Computers & Security 92 (2020) 101740

Contents lists available at ScienceDirect

Computers & Security

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

Byte-level malware classification based on markov images and deep learning

Baoguo Yuan[a], Junfeng Wang[b,*], Dong Liu[c], Wen Guo[c], Peng Wu[a], Xuhua Bao[d]

[a] *College of Computer Science, Sichuan University, Chengdu 610065, China*
[b] *School of Aeronautics and Astronautics, Sichuan University, Chengdu 610065, China*
[c] *Science and Technology on Reactor System Design Technology Laboratory, Nuclear Power Institute of China, Chengdu 610065, China*
[d] *Qi anxin technology group co. LTD, Beijing 100089, China*

A R T I C L E I N F O

Article history:
Received 28 September 2019
Revised 14 January 2020
Accepted 30 January 2020
Available online 31 January 2020

Keywords:
Cyber security
Malware classification
Gray images
Markov images
Deep learning

A B S T R A C T

In recent years, malware attacks have become serious security threats and have caused huge losses. Due to the rapid growth of malware variants, how to quickly and accurately classify malware is critical to cyber security. As traditional methods based on machine learning are limited by feature engineering and difficult to process vast amounts of malware quickly, malware classification based on malware images and deep learning has become an effective solution. However, the accuracy rate of existing method based on gray images and deep learning (GDMC) still needs to be improved. Moreover, it is heavily dependent on the amount of training dataset. To improve the accuracy, this paper proposes a byte-level malware classification method based on markov images and deep learning referred to as MDMC. The main step in MDMC is converting malware binaries into markov images according to bytes transfer probability matrixs. Then the deep convolutional neural network is used for markov images classification. The experiments are conducted on two malware datasets, the Microsoft dataset and the Drebin dataset. The average accuracy rates of MDMC are respectively 99.264% and 97.364% on the two datasets. Further experiments on different proportions of training dataset and testing dataset also show that MDMC has better performance than GDMC.

© 2020 Elsevier Ltd. All rights reserved.

1. Introduction

Malware is the computer program that infiltrates and damages a computer without the user's consent. With the popularity of internet, malware has become a profitable tool and political weapon for criminals. The cyber attacks and cyber crimes using malware have increased dramatically in recent years[1]. For example, the global ransomware, WannaCry, infected more than 230,000 computers and caused losses of $8 billion in April 2017 (BBC, 2017). In March 2019, Norway's Norsk Hydro, one of the world's largest aluminum producers, was attacked by a new ransomware, LockerGoga, and forced to temporarily close multiple plants (Ionut, 2019). Malware attacks have become serious security threats to industrial production and have caused huge losses.

According to Rising's 2018 annual report (Rising, 2018), the Rising Cloud Security System intercepted a total of 77.86 million virus samples and found 1.125 billion virus infections. The total number of viruses increased by 55.63% compared with the same period in 2017. Especially in popular malware attacks such as mining malware and ransomware, the number of malware has increased significantly. The vast majority of new malware variants come from known malware with less than 2% code differences between variants according to the study of Greengard and Samuel (Greengard, 2016). It indicates that most malware samples of the same family have certain similarity in functionality, which is an important basis for malware classification. Faced with the endless of malware variants, how to quickly and accurately classify malware variants is critical to cyber security.

The current methods (Catak and Yazı, 2019; Gandotra et al., 2014; Pektaş and Acarman, 2017; San et al., 2019; Shalaginov et al., 2018) for malware classification are mainly based on static analysis or dynamic analysis to obtain features. Most of these methods analyze the bytecode, assembly code, file structure or dynamic behavior of malware, and then use machine learning algorithms for classification. However, extracting static features of malware disassembly code often meets the problem of reverse analysis. When static analysis is not effective, dynamic analysis can be directly used to

* Corresponding author.
E-mail address: wangjf@scu.edu.cn (J. Wang).

[1] https://cybersecurityventures.com/global-ransomware-damage-costs-predicted-to-exceed-8-billion-in-2018/.

https://doi.org/10.1016/j.cose.2020.101740
0167-4048/© 2020 Elsevier Ltd. All rights reserved.

analyze malware behaviors. But dynamic analysis can only analyze the behavior of a specific execution path, it is hard to traverse all the paths. Meanwhile, dynamic analysis is time-consuming. These methods are not only limited by reverse analysis and time consumption, but also rely heavily on manually constructing features. Therefore, it is difficult to cope with the explosive growth of malware variants.

As traditional methods based on machine learning are limited by feature engineering and difficult to process vast amounts of malware quickly, malware classification based on malware images and deep learning has become an effective solution. However, most of existing methods (Gibert et al., 2019; Han et al., 2015; Liu and Wang, 2016; Ni et al., 2018; Yue, 2017) based on malware images and deep learning directly convert malware binaries into gray images (Nataraj et al., 2011). When training with deep learning, the gray images must be converted to a uniform size. Due to the difference of malware size, it is common to use byte truncation or image scaling methods to unify the size of images. There are two factors influencing the classification accuracy rate during the conversion. First of all, the original binary information is partially missing when the gray images are converted into a uniform size. Therefore, the accuracy rate of GDMC still needs to be improved. Secondly, there is a lot of redundant information during the conversion becausse not all bytes of binary information are conducive to malware classification. When classification of malware gray images with deep learning, it is necessary to rely on vast amounts of labeled training samples to achieve better performance.

To improve the accuracy of malware classification, this paper proposes a byte-level malware classification method, MDMC, based on markov images and deep learning. Firstly, a markov image is constructed to retain the global statistics of malware bytes. Secondly, it is combined with deep convolutional neural network to explore the potential characteristics of markov images. The proposed method only uses the binary information of malware without the reverse analysis and dynamic analysis. It can be applicable to various systems such as windows and android. The main contributions of this paper are summarized as follows:

- A kind of malware image based on markov transfer probability matrix is proposed, which can generate malware images with fixed size and effectively reduce the redundancy of bytes information.
- The deep convolutional neural network designed in this paper has fewer fully connected layers and lower output dimensions than that of VGG16 (Simonyan and Zisserman, 2014). Therefore, its parameters of fully connected layer are less.
- A framework combined markov images with deep convolutional neural network is proposed for malware classification. It has better performance than GDMC and doesn't rely on pre-trained models.

The remainder of this paper is structured as follows: The Section 2 reviews the related work of malware classification. Then Section 3 details the malware classification method based on markov images and deep learning. Experimental evaluation is given in Section 4. Finally, Section 5 concludes this paper.

2. Related work

Traditional malware classification methods were mainly based on static analysis or dynamic analysis to obtain features (Gandotra et al., 2014). Then machine learning algorithms were used for classification. On the aspect of malware images classification, Nataraj et al. (2011) first proposed the malware analysis method based on gray images of malware binaries. The binary files were converted to gray images and then the gist texture features were extracted for malware classification. As deep convolutional neural networks such as VGGNet, ResNet, and DesNet have showed significant advantages for large-scale image classification tasks (Russakovsky et al., 2015), traditional malware images classification methods have gradually been replaced by deep learning (Hasanpour et al., 2018). Therefore, it reviews the related work of malware classification from two main aspects: the methods based on feature extraction and machine learning, the methods based on malware images and deep learning.

2.1. The methods based on feature extraction and machine learning

According to the different feature extraction methods, malware classification methods based on machine learning can be divided into two categories: based on static features, based on dynamic features. The texture feature of malware images also belongs to a kind of static features.

