Detection of Malicious Code Variants Based on Deep Learning

Zhihua Cui, Fei Xue, Xingjuan Cai, Yang Cao, Gai-ge Wang, *Member, IEEE*, and Jinjun Chen, *Senior Member, IEEE*

***Abstract*—With the development of the Internet, malicious code attacks have increased exponentially, with malicious code variants ranking as a key threat to Internet security. The ability to detect variants of malicious code is critical for protection against security breaches, data theft, and other dangers. Current methods for recognizing malicious code have demonstrated poor detection accuracy and low detection speeds. This paper proposed a novel method that used deep learning to improve the detection of malware variants. In prior research, deep learning demonstrated excellent performance in image recognition. To implement our proposed detection method, we converted the malicious code into grayscale images. Then, the images were identified and classified using a convolutional neural network (CNN) that could extract the features of the malware images automatically. In addition, we utilized a bat algorithm to address the data imbalance among different malware families. To test our approach, we conducted a series of experiments on malware image data from Vision Research Lab. The experimental results demonstrated that our model achieved good accuracy and speed as compared with other malware detection models.**

***Index Terms*—Malware variants, grayscale image, deep learning, convolution neural network, bat algorithm.**

Manuscript received September 29, 2017; revised January 9, 2018, March 1, 2018, and March 26, 2018; accepted March 28, 2018. Date of publication April 3, 2018; date of current version July 2, 2018. This work was supported in part by the National Natural Science Foundation of China under Grant 61003053 and Grant 61403271, and in part by the Natural Science Foundation of Shanxi Province under Grant 201601D011045. Paper no. TII-17-2263. *(Corresponding author: Xingjuan Cai.)*

Z. Cui and X. Cai are with the Complex System and Computational Intelligence Laboratory, TaiYuan University of Science and Technology, Taiyuan 030024, China (e-mail: zhihua.cui@hotmail.com; xingjuancai@163.com).

F. Xue is with the School of Information, Beijing Wuzi University, Beijing 101149, China (e-mail: xuefei2004@126.com).

Y. Cao is with the Faculty of Information Technology, Beijing University of Technology, Beijing 100124, China (e-mail: caoyangcwz@gmail.com).

G.-G. Wang is with the Department of Computer Science and Technology, Ocean University of China, Qingdao 266100, China (e-mail: gaigewang@gmail.com).

J. Chen is with the Complex System and Computational Intelligence Laboratory, Taiyuan University of Science and Technology, Taiyuan 030024, China, and also with Swinburne Data Science Research Institute, Swinburne University of Technology, Melbourne, VIC 3122, Australia (e-mail: jinjun.chen@gmail.com).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TII.2018.2822680

I. Introduction

WITH the rapid development of information technology, the exponential growth of malicious code has become one of the main threats to internet security. A recent report from Symantec showed that 401 million malicious codes were found in 2016, including 357 million new malicious code variants [22]. The appearance of malware in mobile devices and the Internet of Things (IoT) also grew rapidly. Till date, 68 new malicious code families and more than 10 000 malicious codes have been reported. This growth has posed a challenge for malicious code detection in cloud computing [14], [15].

As a key part of security protection, uncovering malicious code variants is particularly challenging. Malware detection methods consist primarily of two types of approaches: static detection and dynamic detection. Static detection works by disassembling the malware code and analyzing its execution logic. Dynamic detection analyzes the behavior of malicious code by executing the code in a safe virtual environment or sandbox.

Both static and dynamic detection are feature-based detection methods. First, the textual or behavioral features of the malicious code are extracted and, then, the malicious code is detected or classified by analyzing these extracted features. In recent years, several scholars have used data mining methods to analyze the features of malicious code [28]. This approach has become the mainstay of malware detection because it is highly efficient and has a low rate of false positives compared with traditional heuristic-based detection methods. Fig. 1 illustrates the process of detecting malicious code using data mining.

Unfortunately, methods based on feature analysis are often disrupted. The effectiveness of static feature analysis can be hampered by the obfuscation techniques that transform the malware binary into a self-compressed or uniquely structured binary. Dynamic feature analysis is often challenged by many kinds of countermeasures developed to produce unreliable results. Furthermore, some types of malicious code may be ignored by dynamic analysis because the execution environment does not comply with the rules.

Recently, rather than focusing on nonvisible features for malware classification, Nataraj *et al.* [19] proposed malware visualization, a new approach based on image processing techniques. This work transformed the structure of packed binary samples into two-dimensional (2-D) grayscale images. Then, the image features were used for classification. Moreover, two different feature design strategies are studied comparatively for malware

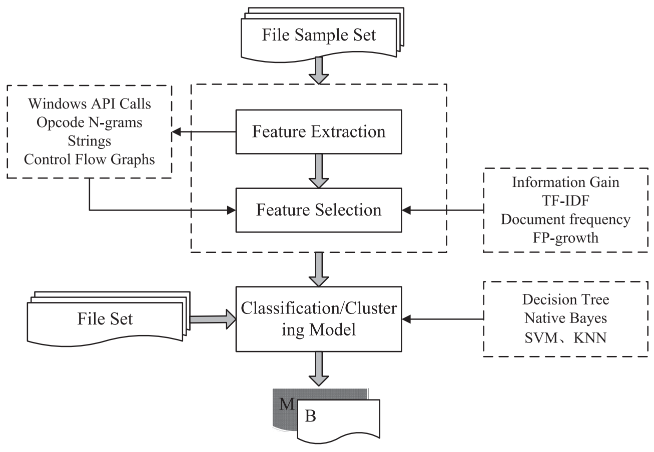


Fig. 1. Malicious code detection based on data mining.

analysis [20]. The results illustrated that the binary texture-based strategy provided performance roughly equivalent to the dynamic analysis techniques, but in less time.

Challenges: Malware detection methods rely mainly on analysis of the features of malicious codes (e.g., static features and dynamic features). More powerful detection methods based on various machine learning techniques also use these features to uncover malicious codes or their variations. However, these approaches become less effective when detecting malicious code variants or unknown malware. The malware visualization method can handle code obfuscation problems, but it suffers from the high time cost needed for complex image texture feature extraction (e.g., GIST and GLCM). Moreover, these feature extraction methods also demonstrate low efficiency when exposed to large datasets. The challenge for building malware detection models is to find a means for extracting features effectively and automatically.

Moreover, the data imbalance problem imposes another challenge. Of the large quantity of malware generated each year, a substantial portion includes variants that belong to existing malicious code families or groups. Usually, the number of malicious code variations differs greatly among various code families. The challenge is to build a universal detection model that can deal with the huge volume of variations, so that it can work well across malware families.

Contributions: To address the above-mentioned challenges, this paper offered the following contributions.

1) We introduced a technique for converting a malware binary to an image, thereby transforming malware detection into an image classification problem.
2) We proposed a novel method for detecting malware variants based on a convolutional neural network (CNN).
3) To resolve the data imbalance problem among different malware families, we designed an effective data equilibrium approach based on the bat algorithm.
4) Extensive experimental results demonstrated that our proposed method was an effective and efficient approach for malware detection.

