurity-across-mobile-ecosystem/

[4] H. Li, K. Ota, and M. Dong, "Learning IoT in edge: Deep learning for the Internet of Things with edge computing," *IEEE Netw.*, vol. 32, no. 1, pp. 96–101, Jan./Feb. 2018.

[5] M. Ammar, G. Russello, and B. Crispo, "Internet of Things: A survey on the security of IoT frameworks," *J. Inf. Security Appl.*, vol. 38, pp. 8–27, Feb. 2018.

[6] L. Zhou, S. Du, H. Zhu, C. Chen, K. Ota, and M. Dong, "Location privacy in usage-based automotive insurance: Attacks and countermeasures," *IEEE Trans. Inf. Forensics Security*, vol. 14, pp. 196–211, 2019.

[7] Flashpoint. *Mirai Botnet Linked to Dyn DNS DDoS Attacks*. Accessed: Dec. 18, 2018. [Online]. Available: https://www.flashpointintel.com/blog/cybercrime/mirai-botnet-linked-dyn-dns-ddos-attacks/

[8] *Internet Security Threat Report*, Symantec, Tempe, AZ, USA, 2018.

[9] A. Khoshkbarforoushha, R. Ranjan, R. Gaire, E. Abbasnejad, L. Wang, and A. Y. Zomaya "Distribution based workload modelling of continuous queries in clouds," *IEEE Trans. Emerg. Topics Comput.*, vol. 5, no. 1, pp. 120–133, Jan.–Mar. 2017.

[10] P. Li, J. Zhao, Z. Xie, W. Li, and L. Lv, "General central firefly algorithm based on different learning time," *Int. J. Comput. Sci. Math.*, vol. 8, no. 5, pp. 447–456, 2017.

[11] I. Firdausi, C. Lim, A. Erwin, and A. S. Nugroho, "Analysis of machine learning techniques used in behavior-based malware detection," in *Proc. 2nd Int. Conf. Adv. Comput. Control Telecommun. Technol.*, Jakarta, Indonesia, 2010, pp. 201–203, doi: 10.1109/ACT.2010.33.

[12] D. Li, K. Ota, M. Dong, J. Wu, and J. Li, "DeSVig: Decentralized swift vigilance against adversarial attacks in industrial artificial intelligence systems," *IEEE Trans. Ind. Informat.*, vol. 16, no. 5, pp. 3267–3277, May 2020.

[13] A. Saracino, D. Sgandurra, G. Dini, and F. Martinelli, "MADAM: Effective and efficient behavior-based android malware detection and prevention," *IEEE Trans. Dependable Secure Comput.*, vol. 15, no. 1, pp. 83–97, Jan./Feb. 2018.

[14] Y. Ye, L. Chen, S. Hou, W. Hardy, and X. Li, "DeepAM: A heterogeneous deep learning framework for intelligent malware detection," *Knowl. Inf. Syst.*, vol. 54, no. 2, pp. 265–285, 2018.

[15] K. S. Han, J. H. Lim, B. Kang, and E. G. Im, "Malware analysis using visualized images and entropy graphs," *Int. J. Inf. Security*, vol. 14, no. 1, pp. 1–14, 2015.

[16] D.-L. Vu, T.-K. Nguyen, T. V. Nguyen, T. N. Nguyen, F. Massacci, and P. H. Phung, "A convolutional transformation network for malware classification," in *Proc. IEEE 6th NAFOSTED Conf. Inf. Comput. Sci. (NICS)*, Hanoi, Vietnam, 2019, pp. 234–239.

[17] M. Ozsoy, K. N. Khasawneh, and C. Donovick, I. Gorelik, N. Abu-Ghazaleh, and D. Ponomarev, "Hardware-based malware detection using low-level architectural features," *IEEE Trans. Comput.*, vol. 65, no. 11, pp. 3332–3344, Nov. 2016.