2.1.1. Based on static features

Most of these methods analyze the bytecode, disassembly code, file structure, etc. Without actual execution, static analysis has the advantages of fast speed and high efficiency. There were various studies of malware classification with static features. For example, Shalaginov et al. (2018) conducted an indepth survey of different machine learning methods for classification of static characteristics of Windows portable executable (PE) files and offered a tutorial on how different machine learning techniques can be utilized in extraction and analysis of a variety of static characteristic of PE binaries. A labeled benchmark dataset, EMBER, was built by Anderson and Roth (2018) for training machine learning models to statically malware analysis. The dataset included features extracted from 1.1M PE binaries: 900K training samples (300K malicious, 300K benign, 300K unlabeled) and 200K test samples (100K malicious, 100K benign). Ahmadi et al. (2016) extracted static features from malware binary files and disassembled files. Integrating multi-dimensional features, this method classified 9 malware families with an accuracy rate of 99.77%. In terms of the methods based on texture feature of malware images, it converts the problem of malware classification into the problem of images classification. For instance, Liu and Wang (2016) designed a system to detect and classify the malware by converting disassembly files into gray images. The gray images were compressed by local mean method and mapped to feature vectors. To classify malware, an ensemble learning method based on k-means and diversity selection was proposed. Han et al. (2015) proposed a malware classification method that converted malware binary files into gray images and entropy graphs. The experimental results showed that their method can effectively distinguish malware families. However, extracting static features of malware disassembly code often meets the problem of reverse analysis. The method based on static features would be invalid, when it failed to obtain malware disassembly code.

2.1.2. Based on dynamic features

In dynamic analysis, malware is automatically executed in sandbox or controlled physical environment to observe its behavior. When static analysis is noneffective, dynamic analysis can be directly used to analyze malware behaviors. The dynamic analysis has the advantages of low false alarm rate and high accuracy rate because it obtains the actual runtime behaviors of malware, such as file access, API call, network communication, etc. For example, Lim et al. (2015) presented a malware classification method based on clustering of flow features and sequence alignment algorithms. With real malware traffic, they identified the most appropriate method for classifying malware by the similarity of network activity. In 2017, a methodology was presented by Pektaş and Acarman (2017) to build the feature vector by using run-time behaviors and apply online machine learning algorithms for malware classification. Their experimental results showed that the training and testing accuracies are 94% and 92.5% respectively. In 2018, Kim (2018) proposed a NLP method to model API sequences. The features were extracted by TF-IDF, word bag and n-gram models, and linear SVM was used for malware classification, achieving an accuracy rate of 95%. San et al. (2019) proposed malware family classification system for 11 malicious families by extracting their prominent API features from the reports of cuckoo sandbox. To classify malicious software, the classifiers, Random Forest (RF), K-Nearest Neighbor (KNN), and Decision Table (DT) were used in the system. Catak and Yazı (2019) analyzed 7107 different malicious software belonging to various families such as virus, backdoor, trojan in an isolated sandbox environment by recording the API calls made with the Windows operating system. Then sequentially analysis results were transformed into a format where different classification algorithms and methods can be used. Although dynamic analysis has higher accuracy, this kind of method can only analyze the behavior of a specific execution path, and it is difficult to traverse the behavior characteristics of all the execution paths. Meanwhile, dynamic analysis is time-consuming. Therefore, it is difficult to cope with the explosive growth of malware variants.

2.2. The methods based on malware images and deep learning

As traditional methods based on machine learning have some limitations, malware classification based on malware images and deep learning has become an effective solution because it eliminates a lot of feature engineering works.

In the past two years, malware classification with deep learning has become more attractive. For example, Yue (2017) proposed a deep convolutional neural network with weighted softmax loss for malware images classification. Their experiments indicated that the new loss function can fit other typical convolutional neural networks with an improved classification performance. Ni et al. (2018) proposed a malware classification algorithm called MCSC, which converted the disassembled malware codes into gray images based on simhash and then identified their families by convolutional neural network. Kalash et al. (2018) also proposed a CNN-based architecture for malware classification. This method used the pre-training models of VGG16 and achieved high accuracy. Similarly, Rezende et al. (2017) also used the transfer learning to apply the ResNet-50 architecture for malware classification. The deep neural network was trained by freezing the convolutional layers of ResNet-50 pre-trained on the ImageNet dataset. Motivated by the visual similarity between malwares from same family, Gibert et al. (2019) proposed a file agnostic deep learning approach for malware categorization based on a set of discriminant patterns extracted from malware images. Bhodia et al. (2019) compared the malware classification based on image transfer learning with a simple machine learning algorithm, KNN. The experimental results showed that the method based on image transfer learning is better than KNN in the simulated zero-day experiment.

Most of the above methods based on deep learning convert malware binaries into gray images. Gray images must be converted into a uniform size when training with deep neural networks. Existing methods pay little attention to the problem that some byte information is missing during the conversion. On the other hand, there is a lot of redundant information during the convertion. Most current methods rely on pre-trained models or vast amounts of labeled training samples to achieve better performance.

3. The proposed method

To improve the accuracy of malware classification, the MDMC is proposed based on markov images and deep learning. The framework of MDMC is shown in Fig. 1, which includes two steps: markov images generation and malware images classification. The first step is converting malware binaries into markov images. The markov images based on bytes transfer probability matrixs are pixel matrixs with fixed size. Compared with gray images, it does not consider the problem of resizing. The second step is malware images classification with deep convolutional neural network. The malware images are divided into training dataset and testing dataset. When training, only the samples in training dataset are used. The testing dataset is used to evaluate the trained models.

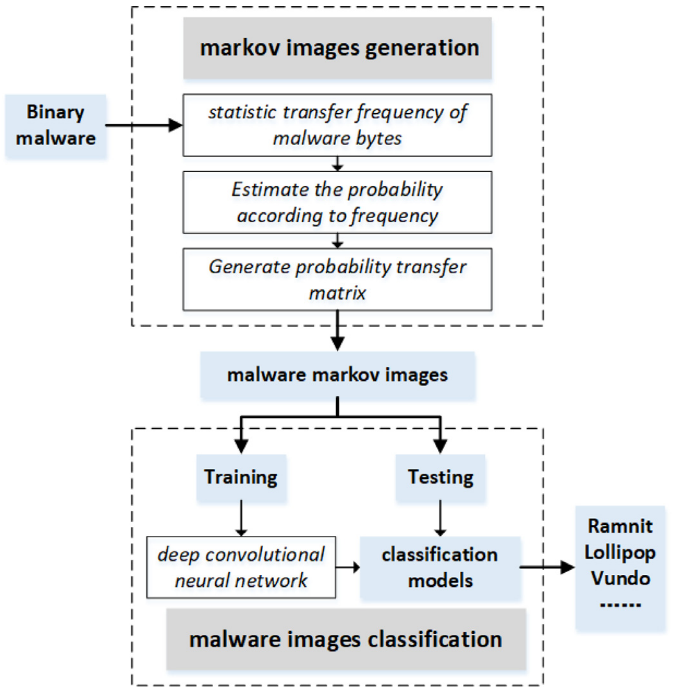

markov images generation
Binary malware
statistic transfer frequency of malware bytes
Estimate the probability according to frequency
Generate probability transfer matrix
malware markov images
Training
Testing
deep convolutional neural network
classification models
Ramnit
Lollipop
Vundo
......
malware images classification

Fig. 1. The framework of malware classification based on markov images and deep learning.

