Received June 9, 2019, accepted July 6, 2019, date of publication July 9, 2019, date of current version July 25, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2927552

Malware Classification Using Probability Scoring and Machine Learning

DI XUE, JINGMEI LI, TU LV, WEIFEI WU, AND JIAXIANG WANG
College of Computer Science and Technology, Harbin Engineering University, Harbin 150001, China

Corresponding author: Jingmei Li (lijingmei@hrbeu.edu.cn)

This work was supported by the National Key Research and Development Plan of China under Grant 2016YFB0801004.

ABSTRACT Malware classification plays an important role in tracing the attack sources of computer security. However, existing static analysis methods are fast in classification, but they are inefficient in some malware using packing and obfuscation techniques; the dynamic analysis methods have better universality for packing and obfuscation, but they will cause excessive classification cost. To overcome these shortcomings, in this paper, we propose a classification system Malscore based on the probability scoring and machine learning, which sets the probability threshold to concatenate static analysis (called Phase 1) and dynamic analysis (called Phase 2). The convolutional neural networks with spatial pyramid pooling were used to analyze the grayscale images (static features) in Phase 1, and the variable *n-grams* and machine learning were used to analyze the native API call sequences (dynamic features) in Phase 2. Malscore combined static analysis with dynamic analysis not only accelerated the static analysis process by taking advantage of the CNN in image recognition but also appeared to be more resilient to obfuscation by the dynamic analysis. Different from other static and dynamic analysis techniques, when malware is detected, due to the fact that malware will most likely be labeled only by static analysis, we could reduce the overheads by dynamically analyzing a few malware that has less obvious features or greater confusion in static analysis. We performed experiments on 174 607 malware samples from 63 malware families. The result showed that Malscore achieved 98.82% accuracy for malware classification. Furthermore, Malscore was compared with the method of using static and dynamic analysis. The preprocessing and test time represented a reduction of 59.58% and 61.70%, respectively.

INDEX TERMS Grayscale image, native API call, malware, machine learning, probability scoring, static and dynamic analysis.

I. INTRODUCTION

The emergence of various automated tools has shown that the speed with which malware mutates on the Internet is far faster than people realized. Kaspersky Labs detected 15,714,700 malicious objects in 2017 [1], while the number of malicious files detected by McAfee Labs increased to 79 million per day in 2018 Q1 (Q1 means the first quartal), up from 45 million in 2017 Q4 (Q4 means the fourth quartal) [2]. Although the speed with which malware mutates on the Internet is getting faster and faster, most unknown malware is derived from known malware. Therefore, finding the homology among samples plays an important role in tracing attack sources, restoring operating environments, and preventing attacks.

The associate editor coordinating the review of this manuscript and approving it for publication was Minho Jo.

Most malware can be classified by analyzing static features, but the proliferation of the packing and obfuscation techniques easily facilitates the creation of malware with consistent behavior and inconsistent static features. Dynamic analysis is required for such malware. Although dynamic analysis is very effective in behavioral analysis, it also means more cost than static analysis [3]. Thus, it is necessary to find an effective combination scheme to solve these problems. That is, static analysis can be used to classify most malware, and dynamic analysis can be fully utilized to analyze the behavior of the malware.

In this work, we propose a malware classification system Malscore based on probability scoring and machine learning. We first generate grayscale images from raw malware as static features and extract native API call sequences by executing malware in the sandbox as dynamic features. Grayscale images can reflect the overall outline and static structure of malware, and have been demonstrated to be an effective static feature [4]. API call sequences with rich semantics and not easily changed with obfuscation techniques are the most commonly used dynamic features for mining malware behavior. Then, we train the classifier S and the classifier D using grayscale images and native API call sequences, respectively. The classifier S is based on Convolutional Neural Networks (CNNs) with Spatial Pyramid Pooling (SPP) [5], and the classifier D is based on variable $n-grams(n = 2, 3, 4)$ and machine learning.

To concatenate the classifier S and the classifier D, this paper proposes a kind of probability scoring with probability threshold (PT). We set the output of the softmax layer in the classifier S as the probability value vector, and judge the credibility of the classification result by comparing the probability value vector of each malware sample with PT. The classifier D will further analyze malware with lower credibility. Malscore mainly utilizes the detection speed and low cost of static analysis, and the better robustness of dynamic analysis for packed/obfuscated malware. Packing/obfuscation malware will get a probability value vector after it is analyzed by the classifier S. Such malware is dispersed and will not be absolutely similar to any family. But such malware can easily be filtered out and entered into classifier D. In this way, Malscore improves the robustness for packed/obfuscated malware, and thus improves the classification accuracy. Because of the high-cost of dynamic analysis, if all malware samples are input into classifier D, this will greatly increase the detection cost. We use probability scoring to filter out most malware that get reliable classification results in classifier S, and only input unreliable malware into classifier D. Through this method, the execution times of dynamic analysis is reduced, and the detection cost of Malscore is reduced.