The remainder of this paper is structured as follows. Section II reviews related work, then Section III details our approach to malicious code detection based on a CNN. Section IV presents our data equalization method based on a bat algorithm and data augmentation. Experimental evaluations of the proposed methods are given in Section V. Finally, Section VI concludes this paper.

II. Related Work

In this section, we present the related research regarding malware, including malware detection based on feature analysis, malicious code visualization, image processing techniques for malware detection, and malware detection based on deep learning.

A. Malware Detection Based on Feature Analysis

As described previously in this paper, there are two main categories of feature analysis techniques for malware detection: static analysis and dynamic analysis. With respect to static analysis, several methods have been put forward by analyzing the codes. For example, Isohara *et al.* [12] developed a detection system using kernel behavior analysis that performed well in detecting the malicious behaviors of unknown applications. However, the static method is easily deceived by obfuscation techniques [18].

To solve this problem, Christodorescu *et al.* [4] proposed a novel malware-detection algorithm that used trace semantics to characterize the behavior of malware. This method proved effective in combating the obfuscation of instructions (e.g., rearrangement of instructions, insertion of garbage code, and register redistribution). However, the approach was limited to feature extraction and analysis at the instruction level. Moreover, the pattern-matching was complex.

Dynamic analysis monitors and analyzes the runtime characteristics of applications (apps) based on assessment of behaviors, such as accessing private data and using restricted API calls. Given this information, a behavior model is established to detect malicious code. Such techniques achieved improved detection performance, but they were still challenged by the variety of countermeasures developed to generate unreliable results [20]. In addition, dynamic analysis is time consuming because of the large amount of computational overhead, leading to low efficiency when exposed to a large dataset.

B. Malicious Code Visualization

Currently, many tools can visualize and manipulate binary data, e.g., common text editors and binary editors. Several studies have proposed utilization of malware visualization [24]. Yoo *et al.* [29] employed self-organizing maps to visualize computer viruses. A similar but richer approach was proposed by Trinius *et al.* [23], who used two visualization techniques—tree maps and thread graphs—to detect and classify malware. Rather than a single detection result, Goodall *et al.* [10] aggregated the results of different malware analysis tools into a visual environment, providing an increase in the vulnerability of detection coverage of single malware tools.

The above-mentioned studies focused mainly on the visualization of malware behavior, but the software source code may imply more meaningful patterns. As previously mentioned, Nataraj *et al.* [19] presented a new visualization approach for malware detection based on binary texture analysis. First, they converted the malware executable file into grayscale images. Then, they identified malware according to the texture features of these images. Compared with the dynamic analysis method, their approach produced equivalent results [20]. In similar work, Han *et al.* [11] transformed malware binary information into color image matrices, and classified malware families by using an image processing method.

C. Image Processing Techniques for Malware Detection

Once the malware has been visualized as grayscale images, malware detection can be converted into an image recognition problem. In [19], Nataraj *et al.* used a GIST algorithm to extract the features of malware images. However, the GIST algorithm was time consuming. Recently, more powerful image processing techniques have been proposed.

Daniel *et al.* [8] developed a bio-inspired parallel implementation for finding the representative geometrical objects of the homology groups in a binary 2-D image. For image fusion, Miao *et al.* [17] proposed an image fusion algorithm based on shearlet and genetic algorithm. In their approach, a genetic algorithm is employed to optimize the weighted factors in the fusion rule. Experimental results demonstrated that their method could acquire better fusion quality than other methods.

These traditional approaches, however, were challenged by the high time cost required for complex image texture feature extraction. To address this challenge, we employed deep learning to identify and classify images efficiently. In the next subsection, we present our research on malware detection based on deep learning.

D. Malware Detection Based on Deep Learning

Deep learning [21] is an area of machine learning research that has emerged in recent years from the work on artificial neural networks. Neural networks can approximate complex functions by learning the deep nonlinear network structure to solve complex problems. Deep learning, which is more powerful than back propagation (BP), uses a deep neural network to simulate the human brain's learning processes. Deep learning has the ability to learn the essential characteristics of data sets from a sample set. As a powerful tool of artificial intelligence, deep learning has been applied widely in many fields, such as recognition of handwritten numerals, speech recognition, and image recognition.

Because of it powerful ability to learn features, many scholars have applied deep learning to malware detection. Using deep learning techniques, Yuan *et al.* [30] designed and implemented an online malware detection prototype system, named Droid-Sec. Their model achieved high accuracy by learning the features extracted from both static analysis and dynamic analysis of Android apps. David *et al.* [7] presented a similar but more compelling method that did not need the type of malware behavior. Their work was based on a deep belief network (DBN) for automatic malware signature generation and classification.

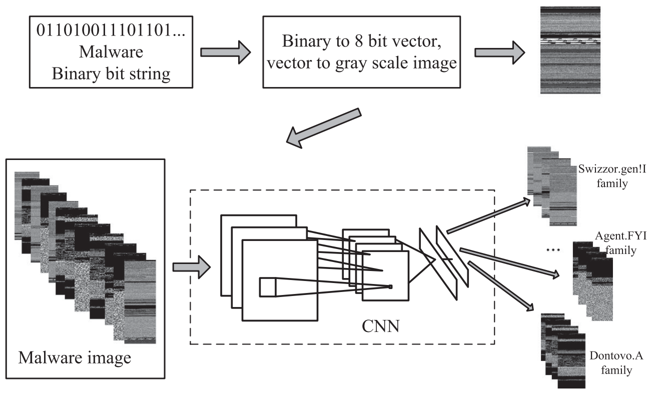


Fig. 2. Overview of the proposed method.

Compared with conventional signature methods for malware detection, their approach demonstrated increased accuracy for detecting new malware variants.

Unfortunately, these methods remained based on the analysis of features extracted by static analysis and dynamic analysis. Therefore, to a greater or lesser extent, they continued to be subject to the limitations of feature extraction. To address this problem, we employed a CNN network to learn the malware image features and classify them automatically.

III. Malware Detection Based on a CNN

This section presents our improved malicious code variant detection method based on a CNN, which included: 1) the malicious code mapping as the grayscale image; and 2) the CNN design for grayscale image detection. Fig. 2 presents an overview of these two processes. First, the binary files of malicious code are transformed into the grayscale images. Next, the convolution neural network is employed to identify and classify the images. According to the results of image classification, we realized the automatic recognition and classification of malicious software.

