Received December 31, 2017, accepted January 28, 2018, date of publication February 12, 2018, date of current version April 4, 2018.

Digital Object Identifier 10.1109/ACCESS.2018.2805301

Malware Visualization for Fine-Grained Classification

JIANWEN FU, JINGFENG XUE, YONG WANG, ZHENYAN LIU, AND CHUN SHAN
School of Software, Beijing Institute of Technology, Beijing 100081, China

Corresponding author: Yong Wang (wangyong@bit.edu.cn)

This work was supported by the National Key Research and Development Program of China under Grant 2016YFB0801304.

ABSTRACT Due to the rapid rise of automated tools, the number of malware variants has increased dramatically, which poses a tremendous threat to the security of the Internet. Recently, some methods for quick analysis of malware have been proposed, but these methods usually require a large computational overhead and cannot classify samples accurately for large-scale and complex malware data set. Therefore, in this paper, we propose a new visualization method for characterizing malware globally and locally to achieve fast and effective fine-grained classification. We take a new approach to visualize malware as RGB-colored images and extract global features from the images. Gray-level co-occurrence matrix and color moments are selected to describe the global texture features and color features, respectively, which produces low-dimensional feature data to reduce the complexity of training model. Moreover, a series of special byte sequences are extracted from code sections and data sections of malware and are processed into feature vectors by Simhash as the local features. Finally, we merge the global features and local features to perform malware classification using random forest, K-nearest neighbor, and support vector machine. Experimental results show that our approach obtains the highest accuracy of 97.47% and the highest F-measure of 96.85% of 7087 samples from 15 families. Color features and the local features effectively assist in the classification based on texture features and enhance the F-measure by 3.4% and 1%, respectively. Overall, the combination of global features and local features can realize fine-grained malware classification with low computational cost.

INDEX TERMS Malware visualization, fine-grained classification, RGB-colored image.

I. INTRODUCTION

With the widespread use of automatic generation tools, a large number of new variants of malicious code has been generated rapidly. According to 2017 Internet security threat report of Symantec [1], 357 million new malware variants were identified in the last year, averaging more than 1 million per day. The huge amount of new variants become a big challenge for malware analysts.

Among the existing analysis methods, static analysis and dynamic analysis are the most used. Static analysis analyzes disassembled code without performing malicious samples. In static analysis, researchers typically extract opcodes [4], API sequences [13], [14] and function call graph [3], [10] from disassembled code as the original feature for analysis. Static analysis can quickly capture syntax and semantic information for thorough analysis, but this approach is easily disturbed by code obfuscation and encryption technology. Dynamic analysis usually analyzes behavioral information such as network activity [12], system calls [11], file operations and registry modification records [6] by executing samples in the monitored virtual environment. It is more robust, but the cost of time and resources for malware execution is expensive. Both static analysis and dynamic analysis have their own advantages and disadvantages, but in view of the time and resources required in dynamic analysis, static analysis is more advantageous for analyzing a large amount of malware.

In recent years, visualization-based approaches [19], [22], [26] have been proposed to directly analyze malware binaries for classification, which further improves the efficiency of malware analysis because no in-depth analysis is required. The reason that this kind of method works well is that most of the variants are generated by using automation technology or reusing some important modules, so they have some similarity in binary code. However, most of the existing methods of visualization are based on grayscale and evaluate similarity by texture, which can not work for malware that is evenly distributed on bytes. Moreover, almost all of them use only global features to characterize malware, resulting in a classification model that is not sufficiently stable against complex malware and can only apply on a small number of malicious samples with marked image features.

Therefore, in this paper, we present a novel malware visualization method, which combines global features and local features to characterize and classify malware. For global features, we create a new method to visualize malware as a RGB-colored image and extract texture features and color features. The RGB-colored images can demonstrate the similarities between families, but not obvious for some variants within the family. Thus, some sequences of the continuous byte values which can be visualized string through ASCII conversion are considered to be as local feature to reveal similarities of these variants. Experimental results show our approach obtains high classification accuracy and maintains high performance with the way of combining the global features and the local features. Our method mainly contributes to providing a new malware visualization method, a malware local feature extraction method and a malware classification method that can achieve high efficiency, high accuracy for a variety of complex malware in practice.

The rest of this paper is organized as follows. Some related researches in malware visualization and classification are presented in Section II. Section III provides an overview of our approach. Section IV describes the malware visualization method and the malware classification process. Section V shows the results and analysis of malware visualization and classification. Finally, the limitations and future work are discussed in Section VI, and conclusions are presented in Section VII.

II. RELATED WORK

At present, most of the traditional malware analysis methods are based on static analysis and dynamic analysis. Kang *et al.* extracted opcode from disassembled file and organized them into feature vectors using N-gram in [4]. Besides, they compared the classification accuracy on different length of opcode. The results indicated that opcode is a valid feature for malware classification, and the short opcode is better. Imran *et al.* [2] proposed a similarity based classification method. They extracted API sequences to train the Hidden Markov Models (HMMs) to evaluate resemblance between training sequences and testing sequences. The relevance scores were computed to classify malwares. This method can effectively utilize API sequence information, but it requires a lot of computational overhead to train the HMM models. Iwamoto and Wasaki [5] also extracted API sequences, but instead of calculating the similarity directly, they converted the API sequences to function call graphs(FCG) and shrank those graphs. The method takes into account the calling relationship between the APIs to make them more differentiated. These methods enable efficient classification, but them are susceptible to code obfuscation. So several dynamic analysis methods such as system calls analysis and network activity analysis were proposed. Xu *et al.* [9] extracted system calls and represented them with several methods: system call histogram, N-gram and Markov Chain. Kim *et al.* [7] collected system calls in the process of malware execution. The system calls were clustered and generated behavior chain to characterize the common pattern of a family. Malware classification were performed through calculating similarity between the behavior sequence chains and system calls of a testing malware. While, some different ideas in the dynamic behavior analysis were presented. In [8], application layer protocols were extracted and summarized in a graph with the way of protocols as graph's nodes and the commonalities between them as edges. Then, a graph distance measure method was used to compute similarities between network activity graphs. Nari and Ghorbani [12] also converted network behavior to a graph, but they selected statistical properties of graphs such as graph size and graph degree as the graph features. Except network activity, Cabau *et al.* [6] used the changes made to filesystem and registry keys to conduct dynamic analysis.

With the development of image processing technology, some visualization-based methods have been proposed for malware analysis. In the beginning, visualization techniques were usually applied to traditional static features [15] and dynamic features [16], [21]. Zhang *et al.* [19] converted opcode sequences to binary images. The results proved the method has good accuracy for a small training set. Han *et al.* [23] also proposed a way to visualize opcode sequences, but their opcode extraction methods included disassembly and dynamic execution, which make the methods are effective for packers and encryption malware. While, Trinius *et al.* [17] used tree map and thread graph to visualize the overall behaviors and individual thread, respectively. [16], Saxe *et al.* [16] chose to visualize system call log, they generated Markov chains from system calls and used them to compute similarity matrix. Shaid and Maarof [21] proposed a method of assigning color to malicious APIs based on their maliciousness degree and used it to convert behavioral information into images for classification. In addition, some novel approaches [18], [24] of visualizing the entire malware as images has been proposed. Nataraj *et al.* [26] initially used visual methods for malicious code classification. They converted malware binary code into gray image and extract texture features using GIST [32], [33] to classify them. They obtained high classification of 98% and proved their method is faster than n-gram. Xiaofang *et al.* [18] visualized malwares as gray images and extracted image feature with SURF. The research of Liu and Wang [20] also focused on gray image, and the local mean method was used to reduce the image size to speed up ensemble learning process. Han *et al.* [22] proposed a malware visualization method, entropy graph generated based on gray image were used to realize automatic analysis. But this method can not be applied to packed malware, because the entropies of packed malware usually are very high and can not indicate any specific pattern.