The contributions of this paper are four-fold:

1) We take grayscale images as static features and classify malware using CNN with SPP layer. Thus, Malscore can reduce the information loss of malware caused by the image preprocessing.
2) We use variable n-grams as dynamic features and apply DF.IDF to select meaningful features. Thus, Malscore can support association between features so that the semantic information in the malware is retained as much as possible.
3) We propose a probability scoring to concatenate static analysis and dynamic analysis, which can improve the classification accuracy of Malscore and reduce the classification cost of dynamic analysis in Malscore.
4) We perform a series of evaluation experiments for Malscore on a real and large malware data set. The results show that Malscore has higher accuracy and lower classification cost.

The remainder of the paper is organized as follows. Section II surveys the key research content of the related work. Section III describes the system framework of Malscore. Sections IV and V explain the process that grayscale images are analyzed by using CNN with SPP, and native API call sequences are analyzed by using variable n-grams and machine learning. The experimental and results are presented in Section VI. Section VII describes the limitations of our paper. Finally, Section VIII summarizes this paper and proposes future work.

II. RELATED WORK

A. MALWARE CLASSIFICATION USING STATIC ANALYSIS AND VISUALIZED IMAGES

In static analysis, the visualized images are extracted from the raw executable file, which represents the overall feature of malware [3], [4], [6]–[15].

Nataraj *et al.* [4] were the first to classify malware families by visualizing malware binary files as grayscale images and using similarity calculations between images. The authors perform experiments on 9,458 samples representing 25 malware families and reach a classification accuracy of 98%. A number of researchers extend the method proposed by Nataraj by using multiple classification methods [3], [6], [7], including machine learning models, to test multiple malware corpora containing more than 100,000 malware samples. Unlike Nataraj, Han *et al.* [8] propose a weighted, synthetic, multi-segmented texture fingerprint similarity matching method to detect malicious code variants and unknown malicious code. When this method classify six malicious code families, its highest accuracy can reach 85.77%. Hanbing *et al.* [9] propose a Pairwise rotation invariant co-occurrence local binary pattern feature for analyzing the grayscale image generated from binary file. This method has a good effect in solving code obfuscation. Liu *et al.* [10] and Makandar and Patrot [11] use K-Nearest Neighbor (KNN) and wavelet transform to process grayscale images generated from malware, respectively. And the accuracy of these malware classification approach can reach 98.2% and 98.88%, respectively. Furthermore, given the stirring of interest in deep learning in recent years, an increasing number of researchers propose using deep learning methods to classify malware using visualized images [12]–[15]. Tobiyama *et al.* [14] were the first to extract features from a trained recurrent neural network (RNN) and generate feature images. Then, feature images with labels were assembled to form the input to the CNN. The system classify 26 types of malware from 11 families. This method can reach a classification accuracy of 96%. In addition to extracting features from the raw malware, there are also other methods that extract higher-level features to detect malware. For example, MaMaDroid modele behavior as abstract API call sequences and transform them into Markov chains [16]. DroidSieve build features model from lightweight code and resource parsing for obfuscated malware detection and classification [17]. And data streams from sensitive resources are also proved to be an effective method for detecting malware [18].

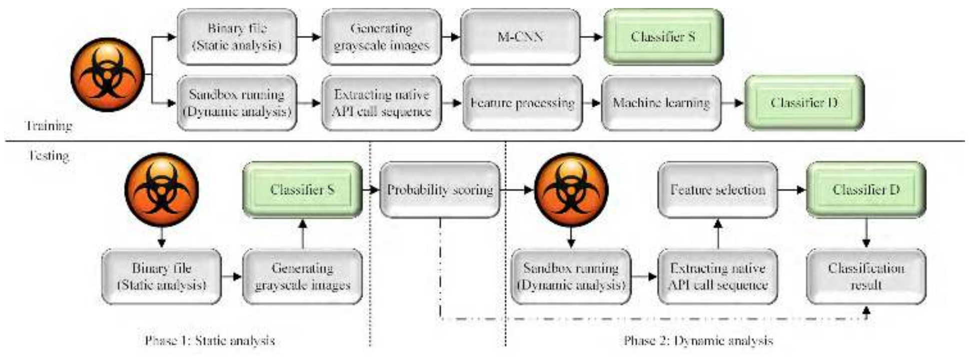
Binary file (Static analysis)
Generating grayscale images
M-CNN
Classifier S
Sandbox running (Dynamic analysis)
Extracting native API call sequence
Feature processing
Machine learning
Classifier D
Training
Testing
Classifier S
Probability scoring
Feature selection
Classifier D
Binary file (Static analysis)
Generating grayscale images
Sandbox running (Dynamic analysis)
Extracting native API call sequence
Classification result
Phase 1: Static analysis
Phase 2: Dynamic analysis

FIGURE 1. Overview of Malscore.