A. Binary Malware to Gray Image

In general, there are several ways to transform binary code into images. In this paper, we used the visualization of executable malware binary files [19]. A malware binary bit string can be split into a number of substrings that are 8 bits in length. Each of these substrings can be seen as a pixel, because the 8 bits can be interpreted as an unsigned integer in the range 0–255. For example, if a bit string is *0110000010101100*, the process is *0110000010101100→01100000,10101100→96 172*. An eight-bit binary number $B = (b_7, b_6, b_5, b_4, b_3, b_2, b_1, b_0)$ can be converted into a decimal number I as follows:

$$I = b_0 * 2^0 + b_1 * 2^1 + b_2 * 2^2 + b_3 * 2^3 + b_4 * 2^4 + b_5 * 2^5 + b_6 * 2^6 + b_7 * 2^7. \quad (1)$$

After binary conversion, the binary malware bit string has been converted into a 1-D vector of decimal numbers. According to a specified width, this 1-D array can be treated as a 2-D matrix of a certain width. Finally, the malicious code matrix is interpreted as a grayscale image. For simplicity, the width of

TABLE I
IMAGE WIDTH FOR VARIOUS FILE SIZES

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

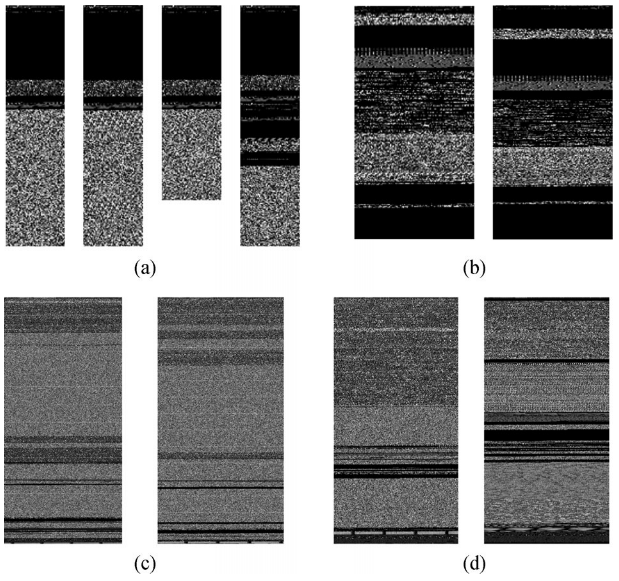


Fig. 3. Illustration of malware images. (a) Agent.FYI family. (b) Dontovo.A family. (c) Swizzor.gen!E family. (d) Swizzor.gen!I family.

the image is fixed, and the height of the image varies depending on the size of the file. Table I, taken from [19], presents some recommended image widths for various file sizes, based on empirical observations.

Fig. 3 shows examples of malware images from different families [19]. As we can see, the images of the same malware family are visually similar, and they differ noticeably from images belonging to another family. For example, Fig. 3(a) shows four variants of the malware agent, called the Agent.FYI family. The size of each member is different, but they still have similarities because a new malware is often created from the old software. In addition, when families are similar, images can display their differences clearly. For example, compared with the Swizzor.gen!I family in Fig. 3(c), the Swizzor.gen!E family in Fig. 3(d) has black stripes. Inspired by the visual similarity of malware images, we can classify and detect malicious software by using image recognition methods. In our current work, we used a convolutional neural network (CNN) to recognize malware images.

B. Malware Image Classification Based on CNN

As a hot topic in speech analysis and image recognition, CNNs have developed quickly. Their local perception and weight sharing network structure reduce the complexity of network models and the number of weights. A CNN has even further advantages when the input is multidimensional. For our work, the image can be considered as the input of the network. Compared with traditional recognition algorithms, this method is not hampered by complicated feature extraction and data reconstruction. Convolutional networks are a multilayer perceptron designed for identifying 2-D shapes, and they are robust with respect to image deformation (e.g., translation, scale, and rotation).

In this paper, we developed a CNN to classify malware. The structure of the CNN for grayscale image recognition consists of several components, as shown in Fig. 4. First is the input layer, which brings the training images into the neural network. Next are the convolution and subsampling layers. The former layer can enhance signal characteristics and reduce noise. The latter can reduce the amount of data processing while retaining useful information. Then, there are several fully connected layers that convert a 2-D feature into a 1-D feature that conforms to the classifier criteria. Finally, the classifier identifies and sorts the malware images into different families according to their characteristics. The detailed iterative formula of each layer can be given as follows.

1) Convolution Layers: The convolution layer can reduce the number of image parameters while preserving the main features, termed *invariance*, including translation invariance, rotation invariance, and scale invariance. This process can avoid overfitting effectively, and improve the generalization ability of the model. The input is several maps, and the output is the maps after dimension reduction. Each map is a combination of convolution values of input maps that belong to the upper layer [1], and can be given by the following equation:

$$x_j^l = f\left(\sum_{i\in M_j} x_j^{l-1} * k_{ij}^l + b_j^l\right) \tag{2}$$

where M_j is the collection of input maps, k_{ij}^l is the convolution kernel used for the connection between the ith input feature map and the jth output feature map, b_j^l is the bias corresponding to jth feature map, and f is the activation function. The sensitivity is

$$\delta_j^l = \delta_j^{l+1} W_j^{l+1} \circ f'(u^l) = \beta_j^{l+1} up(\delta_j^{l+1}) \circ f'(u^l) \tag{3}$$

where l+1 layer is the sampling layer, the weight W represents a convolution kernel, and its value $\beta_j^{l+1} \circ up(.)$ is an up-sampling operation.

The partial derivative of the error cost function with respect to bias b and convolution kernel k can be given as follows:

$$\frac{\partial E}{\partial b_j} = \sum_{u,v} (\delta_j^l)_{u,v} \tag{4}$$

$$\frac{\partial E}{\partial k_{ij}^l} = \sum_{u,v} (\delta_j^l)_{u,v} (p_i^{l-1})_{u,v} \tag{5}$$

where $(p_i^{l-1})_{u,v}$ is the *patch* for each convolution of x_i^{l-1} and k_{ij}^l, (u, v) is the center of the *patch*, and its position value can be obtained by convolution of the *patch* of the (u, v) position in the input feature map and the convolution kernel k_{ij}^l.

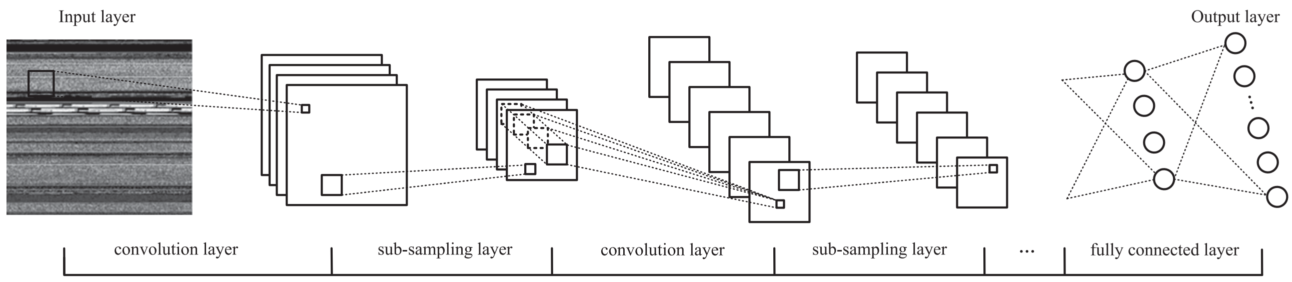


Fig. 4. Illustration of malware classification based on a CNN.