[18] M. H. Nguyen, D. L. Nguyen, X. M. Nguyen, and T. T. Quan, "Auto-detection of sophisticated malware using lazy-binding control flow graph and deep learning," *Comput. Security*, vol. 76, pp. 128–155, Jul. 2018.

[19] Z. Kan, H. Wang, G. Xu, Y. Guo, and X. Chen, "Towards light-weight deep learning based malware detection," in *Proc. IEEE 42nd Annu. Comput. Softw. Appl. Conf. (COMPSAC)*, Tokyo, Japan, 2018, pp. 600–609.

[20] A. Moser, C. Kruegel, and E. Kirda, "Limits of static analysis for malware detection," in *Proc. 23rd Annu. Comput. Security Appl. Conf.*, Miami Beach, FL, USA, 2007, pp. 421–430.

[21] T. Wüchner, A. Cisłak, M. Ochoa, and A. Pretschner, "Leveraging compression-based graph mining for behavior-based malware detection," *IEEE Trans. Dependable Secure Comput.*, vol. 16, no. 1, pp. 99–112, Jan./Feb. 2019.

[22] E. Barshan, A. Ghodsi, Z. Azimifar, and M. Z. Jahromi, "Supervised principal component analysis: Visualization, classification and regression on subspaces and submanifolds," *Pattern Recognit.*, vol. 44, pp. 1357–1371, Jul. 2011, doi: 10.1016/j.patcog.2010.12.015.

[23] L. Nataraj, V. Yegneswaran, P. Porras, and J. Zhang, "A comparative assessment of malware classification using binary texture analysis and dynamic analysis," in *Proc. 4th ACM Workshop Security Artif. Intell.*, 2011, pp. 21–30.

[24] L. Nataraj, S. Karthikeyan, G. Jacob, and B. S. Manjunath, "Malware images: Visualization and automatic classification," in *Proc. 8th Int. Symp. Visual. Cyber Security*, 2011, pp. 1–7.

[25] S. Z. M. Shaid and M. A. Maarof, "Malware behavior image for malware variant identification," in *Proc. Int. Symp. Biometrics Security Technol. (ISBAST)*, Kuala Lumpur, Malaysia, 2014, pp. 238–243.

[26] Z. Cui, F. Xue, X. Cai, Y. Cao, G.-G. Wang, and J. Chen, "Detection of malicious code variants based on deep learning," *IEEE Trans. Ind. Informat.*, vol. 14, no. 7, pp. 3187–3196, Jul. 2018.

[27] T. Lei, Z. Qin, Z. Wang, Q. Li, and D. Ye, "EveDroid: Event-aware android malware detection against model degrading for IoT devices," *IEEE Internet Things J.*, vol. 6, no. 4, pp. 6668–6680, Aug. 2019.

[28] D. Wei and X. Qiu, "Status-based detection of malicious code in Internet of Things (IoT) devices," in *Proc. 6th IEEE Conf. Commun. Netw. Security (CNS)*, 2018, pp. 1–7.

[29] M. M. Hossain, R. Hasan, and S. Zawoad, "Probe-IoT: A public digital ledger based forensic investigation framework for IoT," in *Proc. IEEE INFOCOM Conf. Comput. Commun. Workshops (INFOCOM WKSHPS)*, 2018, pp. 1–2.