B. MALWARE CLASSIFICATION USING DYNAMIC ANALYSIS AND API CALLS

As background to the method proposed in this paper, this section introduces some malware classification work using API calls by dynamically executing malware. Ki *et al.* [19] extract common malicious API call sequence model from different kinds of malware using DNA sequence alignment algorithms. The model divide 26 different types of API calls with a standard of API call functions. Their work achieve a recall of 99.8%. Pekta *et al.* [20] also modele API calls with a standard of API call functions. They focus on mining meaningful behavioral features from the API call sequence, and use the TFIDF of each n-grams as the weight of each feature. This method can reach a classification accuracy of 98%. Lee *et al.* [21] collect API call sequences through the Cuckoo sandbox, then convert API calls into same type of code, and finally detect malware using n-grams and clustering coefficients. Comparing the similarity of malware take 0.0097s on a server with a 2.5GHz CPU in this method. Natani and Vidyarthi *et al.* [22] use API function frequencies as feature vectors and classify malware using classifier with ensemble learning. This method has a good classification performance in a data set with 200 malware samples. Kolosnjaji *et al.* [23] were the first to analyze API sequences using deep learning, which process dynamically API call sequences by using a neural network with convolutional and recurrent layers. From a dataset that included 10 malware families, this method can reach a classification accuracy of 89.4%. In addition to directly using API sequences as features, there are also other features model applied to dynamic analysis. For example, L. Onwuzurike *et al.* model dynamically acquired API sequences as Markov chains [24]. Afonso *et al.* [25] develop a system to monitor malware dynamically. The system detects malicious behavior of mobile applications by monitoring API function calls and system calls. DroidScribe [26] uses Support Vector Machines with Conformal Prediction to process sparse data to complete malware detection. It uses system calls and OS objects as malware features. In addition, system call sequences in dynamic analysis can also be affected by obfuscation technologies. Droidcat is a method to solve the obfuscation problem of system calls in dynamic analysis by using method calls and inter-component communication Intents [27], [28].

C. MALWARE CLASSIFICATION INTEGRATING STATIC ANALYSIS AND DYNAMIC ANALYSIS

Han *et al.* [29] propose a malware visualization method based on static and dynamic analysis in which RGB images are first generated from opcode sequences extracted from malware samples. Then, key blocks are selected to extract opcode sequences using a method that dynamically executes malware. The method calculates image similarity using pixel color information from RGB images. This method can reach a classification accuracy of 98.96%. Shijo and Salim *et al.* [30] use printable string information extracted from malware as static feature and API call sequence extracted from Cuckoo sandbox as dynamic feature. These extracted features are then converted into binary vectors, which are assembled to form the input to two machine learning algorithms, Support Vector Machine (SVM) and Random Forest (RF). Their work can reach a classification accuracy of 98.71% in the SVM algorithm and 97.68% in the RF algorithm. Kilgallon *et al.* [31] extract four static features and one dynamic feature. Static features include binary features, PE input features, string features, and PE metadata features that are processed by hash functions and combine into a fixed-dimension vector. Dynamic feature is Windows API system calls. Finally they assemble these features to form the input to the machine learning for classification. This method's best accuracy can reach 85.77%. StormDroid [32] presents a combined set of contributed features for machine learning classifiers, compared with the traditional combination of static and dynamic features. At the same time, StormDroid proposes a detection method based on Streaminglized Machine Learning, which uses the concept of flow to solve the problem of large-scale data analysis.

III. SYSTEM FRAMEWORK

System framework of Malscore is shown in Figure 1, which consists of training and testing. The training includes feature learning of the classifier S and the classifier D, and the features includes grayscale images and native API call sequences. The testing includes two Phases, named Phase 1 and Phase 2. The CNN with SPP layer is used to analyze grayscale image (static feature) in Phase 1, and the variable n-grams and machine learning are used to analyze native API call sequence (dynamic features) in Phase 2. Malscore sets the probability scoring to concatenate Phase 1 and Phase 2. In the testing Phase, we propose a probability scoring to reduce detection time. Specifically, the classifier S in Phase 1 generates a one-dimensional probability value vector $pvv(p_1, p_2, p_3, \ldots, p_{63})$, and $p_k(k = 1, 2, 3, \ldots, 63)$ represents the probability that the malware sample belongs to the family k. According to whether $max(p_k)$ is greater than PT, we can judge the credibility of the classification result. Phase 2 continues to analyze the malware samples with low credibility. The classification results of Phase 1 and Phase 2 complemented each other. Phase 2 only focused on the malware samples with low credibility in Phase 1. Whether or not the classification result of Phase 2 was the same as that of Phase 1, Malscore took the classification result of Phase 2 as the classification criterion. Because low credibility had proved that the classification results of these samples in Phase 1 are unreliable.

In Malscore, we used probability scoring to decide whether to perform dynamic analysis or not, instead of directly using entropy-based detection method [33]. This is because the method based on entropy can only detect whether malware is packed or not, and cannot detect whether malware uses dead code insertion, code transposition, subroutine reorder and other technologies that can avoid static analysis. In addition, there are still some problems in malware, such as less obvious static features and smaller packed/encryption blocks which do not affect the distribution of static features. Such malware need dynamic analysis to further judge and give more accurate classification result. So Malscore didn't directly divide samples of static analysis and dynamic analysis, but evaluated the reliability of malware in static analysis by probability scoring to decide whether to use dynamic analysis or not.