2) Subsampling Layers: The subsampling layer is also called the pooling layer. Generally, its convolution kernels are the maximum or mean of *patch* (maximum pooling, average pooling), and are not modified by backward propagation. This layer can weaken the influence of image deformation (e.g., translation, scale, and rotation). It reduces the dimension of the feature map, improves the model's accuracy, and avoids overfitting. In CNN, for each output of the sampling layer, the feature map is given as follows:

$$x_j^l = f(\text{down}(x_j^{l-1}) + b_j^l) \tag{6}$$

where down(.) represents a subsampling function and b is bias. The sensitivity is calculated as shown

$$\delta_j^l = \delta_j^{l+1} W_j^{l+1} \circ f'(u^l). \tag{7}$$

The partial derivative of the error cost function with respect to bias b can be given as follows:

$$\frac{\partial E}{\partial b_j} = \sum_{u,v} (\delta_j^l)_{u,v}. \tag{8}$$

So far, we can get the weight updating formula of the CNN, and use it to classify malware images. However, in the CNN, a fixed size image is needed for a structured network because that the connection matrix W of the full connection layer is a fixed size after training. For example, if the input and output sizes are 30 and 20 neurons, respectively, from the convolution layer to the full connection layer, then the size of the weight matrix must be (30,20). Unfortunately, the image size of malware is not fixed, but varies with the size of the software. Therefore, we cannot input these malware images directly into a CNN.

To solve this problem, in this paper, we reshaped the malware image to a fixed size square image (e.g., 48*48, 96*96). In this way, the malware images were normalized and could be entered directly into the CNN network for classification. The advantage of normalization is that it reduces the dimensionality of the image effectively, and is more conducive to the training of the model. In the dimensionality reduction process, inevitably some characteristic information is lost.

For most images in our malware dataset, texture features can be preserved effectively whether there is an enlargement or reduction. An example can be found in the variants of Dontovo.A family shown in Fig. 5(a). The original image size was 64*257. After compression (24*24 and 48*48) or expansion (96*96 and 192*192), the texture feature remained sharp (the upper part is black and the bottom is gray). However, for those big images

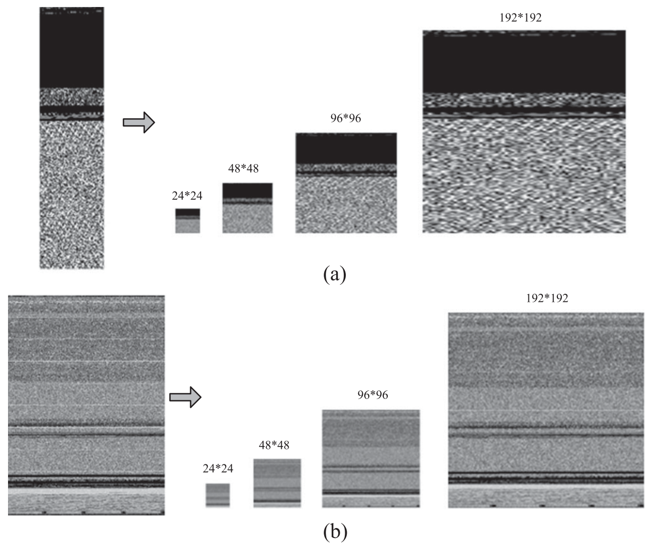


Fig. 5. Reshape the malware image to a fixed size square image. (a) Dontovo.A family (b) Swizzor.gen!E family

with small textures feature, the reshape operation may cause the loss of critical information. For example, the textures feature (black dots at the bottom) of a variant from the Swizzor.gen!E family in Fig. 5(b) was damaged when it was compressed too much (from 512*718 to 24*24 or 48*48).

IV. Malware Image Data Equilibrium

High-quality data is the key to machine learning and deep learning. Rough sample data will affect the quality of the model. In contrast, a model trained by high-quality data is more robust (avoids overfitting), and can speed up model training.

Malicious software usually consists of several families, and the number of samples of each family varies widely. This variation in data will lead to low accuracy and poor robustness of the training model. To address this problem, we employed a data augmentation technique to improve data quality. We proposed a data equalization method based on an intelligent optimization algorithm (i.e., dynamic resampling based on a bat algorithm) to resolve the problem of imbalanced data between different families.

A. Image Data Augmentation (IDA) Technology

In deep learning, to avoid overfitting, we usually need to enter sufficient data to train the model. If the data sample is

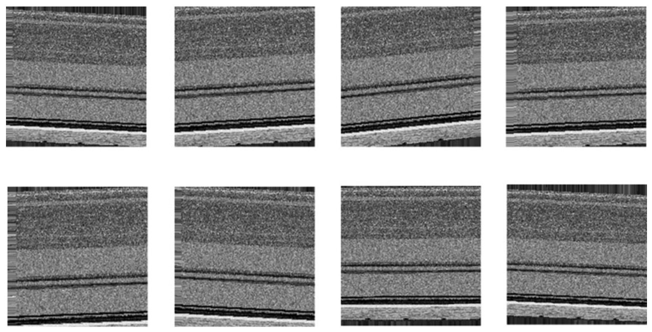


Fig. 6. Images generated by data augmentation for Swizzor.gen!E family.

relatively small, we can use data augmentation to increase the sample, thereby restraining the influence of imbalanced data. The appropriate data augmentation method can avoid overfitting problems effectively, and improve the robustness of the model.

Generally, transformation of the original image data (changing the location of image pixels and ensuring that the features remain unchanged) is used to generate new data. There are many kinds of data augmentation techniques for images, for example, $rotation/reflection$, $flip$, $zoom$, $shift$, $scale$, $contrast$, $noise$, and $color$ transformation.

Let $A = [a_1, a_2, \ldots, a_8]$ be a collection of these techniques. $M_i = a_m, \ldots, a_n$ is the sequence of operations, and i is the length of M. $M_2 = a_1, a_2$ represents data augmentation by using $rotation$ transformation and $flip$ transformation.

Typically, each augmentation technique has a weight λ_i, so the sequence of operations of weighted data augmentation techniques can be given by

$$M_i = \lambda_m a_m, \ldots, \lambda_n a_n \tag{9}$$

where λ_m is the weight of the augmentation technology and $a_m \subseteq A$. Usually, it is necessary to perform multiple data augmentations. For a grayscale image matrix m, $M_k(m)$ represents k augmentations. Fig. 6 shows some images generated by data augmentation for the malware Swizzor.gen!E family. Clearly, most of the newly generated images retained the original texture features. However, a few images lost some information through inappropriate transition (i.e., overrotation). The images viewed in the lower-left corner lack a texture feature (black dots at the bottom) of the Swizzor.gen!E family.

B. Data Equalization Based on a Bat Algorithm

Imbalanced data is a common problem in machine learning, and it can affect the accuracy of the model in classification problems. In this paper, the malware image data was very uneven, as shown in Fig. 7. The maximum ratio was about 36:1. For example, the Allaple.A family had 2949 images, while the Skintrim.N family had 80. Such differences have a very big impact on a CNN's performance in image classification. The classifying model trained according to the data of the two families may achieve 97% accuracy when all the samples in the Skintrim.N family are not identified. However, it is not a reasonable classifier because it can only identify some data.