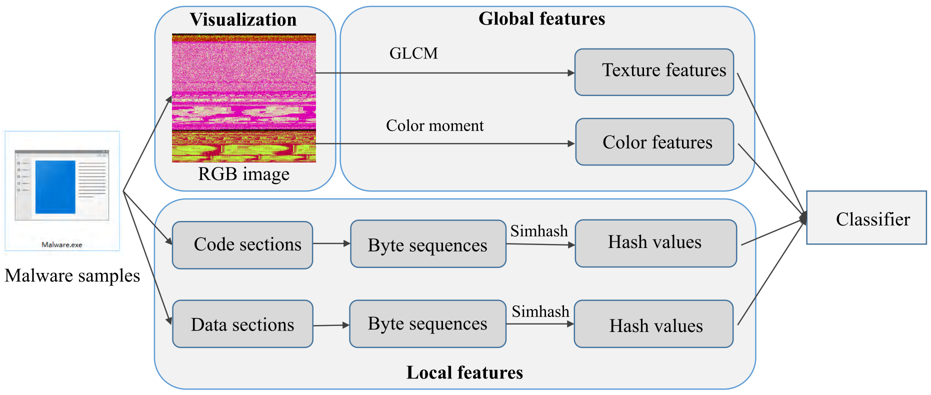


FIGURE 1. Overview of the proposed method.

Different from the traditional static and dynamic analysis, visualization method does not require complex disassembly and time-consuming execution process, and the difficulty of visualization is not affected by the amount of malware. This greatly increases the processing efficiency so that it can be applied to large-scale malware classification. For instance, for the traditional analysis method in [4] and [7], the time it takes to get and handle opcode or system calls will be unacceptable when the amount of malware is huge. Compared with other visualization methods, our method avoids their shortcomings. In addition to transforming gray values into texture features like [26], we add color feature based on entropy values and relative sizes of malware sections, which makes our approach more robust. The addition of the local features increases the cohesion of samples within the same family, which can reduce misclassification among similar families. In the end, the combination of global features and local features can be relatively complementary to further improve the classification accuracy.

III. MECHANISM OVERVIEW

Our approach is composed of three steps which are malware visualization, feature extraction and malware classification. The specific process is shown in Fig. 1. In the process of malware visualization, a new RGB-colored image generation method was proposed to solve the problem of insufficient information in grayscale images. We did not consider looking for a way to convert grayscale to RGB-color directly, but to populate the red, green and blue channels of RGB-colored images with more useful information. Byte values can directly reflect the nature of malware, but are subject to change, so we added the entropy and relative size of malware sections to enhance the stability of malware images. This method makes the difference between sections more obvious, and the malware easier to distinguish their families for helping to categorize. In order to enhance the robustness of the method when dealing with the complex dataset, we decided to find local features from the code and data sections of malware. We focused only on the sequences of consecutive byte values that can be converted to strings, because these byte sequences usually represent key information such as string constants, API calls, and DLLs, which indicates there is a high probability that they can be inherited stably in the variant production process. For the feature extraction, we extracted the global features from RGB-colored images and extracted the local features from the sections of malware, respectively, and then merged them. Gray Level Co-occurrence Matrix (GLCM) was used to extract texture features because its computational complexity is lower than other algorithms. Color moment is a simple and effective method and was used to describe color features. For local features of images, spots and corners are usually extracted as feature points, but these feature points are not meaningful for malware. So we chose byte sequences that can be converted ASCII strings as local features, and arranged them into feature vectors. As for last classification, three different types of classifiers RF (Random Forest), KNN (K-Nearest Neighbor), and SVM (Support Vector Machine) were selected to classify malware.

IV. MALWARE VISUALIZATION AND CLASSIFICATION

A. MALWARE VISUALIZATION

The core steps of malware visualization are section division and feature computation. First of all, malicious code were filtered to ensure that samples follow PE format and retain the original structure. Then, malware was divided into several sections according to the PE format and characterized these sections using entropy, byte value and relative size. The red channel, green channel, and blue channel of each pixel were filled by these values and combined into a RGB-colored image at the end. The detailed process is shown in Fig. 2.

Malicious code filtering is a prerequisite to ensure the effectiveness of the method. Non-PE files are difficult to divide and keep structural information, so they are temporarily not considered in this paper. Malware was judged whether it is a PE file by parsing it according to the PE format. The PE file structure and the required field information for filtering malware are shown in Fig. 3. There are two main field

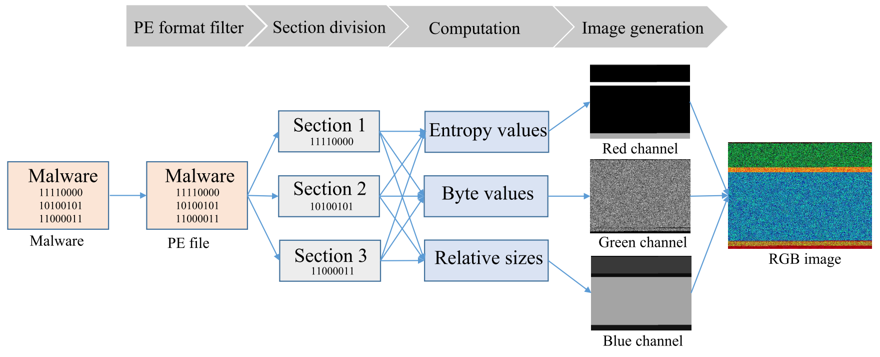
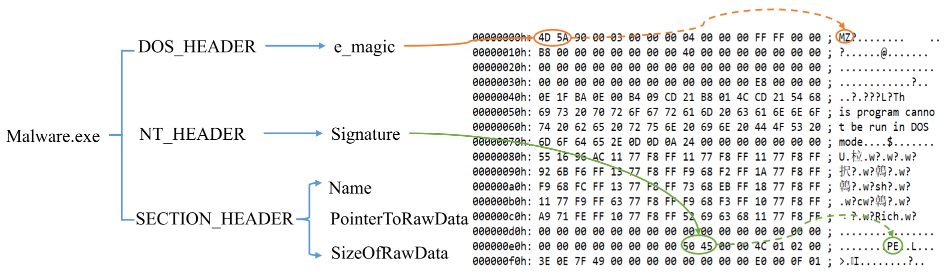


FIGURE 2. The process of malware visualization.



```
00000000h: 4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00 ; MZ?........    ..
00000010h: B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00 ; ?.......@.......
00000020h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000030h: 00 00 00 00 00 00 00 00 00 00 00 00 E8 00 00 00 ; ............?..
00000040h: 0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68 ; ..?.???L?Th
00000050h: 69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F ; is program canno
00000060h: 74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20 ; t be run in DOS
00000070h: 6D 6F 64 65 2E 0D 0D 0A 24 00 00 00 00 00 00 00 ; mode....$.......
00000080h: 55 16 96 AC 11 77 F8 FF 11 77 F8 FF 11 77 F8 FF ; U.粒.w?.w?.w?
00000090h: 92 6B F6 FF 13 77 F8 FF F9 68 F2 FF 1A 77 F8 FF ; 抶?.w?鶫?.w?
000000a0h: F9 68 FC FF 13 77 F8 FF 73 68 EB FF 18 77 F8 FF ; 鶫?.w?sh?.w?
000000b0h: 11 77 F9 FF 63 77 F8 FF F9 68 F3 FF 10 77 F8 FF ; .w?cw?鶫?.w?
000000c0h: A9 71 FE FF 10 77 F8 FF 52 69 63 68 11 77 F8 FF ; ?.w?Rich.w?
000000d0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
000000e0h: 00 00 00 00 00 00 00 00 50 45 00 00 4C 01 02 00 ; ........PE..L...
000000f0h: 3E 0E 7F 49 00 00 00 00 00 00 00 00 E0 00 0F 01 ; >.I........?..
```

FIGURE 3. The structure of PE file.

"e_magic" and "Signature" need to be validated for PE file. The field "e_magic" is in "DOS_HEADER" of PE format and usually locate the first byte of malware. If the malware follows the PE format, the value of the field "e_magic" must be 0x4D5A (hexadecimal) and the corresponding ASCII string is "MZ". In the meantime, the value of the field "Signature" in "NT_HEADER" must be 504500 (hexadecimal) and the corresponding ASCII string is "PE". More details on PE structure and the corresponding parse methods can be found in [27]. In the process of section division, the original section structure of the PE file was preserved. The code section, data section and other natural sections were still viewed completed sections, but the rest regions such as the Dos header, the PE header and the additional data at the end of file were merged according their locations. After section division, the entropy and relative size were computed to represent these section because they can demonstrate high-level characteristics and reflect the similarities of the malware in general.