IV. ANALYSIS OF GRAYSCALE IMAGES USING CNN WITH SPP

A. GENERATION OF GRAYSCALE IMAGES

Inspired by Nataraj *et al.* [4], it is an effective static analysis method to convert malware into grayscale images. Figure 2 shows the process of converting malware into grayscale images, which requires the following steps:

1) The executable file is treated as an original binary stream.
2) Splitting the binary stream with every 8-bits and converting each 8-bits to 256-level gray value. The conversion scheme maps byte values from 0 (black) to 255 (white).
3) Gray values are converted into a two-dimensional grayscale matrix in an orderly manner. The width and height of the matrix are determined based on the size of the malware file [4] to obtain grayscale images.

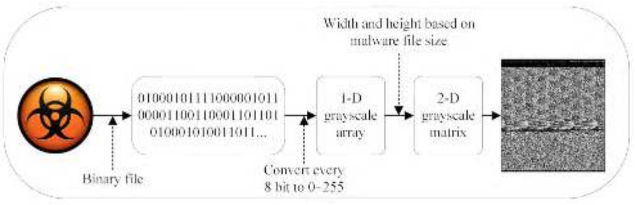

FIGURE 2. Grayscale image conversion process.

According to the scheme of Figure 2, grayscale images with different sizes were generated. But the last fully-connected layer of the CNN requires that the input feature maps be the same size, the general CNN network structure needs to preprocess the image to unify the image size. The existing processing methods segment the grayscale images [10]–[13]. However, these methods not only complicate the preprocessing of malware, but also reduce the correlation between image blocks and the effective information of images. To solve this problem, this paper constructed CNN based on VGGNet-16 [34] and SPP [5], which was named M-CNN.

B. CONSTRUCTION OF M-CNN

Considering that the size of malware may be less than 1 KB (for example, the size of Backdoor.Win32.Agent.amjd is only 640 B, which was converted into a grayscale image of 32 × 20.), M-CNN used smaller convolution filters in deeper parts of the network. As shown in Figure 3, the M-CNN has an input layer, 13 convolutional layers, 5 pooling layers, 3 full-connection layers, and an output layer. All the convolutional layers use a 3 × 3 convolution kernel with a step size of 1. Because the size of the feature map does not change when the feature map passes through a convolutional layer, a 1-pixel edge fill is performed on each input feature map in the convolution layer. The first four pooling layers use max pooling with a 2 × 2 sliding window and a step size of 2. The last pooling layer uses SPP instead of max pooling. The output of the SPP layer is a $k \times B$ dimension vector, where B represents the number of bins and k represents the number of filters in the last convolution layer. This fixed-dimensional vector forms the input to the fully-connected layer, allowing the inputs to be images of any size. From Figures 3 and 4, Malscore uses 3-layer pyramid pooling and obtains vectors of $4 \times 4 \times 512$, $2 \times 2 \times 512$, and $1 \times 1 \times 512$ dimensions. Then, the output of SPP is connected to a 21×512 dimensional vector and output to the fully connected layer.

To prevent network overfitting, the M-CNN included a dropout regularization layer with a probability of 0.5 after each pair of convolutional and pooling layers. In addition, to enhance the convergence performance of the M-CNN network, Malscore used the Leaky ReLU activation function [35], a uniformly distributed weight initialization and batch normalization [36].

To get the probability value vector of each malware sample, the output layer uses the softmax function. The classification result of the softmax is expressed by probability,

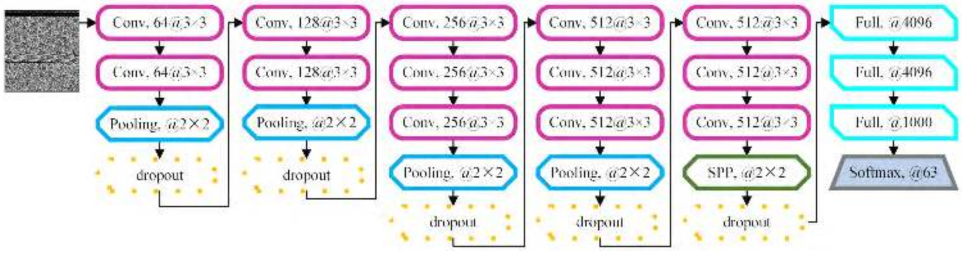

FIGURE 3. Network structure of M-CNN.