3.1. Markov images generation

In most current malware classification methods based on malware images, the bytes of malware binaries are directly used as pixel points in gray images. The influence of information loss in the pre-processing of truncation or scaling of malware images on the final accuracy is not considered. Moreover, it does not take into account the negative impact of large amount of information redundancy brought by this direct conversion. To alleviate the above problems, the malware markov images based on bytes transfer probability matrixs are proposed. The bytes of malware binaries are viewed as bytes stream which can be represented as a stochastic process as follow:

$$Byte_i,\ i \in \{0, 1, ..., N-1\} \tag{1}$$

where N refers to the length of malware binary. Because most new malware comes from known malware with minimal code differences, malware of the same family also has great similarity in bytes distribution. Assuming that the probability of $Byte_i$ is only related to $Byte_{i-1}$, then the random variable $Byte_i$ is a markov chain:

$$P(Byte_{i+1}|Byte_0, ..., Byte_i) = P(Byte_{i+1}|Byte_i) \tag{2}$$

Since the value of each byte ranges from 0 to 255, $Byte_i$ has 256 possible states, $Byte_i \in \{0, 1, ..., 255\}$. The markov transfer probability matrix is generated based on the transfer probability of each state. If $P_{m,n}$ indicates the transfer probability that the subsequent

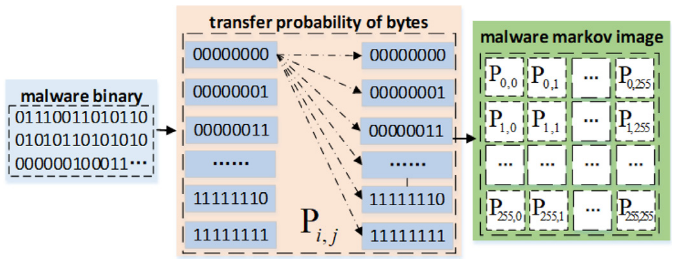

transfer probability of bytes
malware markov image
malware binary
01110011010110
01010110101010
000000100011…
00000000
00000001
00000011
……
11111110
11111111
$P_{i,j}$
$P_{0,0}$ $P_{0,1}$ … $P_{0,255}$
$P_{1,0}$ $P_{1,1}$ … $P_{1,255}$
$P_{255,0}$ $P_{255,1}$ … $P_{255,255}$

Fig. 2. The schematic diagram of converting malware binaries into markov images.

byte of m is n, then the formula for $P_{m,n}$ is as follow:

$$p_{m,n} = P(n|m) = \frac{f(m,n)}{\sum_{n=0}^{255} f(m,n)} \tag{3}$$

where $f(m, n)$ indicates the frequency at which m is followed by n, and $m, n \in \{0, 1, ..., 255\}$. Considering the probability that all bytes are transferred to each other, the transfer probability matrix M is formed:

$$M = \begin{bmatrix} p_{0,0} & p_{0,1} & \cdots & p_{0,255} \\ p_{1,0} & p_{1,1} & \cdots & p_{1,255} \\ \vdots & \vdots & \ddots & \vdots \\ p_{255,0} & p_{255,1} & \cdots & p_{255,255} \end{bmatrix} \tag{4}$$

Malware markov images are generated based on the matrix M. In markov images, each transfer probability $P_{m,n}$ corresponds to a pixel point at $M_{m,n}$.

The schematic diagram of converting malware binaries into markov images is shown in Fig. 2. The steps of markov images generation are summarized as follows:

1. **Step 1:** The malware binaries are viewed as byte streams. The transfer frequency of each byte to other bytes is counted.
2. **Step 2:** The probability is estimated by frequency according to the formula 3. The transfer probability of each byte to other bytes is calculated.
3. **Step 3:** The byte transfer probability matrix is generated according to the formula 4. The markov images of malware are generated based on the matrix M.

Because the size of malware markov images is fixed, they can be directly used as input when classification with deep learning. On the other hand, the pixel values of malware markov images are byte transfer probabilities rather than the actual values of bytes. They represent the distribution characteristics of malware binaries and can reduce the information redundancy to some extent.

3.2. Malware images classification

The deep convolutional neural network (DCNN) is used for malware images classification. The structure of DCNN designed in this paper is shown in Fig. 3, which is based on VGG16 (Simonyan and Zisserman, 2014). The DCNN is comprised of neurons with learnable weights and biases which consists of input layer, convolution layer, pooling layer, full connection layer and output layer.

Considering that the size of malware markov images is fixed, the dimension of input layer is two-dimensional matrix whose length and width are both 256. The depths of convolution layer and pooling layer are the same as that of VGG16, including 13 convolution layers and 5 pooling layers. In terms of the fully connected layer, it has only one layer whose output dimension is 1024 which is less than that of VGG16. This fully connected layer is directly connected to the output layer. And the label of malware class is output through the softmax function. During the training phase, the parameters of the model are learned using adam optimizer and the cross-entropy loss is used to train the network. The initial value of convolution kernel is generated randomly. The values of parameters are continuously updated in real time until reasonable values are learned.

The DCNN designed in this paper has fewer fully connected layers and lower output dimensions than VGG16. Its parameters of fully connected layer are less. Therefore, the time and space consumption during training is much less than that of VGG16. Compared with the methods that depend on transfer learning, it doesn't need pre-trained models.

Table 1
The Microsoft dataset and Drebin dataset.

Dataset	Family	ClassID	samples
Microsoft	*Ramnit*	1	1541
	Lollipop	2	2478
	Kelihos_Ver3	3	2942
	Vundo	4	475
	Simda	5	42
	Tracur	6	751
	Kelihos_Ver1	7	398
	Obfuscator.ACY	8	1228
	Gatak	9	1013
	Total		**10868**
Drebin	*Plankton*	1	624
	DroidKungFu	2	667
	GinMaster	3	339
	FakeDoc	4	132
	FakeInstaller	5	925
	Opfake	6	613
	BaseBridge	7	329
	Kmin	8	147
	Iconosys	9	152
	Geinimi	10	92
	Total		**4020**

4. Experimental evaluation

The MDMC is based on markov images and deep learning, which can effectively improve the accuracy of malware classification compared with GDMC. Secondly, most current methods (Bhodia et al., 2019; Gibert et al., 2019; Kalash et al., 2018; Ni et al., 2018; Rezende et al., 2017) only consider the case that the training samples are relatively sufficient. For example, when the training dataset accounts for 90% or 80% of the whole dataset, these methods can achieve high accuracy. Therefore, it also carried out experimental analysis when lack of training samples. Experiments are conducted on two kinds of malware datasets with the same structure of DCNN. The following is detailed introduction from three aspects: the datasets and experimental settings, evaluation indicators, and experimental results.

4.1. The datasets and experimental settings

The comparative experiments are conducted on two malware datasets, the Microsoft dataset (Ronen et al., 2018) and the Drebin dataset (Arp et al., 2014), as shown in Table 1. The Microsoft dataset has 10,868 labeled samples that come from 9 malware families. The malware samples in the Microsoft dataset are unpacked. When converting malware into markov images, only the binary file of each malware is used. In the Drebin dataset, the top 10 malware families are selected and a total of 4020 android malware samples are used for experiments. To eliminate the influence of irrelevant data, only the executable, *.dex* file, is used for malware markov images generation.

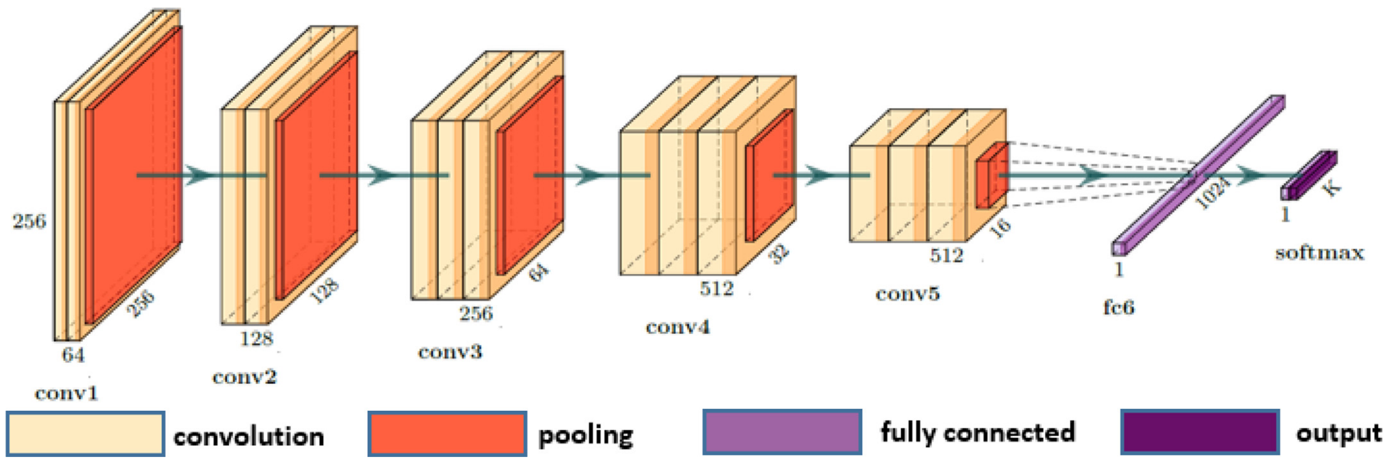


Fig. 3. The structure of deep convolutional neural network for malware images classification.