Entropy can represent chaotic extent of byte values in the section. When the variants change less, the entropies of their sections are almost the same. Entropy can be calculated as:

$$SectionEntropy = -\sum_{i=0}^{255} p(c_i)\log_2 p(c_i) \tag{1}$$

where $p(c_i)$ is the probability of occurrence of byte value i. When all of byte values in a section are the same, the entropy obtains the minimum value of 0; in contrast, the entropy

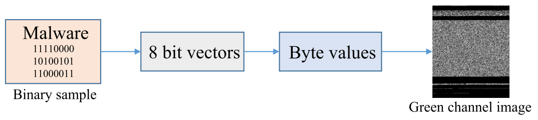


FIGURE 4. The process of filling green channel of RGB-colored image.

obtains the maximum value of 8 when all the byte values are different. But the range of 0 to 8 is so small that the grayscale may not be observed clearly by the human eye. Thus, entropy was magnified by 31.875 times for better visualization of red channel images.

$$\begin{aligned} RedComponent &= SectionEntropy \times \frac{255}{8} \\ &= (-\sum_{i=0}^{255} p(c_i)\log_2 p(c_i)) \times 31.875 \end{aligned} \tag{2}$$

We used the visualization method in [26] to fill green channel of the RGB-colored image, as shown in Fig. 4. Malware binaries are composed of a series of 0 and 1. A byte contains 8 bits (8 binary numbers), which can be converted into a decimal number (byte value) in the range of 0 to 255. Each byte value was used to represent a gray level (0 represents black, 255 represents white, and other values represent different degrees of grey). At last, byte values were organized into a two-dimensional matrix and then visualized as an image. The width of the image was determined by the size of the file and the height was changed accordingly. The size is a basic way to describe section, but the relative size of section to whole file is

TABLE 1. Byte ranges and the corresponding colors.

ASCII type	Byte range	Color	RGB
Number	[48, 57]	Blue	(0, 0, 255)
Upper case letter	[65, 90]	Green	(0, 255, 0)
Lower case letter	[97, 122]	Red	(255, 0, 0)
Special character	[32, 47], [58, 64], [91, 96], [123, 126]	Yellow	(255, 255, 0)
Other	[0, 31], [127, 255]	Black	(255, 255, 255)

more appropriate for comparison. The relative size also need to enlarge 255 times to be a pixel of the blue channel image by the equation as below.

$$BlueComponent = \frac{SectionSize}{FileSize} \times 255 \tag{3}$$

Before extracting texture features, RGB-colored images need to be converted to gray images. In the conversion process, the different proportion of the three channels of red, green, and blue were given, which was equivalent to assign weights for entropies, byte values and relative sizes. A famous transformation method to convert RGB color to grayscale are shown as below.

$$Gray = R * 0.299 + G * 0.587 + B * 0.114 \tag{4}$$

where R, G and B represent red component, green component and blue component separately. As for their match relationship, we assigned entropy, byte values and relative size to R, G and B respectively, which meant byte values obtained the maximum weight of 0.587 and relative size obtained minimum weight of 0.114. In the section III, we mentioned that we extracted local features from special byte sequences in code sections and data sections. The each byte value in the sequences must be guaranteed it is between 32 and 126 (decimal numbers), because which can be converted into ASCII string only in this range. Moreover, the local features were extracted from continuous byte sequences, because these sequences contain some special information. For instance, the string constants, Dynamic Link Library(DLL) names, and system call function names are string type and usually converted binary through ASCII and then stored in code or data sections. These strings are more likely to be retained when malware changes and can play a import role in the classification of samples from similar families. So we can display these sequences as images and compare them to verify the effectiveness of local features. In order to illustrate the local features, we proposed a new technique to visualize byte sequences according to their types of corresponding ASCII strings. The method divided the byte values into five ranges, each of them was represented by a kind of color. The detailed relationships are as shown in Table 1. To clearly show the differences between colors, we did not use the byte value to control the depth of the color, but only determined the basic color according the ASCII type. Fig. 5 is an example of visualization result of code sections, the lengths of sequences in the image are limited to more than 6 bytes.

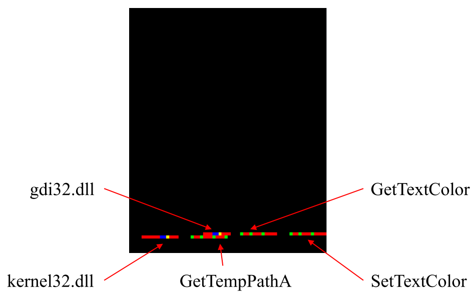


FIGURE 5. The code section image of Trojan.Win32.Buzus.aayv.

B. FEATURE EXTRACTION

1) GLOBAL FEATURES

The next step after malware visualization is to extract features for malware classification. Image feature extraction generally includes two methods: one is to extract the global features from the whole image; the other is to extract the local feature points and then describe them with appropriate features. Global features generally describe texture, color, shape, and space of the image. By analyzing the characteristics and contained information of maware images, we thought texture features and color features were more appropriate and sufficient as the global features of malware. Common used texture feature extraction algorithms include Gray Level Co-occurrence Matrix (GLCM), Local Binary Pattern (LBP) and Gabor transformation, whereas GLCM is the best fit for our needs of low computational complexity. Color Moment was selected to extract color features because of it has more efficient expression to color distribution and lower feature dimension when compare with the methods of color histogram, color set, and color correlogram.

a: TEXTURE FEATURES

Gray Level Co-occurrence Matrix (GLCM) was proposed by Haralick *et al.* [34]. It is based on the assumption that the spatial distribution of pixels in an image contains image texture information. Co-occurrence Matrix is the joint probability distribution of two gray pixels at the distance of d in the image. GLCM can reflect the integrated information of grays from aspects of direction, adjacent interval, and variation amplitude. The direction (θ), offset (d), and gray level are the three important parameters of GLCM. The direction of GLCM refers to the change direction of grayscale, we selected four directions: 0°, 45°, 90°, and 135°, which contains the main direction of texture changes. Offset is the distance between two gray pixels, two adjacent pixels represent they are in a given direction and offset is 1. The gray level actually is the maximum of grayscale in a image and plus one, which is used in grayscale compression. GLCM mainly contains three steps: grayscale compression, co-occurrence matrix generation and feature calculation.

Fig. 6 shows the feature extraction process of GLCM with direction of 0 degrees, offset of 1 and gray level of 3. The first step was to convert the RGB-colored image to

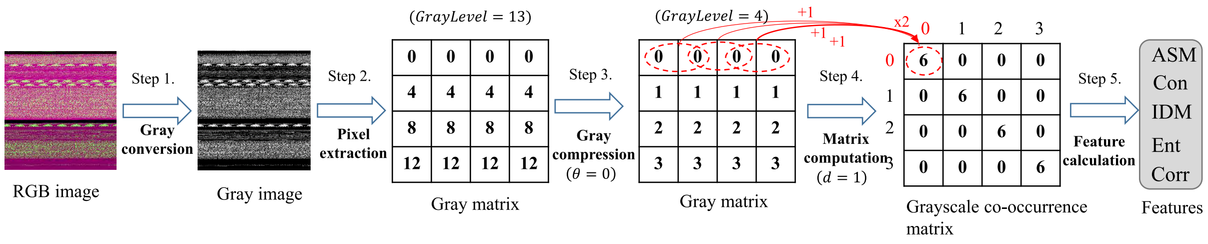


FIGURE 6. **The process of extracting texture features by GLCM.**

grayscale image. The related method has been introduced in Section IV-A. Grayscale compression was used to reduce the matrix dimension and the amount of computation. In Fig. 6, the matrix dimension was compressed from 12×12 to 4×4, and the amount of computation was reduced by around 10 times. The way of generating the co-occurrence matrix was to find out the number of pixel pairs corresponding to the position (i, j) for each matrix element. The number should be multiplied by 2, because it needs to be recalculated from the opposite direction. The co-occurrence matrices were generally not directly treated as texture features because the elements of matrices are so many. Therefore, Haralick *et al.* proposed 14 kinds of statistics to represent texture features, which were calculated based on the co-occurrence matrix. But only 5 of them are common used, which include *ASM (Angular Second Moment)*, *Entropy*, *Contrast*, *IDM (Inverse Different Moment)*, and *Correlation*. So we can speculate the dimension of the texture feature to be 20 (4 directions $\times$ 5 indicators). The calculation formulas are as follows (M is the co-occurrence matrix):

1) ASM (Angular Second Moment)

$$ASM = \sum_i \sum_j M(i,j)^2 \tag{5}$$

ASM is the sum of the squares of each matrix element. The ASM reflects the uniformity of grayscale distribution and the coarseness of texture. If all the values of the co-occurrence matrix are equal, the ASM will be small. Otherwise, if some of the values are great and the other values are small, the ASM will be large. Large ASM represents a more homogeneous and regularly changing texture pattern.