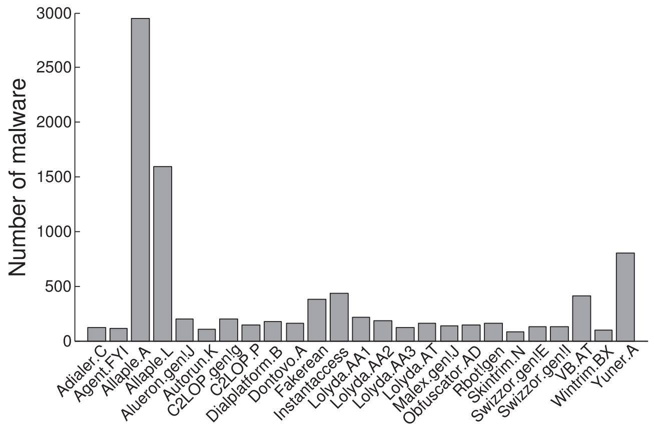


Fig. 7. Number of malware images in different families.

Algorithm 1: DRBA($dataset$) Operation.

```
    Input: $dataset$: training set for CNN; $bats$: bat population;
    Output: $trainset$: training set for each epoch;
 1: $(p, v) \leftarrow \mathbf{getState}(bats)$;
 2: while $t < maxIterations$ do
 3:     for $bat_i \subset bats$ do
 4:         $(p_i^{t+1}, v_i^{t+1}) \leftarrow \mathbf{updateState}(p_i^t, v_i^t)$;
 5:         $templeset \leftarrow \mathbf{reSample}(dataset, p_i^{t+1})$;
 6:         $accuracy_i^{t+1} \leftarrow \mathbf{trainModel}(templeset)$;
 7:     end
 9:     $bats \leftarrow \mathbf{localSearch}(bats)$;
 9:     $p^* \leftarrow \mathbf{getBest}(bats)$;
10:     $t = t + 1$;
11: end
12: $trainset \leftarrow \mathbf{reSample}(dataset)$;
13: $\mathbf{Return}(trainset)$;
```

In classification, resampling is a simple method used to handle an unbalanced training set. The resampling method converts the imbalanced data set into a balanced dataset by processing the training set. It consists of two implementations: oversampling and undersampling. Oversampling is used to make multiple copies of subsets. Undersampling is used to remove some samples from the sample set (i.e., to select only some samples).

For instance, take the Allaple.A family (2949 images) and the Skintrim.N family (80 images). If the training set for the former consists of 1000 samples, we can choose 500 of them. For the latter, because of the insufficient number of data, we must repeat sampling until the training sample reaches 500. Because the training sample is less than the total sample, the trained classification model for the Skintrim.N family will suffer from overfitting. Therefore, the proper proportion of samples is important for the training set. Unfortunately, it is difficult to find an optimal proportion due to the complex weight combinations with dozens of families.

A swarm intelligent algorithm is an effective solution for complex optimization problems. Many researchers have made several in-depth studies about swarm intelligence and its applications [9], [16], [26]. Wang *et al.* [25] used the evolutionary multiobjective optimization algorithms to deal with the floorplan problem in cyber-physical social system. Cui *et al.* [6] applied a modified cuckoo search algorithm to improve the performance of DV-Hop.

To address the problem of weight combination, we proposed a dynamic resampling method based on the bat algorithm (DRBA). The bat algorithm is a novel swarm intelligence algorithm [27], inspired by the echolocation behavior of microbats. Because of its potential parallelism and robustness, several researchers applied the bat algorithm to the optimization problems [2], [3], [5]. We employed it to optimize the sampling weights of multiple malware families. In the model training process, to balance samples properly, each class is resampled according to the weight for each epoch. Suppose the number of malicious code families is N, the resampling weights are denoted by $w_i, i \subset N$. For this optimization, the position of a bat individual is a combination of sampling weights that can be given by the following equation:

$$\text{position} = w_1, \ldots, w_n. \tag{10}$$

The accuracy of the training model is an objective function. The aim of the optimization is to find the optimal position of the bat when the accuracy reaches the set threshold. The detailed resampling process is shown in Algorithm 1. The function $updateState$ (line 4) is used to update the velocity and position of bats according to the rules in [2]. The function $localSearch$ is the local searching of the bat population. The $accuracy$ is the fitness, which can be obtained by the functions $reSample$ and $trainModel$ (lines 5,6). As the names suggest, the former is a resampling function, and the latter is a function that trains the model according to resampling data.

V. Experimental Evaluation

In this section, we present our evaluation of the effectiveness of the proposed method. Our experiments were based on a malware dataset from the Vision Research Lab [19]. We employed the neural network framework Caffe [13] to create and train a CNN. We ran our experiments on a PC with an Intel Core i5-4590 CPU (3.3 GHz), Nvidia GeForce GTX 750Ti GPU (2 GB), and 8 GB RAM.

A. Dataset and Experimental Setting

The dataset consisted of 9342 grayscale images of 25 malware families [19]. The number of samples in each family was different. As shown in Fig. 7, there was a great deal of imbalance in the number of raw data samples. Therefore, it was necessary to use a data balancing method to balance the number of training samples.

First, we used data augmentation techniques to improve the quality of small data samples. The expansion of small data samples can solve problems (e.g., overfitting) caused by the unbalanced training data, effectively. The specific settings for image augmentation are shown in Table II.

In Table II, $rotation\_range$ represents the rotation range, $width\_shift$ is the range of horizontal translation, $height\_shift$ is the range of vertical translation, $rescale$ is the ratio of image magnification or reduction, $shear\_range$ is the range of projection transformation in the horizontal or vertical direction, and $zoom\_range$ is the ratio of randomly zooming image. In addition, we used the $nearest$ method to fill the image when flipping or expansion.

TABLE II
Settings of Grayscale Malware Image Augmentation

Methods	Settings	Methods	Settings
$rotation\_range$	0.1	$shear\_range$	0.1
$width\_shift$	0.1	$zoom\_range$	0.1
$height\_shift$	0.1	$horizontal\_flip$	true
$rescale$	1/255	$fill\_mode$	nearest

For our model, we designed different architectures of the CNN for malware images with different sizes. For the 24*24 inputs, our model had seven layers, including five hidden layers. The detailed structure was as follows: C1:8*20*20, S2:8*10*10, C3:16*8*8, S4:16*4*4, and C5:80*1*1. Each type of convolution kernel corresponded to a type of feature map. The maps per layer referred to the number of feature maps per layer. For the input of other sizes, as the size increased, the number of layers increased. Furthermore, we used the *imresize* function with interpolation method *bicubic* (default) in MATLAB R2017a to resize/normalize the images.

The initial learning rate was set to 0.01, and the decay policy was inv. The maximum number of iterations was 10000. Finally, the training was carried out using the GPU. Moreover, in the image data balancing method DRBA, for BA, the population size was 20, other parameters were the same with [2].

