Computers & Security 126 (2023) 103084

Contents lists available at ScienceDirect

Computers & Security

journal homepage: www.elsevier.com/locate/cose

MCTVD: A malware classification method based on three-channel visualization and deep learning

Huaxin Deng[a], Chun Guo[a,*], Guowei Shen[a], Yunhe Cui[a], Yuan Ping[b]

[a] State Key Laboratory of Public Big Data, College of Computer Science and Technology, Guizhou University, Guiyang 550025, PR China
[b] School of Information Engineering, Xuchang University, Xuchang 461000, PR China

ARTICLE INFO

Article history:
Received 25 August 2022
Revised 17 November 2022
Accepted 27 December 2022

Keywords:
Malware classification
Malware visualization
Markov transfer matrix
Deep learning
Convolutional neural network

ABSTRACT

With the rapid increase in the number of malware, the detection and classification of malware have become more challenging. In recent years, many malware classification methods based on malware visualization and deep learning have been proposed. However, the malware images generated by these methods do not retain the semantic and statistical properties with a small and uniform size. This article gives definitions of extracted content and filling mode to characterize the critical factors for the malware visualization task and proposes a new malware visualization method based on assembly instructions and Markov transfer matrices to characterize malware. Thus, a **m**alware **c**lassification method based on **t**hree-channel **v**isualization and **d**eep learning (MCTVD) is proposed. In MCTVD, its malware image has a small and uniform size, and its convolutional neural network has few convolutional and pooling layers. Experimental results show that MCTVD can achieve an accuracy of 99.44% on Microsoft's public malware dataset under 10-fold cross-validation and thus could be a highly competitive candidate for malware classification.

1. Introduction

Malware is any type of software that harms or exploits the normal operation of a system. In recent years, with the rapid development of the internet and computer technologies, the number of malware in the past decade has increased year by year. As reported by AV-TEST (AV-TEST), the total number of malware cases was 1218.68 million as of 2022. Malware classification is a necessary task for malware analysis. It distinguishes different malware families to better understand the capabilities of malware variants from the same family and thus can reduce the work of security analysts and facilitate their research on new malware or malware variants (Gibert et al., 2020). Unfortunately, classifying malware efficiently and accurately is a challenging task.

To improve the efficiency of classifying malware, traditional machine learning methods, such as decision tree (DT), naive Bayes (NB), support vector machine (SVM), and k-nearest neighbor (KNN), have been widely used in a variety of malware classification methods (Pachhala et al., 2021). However, such methods have the limitations of complex feature engineering and difficulty in processing large amounts of data. This makes such methods still unsatisfactory. Deep learning methods can overcome these shortcomings of machine learning. A convolutional neural network (CNN) is a deep learning approach that has proven very effective in tackling problems such as image recognition and classification (Basha et al., 2020). Correspondingly, deep learning was also introduced in malware detection and classification (Kargarnovin et al., 2022; Li et al., 2022; Wang et al., 2019b; Yadav and Tokekar, 2021), and some malware classification approaches based on image visualization and deep learning were proposed in recent years. This type of method turns malware classification into an image classification problem. Its feasibility lies in the fact that when different malware from the same family is converted into images, they appear to be similar in texture and layout (Verma et al., 2020). The generated malware image is the core of this type of malware classification method. However, how to generate a high-quality image from a malware file is an issue that has not been deeply discussed. The most widely used malware visualization method uses malware binaries directly as input, converting every 8-bit binary to one pixel to generate a grayscale image. This requires compression or interception to keep the image size uniform when training with CNNs. There is undoubtedly a loss of effective information in the original binary file during the conversion. A few malware visualization methods use opcode n-grams extracted from Windows Portable Executable (PE) files as pixels in generated malware images. Such methods generally use only the frequency of the unique

* Corresponding author.
E-mail address: cguo@gzu.edu.cn (C. Guo).

https://doi.org/10.1016/j.cose.2022.103084
0167-4048/

combination of n consecutive opcodes to visualize malware, ignoring the role of the information about the quality of opcodes and operands in assembly instruction. Therefore, this article aims to explore the quality of malware images in terms of "extracted content" and "filling mode" defined in Section 3 and proposes a malware visualization method to generate a high-quality malware image.

To achieve this goal, we analyze the critical factors in generating an image from malware and use the information extracted from the assembly instructions (containing opcodes and operands) of a PE file to generate a three-channel image. A **m**alware **c**lassification method based on **t**hree-channel **v**isualization and **d**eep learning (MCTVD) is then designed. MCTVD extracts the sequence of assembly instructions from the code section (also known as ".text" section) of malware and uses the transfer probabilities of the unique combination of every 2 consecutive letters or numbers from the sequence of assembly instructions to construct the three-channel image. This image contains richer information about assembly instructions than the grayscale image and the opcode n-gram image, and is beneficial to improving the accuracy of malware classification. In addition, it does not require compression or interception of the sizes of the generated images. The main contributions of this article are as follows:

1) A three-channel malware visualization method based on assembly instructions and Markov transfer matrices is proposed. The extracted content and filling mode are defined to characterize the critical factors for the malware visualization task. Subsequently, a new malware visualization method is proposed. The image generated by this method focuses on retaining the information about assembly instructions in the code section of malware with a reduced and equal size, which is helpful to improve the accuracy and efficiency of malware classification.

2) A CNN is designed to effectively classify the three-channel images generated by our malware visualization method. Compared with common CNNs, such as AlexNet (Krizhevsky et al., 2012), VGG16 (Simonyan and Zisserman, 2014), and VGG19 (Simonyan and Zisserman, 2014), our presented architecture has fewer convolutional and fully connected layers, which is conducive to less time consumption during training.

3) A malware classification method called MCTVD combined the three-channel images with our presented CNN is proposed. Experiments on a public dataset from Microsoft Corporation show that MCTVD is superior to the traditional grayscale image-based, byte-level Markov-based, and RGB color image-based methods in terms of accuracy and macro F1-score.

The rest of this article is organized as follows. Section 2 gives a brief introduction to the current malware classification methods. Section 3 describes the motivations of our proposed method. The proposed MCTVD is detailed in Section 4. In Section 5, experimental results regarding our method are presented and compared with other works. Finally, Section 6 summarizes the work of the article.

2. Related work

Over the last 20 years, an increasing number of researchers have proposed many malware classification methods based on machine learning technologies. They can be roughly divided into two types: methods based on traditional machine learning and methods based on malware images and deep learning.

2.1. Malware classification methods based on traditional machine learning

These methods rely on handcrafted features based on expert knowledge, and their feature engineering process are generally time-consuming (Gibert et al., 2020). These methods can be classified into two categories according to the different kinds of features used: static and dynamic.

2.1.1. Static feature-based methods

Static features are those that can be obtained without running the malware. These are features such as byte sequences (Yousefi-Azar et al., 2018), opcode sequences (Yeboah et al., 2021; Zhang et al., 2019), API calls (Soni et al., 2022), and function call graphs (FCGs) (Hassen and Chan, 2017). Shalaginov et al. (2018) conducted an in-depth survey of different machine-learning methods for the classification of static characteristics of Windows PE files. A framework to detect malicious applications and to categorize benign applications with an ensemble of multiple classifiers, namely, SVM, KNN, NB, classification and regression tree (CART), and random forest (RF) was proposed in (Wang et al., 2018). This framework extracts as many as 2,374,340 static features that fall into 11 types (Restricted API calls, Suspicious API calls, and so on) from each APK file and chooses the top-ranked 34,630 static features for detection and categorization. Zhang et al. (2019) proposed an opcode sequence-based ransomware classification method. This method first converts the opcode sequences from ransomware samples into n-gram sequences, and then a vector consisting of term frequency values of the n-gram feature is used as the feature vector. Finally, five machine learning methods are used to perform ransomware classification. Soni et al. (2022) proposed a malware classification method using the features extracted from API calls and opcode sequences. After extracting the features, four machine learning algorithms, NB, logistic regression, RF, and SVM, are used to classify malware. Hassen and Chan (2017) proposed an FCG vector representation based on function clustering that has significant performance gains which is then used for malware classification.

2.1.2. Dynamic feature-based methods

These features are obtained by dynamic analysis methods. Dynamic analysis observes the interaction between malware and the system by executing the executable file of the malware in a controlled environment. Registry changes, memory writes, and API call traces are commonly used as dynamic features of malware. Amer and Zelinka (2020) used various API functions with similar contextual characteristics as a cluster by studying the contextual relationships that exist between API functions in malware. This article proved that there is indeed a clear difference between the API call sequence of malware and benign software. San et al. (2019) proposed a malware family classification system by extracting the prominent API features of 11 malware families from a cuckoo sandbox. Xiao et al., 2020 proposed a graph repartition algorithm to extract fragment behaviors from original API call graphs and then obtained the crucial N-order subgraph for malware detection and classification. An association rule-based malware classification using common subsequences of API calls was proposed in (D'Angelo et al., 2021). This method exploits the probabilities of transitioning from two API invocations in the call sequence.

2.2. Malware classification methods based on malware images and deep learning

The image-based malware classification method was first introduced by Nataraj et al. (2011), who used the binary content of malware to generate a grayscale image and applied GIST and KNN to extract texture features to classify malware. A few years later, image-based malware classification methods using machine learning were also proposed (Ghouti and Imam, 2020), which have the limitation of needing a complex feature engineering process. With the rapid development of deep learning technology in recent years

and its excellent performance in the image classification field, malware classification methods based on malware images and deep learning have become a research hotspot in the malware classification field in recent years. This type of approach can eliminate many feature engineering works and obtain good classification accuracy (Yuan et al., 2020).

Within the existing malware classification methods based on malware images and deep learning, the grayscale image converted directly from the binary sequence of malware is used by a majority of methods (Cui et al., 2018; Pinhero et al., 2021; Yan et al., 2018; Zhao et al., 2020). For instance, Cui et al. (2018) converted the binary sequence of malware into grayscale images. Then, these images were classified by using a CNN. Since only images with uniform sizes can be directly applied to CNNs, this type of method requires compression or interception of the grayscale images when they are trained with CNNs. Although Yuan et al. (2020) also used binary sequences of malware, they converted the binary sequence into a Markov image with a fixed size of 256*256 through the byte transfer probability matrix. Then, a deep CNN was used for training a model to classify malware. Their Markov image contains the binary information and global structure of malware while ignoring specific semantic information. The literature has shown that opcode sequences can represent program behaviors (Jian et al., 2021). Correspondingly, opcode sequences have also been used in some malware classification methods based on malware images and deep learning to generate images (Ni et al., 2018; Zhang et al., 2016). Zhang et al. (2016) collected opcode sequences from binary files in the dataset and used them to construct images. However, the generated images are generally very sparse because the opcodes contained in a single sample are limited. In addition to grayscale images, some other forms of malware images have also been proposed recently for malware classification. For instance, Wang et al. (2019a) converted the byte sequence into an RGB color image. This conversion maps every 8-bit binary to an integer value of RGB in sequence. An RGB image named “CoLab image” was proposed by Xiao et al. (2021). The CoLab image uses colored label boxes to mark the sections of malware. A malware classification method based on CoLab image, VGG16, and SVM was constructed. Yuan et al. (2022) converted a malware binary file into a multidimensional Markov image. This image is a combination of several byte transfer probability matrices and contains richer information about byte distribution of a malware binary file than the Markov image proposed by Yuan et al. (2020).