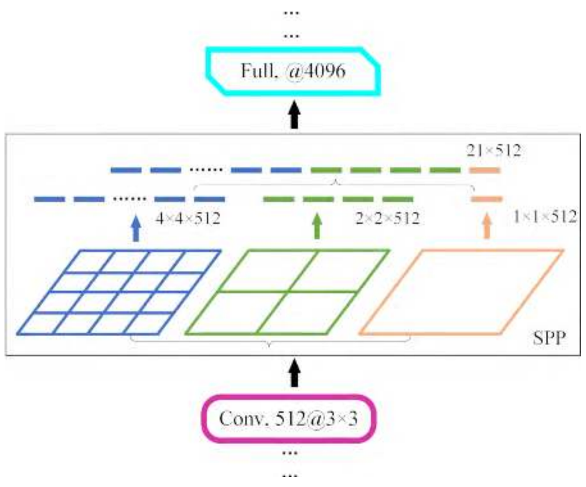

FIGURE 4. The workflow of CNN with SPP.

which increases the interpretability of classification. We used equation (1) to calculate the probability that sample x belongs to family k:

$$P(y = k|x) = \frac{e^{z_k}}{\sum_{k=1}^{63} e^{z_k}} \tag{1}$$

Whereby, z_k is the input vector of softmax.

According to equation (1), a probability value vector pvv can be calculated for each malware sample x:

$$pvv(x) = (P(y = 1|x), P(y = 2|x), \ldots, P(y = 63|x)) \tag{2}$$

When $max(P(y = k|x))$ is greater than or equal to PT, we consider that sample x must belong to family k; otherwise, sample x does not necessarily belong to family k. Further analysis using classifier D for sample x is needed to determine the family to which the sample x belongs.

V. ANALYSIS OF NATIVE API CALL SEQUENCES USING VARIABLE *N-GRAMS* AND MACHINE LEARNING

For the grayscale images generated in Section IV-A, there may be some samples of the same family whose static features are not very obvious. The classifier S may obtain classification results with low credibility for such malware. Then, Malscore used classifier D to analyze such malware for obtaining higher classification accuracy. To analyze the behavior of malware, Malscore extracted the native API call

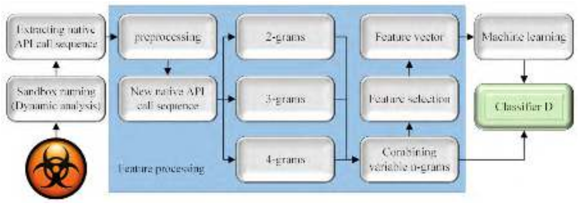
Extracting native API call sequence
Sandbox running (Dynamic analysis)
preprocessing
New native API call sequence
Feature processing
2-grams
3-grams
4-grams
Feature vector
Feature selection
Combining variable n-grams
Machine learning
Classifier D

FIGURE 5. Feature Processing.

sequences by executing malware in the Cuckoo sandbox [37]. Figure 5 shows the process of feature processing.

A. EXTRACTION OF VARIABLE N-GRAMS FEATURES

N-grams are a language model that enables computers to automatically classify text [38]. In this paper, *n-grams* were short sequences including n native API calls. In the past research work of researchers, $3-gram$ is the most widely used and $n-gram(n > 4)$ is less used. When $n \geq 5$, the greater the value of n, the greater the proportion of accuracy reduction and the more the number of parameters. The accuracy of $n-gram(n \geq 5)$ in many machine learning algorithms is 5% - 26% lower than that of $n < 5$ [10]. As long as there is enough training set theoretically, the greater the value of n, the higher the performance. In this way, the next word will have more constraints and $n-gram$ will have greater discrimination. But the actual situation is that the training set is very limited, which is easy to generate sparse data which does not satisfy the law of large numbers [38]. Meanwhile, if n is great, the parameter space will be too large to cause dimension disaster, so it will not be practical. On the contrary, the smaller the value of n, the more times $n-gram$ appears in the training set, which brings more reliable statistical information and higher practicability.

Therefore, Malscore made up *n-grams* with lengths $n = 2, 3, 4$ for mining some meaningful native API calls. We found that native API call sequences of some samples have one native API call or native API call subsequence that was called continuously, which may cause information redundancy. There were also some continuous native API calls (subsequences) that are truly important features of the malware family. According to our research experience, we set a threshold of 80%. When 80% of the malware samples in a certain malware family included the same continuous native API calls (subsequences), we identified them as salient