For evaluation metrics, we used the accuracy, precision, and recall. These evaluation metrics are adopted frequently in the research community to provide comprehensive assessments of imbalanced learning problems. These metrics are defined as follows:

$$\text{precision} = \text{TP}/(\text{TP} + \text{FP}) \tag{11}$$

$$\text{recall} = \text{TP}/(\text{TP} + \text{FN}) \tag{12}$$

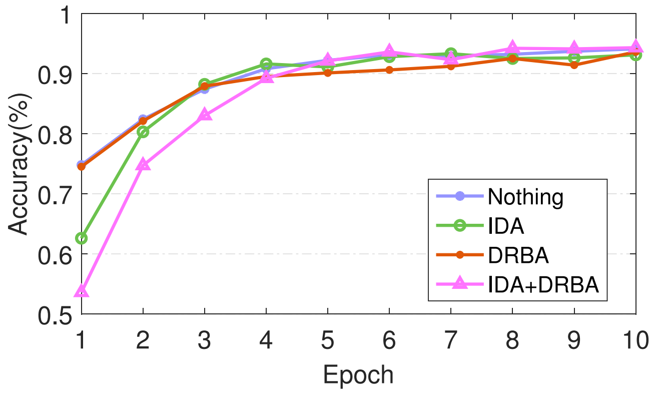
$$\text{accuracy} = (\text{TP} + \text{TN})/(\text{TP} + \text{TN} + \text{FP} + \text{FN}) \tag{13}$$

where true positive (TP) and false positive (FP) are the number of file samples correctly and wrongly classified as malicious, respectively. Similarly, true negative (TN) and false negative (FN) are the number of file samples correctly and wrongly classified as benign, respectively.

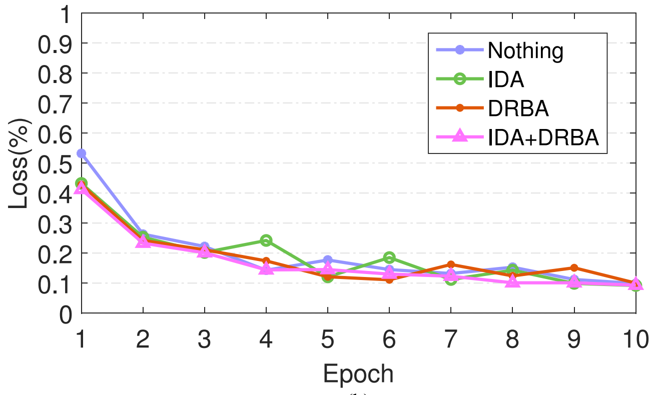
B. Experimental Results

To validate the effectiveness and efficiency of the proposed approach, we designed experiments to

1) verify the efficiency of different data equalization methods,
2) ascertain the effects of different malware image sizes, and
3) compare the results of our proposed approach with the outcomes of other methods for detecting malicious code.

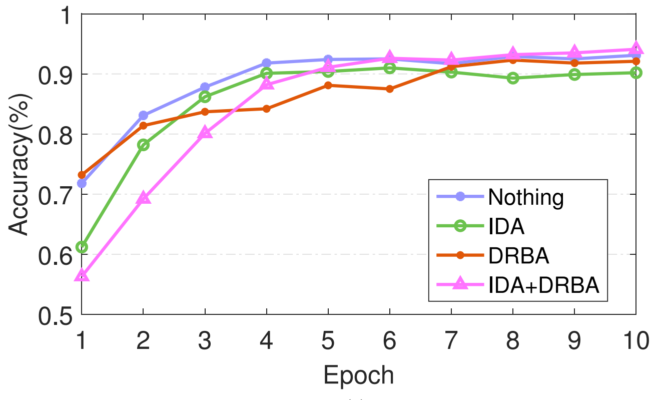


(a)

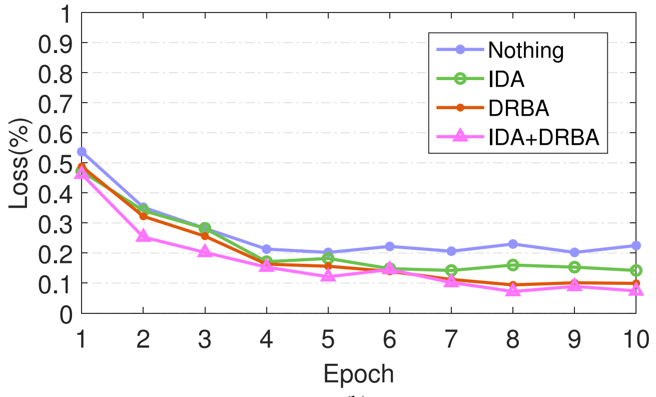


(b)

Fig. 8. Performance of different classification models on the training set. (a) Accuracy. (b) Loss.



(a)



(b)

Fig. 9. Performance of different classification models on the validation set. (a) Accuracy. (b) Loss.

1) Efficiency of Malware Image Data Equalization: To verify the validity of our proposed data equalization method, including the IDA technique and DRBA, we compared the results when no equalization method was used (nothing) with models using IDA, DRBA, and IDA+DRBA. The performance of the different classification models on the training set is shown in Fig. 8. As seen from Fig. 8(a), all methods achieved a good result with respect to accuracy, even if the data set was not processed. The nothing method may have demonstrated overfitting. Fig. 8(b) shows the loss curve, for which the results of the four methods were similar.

Fig. 9 shows the performance of the different classification models on the validation set. As we can see from Fig. 9(a), the nothing method reflected the classification of all the data with the basic CNN model. Because of the imbalance of the samples, it was easy to cause overfitting, so the nothing method appeared to show a high level of accuracy. The accuracy of the DRBA method was the best. The accuracy rate increased continuously with the increase of the epoch. The accuracy of the IDA method was not as good as its performance on the training set, perhaps because some features of the image were lost in the image transformation process. There were some differences between training data and test data. As seen from Fig. 9(b), the loss curve of DRBA and IDA+DRBA was decreasing, while the curve of nothing was oscillating within a certain range, which meant it was overfitting.

2) Model Performance With Different Malware Image Size: The CNN framework required all input images to have a fixed size. Therefore, we reshaped the malware image to a fixed size square image. The smaller the transformed image, the more obvious the texture features were destroyed. In contrast, the larger images had good fidelity, and the texture features were clear. We conducted experiments setting the malware image at 24*24, 48*48, 96*96, and 192*192, respectively. The confusion matrix is shown in Fig. 10.

From the results, we can see that our model performed better as the image became larger. However, a bigger image requires more training time. We noted that the performances at 96*96 and 192*192 were similar, as shown in Fig. 10(c). Since the classification model would take more time to train using the larger image, the 96*96 image was a good choice for training on this malware dataset.