2) Entropy

$$Ent = -\sum_i \sum_j M(i,j) \log M(i,j) \tag{6}$$

Entropy represents the degree of non-uniformity or complexity of the texture in the image. When the pixels are close to random or the image has a lot of noise, the entropy will be large. The larger the entropy, the more complicated the image.

3) Contrast

$$Con = \sum_i \sum_j (i-j)^2 M(i,j) \tag{7}$$

Contrast directly reflects the contrast of the brightness of a pixel value and its neighboring pixel values and indicate the clarity of the image and the groove depth of texture. The deeper the groove of the texture, the greater the contrast and the texture looks more clear. On the contrary, the smaller the contrast, the more shallow groove and the texture looks more blurred.

4) IDM (Inverse Differential Moment)

$$IDM = \sum_i \sum_j \frac{M(i,j)}{1+(i-j)^2} \tag{8}$$

IDM reflects the homogeneity of image texture and measures the local change of image texture. A large IDM indicates that there is no change among different regions of the image and the local uniformity must be very high. While, if the elements in co-occurrence matrix diagonal contain a large value, IDM will take a large value.

5) Correlation

$$\begin{aligned}
Corr &= \sum_i \sum_j \frac{(i,j)M(i,j) - \mu_i \mu_j}{s_i s_j} \\
\mu_i &= \sum_i \sum_j i \cdot M(i,j) \\
\mu_j &= \sum_i \sum_j j \cdot M(i,j) \\
{s_i}^2 &= \sum_i \sum_j M(i,j)(i-\mu_i)^2 \\
{s_j}^2 &= \sum_i \sum_j M(i,j)(j-\mu_j)^2
\end{aligned} \tag{9}$$

The correlation reflects the direction of the texture and represents the extended length of the gray value along a certain direction, which is common used to assess the similarity between row elements and column elements in the co-occurrence matrix. The longer the grayscale extends, the greater the correlation.

b: COLOR FEATURES

Color moment [28] is a simple and effective description method color feature, which was proposed by Stricker *et al.* in 1995. It usually contains the first moments (mean), the second moments (variance) and the third moments (skewness).

The color information is mainly distributed in the low-order moments, so the first moment, second and third moments are enough to express the color distribution of the image. Compared with other description methods of color features, the color moments do not require color space quantization and the generated feature vectors are low-dimensional. The color moments of the image only generate a total of 9 components (3 color components and 3 low-order moments per component). The specific formulas of moments are as follows:

1) The first moment

$$\mu_i = \frac{1}{N}\sum_{j=1}^{N} p_{i,j} \tag{10}$$

2) The second moment

$$\sigma_i = (\frac{1}{N}\sum_{j=1}^{N}(p_{i,j} - \mu_i)^2)^{\frac{1}{2}} \tag{11}$$

3) The third moment

$$s_i = (\frac{1}{N}\sum_{j=1}^{N}(p_{i,j} - \mu_i)^3)^{\frac{1}{3}} \tag{12}$$

where N is the number of pixels in the image, $p_{i,j}$ is the j-th pixel of the i-th color channel. μ_i denotes the mean of all the pixels on the i-th color channel, σ_i denotes the standard deviation of all the pixels on the i-th color channel, and s_i denotes the cubic root of the skewness of all the pixels on the i-th color channel. The first moment reflects the brightness of the image, the second moment reflects the color distribution range, and the third moment reflects the color distribution symmetry. The color moments of the three components of red, green, and blue form a 9-dimensional vector, expressed as follows:

$$F_{color} = [\mu_R, \sigma_R, s_R, \mu_G, \sigma_G, s_G, \mu_B, \sigma_B, s_B] \tag{13}$$

2) LOCAL FEATURES

Before extracting the local features, we need to get the code and data sections. Since sections of the PE file are named differently, we defined the code sections include ".text", "CODE", and ".code" named sections, and data sections include ".data", "DATA", ".rdata", ".idata", and ".edata" named sections. After that, we extracted byte sequences which can be converted into ASCII string (the byte range is 32 to 126) and processed them into feature vectors with Simhash. Simhash [29] was proposed by Charikar in 2002, and it has been used to remove duplication of hundreds of millions web pages by Google (see [30]). This hashing approach ensures that similar texts still have similarities after hashing and drastically reduce the data dimension. Simhash is a kind of LSH (Locality Sensitive Hash). Its main idea is dimensionality reduction. The high dimensional feature vectors are mapped into low dimension feature vectors and still retain the similarity. The Hamming Distance of two vectors is usually used to measure the similarity. Instead of calculating the Hamming distance, our method directly uses the last-generated feature vector as the local features. The detailed process of extracting and processing local features consists of five steps, as described below:

1) Extract sequences of consecutive byte values, each byte value in the sequence can be converted to a ASCII character (byte value is in [32,126]). The length of the sequence is used as the weight of the sequence.
2) Use a traditional hash algorithm (such as MD5) to hash each sequence to generate a n-bit vector of 0 and 1. Hash algorithm must ensure that the hash values of different sequence are different.
3) Weighting each element S_i in the vector. If S_i is 0, the weighed result takes a negative number $-W_i$, whereas if S_i is 1, the result takes a positive number W_i.
4) For all weighted vectors, the elements are summed into an n-bit vector according to their positions.
5) Traverse all the elements of the cumulative result, then set 1 at the corresponding position if the element is greater than 0, otherwise set to 0. The final result is a feature vector and as our local features.

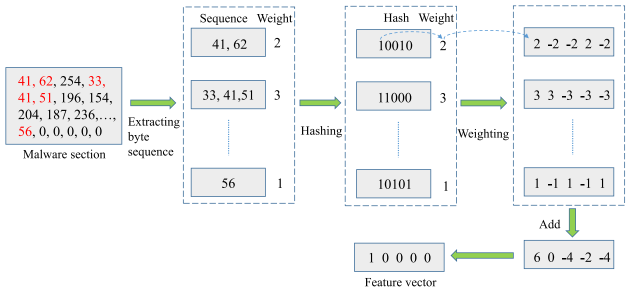


FIGURE 7. The process of extracting and processing local features.

An example of extracting local features is shown in Fig. 7. A sequence of consecutive byte values $n(32 \leqslant n \leqslant 126)$ was treated as a key portion of the malware section, like the feature point of local feature extraction in image processing. The longer the sequence, the more likely it is to be in a locally stable region (such as a file path, Email address), so the corresponding weight should be larger. The weighted result selected plus or minus the corresponding weight based on 1 or 0 of the hash value element. The final feature vector was a n-bit vector consisting of 0 and 1, which was converted by weighted result. n is determined by the traditional hash method. In our approach, we used 64-bit MD5 as the hash algorithm, so the local features has a dimension of 128 (64 for data sections and 64 for code sections).

V. EXPERIMENT

A. EXPERIMENTAL DATA

In order to make the experimental result more convincing, we used a large-scale dataset containing 7087 malware samples from 15 families. The details of malware samples are

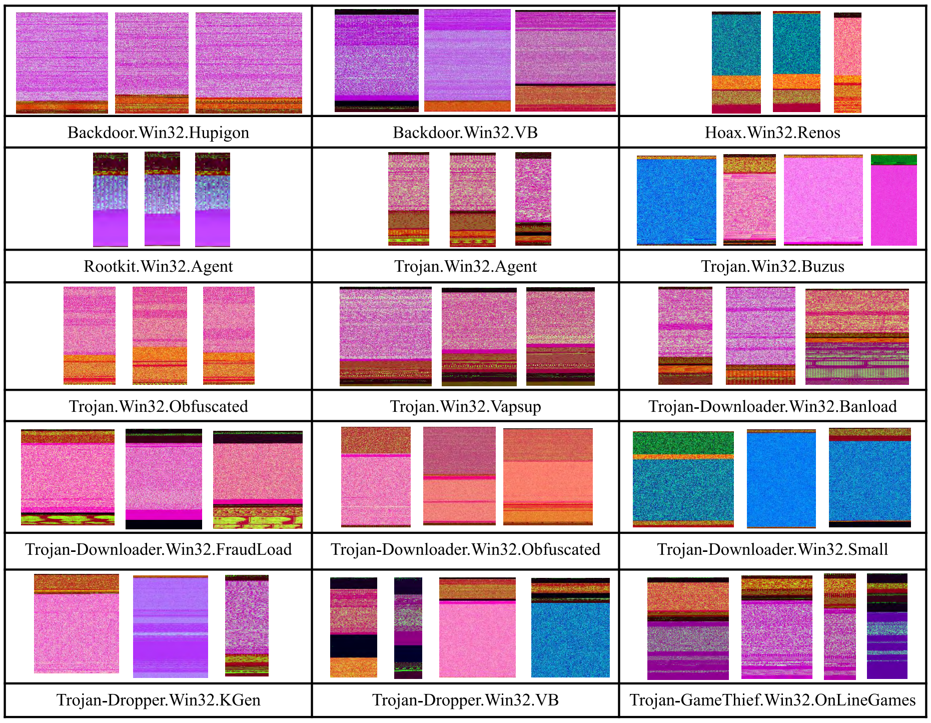

Backdoor.Win32.Hupigon
Backdoor.Win32.VB
Hoax.Win32.Renos
Rootkit.Win32.Agent
Trojan.Win32.Agent
Trojan.Win32.Buzus
Trojan.Win32.Obfuscated
Trojan.Win32.Vapsup
Trojan-Downloader.Win32.Banload
Trojan-Downloader.Win32.FraudLoad
Trojan-Downloader.Win32.Obfuscated
Trojan-Downloader.Win32.Small
Trojan-Dropper.Win32.KGen
Trojan-Dropper.Win32.VB
Trojan-GameThief.Win32.OnLineGames