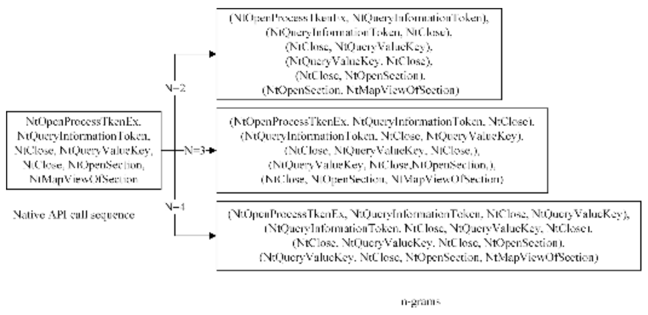
NtOpenProcessTkenEx, NtQueryInformationToken, NtClose, NtQueryValueKey, NtClose, NtOpenSection, NtMapViewOfSection
Native API call sequence
N=2
N=3
N=4
(NtOpenProcessTkenEx, NtQueryInformationToken), (NtQueryInformationToken, NtClose), (NtClose, NtQueryValueKey), (NtQueryValueKey, NtClose), (NtClose, NtOpenSection), (NtOpenSection, NtMapViewOfSection)
(NtOpenProcessTkenEx, NtQueryInformationToken, NtClose), (NtQueryInformationToken, NtClose, NtQueryValueKey), (NtClose, NtQueryValueKey, NtClose,), (NtQueryValueKey, NtClose,NtOpenSection,), (NtClose, NtOpenSection, NtMapViewOfSection)
(NtOpenProcessTkenEx, NtQueryInformationToken, NtClose, NtQueryValueKey), (NtQueryInformationToken, NtClose, NtQueryValueKey, NtClose), (NtClose, NtQueryValueKey, NtClose, NtOpenSection), (NtQueryValueKey, NtClose, NtOpenSection, NtMapViewOfSection)
n-grams

FIGURE 6. **Extraction process of *n-grams* features ($n = 2, 3, 4$).**

features. Thus, we needed to preprocess the native API call sequence before extracting *n-grams* feature, and deleted the continuous native API call or native API call subsequence existing in a few samples of the same family. The process of preprocessing is described as in Algorithm 1.

Algorithm 1 Preprocessing of Native API Call Sequences

```
Require: APISequence = Raw native API call sequence for
    each malware sample;
Ensure: NewAPISequence = Preprocessed native API call
    sequence;
 1: for family in dataset do
 2:   for sample in family do
 3:     Traversal APISequence, APIConcall ← one native
        API call or native API call subsequences that are
        called 4 times or more continuously
 4:     for APIConcall do
 5:       for sample1 in family except sample do
 6:         Traversal APISequence, if APIConcall exist,
            $N \leftarrow N + 1$
 7:       end for
 8:       if $N > 80\%$ of the number of samples in family
          then
 9:         Do not process APIConcall
10:       else
11:         Delete repetitive native API calls in
            APIConcall in the family
12:       end if
13:     end for
14:   end for
15: end for
```

We extracted the *n-grams* features from the NewAPISequence with the length of $n = 2, 3, 4$. Figure 6 shows the extraction process of *n-grams* features. The sequence on the left represents the partial native API call sequence extracted from malware, and the short sequences on the right represents extracted $2-grams$, $3-grams$, *and* $4-grams$ features.

B. SELECTION OF VARIABLE N-GRAMS FEATURES

Most *n-grams* may not be meaningful for the malware classification. We selected some common *n-grams* features in the family training classifier D, which appear less frequently in other families. Malscore proposes Document Frequency-Inverse Document Frequency ($DF \cdot IDF$) to process the selection of *n-grams* features. DF was used to select features that are common to each family, and then IDF was used to select features that appear less frequently in other families. The $DF \cdot IDF$ of *n-grams* feature is calculated by equation (3)–(5):

$$DF(i, j, k) = \frac{\{j : i \in d(k)\}}{D(k)} \tag{3}$$

Whereby, $DF(i, j, k)$ represents the document frequency of the API call sequence j containing *n-gram* i in family k; $\{j : i \in d(k)\}$ is the number of sequences including *n-gram* i in family k, i belongs to j; $D(k)$ represents the number of native API call sequences in family k.

$$IDF(i, j, k) = log_2(\frac{D - D(k)}{\{j : i \in d \cap j : i \notin d(k)\} + 1}) \tag{4}$$

Whereby, $IDF(i, j, k)$ represents the inverse document frequency of the API call sequence j containing *n-gram* i in the data set; $D - D(k)$ represents the total number of API call sequences in the data set (excluding family k); $\{j : i \in d \cap j : i \notin d(k)\}$ is the number of API call sequences containing *n-gram* i in the data set (excluding family k).

$$DF \cdot IDF(i, j, k) = DF(i, j, k) \times IDF(i, j, k) \tag{5}$$

Whereby, $DF \cdot IDF(i, j, k)$ represents the $DF \cdot IDF$ of n-gram i in the malware family k, and i belongs to j.

Based on the $DF \cdot IDF$ of each *n-grams* feature, we can assess the importance of each *n-grams* feature. Malscore selected the top t *n-grams* features in each malware family according to the descending order of $DF \cdot IDF$. Finally, $63 \times t$ *n-grams* features were selected as the input of classifier D. The Term Frequency (TF) of each *n-grams* feature was selected as the feature weight. It can be calculated as:

$$TF(i, j) = \frac{n(i, j)}{\sum_z n(z, j)} \tag{6}$$

Whereby, $TF(i, j)$ represents the frequency of *n-gram* i in the API call sequence j, and $n(i, j)$ is the number of times that i appears in j.