3) Comparison of Our Method With Other Malicious Code Detection Methods: To validate the effectiveness and efficiency of our proposed approach, we compared our model with four other malware detection models that used common image feature extraction methods. Each of these methods extracted features of the malware images, and then applied a machine learning algorithm (e.g., KNN and SVM) to identify and classify the malware. For this comparison, we used a model based on DRBA with a 96*96 input image.

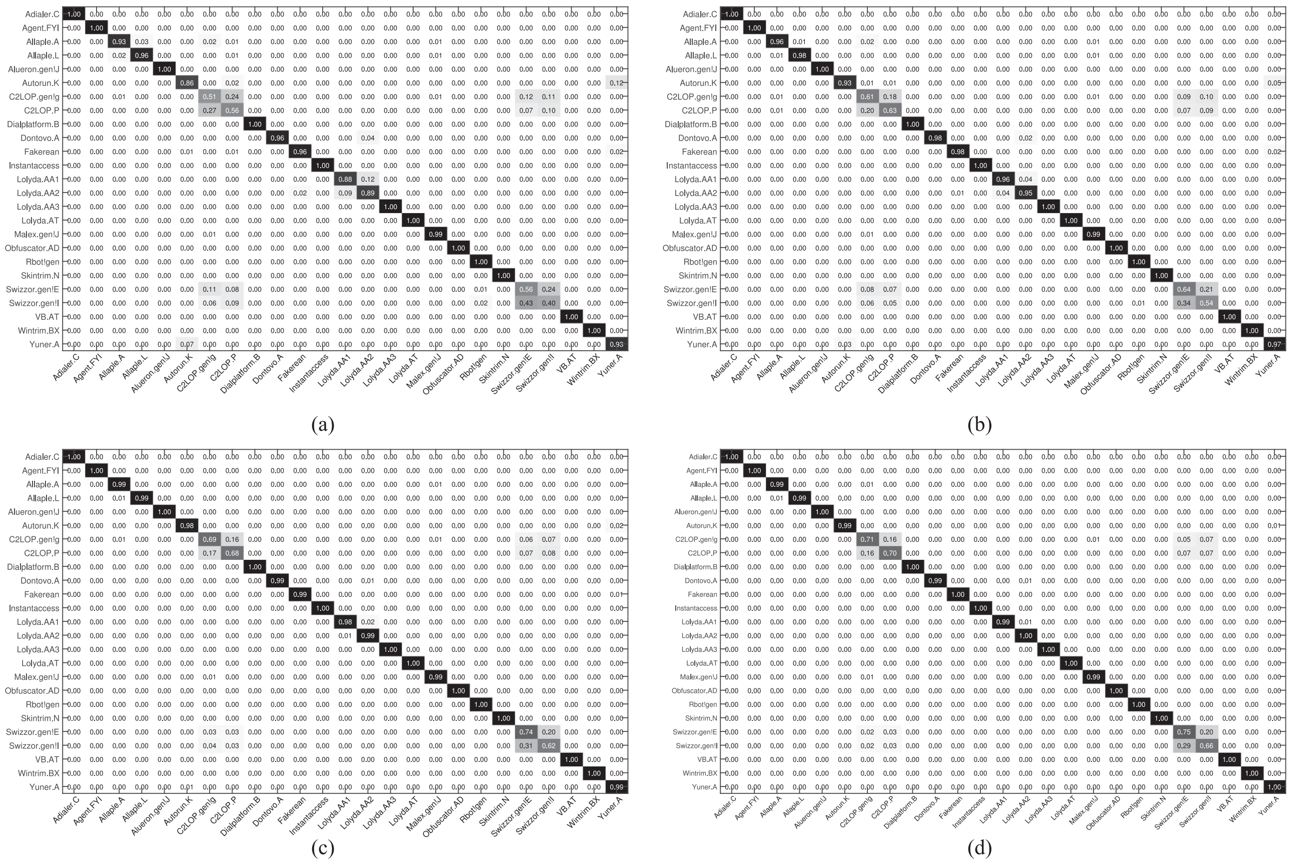


Fig. 10. Confusion matrix of models with different malware image size. (a) 24*24. (b) 48*48. (c) 96*96. (d) 192*192.

TABLE III
PROPOSED MODEL COMPARED WITH OTHER MALICIOUS CODE DETECTION METHODS

Methods	Accuracy	Precision	Recall	Running time
GIST+KNN	91.9	92.1	91.7	60ms
GIST+SVM	92.2	92.5	91.4	64ms
GLCM+KNN	92.5	92.7	92.3	45ms
GLCM+SVM	93.2	93.4	93.0	48ms
Our approach	94.5	94.6	94.5	20ms

Table III shows the results of our approach compared with the outcomes of the four combination models: GIST+KNN, GIST+SVM, GLCM+KNN, and GLCM+SVM. We can see that our method had higher accuracy and faster detection speed. The performance of the other methods was somewhat poorer because their use of GIST or GLCM was complex and time consuming.

VI. CONCLUSION AND FUTURE WORK

This paper proposed a novel method to improve the detection of malware variants through the application of deep learning. First, this method transformed the malicious code into grayscale images. Next, the images were identified and classified by a CNN that could extract the features of the malware images automatically. Because of the effectiveness and efficiency of the CNN for identifying malware images, the detection speed of our model was significantly faster than speeds achieved by other approaches. In addition, our model provided an effective solution for the data imbalance problem among different malware families. Our experimental results on 9342 grayscale images of 25 malware families showed that the proposed approach achieved 94.5% accuracy with good detection speed.

In this study, the CNN framework required all input images to have a fixed size, which limited our model. In future work, we would like to use the SPP-net model to allow images of any size to be used as input. The SPP-net can extract features at variable scales, thanks to a spatial pyramid pooling layer. We can introduce that layer into our model between the last subsampling layer and the fully connected layer to improve our models flexibility. In addition, the transformation of malicious code into color images would be a good topic for future research.

REFERENCES

[1] J. Bouvrie, Notes on Convolutional Neural Networks, 2006.

[2] X. Cai, X.-z. Gao, and Y. Xue, "Improved bat algorithm with optimal forage strategy and random disturbance strategy," *Int. J. Bio-Inspired Comput.*, vol. 8, no. 4, pp. 205–214, 2016.

[3] X. Cai, H. Wang, Z. Cui, J. Cai, Y. Xue, and L. Wang, "Bat algorithm with triangle-flipping strategy for numerical optimization," *Int. J. Mach. Learn. Cybern.*, vol. 9, no. 2, pp. 199–215, 2018.

[4] M. Christodorescu, S. Jha, S. A. Seshia, D. Song, and R. E. Bryant, "Semantics-aware malware detection," in *Proc. 2005 IEEE Symp. Security Privacy*, 2005, pp. 32–46.

[5] Z. Cui, Y. Cao, X. Cai, J. Cai, and J. Chen, "Optimal leach protocol with modified bat algorithm for big data sensing systems in internet of things," *J. Parallel Distrib. Comput.*, 2018, doi:10.1016/j.jpdc.2017.12.014.