Different from existing work, in this article, the sequence of assembly instructions and the transfer probabilities of the unique combination of every 2 consecutive letters or numbers from the sequence of assembly instructions are used as extracted content and filling mode, respectively, to generate a three-channel image from malware. In this way, the byte distribution of assembly instructions, the dependencies of an opcode on its previous opcode, and the quality of each opcode in malware are used to characterize malware. The image thus can provide richer information about assembly instructions of malware than binary and opcode sequences.

3. Motivation

Since excellent performance can be obtained via malware classification methods based on malware images and deep learning, this article focuses on designing a method belonging to this type of method. One key problem of a malware classification method based on malware images and deep learning is how to convert malware into an image. This is because a deep learning model requires a data representation that is convenient for this model to effectively extract key features from malware images. As shown in Fig. 1, for generating an image from malware, there are two critical factors: what content is extracted from malware and how to fill the content.

Definition 1 (**Extracted Content**): Extracted content is the substance that is extracted from a malware file to prepare for filling an image.

Definition 2 (**Filling Mode**): The Filling mode is the method of filling the extracted content into an image.

Extracted content determines the collection of available information that can be used to fill an image. A binary sequence, opcode sequence, and sequence of assembly instructions are instances of extracted content. The filling mode gives the specific

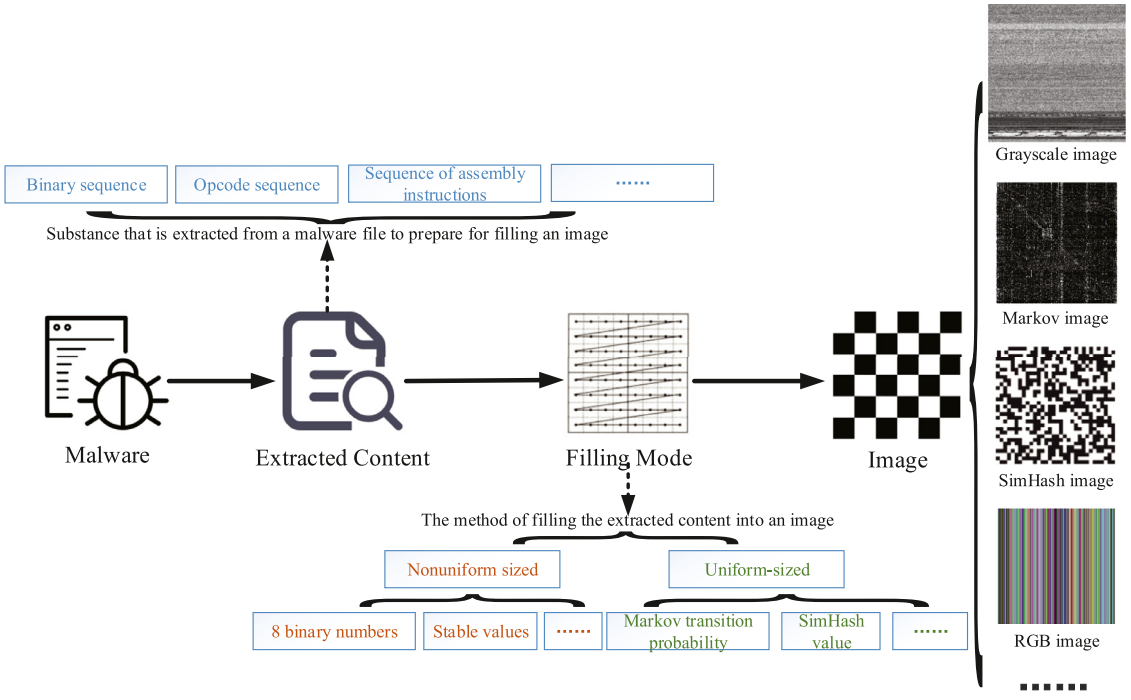


Fig. 1. Extracted content and filling mode for generating a malware image.

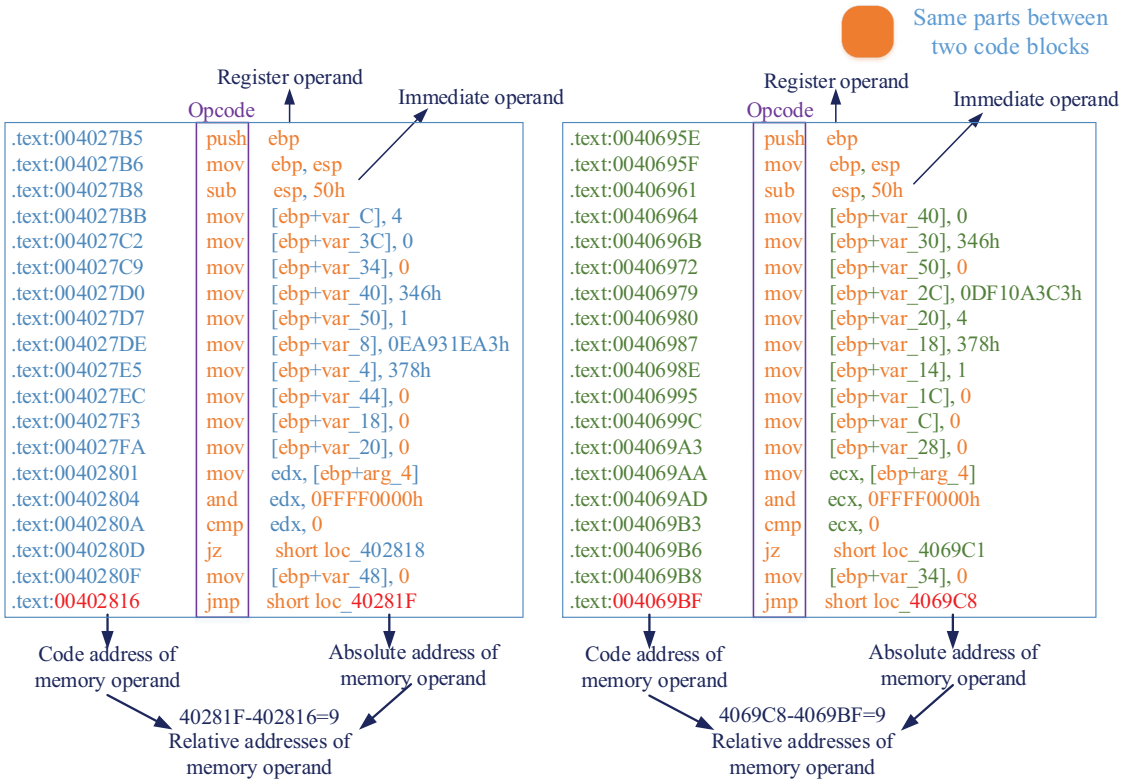


Fig. 2. Comparison of the assembly instructions in code blocks of two malware samples from the same family.

pixel values and the size of the generated image and thus determines what and how much information or property in the extracted content can be provided by the generated image. It can be divided into two classes: nonuniform sized and uniform-sized. When the sizes of the generated image after filling are different and thus cannot be directly applied to a CNN, this filling mode belongs to the class of nonuniform size; otherwise, it belongs to the class of uniform size. The 8 binary numbers and the stable value used in (Fu et al., 2018) are instances of the filling mode of nonuniform size, and the Markov transition probability and the Simhash value are examples of the filling mode of uniform size. Therefore, the extracted content and filling mode undoubtedly influence whether a generated image is of high quality, i.e., the generated image can provide sufficient valuable information about its original malware file, which is conducive to distinguishing the generated images belonging to different malware families.

3.1. Extracted content

Binary and opcode sequences are the two most common types of extracted content used to generate the malware image. On the one hand, the binary sequence preserves binary information and the global structure of a malware file while ignoring the specific semantic information contained in the code section. On the other hand, the opcode sequence preserves partial information in the assembly instructions of malware. The literature has shown that the opcode sequence is better than the binary sequence in terms of analyzing malware files (Manavi and Hamzeh, 2017; Raff et al., 2018). However, the opcode sequence lacks operands which are participants in the execution of assembly instructions, i.e., the objects of various operations; thus, it just contains partial information about the assembly instructions of malware.

The address space of a PE file is flat, and its code and data are stored in different sections in a certain format. Usually, the data in different sections are logically related. For example, the code section stores program codes of a PE file, while the ".data" section stores the data variables of the program. Program code is the core of the program, and different malware from the same family generally have similar program codes. To preserve more information about the assembly instructions of malware, the extracted content used in this article is the sequence of assembly instructions in the code section of a malware file. An assembly instruction includes an opcode and one or more operands. In a PE file, the code section is a block whose main content consists of assembly instructions. Therefore, we focuse on the code section of the PE file rather than on the whole PE file and discard the other sections, such as the ".data" section. Fig. 2 gives the assembly instructions in code blocks of two malware samples from the same family and it is obvious that the assembly instructions of the two malware samples are extremely similar in terms of opcode and execution sequence. There are three main types of operands: immediate operands, register operands, and memory operands. The similarity of the first two types of operands can be observed intuitively, while the similarity of the memory operands can be better observed by their relative addresses rather than the absolute addresses. The relative address of a memory operand is obtained by the absolute address minus the code address. Intuitively, using the sequence of assembly instructions as the extracted content is conducive to preserving richer information about assembly instructions of malware than binary and opcode sequences.