FIGURE 8. **Malware visualization results of 15 families.**

TABLE 2. **Malware dataset of 15 families.**

#	Class	Family	Quantity
1	Backdoor	Hupigon	807
2	Backdoor	VB	306
3	Hoax	Renos	210
4	Rootkit	Agent	126
5	Trojan-Downloader	Banload	441
6	Trojan-Downloader	FraudLoad	487
7	Trojan-Downloader	Obfuscated	795
8	Trojan-Downloader	Small	475
9	Trojan-Dropper	KGen	527
10	Trojan-Dropper	VB	435
11	Trojan-GameThief	OnLineGames	472
12	Trojan	Agent	297
13	Trojan	Buzus	286
14	Trojan	Obfuscated	552
15	Trojan	Vapsup	871

shown in Table 2. The dataset includes four categories, which are Backdoor, Hoax, Rootkit and Trojan. While the Trojan and its subclasses of Trojan-Downloader, Trojan-Dropper, Trojan-GameThief are still divided into several similar families. For example, families of Banload, Frauload, Obfuscated and Small are included in a big family of Trojan-Downloader. In general, samples from such families are highly similar and difficult to classify. But our method has been solved for this problem and achieved fine-grained classification through the global features and local features complement each other.

B. VISUALIZATION RESULTS

Nataraj *et al.* [26] visualized malware as grays images and found that image textures of the same families are particularly similar, and there is a big difference between different families. In our method, we expanded grayscale images to RGB-colored images and found the visualization results of our samples are a litter different from the results of Nataraj *et al.* As shown in Fig. 8, we put the image on the left represent the image has more similar images in that family. We can see that the images in a family are not always similar, the images may display several styles, such as families of Trojan.Win32.Buzus, Trojan-Downloader.Win32.Banload and Trojan-Dropper.Win32.VB. This means that it is hard to separate them accurately with a single feature. Thus, we combined multiple features to complement each other to improve the classification accuracy. Moreover, we can clearly observe the facilitation of features combination through visualization.

For our dataset, the texture features are not enough to classify samples very well. As we can see in Fig. 8, the textures of some malware images in some families such as Trojan.Win32.Buzus, Trojan-Downloader.Win32.Small and Trojan-Dropper.Win32.VB are very similar, but the color features clearly distinguish them. The Fig. 9 gives two specific examples to show the role of color features. The first group images are from the family Backdoor.Win32.VB, their textures look different, but the colors display similarity. The images of second group belong to different families look almost the same from the texture, but it is easy to distinguish

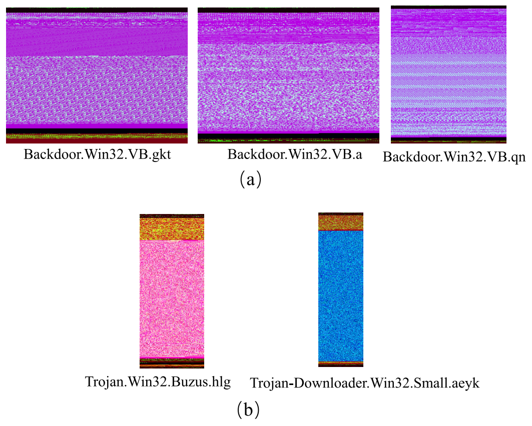


FIGURE 9. Comparison of the color and texture from the same family (a) and different family (b).

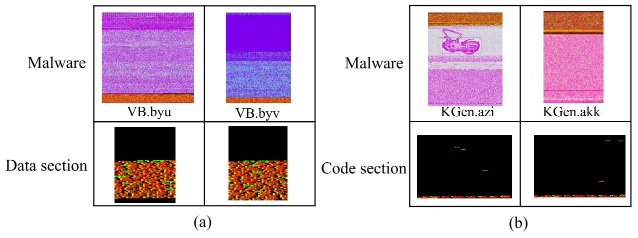


FIGURE 10. Comparison of the whole image and section images.

for the color. In the meantime, for images having fine textures or no obvious textures, using textures alone will result in poor accuracy. However, the color feature can make the above two kinds of images correctly classified to improve the accuracy.

In addition, the visualization of the sections corroborates the effectiveness of extracting local features. The Fig. 10 shows some examples where the textures and colors of the whole images are different but image of the data section or code section are similar. In fact, variants producers often add or modify icons, sounds and other resource to cheat users in different scenarios, but usually keep the core code and data. This allows malware sections to reduce false positives resulting from inconsistent RGB-colored image in overall. Further said that local features can contribute to the accuracy of classification.

C. CLASSIFICATION RESULTS

After the feature extraction, we obtained a low-dimensional feature set composed of 20 texture features, 9 color features and 128 hash features. Moreover, the dimensions of the features will not change according to the data set, which greatly reduces the complexity of the model and improves the usability. For the final classification, we conducted 10-fold cross-validation using three classifiers of RF (Random Forest), KNN (K-Nearest Neighbor), and SVM (Support Vector Machine). In order to minimize the effect of unbalanced data sets on the results, we used stratified folds to ensure that the proportion of each family of training and test sets remains the same. We repeated the classification 100 times and calculated the average accuracy, precision, recall, and F-measure as the classifier evaluation method. The classification results are shown in Table 3. The combination of the global features, local features and RF classifier result in the highest values of accuracy, precision, recall, and F-measure of 0.9747, 0.9711, 0.9672 and 0.9685. The formulas of accuracy, precision, recall and F-measure are as follows:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{14}$$

$$Precision = \frac{TP}{TP + FP} \tag{15}$$

$$Recall = \frac{TP}{TP + FN} \tag{16}$$

$$\begin{aligned} F - measure &= 2 \times \frac{Recall \times Precision}{Recall + Precision} \\ &= \frac{2TP}{2TP + FP + FN} \end{aligned} \tag{17}$$

The TP, TN, FP and FN usually are defined in the binary classification. In this paper, for the family A, the definition of TP, TN, FP and FN in the formulas are shown as follows:

- TP (True Positive) is the number of samples that are labeled as family A and do belong to family A.
- TN (True Negative) is the number of samples that are not labeled as family A and do not belong to family A.
- FP (False Positive) is the number of samples that are labeled as family A but do not belong to family A.
- FN (False Negative) is the number of samples that are not labeled as family A but do belong to the family A.

We performed all malware classification experiments on three classifiers: RF (Random Forest), KNN (K-Nearest Neighbor), and SVM (Support Vector Machine). The classification results in Table 3 show RF is superior to KNN and SVM for all feature combinations except the local features. For local features, the results of SVM are much better than other two classifiers. For the combination of all features, the F-measure of RF is 1.69% higher than that of KNN and 2.55% higher than that of SVM. From the final classification results, RF is the best classifier among RF, KNN, and SVM.

By comparing classification results of different feature combination, we can conclude that the combination of global and local features is superior to individual global features or local features. Since the data set is imbalanced, we select F-measure as the basis for comparison of features combination methods. For single feature, the color features get the highest F-measure, which is supported by results of RF and KNN classifiers. In results of RF, color features obtain high F-measure of 0.9550, which is 3.31% higher than that of texture features and 13.02% higher than that of the local features. This result proves that color features play a major role in the classification. In addition, even though the local features achieves the smallest F-measure, it increases the F-measure of the global feature (the combination of texture features and color features) by 1%. It has been proved that

TABLE 3. Classification results.