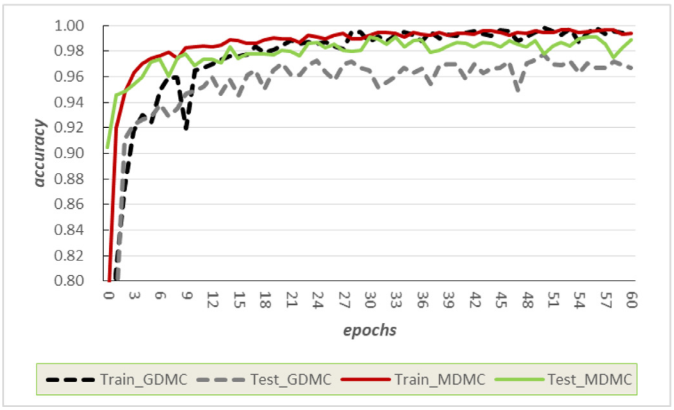


Fig. 4. The change trend of accuracy with training epochs on Microsoft dataset.

Table 2
The average results comparison of 10 folds cross-validation on Microsoft dataset.

fold	GDMC		MDMC	
	accuracy	*logloss*	*accuracy*	*logloss*
1	0.97711	0.16263	0.99451	0.03212
2	0.96697	0.27994	0.98899	0.06448
3	0.96415	0.35283	0.98713	0.07389
4	0.97516	0.22154	0.99540	0.04213
5	0.98068	0.19280	0.99172	0.05878
6	0.97882	0.16235	0.99632	0.01424
7	0.97238	0.29136	0.99079	0.08010
8	0.97698	0.24580	0.99908	0.00876
9	0.96768	0.28188	0.98707	0.09785
10	0.97045	0.30688	0.99538	0.04567
average	**0.97304**	**0.24980**	**0.99264**	**0.05180**

In terms of experimental settings, the DCNN is built in Python with keras[2] and tensorflow[3]. The network is trained on an ubuntu server with 4 GPUs. As for hyperparameters, the learning rate is $1e-3$ which is consistent with the previous studies (Kalash et al., 2018; Simonyan and Zisserman, 2014). According to the memory size of GPU and time consumption of training, it selected suitable values for other hyperparameters. The batch size is 32, the training epoch is 250 and the decay is $1e-6$. The problem of hyperparameter optimization is not further studied because it does not improve classification accuracy significantly.

4.2. Evaluation indicators

To quantitatively evaluate the experimental results, the accuracy and logarithmic loss (**logloss**) are used as indicators to compare the performance of various methods on the whole. The accuracy refers to the fraction of correct predictions. The **logloss** is the cross entropy between the distribution of the true labels and the predicted probabilities, as shown in Eq. 5:

$$logloss = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M} y_{ij} log(p_{ij}) \quad (5)$$

where N is the number of observations, M is the number of class labels and log is the natural logarithm. If observation i is in class j, y_{ij} is 1. Otherwise, it is equal to 0. The p_{ij} represents the predicted probability that observation i is in class j.

Because the distribution of malware families is imbalanced, the accuracy and logloss are difficult to reflect the classifier's performance on each class, especially the minority. Therefore, precision, recall and f1-score are also used as indicators for each malware class, so as to comprehensively evaluate the performance.

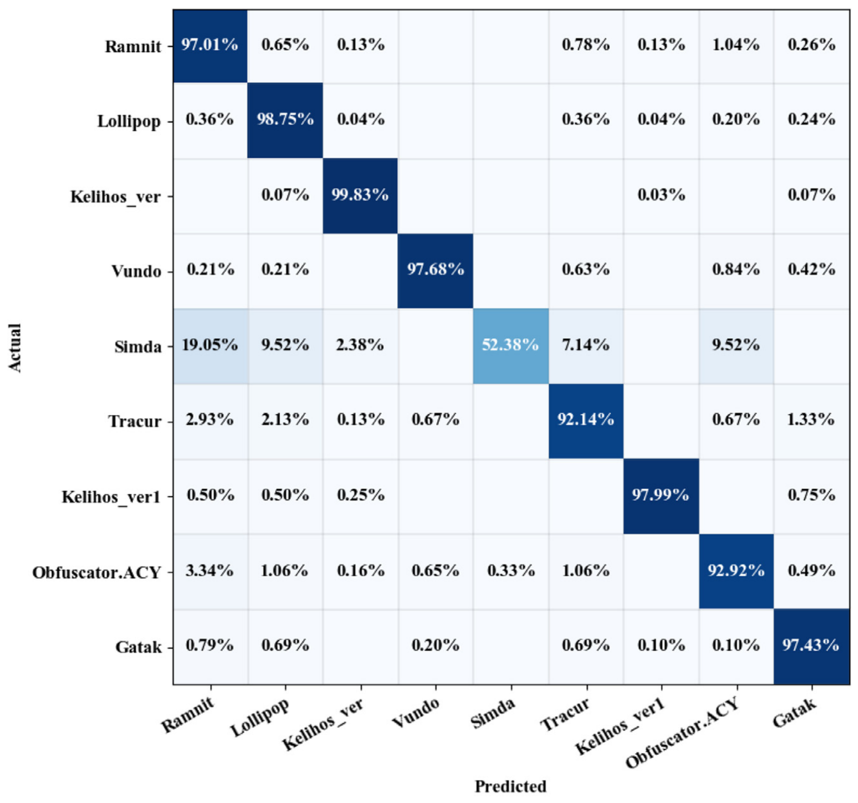
4.3. Experimental results

Similar to most studies on malware classification, it firstly compares the methods, MDMC and GDMC, when training dataset accounts for 90% and testing dataset only accounts for 10%. Two different types of malware datasets are used for experimental analysis. On each dataset, the accuracy and logloss of two methods are compared firstly. Then the precision, recall and f1-score are used to evaluate the performance on each malware class. It also fully considers the case of insufficient training samples by gradually reducing the proportion of training dataset and testing dataset. To make experiments comparable, the experiments are conducted under the same settings of DCNN.