3.2. Filling mode

As mentioned above, there are two types of filling modes: nonuniform sized and uniform-sized. The most commonly used grayscale image uses every 8 binary numbers as a pixel, and thus, its filling mode belongs to the class of nonuniform size. The size of the malware image generated by this filling mode varies with the malware size. Therefore, it requires using byte truncation or image scaling methods to unify the sizes of malware images when

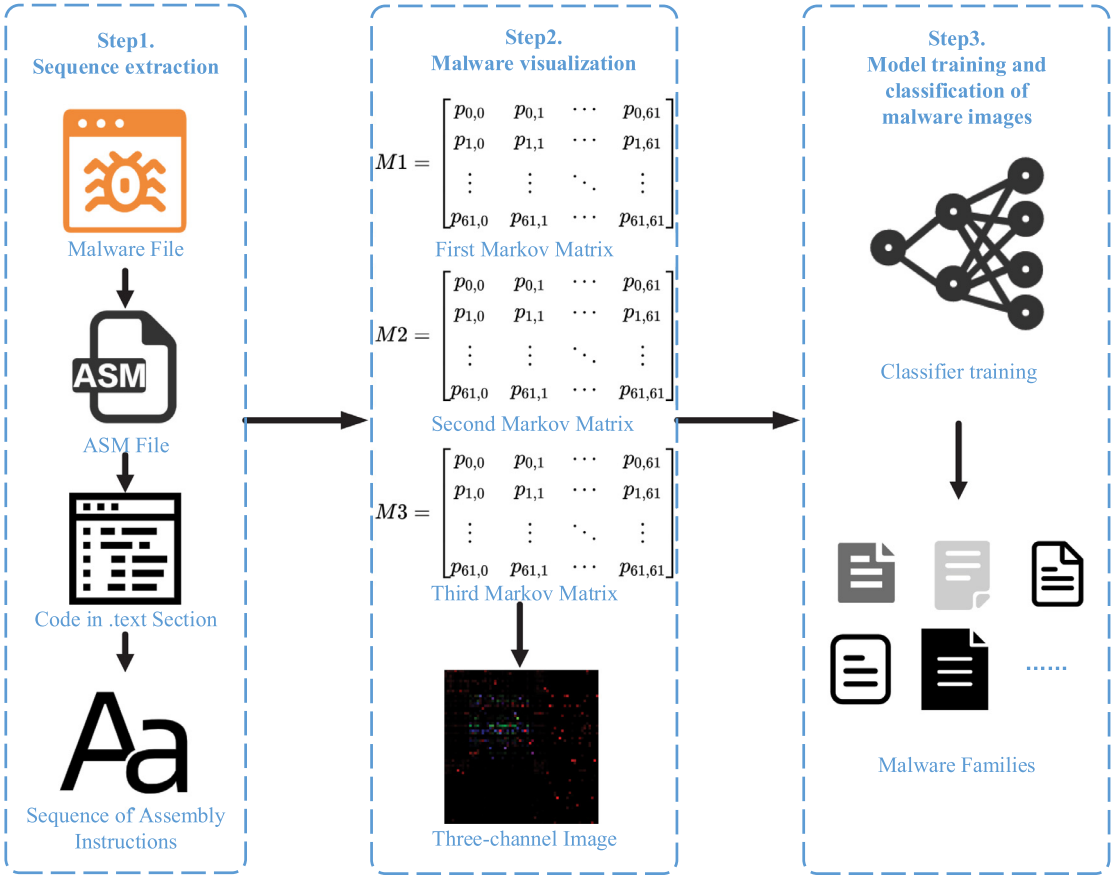


Fig. 3. MCTVD framework.

training with a CNN. However, the original binary information of malware could be partially missing when the sizes of grayscale images are unified (Yuan et al., 2020). In addition, when faced with a malware variant with relocation sections, the similarities between the malware variant's image and its original malware image are low. Specifically, the stable values of "section entropy" and "section size" used in work (Fu et al., 2018) are another filling mode belonging to the nonuniform size class.

For the uniform-sized filling mode, the SimHash value and Markov transition probability are most commonly used approaches. In (Ni et al., 2018), SimHash was used to convert opcode sequences of different malware samples into images of equal size, but it can only generate short-length values; thus, the image that is converted from SimHash values usually requires interpolating, which may introduce meaningless padding information. For Markov transition probability, the sizes of the generated image are fixed and thus can be used directly as inputs for a CNN. In (Manavi and Hamzeh, 2017), the frequency of the unique combination of every 2 consecutive opcodes that are extracted from the opcode sequence of malware is used to directly form a Markov image. Since there are many opcodes, the generated Markov images will be large and sparse. This may lead to the effective information in the image being too sparse and create some difficulties in the training process of a CNN (Sun and Qian, 2021). Although fixing some of the opcodes as detection objects can deal with the above problem, it leads to a partial loss of useful information.

To avoid the useful information being missed due to the discarding of the bytes or opcodes and to alleviate the sparseness of the generated image, we use the transfer probabilities of the unique combination of every 2 consecutive letters or numbers extracted from the sequence of assembly instructions to generate Markov images. Punctuation of the sequence of assembly instructions is omitted because it is only used to separate operands. There are only 62 different letters and numbers, which is helpful to generate matrices with a small and uniform size. Most new malware comes from known malware with some code differences (Sun and Qian, 2021). Therefore, it can be inferred that malware of the same family has great similarity in code structures. Specifically, when generating our three-channel image, the transfer probability of the unique combination of 2 consecutive letters or numbers, the transfer probability of the unique combination of the first letters of 2 consecutive opcodes, and the transfer probability of the unique combination of the last 2 consecutive letters in each opcode are used. They can (approximately) represent the assembly instructions' byte distribution, the dependencies of an opcode on its previous opcode, and the quality of each opcode. Compared with the previous filling modes, it can provide richer information about assembly instructions in a malware file and will have a uniform size, which will not cause missing useful information of assembly instructions and is suitable for classifying its variant with relocation sections. After generating the three-channel image, a CNN is used to train the malware classification model. CNNs can discover the local features of images and are a good choice for classifying images according to current research. Since the size of the proposed three-channel image is only $62 \times 62 \times 3$, a CNN that has a few fully connected layers is designed for the classification of the malware images. The details will be introduced in Section 4.

4. MCTVD method

This section describes how MCTVD classifies malware upon the content in Section 3. The overall framework of MCTVD is shown in Fig. 3. It consists of the following three steps that are as follows.

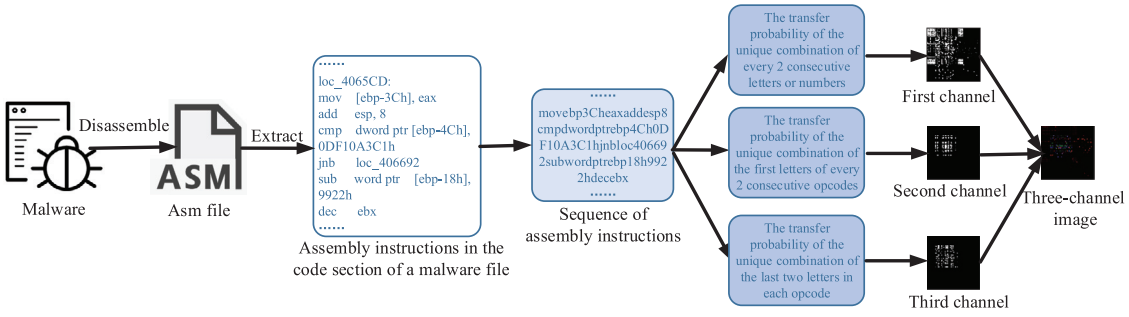


Fig. 4. Schematic diagram of the three-channel image used in MCTVD.

Step 1: Sequence extraction. The assembly instructions in the code section of malware are extracted from executable files with the help of static analysis tools such as IDA Pro. These assembly instructions are viewed as a sequence of assembly instructions in the later steps.

Step 2: Malware visualization. The sequence of assembly instructions obtained in step 1 is used to generate three Markov matrices. The consequence is that a $62 \times 62 \times 3$ three-channel image is generated.

Step 3: Model training and classification of malware images. The three-channel images are used to build a malware classification model by using our presented CNN. Then the three-channel images converted by the samples to be classified can be classified into different families by this model.

4.1. Sequence extraction

The sequence extraction step extracts a sequence of assembly instructions as the extracted contents for the three-channel image of MCTVD. In a PE file, its code section stores the code to be executed in the running process. To obtain more useful and nonredundant features, MCTVD focuses on the assembly instructions in the code section of malware. Since these instructions cannot be obtained directly from the malware itself, it is necessary to use a third-party analysis tool such as IDA Pro to convert the malware files into assembly files. After obtaining the assembly file, MCTVD extracts the assembly instructions in the code section of the assembly file. Then, these instructions are combined into a sequence of assembly instructions. When generating the sequence of assembly instructions, the opcode, immediate operands, and register operands in the extracted assembly instructions are reserved directly. For the memory operands, their relative addresses are reserved in the sequence of assembly instructions and can be obtained as their absolute addresses minus their code addresses. Compared with the binary and opcode sequences, the sequence of assembly instructions contains richer information about the assembly instructions of malware.

4.2. Malware visualization

In the malware visualization step, MCTVD uses the sequence of assembly instructions obtained in the previous step to generate a three-channel image. Specifically, we use the transfer probabilities of uppercase and lowercase letters or numbers of the sequence of assembly instructions as the pixel values. Punctuation is omitted because it is only used to separate operands or to help the computer understand human-written assembly code. The existence of such irrelevant information may lead to difficulties in the learning phase. The uppercase and lowercase letters and numbers of the sequence of assembly instructions can be represented as a byte stream S. Assuming that each letter or number is regarded as a state, then each element in stream S has 62 possible states that include 26 uppercase letters, 26 lowercase letters, and 10 numbers, where $S = \{s_0, s_1, \ldots, s_{61}\}$. We assume that the next state is only related to its current state and hence the sequence of assembly instructions can be regarded as a Markov chain, i.e., $P(s_{i+1}|s_0, \ldots, s_i) = P(s_{i+1}|s_i)$. Assuming that the transfer from the previous state m to the subsequent state n occurs with a certain probability, the transfer probability $P_{m,n}$ is computed by using Formula 1.

$$P_{m,n} = P(n \mid m) = \frac{f(m,n)}{\sum_{n=0}^{61} f(m,n)} \tag{1}$$

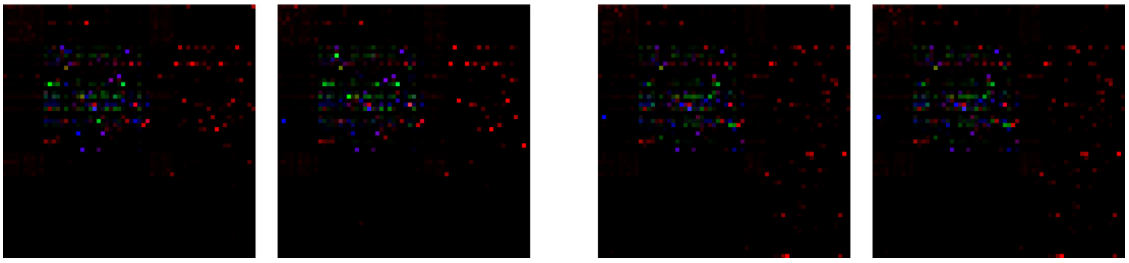
where $f(m,n)$ is the frequency of moving from state m to state n $(m, n \in \{0, 1, \ldots, 61\})$.