Type	Feature	Accuracy			Precision			Recall			F-measure		
		RF	KNN	SVM	RF	KNN	SVM	RF	KNN	SVM	RF	KNN	SVM
G	T	0.9278	0.8737	0.7895	0.9250	0.8713	0.7957	0.9209	0.8650	0.7751	0.9219	0.8650	0.7749
	C	0.9628	0.9458	0.8521	0.9575	0.9405	0.9461	0.9540	0.9484	0.8264	0.9550	0.9486	0.8566
	T+C	0.9653	0.9565	0.9124	0.9607	0.9513	0.9470	0.9576	0.9493	0.8977	0.9585	0.9495	0.9089
L	L	0.8595	0.8729	0.9087	0.8365	0.8506	0.8922	0.8281	0.8434	0.8804	0.8248	0.8355	0.8818
G+L	**T+C+L**	**0.9747**	**0.9623**	**0.9523**	**0.9711**	**0.9548**	**0.9520**	**0.9672**	**0.9502**	**0.9421**	**0.9685**	**0.9516**	**0.9430**

G means global features, L means local features, G+L represents a combination of global features and local features.
T means texture features, C means color features.

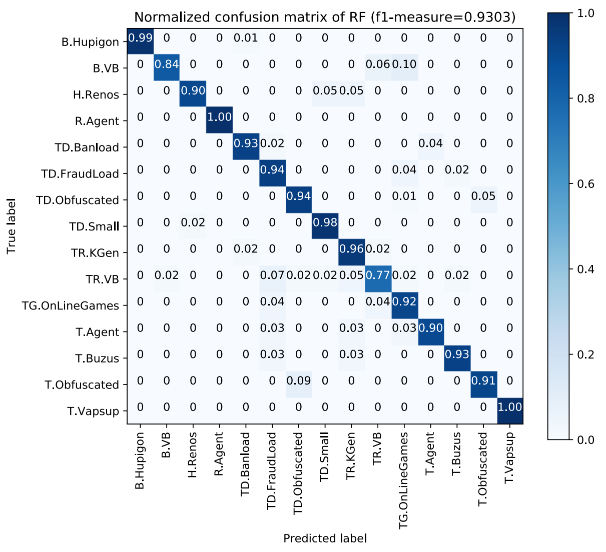


FIGURE 11. **The confusion matrix of texture features.**

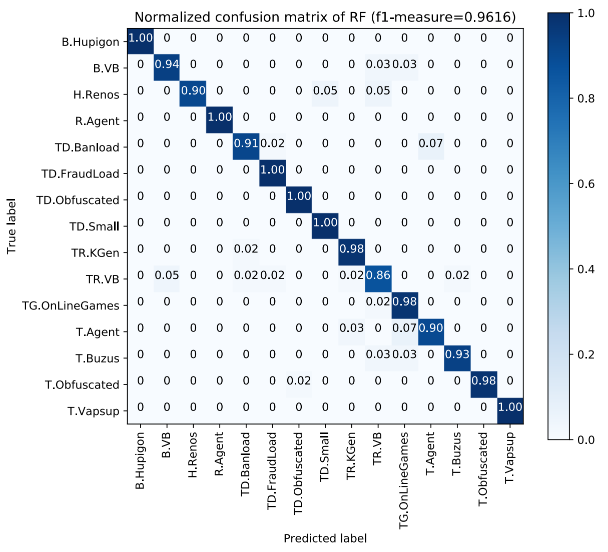


FIGURE 12. **The confusion matrix of combination of texture and color features.**

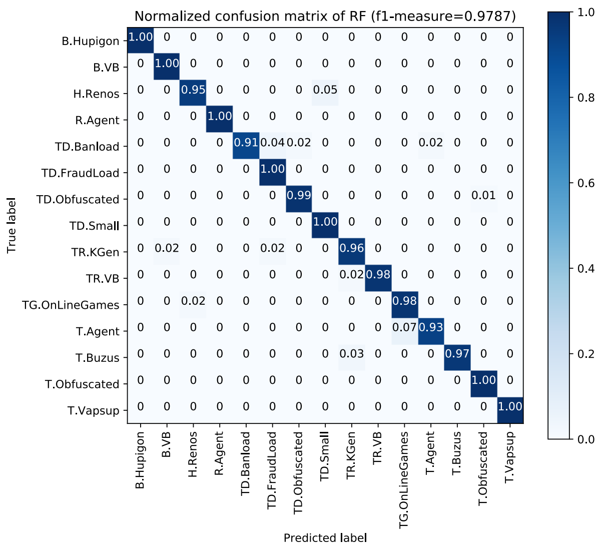
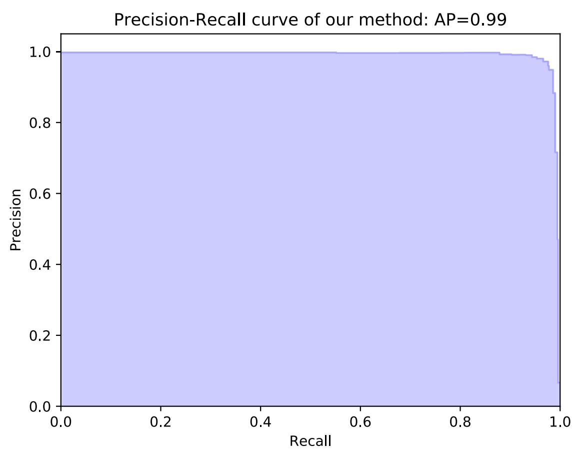


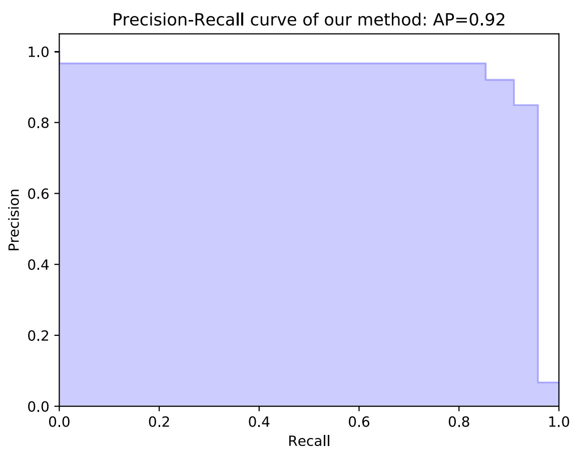
FIGURE 13. **The confusion matrix of combination of texture, color and hash features.**

the addition of local features obviously enhances the effect of classification.

In order to further demonstrate the role of local features, we generated confusion matrixes of the different combinations of texture features, color features, and local features. We randomly picked the training set of 0.9 and the test set of 0.1, and ensured that the proportions of each family in the training set and test set were the same. In Fig. 11, the accuracies of test samples of Backdoor.Win32.VB and Trojan-Dropper.Win32.VB families are under 0.9. However, only accuracy of family Trojan-Dropper.Win32.VB is still under 0.9 and the accuracy of Backdoor.Win32.VB increases 10% and accuracy of Trojan-Dropper.Win32.VB also increases 9% in Fig. 12. This means that the addition of color features enhances the discrimination between these two families and other families and can obviously improve the classification accuracy. As we can see in Fig. 13, the combination of local features further improves the accuracy of family Backdoor.Win32.VB to 100%. The results indicate that local features can increase the internal similarity in a family so that the samples of the family are not easy be classified into other families. Moreover, we found the similar families are easy to mix each other when using texture features, like Trojan-Dropper.Win32.KGen and Trojan-Dropper.Win32.VB in Fig. 11. However, our method can solve this problem by combining global features and local



(a)



(b)

FIGURE 14. Precision-Recall curves of our method (a) and Nataraj's method (b).

features. From this point of view, our method is better than the method proposed by Nataraj *et al.* [26].

Aim to fully compare our method with Nataraj *et al.* proposed method, we added several sets of experiments. Nataraj *et al.* [26] only extracted texture features from gray images using the GIST algorithm and used KNN to perform malware classification. Since we are unable to get their original data and can not guarantee all of the samples are PE files, we decided to implement their method and run it on our dataset to compare with our method. The comparison results are shown in Table. 4. When using the RF as the classifier, our feature obtain an F-measure higher than that of Nataraj by 4.69%. And the F-measure of our method is 3.42% higher when applied the KNN classifier which used in the method of Nataraj. This shows that our method can achieve better results when the malicious variant dataset becomes more confusing and complex.