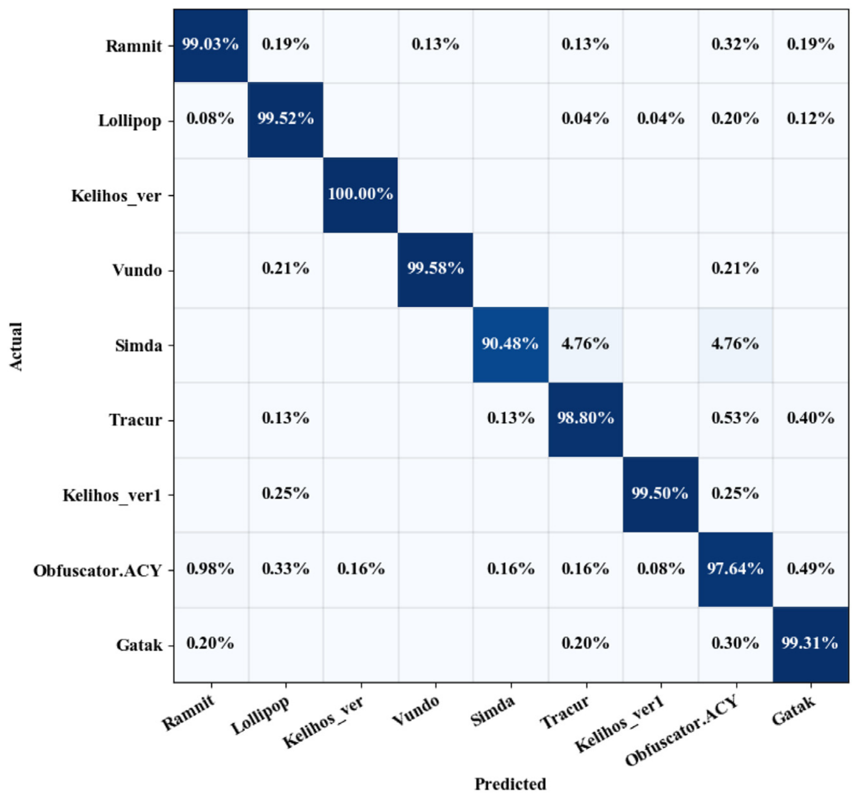
4.3.1. Experiment on Microsoft dataset

As shown in Fig. 4, it is the change trend of accuracy with training epochs. The experimental datas are recorded by TensorBoard[4]. To clear display, the experimental datas of the first 60 epochs are

[2] https://keras.io.
[3] https://github.com/tensorflow/tensorflow.
[4] https://github.com/tensorflow/tensorboard.

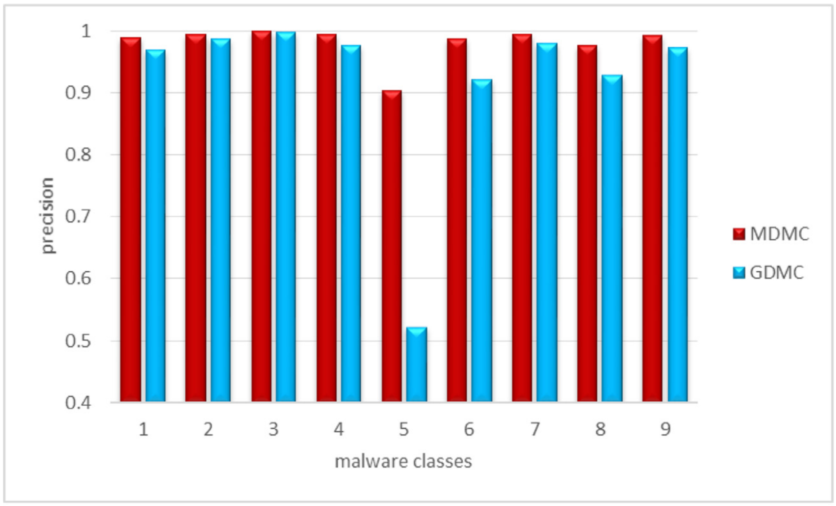


(a) GDMC

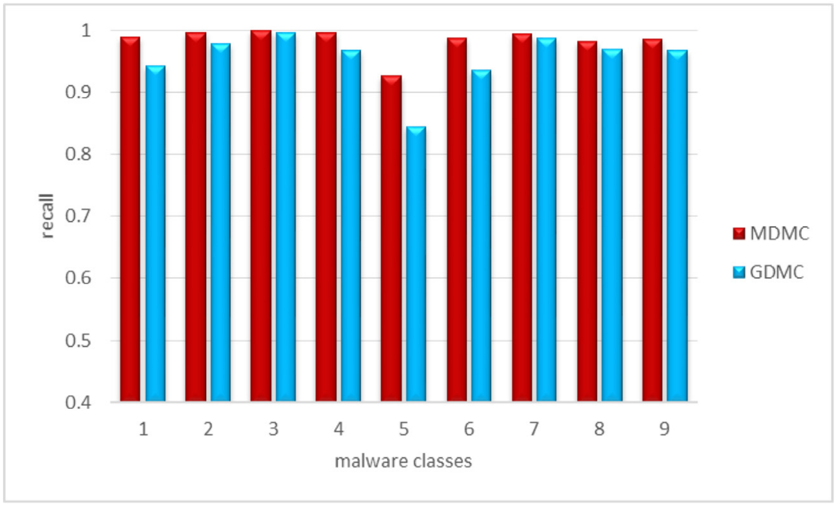


(b) MDMC

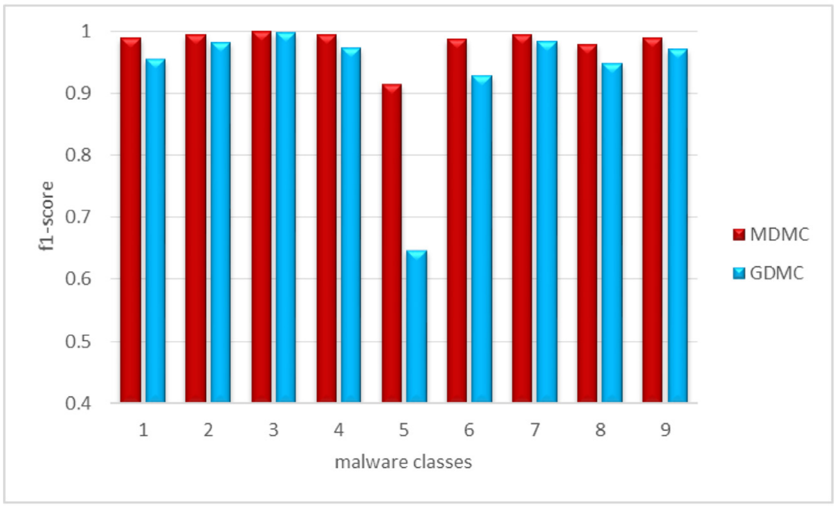
Fig. 5. The comparison of confusion matrix on Microsoft dataset.



(a) Precision



(b) Recall



(c) F1-score

Fig. 6. The comparison of precision, recall, f1-score on Microsoft dataset.

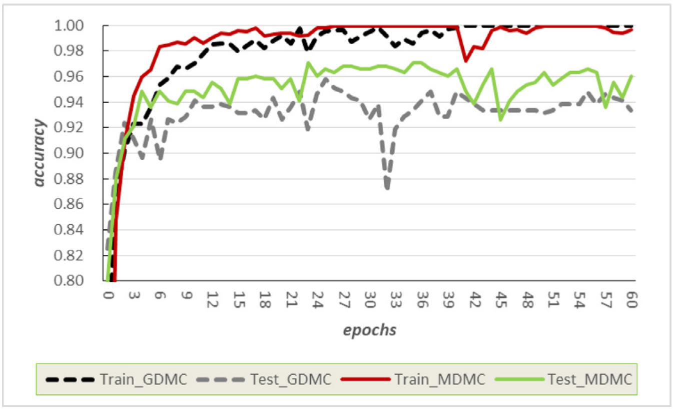


Fig. 7. The change trend of accuracy with training epochs On Drebin dataset.

used. The horizontal axis indicates training epochs, and the vertical axis indicates accuracy. The black and gray dashed lines represent the training accuracy and testing accuracy of GDMC respectively. The red and green solid lines represent the training accuracy and testing accuracy of MDMC respectively. It can be seen from the figure that the training accuracy of two methods converges rapidly with the increase of training epochs. Compared with GDMC, the training accuracy of MDMC converges faster. Secondly, when the training accuracy is stable, the testing accuracy of GDMC can reach about 96%. However, the testing accuracy of MDMC fluctuates in the range of 98% to 100%.

Fig. 4 is just the accuracy comparison of two methods in single experiment. To avoid random errors, the 10 fold cross-validation method is used to quantitatively evaluate the average results. As shown in Table 2, the testing accuracy and logloss of two methods are compared under the 10 folds cross-validation condition. The first row in the table indicates the results of two methods in the first fold. It can be found from the table that the testing accuracy of MDMC is higher and logloss of MDMC is lower in each fold. Comparing the average results, the performance of MDMC is better.