The transfer probability matrix with n states has n^2 transfer probabilities and is a $n \times n$ matrix. Therefore, the size of matrix M used in MCTVD is small and uniform (62×62). As shown in Formula (2), the element $p_{i,j}$ in Row i and Column j of M represents the transfer probability from state i to state j.

$$M = \begin{bmatrix} p_{0,0} & p_{0,1} & \cdots & p_{0,61} \\ p_{1,0} & p_{1,1} & \cdots & p_{1,61} \\ \vdots & \vdots & \ddots & \vdots \\ p_{61,0} & p_{61,1} & \cdots & p_{61,61} \end{bmatrix} \tag{2}$$

To ensure that the generated malware image can provide rich information about the assembly instructions of malware, three transfer probability matrices generated by the sequence of assembly instructions are used to construct a three-channel image in MCTVD. That is, each transfer probability matrix is used as a channel of the three-channel image. A schematic diagram of the three-channel image used in MCTVD is shown in Fig. 4.

The transfer probability of the unique combination of every 2 consecutive letters or numbers is used to fill a matrix M_1 for being the first channel of the three-channel image. It can reflect the byte distribution of the assembly instructions of a malware file. Two-tuple opcodes can reflect the dependencies of an opcode on its previous opcode. To generate the second channel of the three-channel image, the transfer probability of the unique combination of the first letters of every 2 consecutive opcodes in the sequence of assembly instructions are used as the pixel values because they can approximately substitute the transfer probability of 2-tuple opcodes. This transfer probability is used to fill a 62×62 matrix (called M_2) to ensure that its size is the same as that of M_1. The quality of each opcode is a useful statistical characteristic for a PE file and we add this property to our malware image. Specifically, the transfer probability of the unique combination of the last two letters in each opcode of the sequence of assembly instructions is used to approximately substitute the quality of this opcode in a PE file. A 62×62 matrix M_3 filled by these transition probabilities is used to form the third channel of the three-channel image. Finally, a $62 \times 62 \times 3$ matrix $M = [M_1|M_2|M_3]$ is constructed and is used to form a three-channel image. Fig. 5 shows the malware images



(a) Malware images belonging to Ramnit family (b) Malware images belonging to Lollipop family

Fig. 5. Malware images generated by MCTVD.

belonging to two malware families that were generated using the MCTVD. As shown in Fig. 5, images generated by malware from the same family have similar pixels and colors, while images generated by malware from different families have significant differences in some areas. These commonalities and differences make us believe that learning the features of the three-channel images by a CNN could distinguish different malware families. The process of generating the three-channel images is given in Algorithm 1.

Algorithm 1 Three-channel image generation.

Input: PE software E.
Output: Three-channel image M.

$A \Leftarrow$ assembly language file extracted from sample E
$C \Leftarrow$ code Section C is obtained by A
for each line l_i in C **do**
  if l_i contains assembly instruction **then**
    $R_{mo} \Leftarrow$ calculate the relative address of the memory operand in the assembly instruction
    $A_{ai} \Leftarrow$ stores the opcode, immediate operands, register operands, and R_{mo}
  end if
end for
/* F_{ai} is the set of the first letter of every 2 consecutive opcodes in A_{ai}, E_{ai} is the set of last two letters of each opcode in A_{ai} */
$PA_{ai} \Leftarrow$ removes punctuation from A_{ai}, leaving only letters and numbers
$F_{ai} \Leftarrow$ stores the first letter of every 2 consecutive opcodes from A_{ai}
$E_{ai} \Leftarrow$ stores the last two letters of each opcode from A_{ai}
$M_1 \Leftarrow$ Markov matrix M_1 is generated by PA_{ai}
$M_2 \Leftarrow$ Markov matrix M_2 is generated by F_{ai}
$M_3 \Leftarrow$ Markov matrix M_3 is generated by E_{ai}
$M \Leftarrow$ three-channel image M is constructed of M_1, M_2, M_3
return M

4.3. Model training and classification of malware images

This step aims to train a model to classify malware images into different families. In recent years, deep learning models have emerged, and their effectiveness has been proven in many fields. Some existing deep learning models, such as AlexNet, VGG16, and VGG19, show good performance in the field of image recognition. As mentioned above, the size of the three-channel image generated in step 2 is only $62 \times 62 \times 3$. It is not suitable for the adoption of a complex model such as VGG16 to process our three-channel images because of the difficulty of training or the overfitting problem caused by too small images. To address this problem, we construct a CNN (shown in Fig. 6) with fewer layers compared with VGG16, VGG19, or AlexNet. Compared with these above network structures, our presented architecture reduces the numbers of both the convolutional and the fully connected layers. For our presented architecture, the size of the convolution kernel for the convolutional layer is 3×3. Combined with other parameters (stride=1, padding='same'), each convolutional layer of our presented architecture can maintain the same width and height as the previous layers. The number of convolution kernels will gradually double; it begins with 64 and ends with 512. Its pooling layer uses maximum pooling with a 2×2 pool matrix, and the default step size is also 2×2. After pooling, the length and width of the matrix are continuously reduced by half; it begins with 62 and ends with 3. To speed up the training, a ReLU function is selected as the activation function for both the convolutional layer and the fully connected layer. The Relu function is given by Formula 3.

$$\mathrm{h(x)} = \begin{cases} x & (x > 0) \\ 0 & (x \leq 0) \end{cases} \tag{3}$$

After convolution and pooling, the data are flattened into a one-dimensional vector by the flatten function. In addition, our proposed architecture includes a fully connected layer with an output size of 1024. Subsequently, a softmax layer is connected, whose number of neurons is set according to the number of malware families in the training dataset. The softmax function can be used to solve multiclass classification problems and is given by Formula 4.

$$softmax(\mathrm{y}_i) = \frac{\exp(a_i)}{\sum_{i=1}^{n} \exp(a_i)} \tag{4}$$

Table 1 gives a comparison of our presented architecture with AlexNet, VGG16, and VGG19. As shown, compared with these network structures, our presented architecture has fewer convolution layers and fully connected layers. Correspondingly, our presented architecture requires less time during training than AlexNet, VGG16, and VGG19 (see Section 5 for details). During the training process, the Adam optimizer is used to learn the parameters, and the network is trained using the cross-entropy loss.

5. Experimental evaluation

5.1. Dataset and experimental environment

The malware dataset used to evaluate MCTVD was derived from a malware classification contest held by Microsoft at Kaggle in 2015 (Ronen et al., 2018). It has been the benchmark dataset most widely used in the field of static malware analysis since 2016. The dataset consists of two separate parts: a training dataset and a test dataset. The training dataset contains nine malware families with a total of 10,868 samples. The test dataset contains 10,873 samples, but the labels of these samples are not publicly available. Therefore, similar to most of the literature, we used only the training dataset (hereafter referred to as the Microsoft dataset) to obtain experimental results. Table 2 lists the sample distribution of the Microsoft dataset. For each sample, the dataset provides two

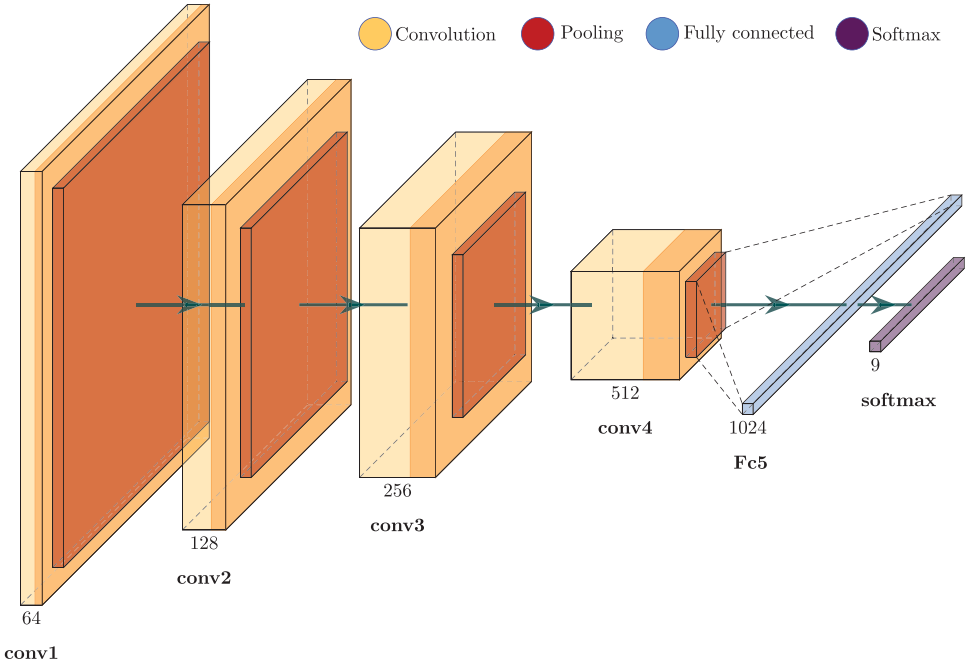


Fig. 6. Network structure of our presented architecture used in MCTVD.

Table 1
Comparison of our presented architecture with AlexNet, VGG16, and VGG19 in the network structure.

Networks	Convolutional layer	Pooling layer	Fully connected layer	Total
AlexNet	5	3	3	11
VGG16	13	5	3	21
VGG19	16	5	3	24
Our presented architecture	4	4	1	9

Table 2
Sample distribution of the Microsoft dataset.

Malware Family	Malware Type	Sample Number
Ramnit	Confucianism	1541
Lollipop	Advertising	2478
Keilhos_ver3	Back Door	2942
Vundo	Trojan Horse	475
Simda	Back Door	42
Tracur	Download Software	751
Keilhos_ver1	Back Door	398
Obfuscator.ACY	Obfuscating Software	1228
Gatak	Back Door	1013
Total		10868

file formats: malware binary files with the suffix ".bytes" (binary stream files without PE headers) and the corresponding assembly files with the suffix ".asm" decompiled by IDA Pro. MCTVD used only the assembly files. Note that 61 samples were removed in the experiment for MCTVD because their assembly files did not contain the code section.