For example, we extracted 1,923,460 and 1,375,083 API short sequences ($n-grams$) from the Backdoor.Win32.Agent (3561 samples) and Trojan.Win32.Monder (1341 samples), respectively. We selected part of $n-grams$ to illustrate the extraction process of API short sequence of malware. Table 1 shows three API short sequence features in each family. From Table 1, the more samples containing $n-grams$ in the family, the greater the DF value. The more samples containing $n-grams$ outside the family, the smaller the IDF value. Interestingly, both Backdoor.Win32.Agent and Trojan.Win32.Monder contain the API short sequence ''NtAllocate VirtualMemory > NtRequestWaitReplyPort > NtQuery Attributes File'', but this short sequence is only a feature of Trojan.Win32.Monder, not of Backdoor.Win32.Agent.

TABLE 1. Some API short sequences (*n–grams*) extracted from the malware family.

Malware family	API Short sequence	Function	DF (num)	IDF (num)	Feature?
Backdoor. Win32. Agent	NtAccessCheckByType >NtClose	Permission query	1 (3561)	5.4486 (3116)	√
	NtQueryInformationProcess >NtQueryAttributesFile >NtQueryAttributesFile	File Attribute Acquisition	0.9677 (3446)	2.9995 (17021)	√
	NtAllocateVirtualMemory >NtRequestWaitReplyPort >NtQueryAttributesFile	File Attribute query	0.1983 (706)	4.5540 (5794)	×
Trojan. Win32. Monder	NtOpenKey >NtOpenThreadTokenEx	Thread operation	0.9582 (1285)	2.8475 (19220)	√
	NtOpenProcessTokenEx >NtQueryInformationToken >NtClose	Process operation	0.9582 (1285)	3.3826 (13264)	√
	NtAllocateVirtualMemory >NtRequestWaitReplyPort >NtQueryAttributesFile	File Attribute query	0.9911 (1329)	4.7498 (5141)	√

C. SELECTION OF MACHINE LEARNING ALGORITHMS

We train the machine learning algorithm to obtain the classifier D by using the *n-grams* feature vectors with known tags. Classifier D can detect the family of malware from unlabeled *n-grams* feature vectors. In our work, we train five machine learning algorithms, namely SVM [39], RF [40], Adaboost [41], Naive Bayes (NB) [42], and KNN [43]. Finally, we validate the performance of these machine learning algorithms and select a better classifier as the classifier D.

SVM [39] is a machine learning algorithm based on VC Dimension theory and Structural Risk Minimization theory of statistical learning theory. Standard SVM is mainly used for 2-classification problems whose basic model is to find the best Separating Hyperplane in the feature space to maximize the interval between positive and negative samples on the training set. This paper belongs to the multi-classification problem, and applies one-against-all SVM. One-against-all SVM establishes N decision boundaries for N classifications, and each decision boundary determines the attribution of one classification to all other classifications. Meanwhile, all decision boundaries can be computed iteratively by modifying the optimization problem of standard SVM.

RF [40] is a machine learning algorithm that integrates multiple trees by ensemble learning, whose basic unit is a decision tree. Unlike decision tree algorithm, RF extracts training samples to train each tree by using Random Sampling With Replacement, and randomly select some features through the tree nodes. Then the results of several weak classifiers (trees) form a strong classifier via a voting mechanism. In RF, we can modify the number of decision tree, number of features to consider for each node, and maximum depth of the decision trees to improve the classifier.

Adaboost [41] is the most popular boosting algorithm in machine learning. The Adaboost classifiers use the weak classifiers as the base classifiers, assign different weight parameters based on the error rate of the classifier, and finally accumulate the weighted prediction results as the output. In our work, we chose the relatively simple NB classifiers based on NB algorithms as the base classifiers. Compared with RF, Adaboost uses an adaptive method to change the probability distribution and learns each base classifier iteratively. In each iteration, the data weights of the last wrong classification and the weights of the base classifier with smaller classification error rate are increased. Then the linear combination of the base classifiers is regarded as a strong classifier.

NB [42] is a machine learning algorithm based on Bayesian theorem and independent assumption of feature conditions. It calculates the probability that an unknown sample is correctly predicted as belonging to given family. Naive Bayes Classifier (NBC) requires few estimated parameters, is not sensitive to missing data, and is relatively simple. In theory, NBC has the smallest error rate compared with other classification methods. But in fact, the independent assumption of feature conditions is often not valid, which has a certain impact on the correct classification of NBC.

KNN [43] is a supervised and lazy learning algorithm in machine learning. The classifier in the KNN algorithm selects K nearest neighbors for the input data point in the training set, and classifies the data point according to the category which appears most frequently in K neighbors (Maximum Voting Rule). In KNN algorithm, the selected neighbors are all correctly classified samples. The classifier does not need to use training set for training, so it has less time complexity, and is especially suitable for multi-classification problems.

VI. EXPERIMENTS AND RESULTS

In this section, we evaluated the efficiency of our proposed system Malscore on a real and large data set. First, we described the experimental environment and the data set. Second, we defined evaluation criteria for Malscore. Third, we evaluated the performance of variable *n-grams* and fixed *n-grams* on different machine learning algorithms, and the effect of changes of PT in Phase 1 on the classification results. Fourth, we showed the experimental results and time costs for multi-classification. Finally, we showed the confusion matrix of malware classification and the comparison results with other similar methods.