In addition to comparison of accuracy and logloss, the performance of two methods on each class is also compared. As shown in Fig. 5, it is the confusion matrix comparison of two methods. Fig. 5 a and b represent the confusion matrix of two methods respectively. In the confusion matrixs, each column represents the actual label of malware, and each row represents the predicted label of malware. If Y is the actual label, and Y' represents the predicted label. Then, the value at (Y, Y') indicates the percentage of samples whose predicted label is Y' but the actual label is Y in all malware samples whose actual label is Y. Therefore, the values on the diagonal of the confusion matrix indicate the precision of each class, and the values elsewhere indicate the misclassification. To avoid random errors, the average of confusion matrix is calculated under the condition of 10 folds cross-validation. As can be seen from Fig. 5a and b, MDMC has higher precision than that of GDMC in each class. Even in the minority class, MDMC can achieve a precision rate of 90.48%.

According to the above average confusion matrixs, the precision, recall and f1-score of two methods are compared. As shown in Fig. 6, the Fig. 6a–c are the comparison of precision, recall, and f1-score respectively. In the subfigures, the horizontal axis indicates the class label, and the vertical axis indicates the values of precision, recall, and f1-score. Blue and red represent GDMC and MDMC respectively. It can be seen from the subfigures that MDMC performs better than GDMC in each class under three evaluation indicators. Especially in the fifth class, the minority, GDMC performs extremely poorly, and the advantages of MDMC are more obvious.

Table 3
The average results comparison of 10 folds cross-validation on Drebin dataset.

fold	GDMC		MDMC	
	accuracy	*logloss*	*accuracy*	*logloss*
1	0.96069	0.31690	0.97052	0.20637
2	0.94103	0.47764	0.97297	0.14124
3	0.93317	0.46315	0.96535	0.25209
4	0.93052	0.58336	0.97022	0.26058
5	0.94527	0.50460	0.97512	0.23287
6	0.95012	0.42220	0.99002	0.04805
7	0.95012	0.48156	0.97257	0.17174
8	0.93985	0.53428	0.97744	0.22880
9	0.93484	0.63452	0.96742	0.16859
10	0.95214	0.38823	0.97481	0.16018
average	**0.94378**	**0.48064**	**0.97364**	**0.18705**

4.3.2. *Experiment on Drebin dataset*

Unlike the Microsoft dataset, the Drebin is an android malware dataset. Experiments on Drebin dataset show that this byte-level malware classification method is also suitable for android malware. Except for different dataset, the experimental settings and evaluation indicators in this section are consistent with the Section 4.3.1.

As shown in Fig. 7, it is change trend of accuracy with training epochs on Drebin dataset. The experimental data source, the representations of axis, the meanings of different colored lines are the same as those in Fig. 4. It can be seen that the training accuracy of two methods can also quickly converge with the growth of training epochs on Drebin dataset. Compared with GDMC, the training accuracy of MDMC also converges faster. When the training accuracy tends to be stable, the testing accuracy of MDMC fluctuates within the range of 94% to 98%. While the testing accuracy of GDMC fluctuates greatly and is less than 88% at the 32nd epoch.

Similar to Table 2, the 10 folds cross-validation method is also used on Drebin dataset. As shown in Table 3, it is the average results comparison of testing accuracy and logloss on Drebin dataset. It can be seen that the testing accuracy of MDMC is still higher and logloss of MDMC is still lower in each fold. Comparing the average results of 10 folds, MDMC has better performance than GDMC on android malware.

Compared with Microsoft dataset, the Drebin dataset has fewer samples and more classes, which makes the multi-classification more difficult. As shown in Fig. 8, it is the confusion matrix comparison on Drebin dataset. The average of confusion matrix is also calculated under the condition of 10 folds cross-validation. It can be seen that MDMC still has higher precision than GDMC on Drebin dataset.

The precision, recall and f1-score of two methods in each class on Drebin dataset are shown in Fig. 9. In the subfigures, the representations of axis, the meaning of different colors are the same as those in Fig. 6. As can be seen from the subfigures, MDMC still has more advantages than GDMC on Drebin dataset.

4.3.3. *Experiment on different proportions of training dataset and testing dataset*

This section fully considers the problem of limited training samples, and simulates the scenario by gradually reducing the proportion of training dataset and testing dataset. The experimental settings in this section are consistent with the previous. Fig. 10 is the statistical histogram of average testing accuracy in different proportions of training dataset and testing dataset.

The Fig. 10a is the result on Microsoft dataset, and the Fig. 10b is the result on Drebin dataset. In the subfigures, the horizontal axis represents the average testing accuracy and the vertical axis represents the proportion of training dataset and testing dataset. Blue and red successively represent the results of GDMC