To verify the effectiveness of MCTVD, we use stratified 10-fold cross-validation. That is, the dataset was divided into 10 subsets of equal size, the i-th subset was used as test data in turn, while the remaining subsets were used as training data.

MCTVD was implemented in Python 3 and trained on Ubuntu 18.04. Experiments were conducted with Intel(R) Xeon(R) Gold 5220, NVIDIA GeForce RTX 2080 Ti * 1, 251G RAM. Table 3 lists the parameters used in MCTVD and other network structures and methods in the experiment.

5.2. Evaluation metrics

Four commonly used evaluation metrics were utilized to assess the classification performance of MCTVD, namely, accuracy, precision, recall, and F1-score. They can be calculated by Formulas 5∼8 based on true positives (TP), true negatives (TN), false-positives (FP), and false negatives (FN).

$$Accuracy = \frac{\sum_{i\in\{1,\ldots,k\}} TP_i}{N} \tag{5}$$

$$Precision = \frac{\sum_{i\in\{1,\ldots,k\}} TP_i}{\sum_{i\in\{1,\ldots,k\}} (TP_i + FP_i)} \tag{6}$$

$$Recall = \frac{\sum_{i\in\{1,\ldots,k\}} TP_i}{\sum_{i\in\{1,\ldots,k\}} (TP_i + FN_i)} \tag{7}$$

$$F1 - score = \frac{2 * Precision * Recall}{Precision + Recall} \tag{8}$$

where N denotes the number of samples in the dataset with k malware families.

The meanings of TP, TN, FP, and FN for a specific malware family $i \in \{1, 2, \ldots, k\}$ are as follows: TP_i and FN_i denote the numbers of samples correctly predicted as family i and not predicted as family i but actually belong to family i, respectively; TN_i and FP_i denote the numbers of samples correctly not predicted as family i and incorrectly predicted as family i but actually do not belong to family i, respectively. Furthermore, we used the receiver operating characteristic (ROC) curve and the area under the ROC (AUC)

Table 3
Parameters of different network structures and methods in the experiment.

Network Structure or Method	Optimizer	Learning Rate	Decay Rate	Batch Size	Epoch
AlexNet	Adam	3e-4	2e-7	16	250
VGG16	Adam	3e-4	2e-7	16	250
VGG19	Adam	3e-4	2e-7	16	250
GDMC	Adam	3e-4	2e-7	16	250
MDMC	Adam	3e-4	2e-7	16	250
RGBDMC	Adam	3e-4	2e-7	16	250
MalCVS	Adam	1e-3	5e-4	256	74
MulMarkov	Adam	3e-4	2e-7	16	250
MCTVD	Adam	3e-4	2e-7	16	250

Table 4
Results of our presented architecture, AlexNet, VGG16, and VGG19 under 10-fold cross-validation.

Network	Accuracy	Macro Precision	Macro Recall	Macro F1-score
AlexNet	0.9924	0.9806	0.9872	0.9838
VGG16	0.9915	0.9763	0.9805	0.9783
VGG19	0.9900	0.9586	0.9764	0.9665
Our presented architecture	0.9944	0.9944	0.9913	0.9929

to show more details about the performance of models. In addition, the training time (for building the model in the model training step) in seconds was considered to measure the efficiency of the training models.

Due to the imbalance of various families of samples in the Microsoft dataset, for example, the Simda family has only 42 samples, and the Kelihos_ver3 family has 2942 samples, we used macro average to measure the average individual evaluation metrics obtained for each category. It can be calculated by Formula 9.

$$Macro\_metrics = \frac{1}{q}\sum_{i=1}^{q} metrics_i \tag{9}$$

5.3. Experimental results

5.3.1. Large training dataset test

A. The results of our presented architecture and other network structures

A CNN is designed in Section 4.3 to classify our three-channel images. In this section, we compare our presented architecture with three well-known network structures, i.e., AlexNet, VGG16, and VGG19, for classifying the same three-channel images. To avoid random errors, 10-fold cross-validation is used to compare the accuracies of different network structures. Note that the value of each valuation metric is an average value of 10-fold in this section. The results obtained by our presented architecture, AlexNet, VGG16, and VGG19, on the whole Microsoft dataset with 10-fold cross-validation are given in Table 4.

On the one hand, as shown in Table 4, the classification accuracies obtained by all four network structures on the Microsoft dataset whose samples are converted to our three-channel images are higher than 99%. This suggests that our three-channel images have high separability between malware families. On the other hand, Table 4 shows that our presented architecture provides better performance than AlexNet, VGG16, and VGG19 in terms of accuracy, macro precision, macro recall, and macro F1-score. In addition, Fig. 7 shows that the training time of our presented architecture is less than that of AlexNet, VGG16, and VGG19, which confirms the lower time consumption during training of our presented architecture. Comparisons of different network structures in terms of ROC and AUC are shown in Fig. 8, our presented architecture and VGG16 behave better than AlexNet and VGG19.

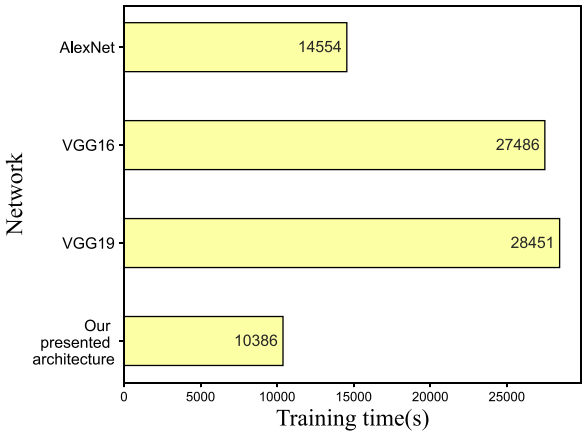


Fig. 7. Training times of different network structures under 10-fold cross-validation.

B. Comparative evaluation

To present better observe the performance of MCTVD, we took the performances of the traditional grayscale image-based method (GDMC) (Nataraj et al., 2011), the byte-level method based on Markov images and deep learning (MDMC) (Yuan et al., 2020), the RGB color image-based method (RGBDMC) (Wang et al., 2019a), the CoLab image-based method (MalCVS) (Xiao et al., 2021) and the multidimensional Markov image-based method (MulMarkov) (Yuan et al., 2022) as our baselines. The grayscale image used in GDMC is the most widely used malware image. MDMC generates the malware image based on the Markov transfer probability matrix and uses a deep CNN to classify its generated images. The malware image generated by RGBDMC is an RGB image converted from binary sequences. MalCVS is a malware classification using CoLab image (also an RGB image), pretrained VGG16, and SVM. MulMarkov is a malware classification method based on multidimensional Markov images and deep learning. In the experiment, GDMC used our presented architecture with the single-channel model, RGBDMC used our presented architecture to classify the RGB images, MulMarkov used the CNN proposed by (Su et al., 2018) to classify the multidimensional Markov images (dimension=3), and the image sizes of GDMC, MDMC, RGBDMC, MalCVS, and Mul-

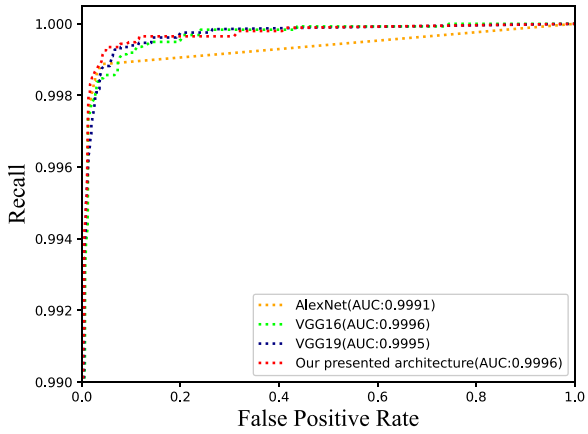


Fig. 8. ROC curves and AUC values of different network structures under 10-fold cross-validation.

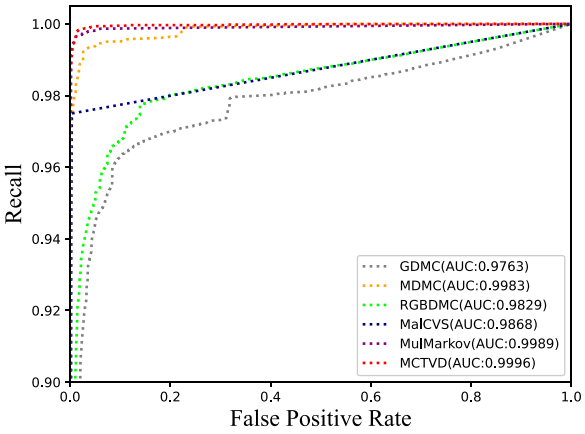


Fig. 10. ROC curves and AUC values of different methods under 10-fold cross-validation.

Table 5
Results obtained by different methods under 10-fold cross-validation.

Method	Accuracy	Macro Precision	Macro Recall	Macro F1-score
GDMC	0.9237	0.8948	0.8452	0.8649
MDMC	0.9826	0.9563	0.9422	0.9487
RGBDMC	0.9410	0.9198	0.8650	0.8839
MalCVS	0.9891	0.9821	0.9748	0.9763
MulMarkov	0.9915	0.9902	0.9628	0.9747
MCTVD	0.9944	0.9944	0.9913	0.9929

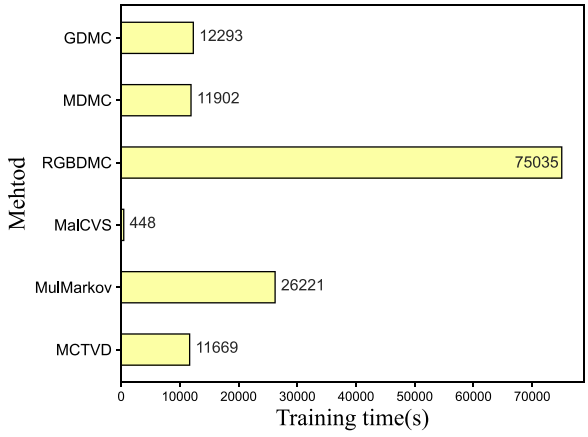


Fig. 9. Training times of different methods under 10-fold cross-validation.

Markov were compressed into 256×256, 256×256, $256 \times 256 \times 3$, $224 \times 224 \times 3$, and $256 \times 256 \times 3$, respectively.