Because of imbalance of the data set, we used Precision-Recall curve to evaluate the two classification methods. From Fig. 14, we can see that our method still maintains a high precision while precision of Nataraj's approach declines rapidly, when recall is close to 1. This proves our method is more stable and credible.

D. PERFORMANCE ANALYSIS

In order to measure the performance of the classifier, we tested the training time of RF, KNN, and SVM under different proportions of training samples. Our computer environment includes Intel Core i7-6700, 8 core CPU, 8G memory, and 1T hard drive. The training time is the average of the 100 training experiments, which ensures the reliability of the result. As shown in Fig. 15, the training time of RF, KNN, and SVM increase with the increase of the proportion of the training set. When the training set proportion is 0.05 (354 samples), the average training time of RF, KNN, and SVM are 0.02s, 0.001s, and 0.05s respectively. When the

TABLE 4. Comparison of our method and Nataraj's method.

Feature	Classifier	Accuracy	Precision	Recall	F-Measure
T+C+L	RF	0.9747	0.9711	0.9672	0.9685
GIST	RF	0.9323	0.9246	0.9205	0.9216
T+C+L	KNN(k=3)	0.9623	0.9548	0.9502	0.9516
GIST	KNN(k=3)	0.9289	0.9207	0.9168	0.9174

T represents texture features, C represents color features, L represents local features.

training set proportion is 1 (7087 samples), the corresponding average training time are 0.15s, 0.03s, and 6.12s. The growth rates of training time for RF and KNN are extremely slow, while the growth rate of SVM is much faster. As the sample increases, SVM will spend much more time than RF and KNN. So, for the large scale dataset, KNN and RF are more suitable.

In fact, the feature extraction time spent much more time than the model training in our experiments. So we recorded the extraction time of different features and compared the average time of each sample. As shown in Fig. 16, the extraction time of texture features is longest compared with color features and local features in our method. The GLCM is used to extract texture features and it spend about 0.66s for a sample. While GIST used by Nataraj *et al.* take an average of 1.45s on a sample, which is more than three times compare with the cost of GLCM. Even compare with the sum of all feature extraction time in our approach, the time of GIST is 0.28s longer. In addition, texture features, color features, and local features are independent of each other and can be extracted simultaneously. So the performance of our method is higher than the method of Nataraj *et al.* proposed.

VI. LIMITATIONS AND FUTURE WORK

Our method can achieve efficient and accurate classification of large and complex malicious code data set, but there are a few limitations to our approach. Our method requires that the

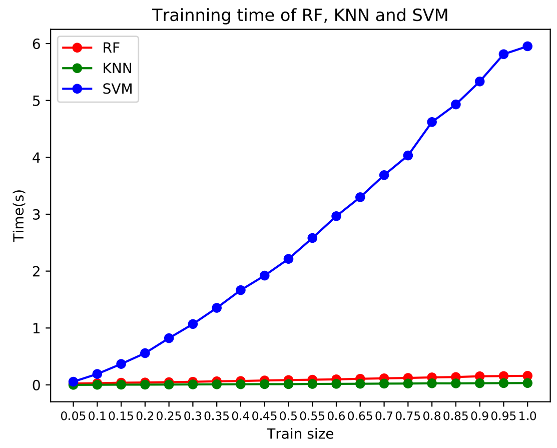


FIGURE 15. **The comparison of training time of RF, KNN, and SVM.**

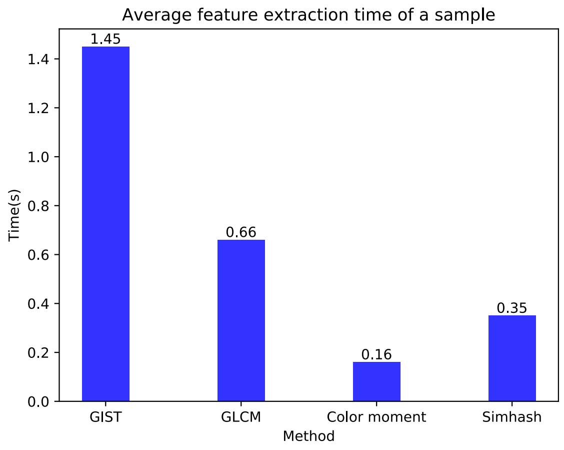


FIGURE 16. **The comparison of feature extraction time for different method.**

structure of malware can be parsed, the format of malware is limited to PE files. The structure of non-PE files such as various malicious pages are complex and cannot be effectively segmented. As a result, it is very difficult to generate RGB-colored images and extract valid local features for these malicious code. Another important reason for choosing a PE file is that because the PE file has a uniform structure, the structure itself contains a lot of useful information, which makes the resulting RGB-colored images inherently better than gray images. Another limitation is the poor adaptability of the method to packed malware, especially encrypted malware. These malwares must be decompressed or decrypted before using our method. However, it is difficult to find an effective decompression and decryption method for the uncommon packers. Sometimes, gray images can better deal with these packed malware, but the prerequisite is having much similarities within the family and distinctions between the family of packed malwares. However, it is obvious that the condition is difficult to satisfy. For example, most malware uses common packers so it is hard to distinguish their families. The most effective way to deal with packed and encrypted malware is to run them in a virtual environment and extract dynamic behavior features.

Therefore, in the future, we will directly determine the section type of the binary file, so as to achieve section partition and extract the local features from specific section. Relevant research has been carried out in [25] and [31], but their methods currently only can classify the type of binary file, and lacks the basis for classification of internal sections. For packed malware, we will consider adding system calls [3], [7], [16] and other dynamic features to break the restrictions on compressed and encrypted malware. Another worthwhile future work is to try some deep learning models such as CNN on malware classification, because deep learning has a very good character, it can automatically learns features and thus reduces manual involvement.

VII. CONCLUSION

In this paper, we present a new method of malware visualization enables effective and efficient malware classification. We divided the malware into sections by parsing the malware structure and computed the entropy and relative size of each section to expand the gray image to a RGB-colored image. RGB-colored images provided a complete characterization of the global features of malware in terms of texture and color. To keep the model simple, we selected low dimensional GLCM and color moments to extract texture features and color features, respectively. In the meantime, we extracted local features from code sections and data sections to further distinguish their families. Experimental results show that the combination of global features and local features obtained the highest accuracy. Compared with the current classification method based on gray images analysis, our method can achieve lower computational cost and finer classification. In addition to the traditional static features and dynamic features, we provide a new idea for malware analysis by combining the image processing technology.

REFERENCES

[1] Symantec Corp. (Apr. 2017). *Internet Security Threat Report*. [Online]. Available: https://www.symantec.com/content/dam/symantec/docs/reports/istr-22-2017-en.pdf

[2] M. Imran, M. T. Afzal, and M. A. Qadir, "Similarity-based malware classification using hidden Markov model," in *Proc. 4th Int. Conf. Cyber Secur., Cyber Warfare, Digit. Forensic (CyberSec)*, Oct. 2015, pp. 129–134.

[3] M. Hassen and P. K. Chan, "Scalable function call graph-based malware classification," in *Proc. 7th ACM. Conf. Data. Appl Secur. Privacy*, 2017, pp. 239–248. [Online]. Available: http://dx.doi.org/10.1145/3029806.3029824

[4] B. Kang, S. Y. Yerima, K. Mclaughlin, and S. Sezer, "N-opcode analysis for Android malware classification and categorization," in *Proc. Int. Conf. Cyber Secur. Protection Digit. Services (Cyber Security)*, Jun. 2016, pp. 1–7.

[5] K. Iwamoto and K. Wasaki, "Malware classification based on extracted API sequences using static analysis," in *Proc. Asian Internet Eng. Conf.*, 2012, pp. 31–38. [Online]. Available: http://dx.doi.org/10.1145/2402599.2402604

[6] G. Cabau, M. Buhu, and C. P. Oprisa, "Malware classification based on dynamic behavior," in *Proc. 18th Int. Symp. Numer. Algorithms Sci. Comput. (SYNASC)*, Sep. 2016, pp. 315–318.