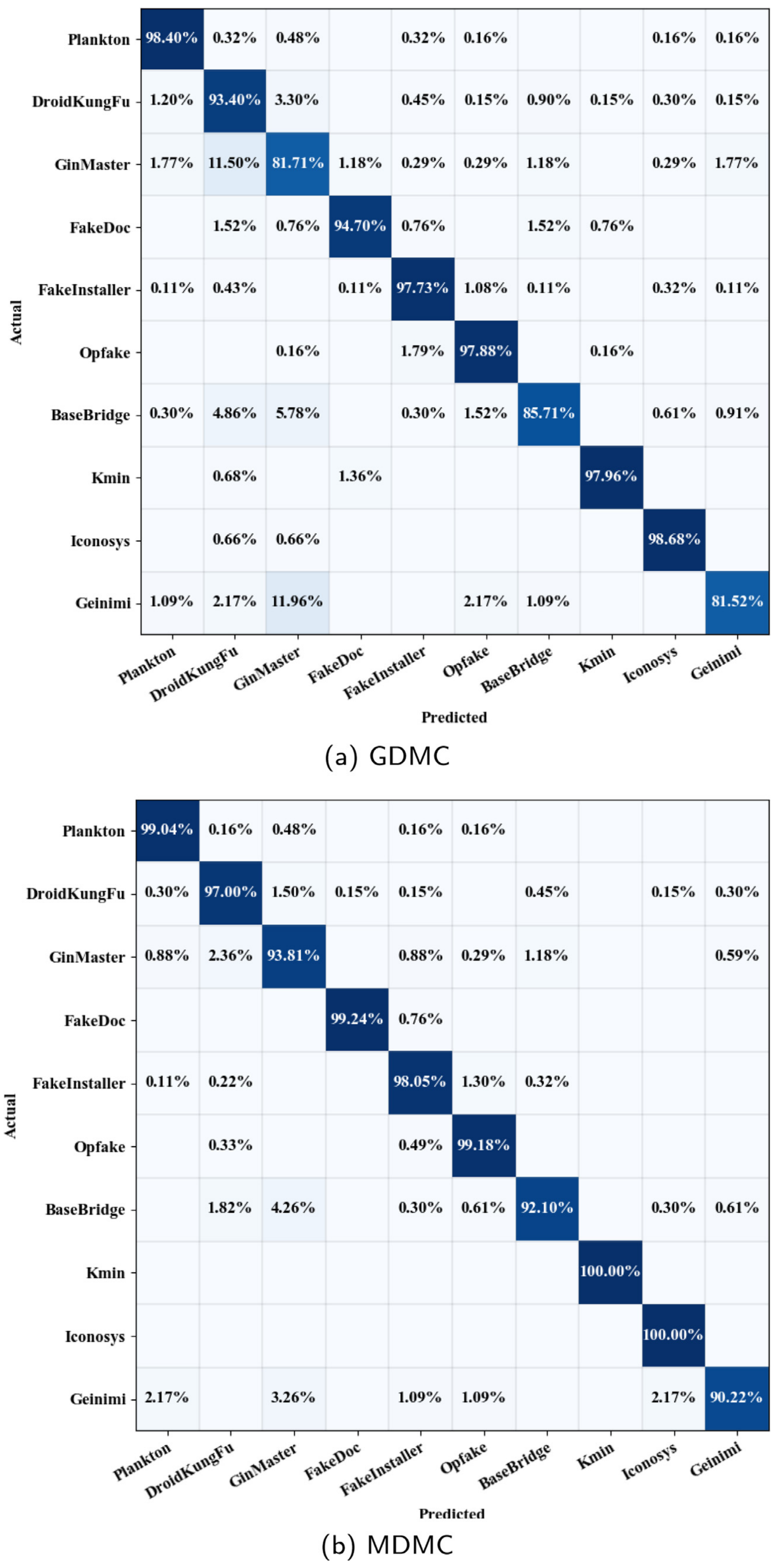
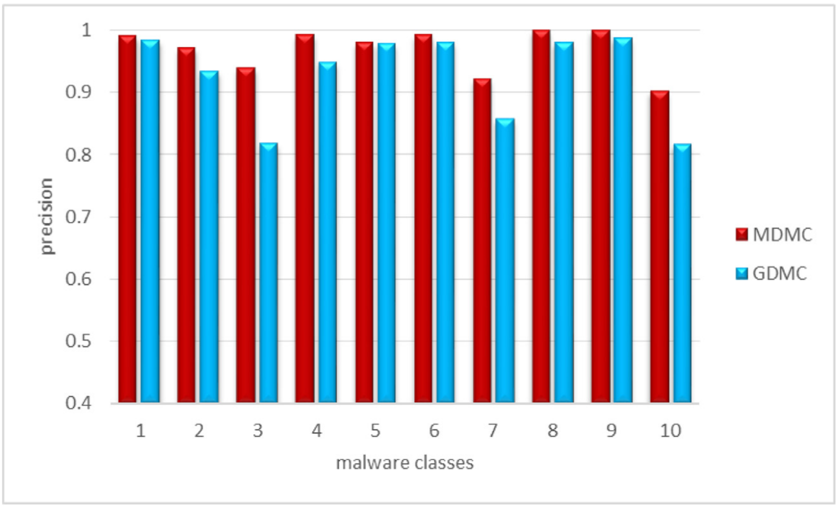
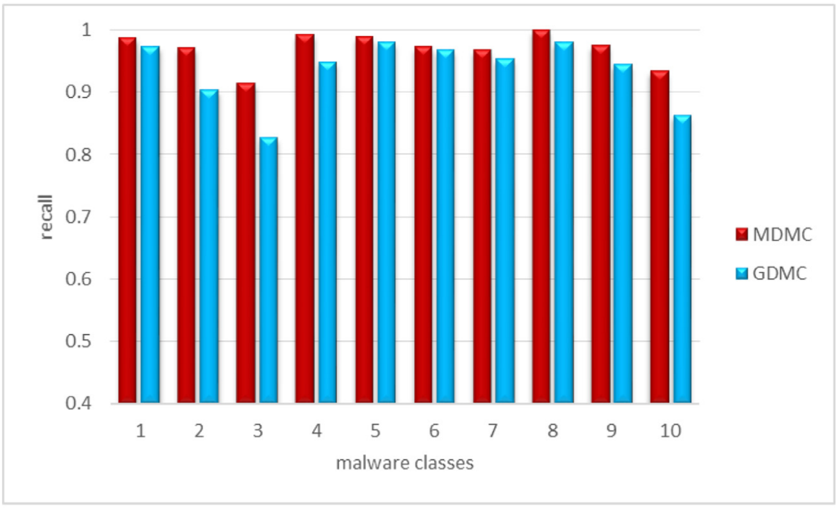


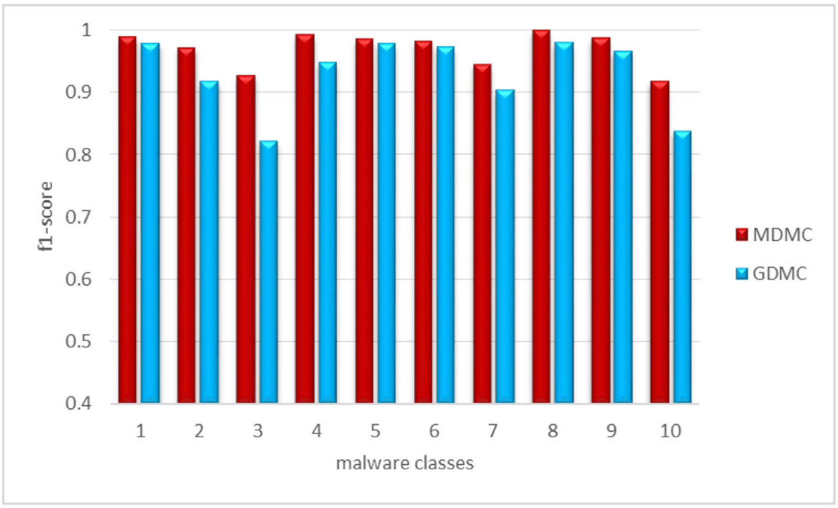
Fig. 8. The comparison of confusion matrix on Drebin dataset.



(a) Precision

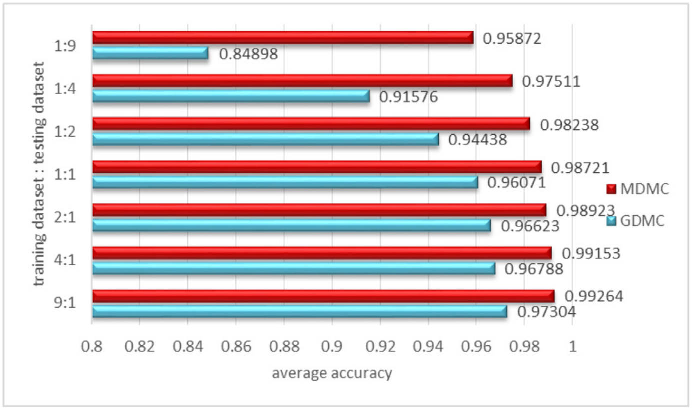


(b) Recall

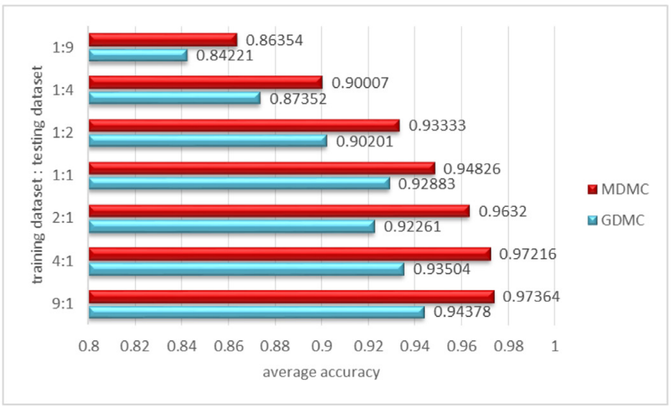


(c) F1-score

Fig. 9. The comparison of precision, recall, f1-score on Drebin dataset.



(a) On Microsoft dataset



(b) On Drebin dataset

Fig. 10. The comparison of average accuracy on different proportions of training dataset and testing dataset.

and MDMC. As can be seen from Fig. 10a, the testing accuracy of MDMC is better than GDMC in different proportions. With the decrease proportion of training dataset, the testing accuracy of GDMC decreases rapidly. However, that of MDMC is not large. Even when the training dataset is only 10% and the testing dataset is 90%, the average accuracy of MDMC is still high, reaching 95.872%. On Drebin dataset, the testing accuracy of MDMC is also higher than GDMC in different proportions as shown in Fig. 10b. Because the Drebin dataset has fewer samples and more classes than the Microsoft dataset, the classification is more difficult. When the proportion of training dataset and testing dataset is greater than 1:1, the testing accuracy rates of MDMC are 96.32%, 97.216%, and 97.364%, respectively. However, that of GDMC are all less than 95% in the same case. The experiment on different proportions shows that the method proposed in this paper can significantly improve the accuracy of malware classification.