As shown in Table 5, among the six methods, MCTVD obtained the best accuracy, macro precision, macro recall, and macro F1-score. It is worth mentioning here that the accuracy of MCTVD is 5.34% higher than that of RGBDMC with the same CNN structure, which reflects that the malware image generated by MCTVD is markedly superior to the RGB image generated by RGBDMC in terms of malware classification. In comparisons of different methods in terms of training time shown in Fig. 9, MalCVS requires far less training time than the other five methods because it relies on a pretrained model and a traditional machine learning algorithm. Among the remaining five methods, the training time of MCTVD is less than that of GMDC, MDMC, RGBDMC, and MulMarkov. One main reason for this is that the size of the image generated in MCTVD is only $62 \times 62 \times 3$, which is smaller than the sizes of the images generated by the other four methods. For the ROC curves and AUC values of different methods as shown in Fig. 10, among all the methods, MCTVD obtains the highest AUC value.

The resulting confusion matrices obtained by the six methods are shown in Fig. 11. It can help understand the detailed accuracy of the six methods for each of the nine malware families. From Fig. 11, we can observe that MCTVD outperforms GDMC, MDMC, RGBDMC, and MalCVS on all nine malware families, and MCTVD is inferior to MulMarkov in three out of the nine malware families.

To further assess the accuracy of MCTVD, the performances of some other state-of-the-art malware classification methods are given. For fairness, only the methods that had used the same dataset and the division (the whole Kaggle's Microsoft training dataset under 10-fold cross-validation) as well as those based on one modality of data (.bytes or .asm) were selected. More specifically, method (Drew et al., 2016), method (Narayanan et al., 2016), method (Drew et al., 2017), and method (Hassen and Chan, 2017) are methods based on static features and traditional machine learning; method (Lin and Yeh, 2022), method (Gibert, Mateu, Planes, Vicens, 2018), method (Ding et al., 2020) and method (Gibert et al., 2018a) are methods based on static features and deep learning; and method (Kim et al., 2017), method (Kim and Cho, 2022), method (Gibert et al., 2019) and method (Ren et al., 2020) can be classified into methods based on malware images and deep learning. Table 6 gives the average accuracies comparison of different methods under 10-fold cross-validation on the Microsoft dataset. As shown in Table 6, MCTVD obtains higher average accuracy than the other methods. Hence it could be a highly competitive candidate for malware classification.

5.3.2. Small training dataset test

To evaluate the performance of MCTVD in the scenario where only limited training samples are available, this section presents the results of an experiment to test the MCTVD trained by a small training dataset. To avoid random errors, a specific 5-fold cross-validation was used. In each fold, 20% of the samples from the whole dataset were used as the training data, and the remaining 80% were used as the test data. We note that the value of each valuation metric is an average value of 5-fold.

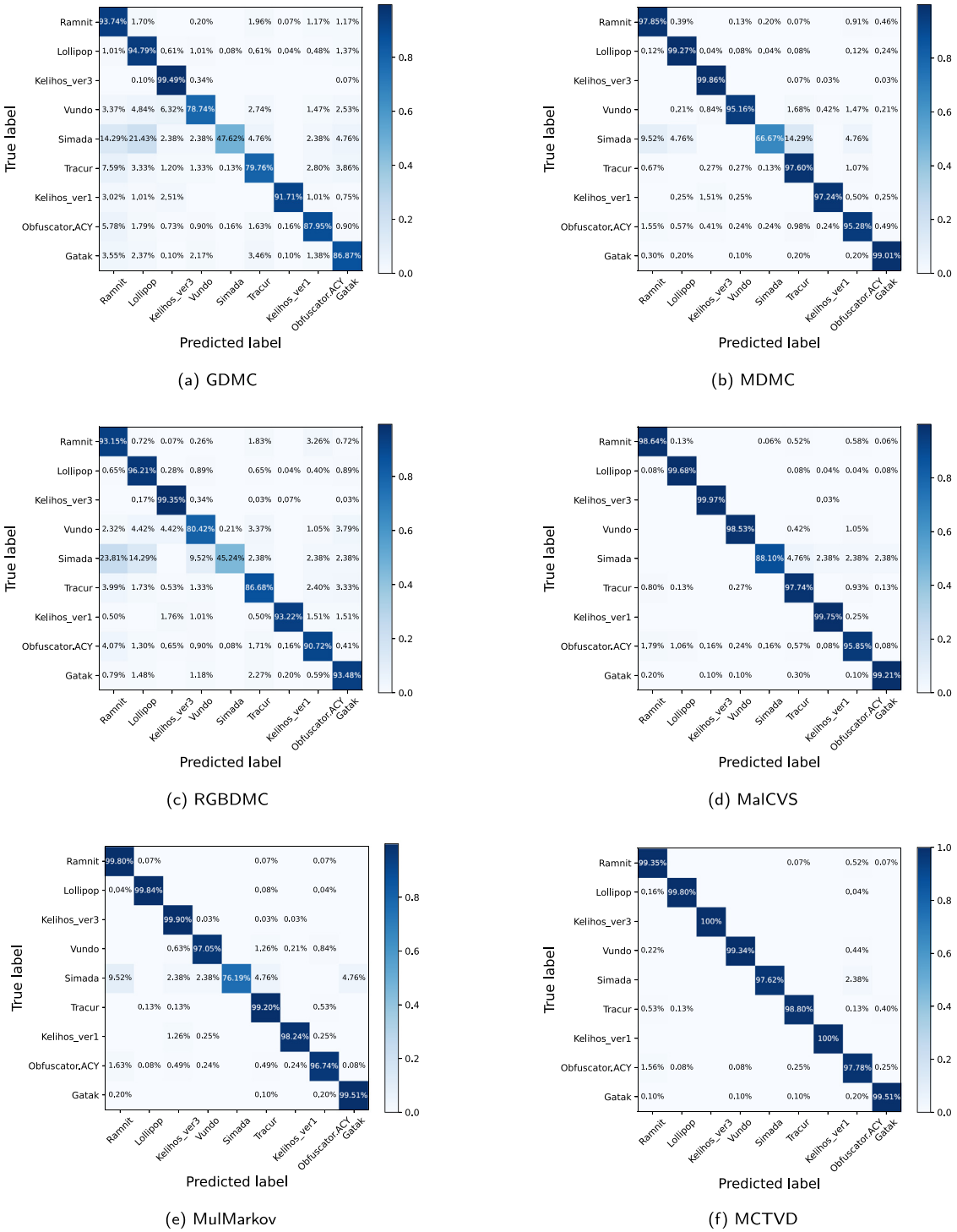


Fig. 11. Confusion matrices obtained by different methods under 10-fold cross-validation.

Table 7 shows the comparison of the six methods under 5-fold cross-validation on the Microsoft dataset. From Table 7, we can see that even when the training dataset only accounts for 20% and the testing dataset accounts for 80%, the accuracy of MCTVD still reaches 98.72%, which is higher than that of the other five methods. Table 7 also shows that MCTVD achieves the best performance in terms of macro recall, macro precision, and macro F1-score under 5-fold cross-validation. As for the ROCs and AUCs of different methods given in Fig. 12, MCTVD obtains the highest AUC value under the 5-fold cross-validation. Fig. 13 gives the training times of different methods. Similar to the results obtained by different methods under 10-fold cross-validation, Fig. 13 shows that MCTVD requires less training time than GMDC, MDMC, RGBDMC, and MulMarkov, while it requires more training time than MalCVS because MalCVS relies on a pretrained model and a traditional machine learning algorithm. The resulting confusion matrices of the

Table 6
The average accuracies comparison of different methods under 10-fold cross-validation on the Microsoft dataset.

Method	Input	Algorithm	Accuracy
(Drew et al., 2016)	Byte sequence	Strand	97.41%
(Narayanan et al., 2016)	Byte image	Linear kNN	96.6%
(Drew et al., 2017)	Opcode sequence	Strand	98.59%
(Hassen and Chan, 2017)	Function call graph	Ensembling multiple RF	99.3%
(Lin and Yeh, 2022)	Bit sequence	1D CNN	96.32%
(Gibert, Mateu, Planes, Vicens, 2018)	Structural entropy	CNN	98.28%
(Ding et al., 2020)	Opcode sequence	Self-attention	98.48%
(Gibert et al., 2018a)	Byte sequence	Residual network	98.61%
(Kim et al., 2017)	Byte image	tGAN	96.39%
(Kim and Cho, 2022)	Byte sequence	VAE+1D CNN+LSTM	97.47%
(Gibert et al., 2019)	Byte image	CNN	97.5%
(Ren et al., 2020)	Markov image	VGG16+SVM	99.08%
MCTVD	Assembly instruction sequence	CNN	99.44%

Table 7
Results obtained by different methods under 5-fold cross-validation.

Method	Accuracy	Macro Precision	Macro Recall	Macro F1-score
GDMC	0.8116	0.7258	0.6648	0.6820
MDMC	0.9620	0.9167	0.9014	0.9086
RGBDMC	0.8673	0.8248	0.7326	0.7520
MalCVS	0.9741	0.9545	0.9363	0.9436
MulMarkov	0.9756	0.9641	0.9318	0.9449
MCTVD	0.9872	0.9789	0.9584	0.9674

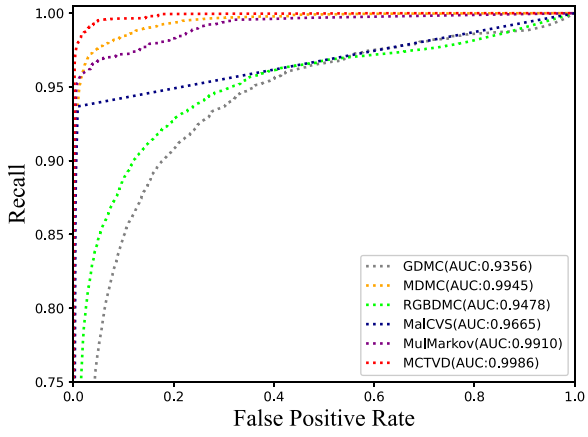


Fig. 12. ROC curves and AUC values of different methods under 5-fold cross-validation.

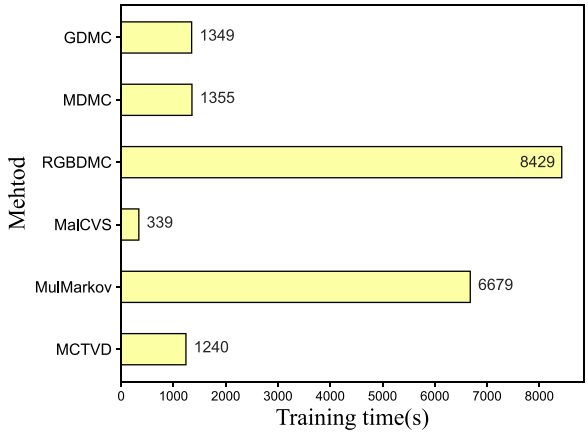


Fig. 13. Training times of different methods under 5-fold cross-validation.