[6] Z. Cui, B. Sun, G. Wang, Y. Xue, and J. Chen, "A novel oriented cuckoo search algorithm to improve dv-hop performance for cyber–physical systems," *J. Parallel Distrib. Comput.*, vol. 103, pp. 42–52, 2017.

[7] O. E. David and N. S. Netanyahu, "Deepsign: Deep learning for automatic malware signature generation and classification," in *Proc. 2015 Int. Joint Conf. Neural Netw.*, 2015, pp. 1–8.

[8] D. Díaz-Pernil, A. Berciano, F. Peña-Cantillana, and M. A. Gutiérrez-Naranjo, "Bio-inspired parallel computing of representative geometrical objects of holes of binary 2d-images," *Int. J. Bio-Inspired Comput.*, vol. 9, no. 2, pp. 77–92, 2017.

[9] H. Gao, Y. Du, and M. Diao, "Quantum-inspired glowworm swarm optimisation and its application," *Int. J. Comput. Sci. Math.*, vol. 8, no. 1, pp. 91–100, 2017.

[10] J. R. Goodall, H. Radwan, and L. Halseth, "Visual analysis of code security," in *Proc. 7th Int. Symp. Vis. Cyber Security*, 2010, pp. 46–51.

[11] K. Han, J. H. Lim, and E. G. Im, "Malware analysis method using visualization of binary files," in *Proc. 2013 Res. Adapt. Convergent Syst.*, 2013, pp. 317–321.

[12] T. Isohara, K. Takemori, and A. Kubota, "Kernel-based behavior analysis for android malware detection," in *Proc. 2011 7th Int. Conf. Comput. Intell. Security*, 2011, pp. 1011–1015.

[13] Y. Jia *et al.*, "Caffe: Convolutional architecture for fast feature embedding," in *Proc. 22nd ACM Int. Conf. Multimedia*, 2014, pp. 675–678.

[14] A. Khoshkbarforoushha, A. Khosravian, and R. Ranjan, "Elasticity management of streaming data analytics flows on clouds," *J. Comput. Syst. Sci.*, vol. 89, pp. 24–40, 2017.

[15] 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. Emerging Topics Comput.*, vol. 5, no. 1, pp. 120–133, 2017.

[16] 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.

[17] Q. Miao, R. Liu, Y. Quan, and J. Song, "Remote sensing image fusion based on shearlet and genetic algorithm," *Int. J. Bio-Inspired Comput.*, vol. 9, no. 4, pp. 240–250, 2017.

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

[19] L. Nataraj, S. Karthikeyan, G. Jacob, and B. Manjunath, "Malware images: visualization and automatic classification," in *Proc. 8th Int. Symp. Vis. Cyber Security*, 2011, Paper 4.

[20] 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.

[21] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural Netw.*, vol. 61, pp. 85–117, 2015.

[22] Symantec, Internet Security Threat Report, 2017.

[23] P. Trinius, T. Holz, J. Göbel, and F. C. Freiling, "Visual analysis of malware behavior using treemaps and thread graphs," in *Proc. 6th Int. Workshop Vis. Cyber Security*, 2009, pp. 33–38.

[24] M. Wagner *et al.*, "A survey of visualization systems for malware analysis," in *Proc. Eurographics Conf. Vis.*, 2015, pp. 105–125.

[25] G.-G. Wang, X. Cai, Z. Cui, G. Min, and J. Chen, "High performance computing for cyber physical social systems by using evolutionary multi-objective optimization algorithm," *IEEE Trans. Emerging Topics Comput.*, to be published.

[26] L. Wang *et al.*, "Particle swarm optimization based dictionary learning for remote sensing big data," *Knowledge-Based Syst.*, vol. 79, pp. 43–50, 2015.

[27] X.-S. Yang, "A new metaheuristic bat-inspired algorithm," *Nature Inspired Cooperative Strategies for Optimization (NICSO 2010)*, New York, NY, USA: Springer, pp. 65–74, 2010.

[28] Y. Ye, T. Li, D. Adjeroh, and S. S. Iyengar, "A survey on malware detection using data mining techniques," *ACM Comput. Surveys*, vol. 50, no. 3, 2017, Art. no. 41.

[29] I. Yoo, "Visualizing windows executable viruses using self-organizing maps," in *Proc. 2004 ACM Workshop Vis. Data Mining Comput. Security*, 2004, pp. 82–89.

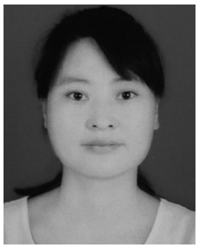
[30] Z. Yuan, Y. Lu, Z. Wang, and Y. Xue, "Droid-sec: Deep learning in android malware detection," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 44, pp. 371–372, 2014.

Zhihua Cui received the Ph.D. degree in control theory and engineering from Xi'an Jiaotong University, Xi'an, China, in 2008.

He is a Professor with the School of Computer Science and Technology, Taiyuan University of Science and Technology, Taiyuan, China. He is the Editor-in-Chief of *International Journal of Bio-inspired Computation*. His research interest includes computational intelligence, stochastic algorithm, and combinatorial optimization.

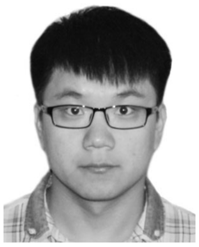
Fei Xue received the M.S. degree in computer application technology from Taiyuan University of Science and Technology, Taiyuan, China, in 2011, and the Ph.D. degree in computer science and technology from Beijing University of Technology, Beijing, China, in 2016.

He is a Lecturer with the School of Information, Beijing Wuzi University, Beijing, China. His current research interests include swarm intelligence and network security.



Xingjuan Cai received the Ph.D. degree in control theory and engineering from Tongji University, Shanghai, China, in 2017.

She is an Associate Professor with the School of Computer Science and Technology, Taiyuan University of Science and Technology, Taiyuan, China. Her research interests include bio-inspired computation and applications.



Yang Cao received the B.S. and M.S. degrees in computer science and technology from Taiyuan University of Science and Technology, Taiyuan, China, in 2011 and 2015, respectively. He is currently working toward the Ph.D. degree at College of Computer Science, Beijing University of Technology, Beijing, China.

His main research interests include swarm intelligence, network security, and big data analysis.

Gai-ge Wang (M'15) is an Associate Professor with the Ocean University of China, Qingdao, China. He has proposed five bio-inspired algorithms (monarch butterfly optimization, earthworm optimization algorithm, elephant herding optimization, moth search algorithm, and rhino herd). His research interests include swarm intelligence, evolutionary computation, and big data optimization.

Dr. Wang served as an Associate Editor of IJCISIM, and an Editorial Board Member of IJBIC from 2016.

Jinjun Chen (SM'13) received the Ph.D. degree in computer science and software engineering from Swinburne University of Technology, Melbourne, VIC, Australia.

He is an Associate Professor with the Faculty of Engineering and IT, University of Technology Sydney, Ultimo NSW, Australia. He is the Director of the Laboratory of Cloud Computing and Distributed Systems. His research interests include cloud computing, big data, workflow management, privacy and security, and various related research topics.