5. Conclusion

Because static reverse analysis and dynamic analysis respectively have their own limitations, traditional machine learning algorithms are generally difficult to process massive unknown malware samples. This paper proposed a byte-level malware classification method called MDMC which converted malware binaries into markov images and classified malware markov images with deep learning. Only the binaries of malware were used without the reverse analysis and dynamic analysis. MDMC could be applicable to various systems such as windows and android. Compared with similar methods such as GDMC, MDMC could significantly improve the accuracy of malware classification.

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.

Acknowlgedgments

This work was supported by the National Key Research and Development Program (2019QY1400, 2018YFB0804503), the National Key Project (GJXM92579), and the National Natural Science Foundation of China (U1836103), and the Technology Research and Development Program of Sichuan, China (19ZDZX0024).

References

Ahmadi, M., Ulyanov, D., Semenov, S., Trofimov, M., Giacinto, G., 2016. Novel feature extraction, selection and fusion for effective malware family classification. In: Proceedings of the sixth ACM Conference on Data and Application Security and Privacy. ACM, pp. 183–194.
Anderson, H. S., Roth, P., 2018. Ember: an open dataset for training static pe malware machine learning models. arXiv:1804.04637.
Arp, D., Spreitzenbarth, M., Hubner, M., Gascon, H., Rieck, K., Siemens, C., 2014. Drebin: Effective and explainable detection of android malware in your pocket.. In: Ndss, 14, pp. 23–26.
BBC, N., 2017. Cyber-attack: Europol says it was unprecedented in scale. http://www.bbc.com/news/world-europe-39907965, Online. Accessed May 13.
Bhodia, N., Prajapati, P., Di Troia, F., Stamp, M., 2019. Transfer learning for image-based malware classification. arXiv:1903.11551.
Catak, F. O., Yazı, A. F., 2019. A benchmark api call dataset for windows pe malware classification. arXiv:1905.01999.
Gandotra, E., Bansal, D., Sofat, S., 2014. Malware analysis and classification: a survey. J. Inform. Secur. 5 (2), 56–64.
Gibert, D., Mateu, C., Planes, J., Vicens, R., 2019. Using convolutional neural networks for classification of malware represented as images. J. Comput. Virol. Hack. Tech. 15 (1), 15–28.
Greengard, S., 2016. Cybersecurity gets smart. Commun. ACM 59 (5), 29–31.
Han, K.S., Lim, J.H., Kang, B., Im, E.G., 2015. Malware analysis using visualized images and entropy graphs. Int. J. Inform. Secur. 14 (1), 1–14.
Hasanpour, S. H., Rouhani, M., Fayyaz, M., Sabokrou, M., Adeli, E., 2018. Towards principled design of deep convolutional networks: introducing simpnet. arXiv:1802.06205.
Ionut, I., 2019. Lockergoga ransomware sends norsk hydro into manual mode. https://www.bleepingcomputer.com/news/security/lockergoga-ransomware-sends-norsk-hydro-into-manual-mode/, Online. Accessed March 19.
Kalash, M., Rochan, M., Mohammed, N., Bruce, N.D., Wang, Y., Iqbal, F., 2018. Malware classification with deep convolutional neural networks. In: 2018 9th IFIP International Conference on New Technologies, Mobility and Security (NTMS). IEEE, pp. 1–5.
Kim, C. W., 2018. Ntmaldetect: a machine learning approach to malware detection using native api system calls. arXiv:1802.05412.
Lim, H., Yamaguchi, Y., Shimada, H., Takakura, H., 2015. Malware classification method based on sequence of traffic flow. In: 2015 International Conference on Information Systems Security and Privacy (ICISSP). IEEE, pp. 1–8.
Liu, L., Wang, B., 2016. Malware classification using gray-scale images and ensemble learning. In: 2016 3rd International Conference on Systems and Informatics (ICSAI). IEEE, pp. 1018–1022.
Nataraj, L., Karthikeyan, S., Jacob, G., Manjunath, B., 2011. Malware images: visualization and automatic classification. In: Proceedings of the 8th International Symposium on Visualization for Cyber Security. ACM, p. 4.
Ni, S., Qian, Q., Zhang, R., 2018. Malware identification using visualization images and deep learning. Comput. Secur. 77, 871–885.
Pektaş, A., Acarman, T., 2017. Classification of malware families based on runtime behaviors. J. Inform. Secur. Appl. 37, 91–100.
Rezende, E., Ruppert, G., Carvalho, T., Ramos, F., De Geus, P., 2017. Malicious software classification using transfer learning of resnet-50 deep neural network. In: 2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA). IEEE, pp. 1011–1014.
Rising, 2018. China cyber security report for the first half of 2018. http://it.rising.com.cn/dongtai/19390.html.
Ronen, R., Radu, M., Feuerstein, C., Yom-Tov, E., Ahmadi, M., 2018. Microsoft malware classification challenge. arXiv:1802.10135.
Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., et al., 2015. Imagenet large scale visual recognition challenge. Int. J. Comput. Vis. 115 (3), 211–252.

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: asurvey 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:1409.1556.
Yue, S., 2017. Imbalanced Malware Images Classification: a cnn Based Approach. Cornell University Library, pp. 1–6.

Baoguo Yuan received the B'S degree in Computer Science and Technology from Sichuan University of China, Chengdu in 2015. He is currently a doctoral candidate of Computer Science and Technology, Sichuan University. His recent research interests include information security, malware analysis and data mining.

Junfeng Wang received the M.S. degree in Computer Application Technology from Chongqing University of Posts and Telecommunications, Chongqing in 2001 and Ph.D. degree in Computer Science from University of Electronic Science and Technology of China, Chengdu in 2004. From July 2004 to August 2006, he held a postdoctoral position in Institute of Software, Chinese Academy of Sciences. From August 2006, Dr. Wang is with the College of Computer Science and the School of Aeronautics & Astronautics, Sichuan University as a professor. He serves as an associate editor for several international journals. His recent research interests include network and information security, spatial information networks and data mining.

Dong Liu received the M.S. degree in Computer Science and Technology from Sichuan University, Chengdu in 2002 and Ph.D. degree in Computer Application from University of Electronic Science and Technology of China, Chengdu in 2007. From 1995, he started his career in Nuclear Power Institute of China for nuclear energy software and information technology research and development. Now he is a professor-level senior engineer and deputy chief designer of the nuclear energy software project. His recent research interests include nuclear energy software development, high performance technology, and nuclear energy systems information security technology.

Wen Guo received the M.S. degree in Computer Science and Technology from Sichuan University of china, Chengdu in 2018. From 2018, she started her career in Nuclear Power Institute of China for nuclear energy software and information technology research and development. Now she is an assistant engineer of Science and Technology on Reactor System Design Technology Laboratory. Her recent research interests include nuclear energy software development, high performance technology, and nuclear energy systems information security technology.

Peng Wu received the M.S. degree in Computer Application Technology from Sichuan University, Sichuan in 2009. He is currently pursuing the Ph.D. degree in Computer science and technology with Sichuan University, Chendu, China. He is currently involved in research work on software homology and software authorship identification, His research interests include software security.

Xuhua Bao received the B.S. degree in electronic engineering and information science from University of Science and Technology, Hefei, China, in 2001 and the Ph.D. degree in information security from Chinese Academy of Sciences, Beijing, China, in 2008. From 2008 to 2019, He has been assigned to manager and researcher in China Information Technology Security Evaluation Center, Chinese Academy of Sciences, NSFOCUS Technology and Huawei. His research interest includes DDoS defend and situation awareness. He published the book King of Destruction: an In-depth Analysis of DDoS Attacks and Prevention.