Table 8
Models in the ablation experiment.

Method	First Channel	Second Channel	Third Channel
MCTVD-F	•	○	○
MCTVD-S	○	•	○
MCTVD-T	○	○	•
MCTVD-FS	•	•	○
MCTVD-FT	•	○	•
MCTVD-ST	○	•	•
MCTVD	•	•	•

Note: • means the channel is included, ○ means it is not included.

average results obtained by the six methods under 5-fold cross-validation are given in Fig. 14. It shows that MCTVD behaves better than GDMC, MDMC, and RGBDMC on all nine malware families, and MCTVD is inferior to MulMarkov and MalCVS in merely one and two out of the nine malware families, respectively.

5.3.3. Ablation experiment

To explore the contribution of the different channels in the three-channel image to the final result, ablation experiments were designed and the results are reported in this subsection. Experiments for single-channel images and two-channel images were conducted. Since each single channel of the three-channel image is a grayscale image, Our presented architecture was changed to the single-channel mode. The each two-channel image was derived from the three-channel image by filling one channel with 0s. The other experimental parameters in this section were the same as in the previous section. The settings of the seven comparison models are shown in Table 8.

Fig. 15 gives the performance of the different models used in the ablation experiment under 10-fold cross-validation on the Microsoft dataset. On the one hand, it shows that among the three channels, the first channel contributes most to the classification accuracy. On the other hand, each channel contributes to the classification accuracy because each two-channel image obtains higher accuracy than the single-channel images constituting them, and the three-channel image obtains the best accuracy. In practice, the number of channels n used will influence the form and quality of the generated images. n can be neither too small nor too large. When a very small value of n is used, the effective information contained in the generated image may not be enough to make highly accurate malware classification. On the other hand, if n is too large, the computation requirement in training a CNN will increase and even lead to less accurate outcomes when some of the channels are filled by invalid properties.

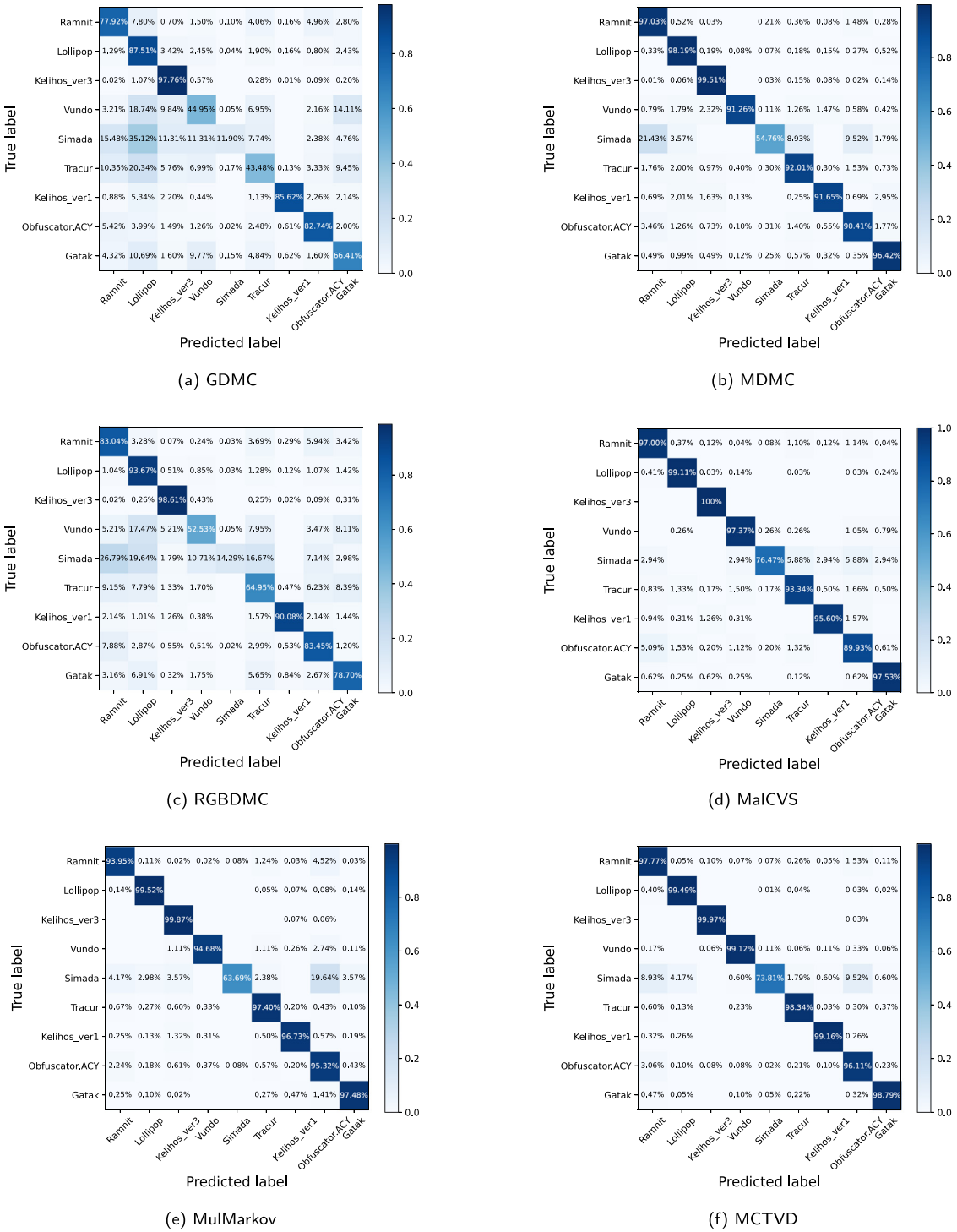


Fig. 14. Confusion matrices obtained by different methods under 5-fold cross-validation.

6. Conclusion

Malware images are the key to malware classification methods based on malware images and deep learning. To analyze the quality of the generated image, definitions of extracted content and filling mode are proposed to characterize the critical factors for malware visualization task. Both of them need to be focused on when generating images from malware. In addition, a three-channel malware visualization method is proposed to improve malware classification accuracy. The three-channel image uses the sequence of assembly instructions in the code section of the PE file as the extracted content and has a small and uniform size. This malware image will not cause missing useful information on extracted content because it does not require interception or compression. Based on the three-channel image and a CNN that has only a few convolution and fully connected layers, a malware classification method called MCTVD is constructed. A series of experiments were conducted on the widely used Microsoft dataset to evaluate the performance of MCTVD. Experimental results show that MCTVD achieves high accuracy in both the scenarios of us-

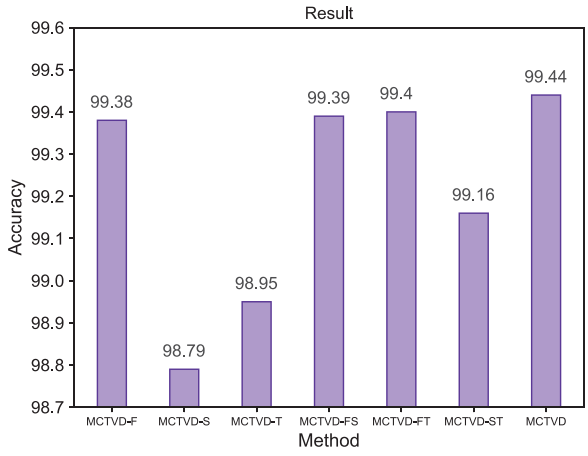


Fig. 15. The accuracies of the different models used in the ablation experiment under 10-fold cross-validation.

ing large and small training datasets. In the future, we will further explore the effect of the different numbers of channels for generating malware images on malware classification and use API calls or FCG features as extracted content to generate high-quality malware images.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

CRediT authorship contribution statement

Huaxin Deng: Methodology, Software, Data curation, Writing – original draft. **Chun Guo:** Conceptualization, Methodology, Formal analysis, Funding acquisition, Writing – review & editing. **Guowei Shen:** Formal analysis, Investigation, Funding acquisition, Resources. **Yunhe Cui:** Investigation, Validation, Writing – review & editing. **Yuan Ping:** Methodology, Writing – review & editing.

Data Availability

Data will be made available on request.

Acknowledgments

The authors thank the anonymous referees for their valuable comments and suggestions, which improved the technical content and the presentation of the article. This work is supported by the National Natural Science Foundation of China under Grant No. 62162009, the Science and Technology Foundation of Guizhou Province under Grant No. [2020]1Y268, the Guizhou Major Special Science and Technology Project under Grant No. 20183001, the Key Technologies R&D Program of He'nan Province under Grant Nos. 212102210084 and 222102210048.

References

Amer, E., Zelinka, I., 2020. A dynamic windows malware detection and prediction method based on contextual understanding of API call sequence. Comput. Secur. 92, 101760.

AV-TEST, Av-test, 2022. https://www.av-test.org/en/statistics/malware/.Online. Accessed: 24 August 2022.

Basha, S.S., Dubey, S.R., Pulabaigari, V., Mukherjee, S., 2020. Impact of fully connected layers on performance of convolutional neural networks for image classification. Neurocomputing 378, 112–119.

Cui, Z., Xue, F., Cai, X., Cao, Y., Wang, G.-g., Chen, J., 2018. Detection of malicious code variants based on deep learning. IEEE Trans. Ind. Inf. 14 (7), 3187–3196.

Ding, Y., Wang, S., Xing, J., Zhang, X., Oi, Z., Fu, G., Qiang, Q., Sun, H., Zhang, J., 2020. Malware classification on imbalanced data through self-attention. In: 2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom). IEEE, pp. 154–161.

Drew, J., Hahsler, M., Moore, T., 2017. Polymorphic malware detection using sequence classification methods and ensembles. EURASIP J. Inform. Secur. 2017 (1), 1–12.

Drew, J., Moore, T., Hahsler, M., 2016. Polymorphic malware detection using sequence classification methods. In: 2016 IEEE Security and Privacy Workshops (SPW). IEEE, pp. 81–87.

D'Angelo, G., Ficco, M., Palmieri, F., 2021. Association rule-based malware classification using common subsequences of API calls. Appl. Soft Comput. 105, 107234.