[30] S. Shen S. Huang L. Zhou, S. Yu, E. Fan, and Q. Cao, "Multistage signaling game-based optimal detection strategies for suppressing malware diffusion in fog-cloud-based IoT networks," *IEEE Internet Things J.*, vol. 5, no. 2, pp. 1043–1054, Apr. 2018.
[31] P. Wu, D. Liu, J. Wang, B. Yuan, and W. Kuang, "Detection of fake IoT app based on multidimensional similarity," *IEEE Internet Things J.*, vol. 7, no. 8, pp. 7021–7031, Aug. 2020.
[32] H. Alasmary *et al.*, "Analyzing, comparing, and detecting emerging malware: A graph-based approach," in *Proc. Netw. Distrib. Syst. Security Symp. (NDSS)*, 2019, pp. 1–2.
[33] A. Azmoodeh, A. Dehghantanha, and K.-K. R. Choo, "Robust malware detection for Internet of (Battlefield) Things devices using deep Eigenspace learning," *IEEE Trans. Sustain. Comput.*, vol. 4, no. 1, pp. 88–95, Jan.–Mar. 2019, doi: 10.1109/TSUSC.2018.2809665.
[34] A. Vaswani *et al.*, "Attention is all you need," in *Advances in Neural Information Processing Systems 30*. Red Hook, NY, USA: Curran, 2017, pp. 5598–6008.
[35] *Malware Capture Facility Project*. Accessed: Mar. 21, 2020. [Online]. Available: https://www.stratosphereips.org/datasets-malware
[36] *VxHeaven*. Accessed: Apr. 27, 2020. [Online]. Available: https://vt.netlux.org/index.html
[37] *Virus Share*. Accessed: Apr. 26, 2020. [Online]. Available: https://virusshare.com/
[38] H. Naeem, B. Guo, F. Ullah, and M. R. Naeem, "A cross-platform malware variant classification based on image representation," *KSII Trans. Internet Inf. Syst.*, vol. 13, no. 7, pp. 3756–3777, 2019
[39] J. Fu, J. Xue, Y. Wang, Z. Liu, and C. Shan, "Malware visualization for fine-grained classification," *IEEE Access*, vol. 6, pp. 14510–14523, 2018.
[40] Y. Qiao, Z. Jiang, Z. Jiang, and L. Gu, "A multi-channel visualization method for malware classification based on deep learning," in *Proc. 18th IEEE Int. Conf. Trust Security Privacy Comput. Commun./13th IEEE Int. Conf. Big Data Sci. Eng. (TrustCom/BigDataSE)*, Rotorua, New Zealand, 2019, pp. 757–762.

Qi Li received the Ph.D. degree in computer science and technology from Beijing University of Posts and Telecommunications, Beijing, China, in 2010.

She is currently an Associate Professor with the Information Security Center, State Key Laboratory of Networking and Switching Technology, School of Computer Science, Beijing University of Posts and Telecommunications. Her current research focuses on information systems and software.

Jiaxin Mi received the B.Eng. degree in information security from Beijing University of Posts and Telecommunications, Beijing, China, in 2020, where she is currently pursuing the M.Eng. degree in cyberspace security.

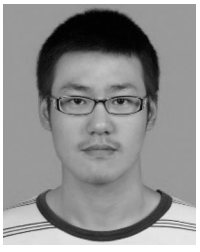
Her research interests include software security and data analysis.

Weishi Li received the B.Eng. degree in telecommunication engineering and management from Beijing University of Posts and Telecommunications, Beijing, China, in 2018, where he is currently pursuing the M.Eng. degree in cyberspace security.

His research interests include the areas of software security and data analysis.

Junfeng Wang received the M.S. degree in computer application technology from Chongqing University of Posts and Telecommunications, Chongqing, China, in 2001, and the Ph.D. degree in computer science from the University of Electronic Science and Technology of China, Chengdu, China, in 2004.

From July 2004 to August 2006, he was a Postdoctoral Fellow with the Institute of Software, Chinese Academy of Sciences, Beijing, China. Since August 2006, he is with the College of Computer Science, Sichuan University, Chengdu, as a Professor. His recent research interests include network and information security, spatial information networks, and data mining.



Mingyu Cheng received the B.S. degree in information security from Xi'an University of Posts and Telecommunications, Xi'an, China, in 2019. He is currently pursuing the M.S. degree in information security with Beijing University of Posts and Telecommunications, Beijing, China.

His research interests are in the areas of software security and data analysis.