[7] H. Kim, J. Kim, Y. Kim, I. Kim, K. J. Kim, and H. Kim, "Improvement of malware detection and classification using API call sequence alignment and visualization," in *Cluster Computing*. New York, NY, USA: Springer-Verlag, 2017, pp. 1–9. [Online]. Available: http://dx.doi.org/10.1007/s10586-017-1110-2

[8] N. Stakhanova, M. Couture, and A. A. Ghorbani, "Exploring network-based malware classification," in *Proc. 6th Int. Conf. Malicious Unwanted Softw. (MALWARE)*, Oct. 2011, pp. 14–20.
[9] L. Xu, D. Zhang, M. A. Alvarez, J. A. Morales, X. Ma, and J. Cavazos, "Dynamic Android malware classification using graph-based representations," in *Proc. IEEE 3rd Int. Conf. Cyber Secur. Cloud Comput. (CSCloud)*, Jun. 2016, pp. 220–231.
[10] S. Cesare, Y. Xiang, and W. Zhou, "Control flow-based malware variant detection," *IEEE Trans. Depend. Sec. Comput.*, vol. 11, no. 4, pp. 307–317, Jul./Aug. 2014.
[11] T. Lee, B. Choi, Y. Shin, and J. Kwak, "Automatic malware mutant detection and group classification based on the n-Gram and clustering coefficient," *J. Supercomput.*, pp. 1–15, Dec. 2015. [Online]. Available: https://link.springer.com/article/10.1007/s11227-015-1594-6, doi: http://dx.doi.org/10.1007/s11227-015-1594-6
[12] S. Nari and A. A. Ghorbani, "Automated malware classification based on network behavior," in *Proc. Int. Conf. Comput. Netw. Commun. (ICNC)*, Jan. 2013, pp. 642–647.
[13] J. Y.-C. Cheng, T.-S. Tsai, and C.-S. Yang, "An information retrieval approach for malware classification based on windows API calls," in *Proc. Int. Conf. Mach Learn. Cybern. (ICMLC)*, Jul. 2013, pp. 1678–1683.
[14] I. Santos, F. Brezo, X. Ugarte-Pedrero, and P. G. Bringas, "Opcode sequences as representation of executables for data-mining-based unknown malware detection," *Inf. Sci.*, vol. 231, pp. 64–82, May 2013. [Online]. Available: http://dx.doi.org/10.1016/j.ins.2011.08.020
[15] J. Donahue, A. Paturi, and S. Mukkamala, "Visualization techniques for efficient malware detection," in *Proc. IEEE Int. Conf. Intell. Secur. Inform. (ISI)*, Jun. 2013, pp. 289–291.
[16] J. Saxe, D. Mentis, and C. Greamo, "Visualization of shared system call sequence relationships in large malware corpora," in *Proc. 9th Int. Symp. Vis. Cyber Secur.*, Seattle, WA, USA, 2012, pp. 33–40. [Online]. Available: http://dx.doi.org/10.1145/2379690.2379695
[17] 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 Secur.*, Oct. 2009, pp. 33–38.
[18] B. Xiaofang, C. Li, H. Weihua, and W. Qu, "Malware variant detection using similarity search over content fingerprint," in *Proc. 26th Chin. Control Decision Conf. (CCDC)*, Piscataway, NJ, USA, May/Jun. 2014, pp. 5334–5339.
[19] J. Zhang, Z. Qin, H. Yin, L. Ou, S. Xiao, and Y. Hu, "Malware variant detection using opcode image recognition with small training sets," in *Proc. 25th Int. Conf. Comput. Commun. Netw. (ICCCN)*, Aug. 2016, pp. 1–9.
[20] L. Liu and B. Wang, "Malware classification using gray-scale images and ensemble learning," in *Proc. 3rd Int. Conf. Syst. Inform. (ICSAI)*, Nov. 2016, pp. 1018–1022.
[21] S. Z. M. Shaid and M. A. Maarof, "Malware behavior image for malware variant identification," in *Proc. Int. Symp. Biometrics Secur. Technol. (ISBAST)*, Aug. 2014, pp. 238–243.
[22] K. S. Han, J. H. Lim, B. Kang, and E. G. Im, "Malware analysis using visualized images and entropy graphs," *Int. J. Inf. Secur.*, vol. 14, no. 1, pp. 1–14, 2015. [Online]. Available: http://dx.doi.org/10.1007/s10207-014-0242-0
[23] K. Han, B. Kang, and E. G. Im, "Malware analysis using visualized image matrices," *Sci. World J.*, vol. 2014, pp. 1–15, Jul. 2014. [Online]. Available: http://dx.doi.org/10.1155/2014/132713
[24] K. Kancherla and S. Mukkamala, "Image visualization based malware detection," in *Proc. IEEE Symp. Comput. Intell. Cyber Secur. (CICS)*, Apr. 2013, pp. 40–44.
[25] G. Conti *et al.*, "Automated mapping of large binary objects using primitive fragment type classification," *Digit. Invest.*, vol. 7, pp. S3–S12, Aug. 2010. [Online]. Available: http://dx.doi.org/10.1016/j.diin.2010.05.002
[26] L. Nataraj, S. Karthikeyan, G. Jacob, and B. S. Manjunath, "Malware images: visualization and automatic classification," in *Proc. Int. Symp. Vis. Cyber Secur.*, Pittsburgh, PA, USA, 2011, Art. no. 4. [Online]. Available: http://dx.doi.org/10.1145/2016904.2016908
[27] *Microsoft Portable Executable and Common Object File Format Specification*. Accessed: Nov. 21, 2017. [Online]. Available: https://msdn.microsoft.com/en-us/library/ms809762.aspx
[28] M. A. Stricker and M. Orengo, "Similarity of color images," *Proc. SPIE*, vol. 2420, pp. 381–392, Mar. 1995. [Online]. Available: http://dx.doi.org/10.1117/12.205308
[29] M. S. Charikar, "Similarity estimation techniques from rounding algorithms," in *Proc. 34th Annu. ACM Symp. Theory Comput.*, Montreal, QC, Canada, 2002, pp. 380–388.
[30] G. S. Manku, A. Jain, and A. Das Sarma, "Detecting near-duplicates for web crawling," in *Proc. 16th Int. Conf. World Wide Web*, Banff, AB, Canada, 2007, pp. 141–150. [Online]. Available: http://dx.doi.org/10.1145/1242572.1242592
[31] G. Conti *et al.*, "A visual study of primitive binary fragment types," Black Hat, Las Vegas, NV, USA, Tech. Rep., 2010, pp. 1–17.
[32] A. Torralba, K. P. Murphy, W. T. Freeman, and M. A. Rubin, "Context-based vision system for place and object recognition," in *Proc. 9th IEEE Int. Conf. Comput. Vis.*, Oct. 2003, pp. 273–280.
[33] A. Oliva and A. Torralba, "Modeling the shape of the scene: A holistic representation of the spatial envelope," *Int. J. Comput. Vis.*, vol. 42, no. 3, pp. 145–175, 2001. [Online]. Available: http://dx.doi.org/10.1023/A:1011139631724
[34] R. M. Haralick, K. Shanmugam, and I. Dinstein, "Textural features for image classification," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. SMC-3, no. 6, pp. 610–621, Nov. 1973.

JIANWEN FU received the B.S. degree in software engineering from Northeastern University, China, in 2016. He is currently pursuing the M.S. degree in software engineering with the Beijing Institute of Technology, China. His research interest includes cyber security and machine learning.

JINGFENG XUE was born in Yan'an, China, in 1975. He received the B.S. degree in information science and the Ph.D. degree in computer application from the Beijing Institute of Technology in 1998 and 2003, respectively.

From 2003 to 2005, he was a Lecturer with the School of Software, Beijing Institute of Technology. From 2005 to 2012, he was an Associate Professor. Since 2012, he has been a Professor. He has authored five books, over 30 articles, and over 10 inventions. His research interests include information security and computer network.

Dr. Jingfeng Xue was a member of the Council of Chinese Association for Artificial Intelligence.

YONG WANG received the Ph.D. degree in computer application from the Beijing Institute of Technology, China, in 2003. She is currently an Associate Professor with the School of Software, Beijing Institute of Technology, China. She has authored or co-authored four books and over 30 papers. Her research interests include cyber security and machine learning. She is a member of the China Computer Federation and the Chinese Association for Artificial Intelligence.

ZHENYAN LIU received the Ph.D. degree in computer architecture from the Institute of Computing Technology, Chinese Academy of Sciences, China. She is currently with the School of Software, Beijing Institute of Technology. Her current research interests include big data, artificial intelligence, and cyber security. She is a Fellow of the China Computer Federation.

CHUN SHAN was born in Songyuan, China, in 1975. She received the B.S., master's, and Ph.D. degrees in computer science from the Beijing Institute of Technology in 1998, 2003, and 2015. She is currently a Lecturer and the Master Supervisor with the Beijing Institute of Technology. She has authored over 20 articles. She is the Leader of over 10 projects. Her research interests include artificial intelligence and network security.