Fu, J., Xue, J., Wang, Y., Liu, Z., Shan, C., 2018. Malware visualization for fine-grained classification. IEEE Access 6, 14510–14523.

Ghouti, L., Imam, M., 2020. Malware classification using compact image features and multiclass support vector machines. IET Inf. Secur. 14 (4), 419–429.

Gibert, D., Mateu, C., Planes, J., 2018. An end-to-end deep learning architecture for classification of malware's binary content. In: International Conference on Artificial Neural Networks. Springer, pp. 383–391.

Gibert, D., Mateu, C., Planes, J., 2020. The rise of machine learning for detection and classification of malware: research developments, trends and challenges. J. Netw. Comput. Appl. 153, 102526.

Gibert, D., Mateu, C., Planes, J., Vicens, R., 2018. Classification of malware by using structural entropy on convolutional neural networks. In: Proceedings of the AAAI Conference on Artificial Intelligence, New Orleans, Louisiana, USA, February 2-7, 2018, AAAI Press, pp. 7759–7764.

Gibert, D., Mateu, C., Planes, J., Vicens, R., 2019. Using convolutional neural networks for classification of malware represented as images. J. Comput. Virol. Hacking Tech. 15 (1), 15–28.

Hassen, M., Chan, P.K., 2017. Scalable function call graph-based malware classification. In: Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy, pp. 239–248.

Jian, Y., Kuang, H., Ren, C., Ma, Z., Wang, H., 2021. A novel framework for image-based malware detection with a deep neural network. Comput. Secur. 109, 102400.

Kargarnovin, O., Sadeghzadeh, A. M., Jalili, R., 2022. Mal2GCN: a robust malware detection approach using deep graph convolutional networks with non-negative weights. arXiv preprint arXiv:2108.12473.

Kim, J.-Y., Bu, S.-J., Cho, S.-B., 2017. Malware detection using deep transferred generative adversarial networks. In: International Conference on Neural Information Processing. Springer, pp. 556–564.

Kim, J.-Y., Cho, S.-B., 2022. Obfuscated malware detection using deep generative model based on global/local features. Comput. Secur. 112, 102501.

Krizhevsky, A., Sutskever, I., Hinton, G.E., 2012. Imagenet classification with deep convolutional neural networks. Adv. Neural Inf. Process. Syst. 25, 1106–1114.

Li, C., Cheng, Z., Zhu, H., Wang, L., Lv, Q., Wang, Y., Li, N., Sun, D., 2022. DMalNet: dynamic malware analysis based on API feature engineering and graph learning. Comput. Secur. 122, 102872.

Lin, W.-C., Yeh, Y.-R., 2022. Efficient malware classification by binary sequences with one-dimensional convolutional neural networks. Mathematics 10 (4), 608.

Manavi, F., Hamzeh, A., 2017. A new method for malware detection using opcode visualization. In: 2017 Artificial Intelligence and Signal Processing Conference (AISP). IEEE, pp. 96–102.

Narayanan, B.N., Djaneye-Boundjou, O., Kebede, T.M., 2016. Performance analysis of machine learning and pattern recognition algorithms for malware classification. In: 2016 IEEE National Aerospace and Electronics Conference (NAECON) and Ohio Innovation Summit (OIS). IEEE, pp. 338–342.

Nataraj, L., Karthikeyan, S., Jacob, G., Manjunath, B.S., 2011. Malware images: visualization and automatic classification. In: Proceedings of the 8th International Symposium on Visualization for Cyber Security, pp. 1–7.

Ni, S., Qian, Q., Zhang, R., 2018. Malware identification using visualization images and deep learning. Comput. Secur. 77, 871–885.

Pachhala, N., Jothilakshmi, S., Battula, B.P., 2021. A comprehensive survey on identification of malware types and malware classification using machine learning techniques. In: 2021 2nd International Conference on Smart Electronics and Communication (ICOSEC). IEEE, pp. 1207–1214.

Pinhero, A., Anupama, M., Vinod, P., Visaggio, C.A., Aneesh, N., Abhijith, S., AnanthaKrishnan, S., 2021. Malware detection employed by visualization and deep neural network. Comput. Secur. 105, 102247.

Raff, E., Barker, J., Sylvester, J., Brandon, R., Catanzaro, B., Nicholas, C.K., 2018. Malware detection by eating a whole EXE. In: Workshops at the Thirty-Second AAAI Conference on Artificial Intelligence, New Orleans, Louisiana, USA, February 2-7, 2018, AAAI Press, pp. 268–276.

Ren, Z., Chen, G., Lu, W., 2020. Malware visualization methods based on deep convolution neural networks. Multimed. Tools Appl. 79 (15), 10975–10993.

Ronen, R., Radu, M., Feuerstein, C., Yom-Tov, E., Ahmadi, M., 2018. Microsoft malware classification challenge. arXiv preprint arXiv:1802.10135.

San, C.C., Thwin, M.M.S., Htun, N.L., 2019. Malicious software family classification using machine learning multi-class classifiers. In: Computational Science and Technology. Springer, pp. 423–433.

Shalaginov, A., Banin, S., Dehghantanha, A., Franke, K., 2018. Machine learning aided static malware analysis: a survey and tutorial. In: Cyber Threat Intelligence. Springer, pp. 7–45.

Simonyan, K., Zisserman, A., 2014. Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556.

Soni, H., Kishore, P., Mohapatra, D.P., 2022. Opcode and API based machine learning framework for malware classification. In: 2022 2nd International Conference on Intelligent Technologies (CONIT). IEEE, pp. 1–7.
Su, J., Vasconcellos, D.V., Prasad, S., Sgandurra, D., Feng, Y., Sakurai, K., 2018. Lightweight classification of IoT malware based on image recognition. In: 2018 IEEE 42Nd Annual Computer Software and Applications Conference (COMPSAC), Vol. 2. IEEE, pp. 664–669.
Sun, G., Qian, Q., 2021. Deep learning and visualization for identifying malware families. IEEE Trans. Dependable Secure Comput. 18 (1), 283–295. doi:10.1109/TDSC.2018.2884928.
Verma, V., Muttoo, S.K., Singh, V., 2020. Multiclass malware classification via first-and second-order texture statistics. Comput. Secur. 97, 101895.
Wang, S.-w., Zhou, G., Lu, J.-c., Zhang, F.-j., 2019. A novel malware detection and classification method based on capsule network. In: International Conference on Artificial Intelligence and Security. Springer, pp. 573–584.
Wang, W., Li, Y., Wang, X., Liu, J., Zhang, X., 2018. Detecting android malicious apps and categorizing benign apps with ensemble of classifiers. Future Gen. Comput. Syst. 78, 987–994.
Wang, W., Zhao, M., Wang, J., 2019. Effective android malware detection with a hybrid model based on deep autoencoder and convolutional neural network. J. Ambient Intell. Humaniz. Comput. 10 (8), 3035–3043.
Xiao, M., Guo, C., Shen, G., Cui, Y., Jiang, C., 2021. Image-based malware classification using section distribution information. Comput. Secur. 110, 102420.
Xiao, F., Sun, Y., Du, D., Li, X., Luo, M., 2020. A novel malware classification method based on crucial behavior. Math. Probl. Eng. 2020, 6804290.
Yadav, B., Tokekar, S., 2021. Recent innovations and comparison of deep learning techniques in malware classification: a review. Int. J. Inform. Secur.Sci. 9 (4), 230–247.
Yan, J., Qi, Y., Rao, Q., 2018. Detecting malware with an ensemble method based on deep neural network. Secur. Commun. Netw. 2018, 7247095.
Yeboah, P.N., Amuquandoh, S.K., Musah, H.B.B., 2021. Malware detection using ensemble n-gram opcode sequences. Int. J. Interact. Mob.Technol. 15 (24), 19–31.
Yousefi-Azar, M., Hamey, L., Varadharajan, V., Chen, S., 2018. Learning latent byte-level feature representation for malware detection. In: International Conference on Neural Information Processing. Springer, pp. 568–578.
Yuan, B., Wang, J., Liu, D., Guo, W., Wu, P., Bao, X., 2020. Byte-level malware classification based on Markov images and deep learning. Comput. Secur. 92, 101740.
Yuan, B., Wang, J., Wu, P., Qing, X., 2022. IoT malware classification based on lightweight convolutional neural networks. IEEE Internet Things J. 9 (5), 3770–3783. doi:10.1109/JIOT.2021.3100063.
Zhang, H., Xiao, X., Mercaldo, F., Ni, S., Martinelli, F., Sangaiah, A.K., 2019. Classification of ransomware families with machine learning based onN-gram of opcodes. Future Gener. Comput. Syst. 90, 211–221.
Zhang, J., Qin, Z., Yin, H., Ou, L., Hu, Y., 2016. IRMD: malware variant detection using opcode image recognition. In: 2016 IEEE 22nd International Conference on Parallel and Distributed Systems (ICPADS). IEEE, pp. 1175–1180.
Zhao, Y., Cui, W., Geng, S., Bo, B., Feng, Y., Zhang, W., 2020. A malware detection method of code texture visualization based on an improved faster RCNN combining transfer learning. IEEE Access 8, 166630–166641. doi:10.1109/ACCESS.2020.3022722.

Huaxin Deng received BS degree in computer science and technology from Chongqing University of Science and Technology in China in 2019. He is currently pursuing the MS degree in computer science and technology from Guizhou University. His recent research interests include information security and malware classification.

Chun Guo received PhD degree in information security from Beijing University of Posts and Telecommunications in July 2014. He is currently an Associate Professor in the College of Computer Science and Technology, Guizhou University, PR China. His research interests include data mining, intrusion detection and malware detection.

Guowei Shenreceived his PhD degree from Harbin Engineering University. He is currently a Professor of Guizhou University. His main research interests include big data, computer network and cybersecurity.

Yunhe Cui received his PhD degree from the Southwest Jiaotong University, Chengdu, Sichuan, PR China. He is currently a lecturer of Guizhou University, Guiyang, Guizhou, PR China. His research interests include software-defined networking, network security, traffic engineering, swarm intelligence algorithm, data centers, edge computing and cloud computing.

Yuan Ping received the BS degree in electronics and information engineering from Southwest Normal University, in 2003, the MS degree in mathematics from He'nan University, in 2008, and the PhD degree in information security from the Beijing University of Posts and Telecommunications, in 2012. He was a Visiting Scholar with the School of Computing and Informatics, University of Louisiana at Lafayette and with the Department of Computing Science, University of Alberta. He is currently a Professor with Xuchang University. His research interests include machine learning, public key cryptography, data privacy and security, and cloud and edge computing.