A. EXPERIMENTAL ENVIRONMENT AND DATA SET

The experimental data set for this paper was collected primarily through the VX Heaven website [44], which contains 271,092 tagged malware samples. There were two criteria for dataset selection used in this paper. One was that the number of samples in malware family cannot be too low, because we use deep learning. The other was all malware samples belong to Win32 platform. Before data processing, we didn't delete any samples in selected data set, and used all native malware as our data set to reduce the impact of human factors on classification results. So we chose 174,607 real and effective malware samples from 63 families. All experiments were performed on a laptop of Windows 10 v1809 operating system with Inter(R) Core i7-7700 @3.6G Hz CPU, 16G RAM and DDR4. In our experiments, the training set contained 139,687 samples and the test set contained 34,920 samples. And because our data set size was enough, our experimental data set was fixed instead of cross validation. The number of samples is shown in Table 4.

B. EVALUATION CRITERIA

We evaluated the classification performance of Malscore using *Precision*, *Recall*, $F1-score$, and *Accuracy*. The four evaluation criteria are calculated as:

$$Precision = \frac{TP}{TP+FP} \tag{7}$$

$$Recall = \frac{TP}{TP+FN} \tag{8}$$

$$F1-score = \frac{2 \times Precision \times Recall}{Precision + Recall} \tag{9}$$

$$Accuracy = \frac{correctly\ classified\ samples}{total\ number\ of\ samples} \tag{10}$$

Whereby, the true positive (TP) is the number of samples belonging to the family k that are correctly labeled as family k; the false positive (FP) is the number of samples not belonging to the family k that are erroneously labeled as family k; the false negative (FN) is the number of samples belonging to the family k that are erroneously labeled as other family.

C. EVALUATION OF N-GRAMS AND MACHINE LEARNING

We experimentally compared the influence of variable $n-grams(n = 2, 3, 4)$ with fixed *n-grams* on the classification results. Meanwhile, we also compared the influence of the five machine learning algorithms mentioned in Section V-C on the classification results. The results are shown in Figure 7. Figures 7(a) to 7(e) show the variation trend of the *Accuracy* of $2-grams$, $3-grams$, $4-grams$, and $n-grams(n = 2, 3, 4)$ with increasing number of features in different machine learning algorithms. These machine learning algorithms include SVM, RF, Adaboost, NB, and KNN.

In Figure 7(a), the *Accuracy* of the classifier with $3-grams$ is better than that of the other classifiers when the number of features is lower than 12. However, the *Accuracy* of the classifier with *n-grams* rises rapidly and exceeds the classifier with $3-grams$ when the number of features is greater than 12. When the number of features is equal to 16, the *Accuracy* of the classifier with *n-grams* is 97.55% and tends to be stable, which is 2.36% to 5.27% higher than that of the other classifiers.

Figures 7(b) and 7(c) illustrate similar growth trends for the *Accuracy* of the all classifiers with increasing number of feature in the RF algorithms and Adaboost algorithms. When the number of features is greater than 13, the *Accuracy* of the classifier with *n-grams* is 97.73% and 97.74%, respectively, which is significantly better than that of the other classifiers.

In Figure 7(d), clearly, the *Accuracy* of all classifiers in the NB algorithm is not good, regardless of the number of feature. The *Accuracy* is not stable until the number of features is equal to 20. The best *Accuracy* of all classifiers is no more than 80%. The *Accuracy* of classifiers with $2-grams$, $3-grams$, $4-grams$ and *n-grams* ranges from 41.21% to 79.69%.

From Figure 7(e), the classifiers with $2-grams$ and $3-grams$ are more accurate than $4-grams$ and *n-grams* when the number of features is lower than 10. The *Accuracy* of all classifier tends to be stable when the number of features is between 11 and 13, and the *Accuracy* of the classifier with *n-grams* is 87.14%.

From Figures 7(a) to 7(e), Although the *Accuracy* of the classifier with *n-grams* is lower than that of the other classifier when the number of features is small, the *Accuracy* of the classifier with *n-grams* is better than that of the other classifier when the *Accuracy* tends to be stable. As the number of features increases, the more the number of *n-grams* features, the more representative the semantic information of malware.

As we can see from Figure 7, the Accuracy of the classifier with $n-grams(n = 2, 3, 4)$ is better than that of the other classifiers when the Accuracy of the classifier with $n-grams$ $(n = 2, 3, 4)$ tends to be stable. Therefore, we compare the classification performance of the classifier with $n-grams(n = 2, 3, 4)$ using different machine learning algorithm in our selected data sets in Malscore. Table 2 shows the performance of Phase 2 in Malscore compared with the methods using different machine learning algorithm and [45] using deep learning on the malware dataset. Zhao *et al.* [45] with CNN can be compared with our experiments. The Phase 2 in Malscore outperforms almost other techniques when it comes to classifying malware, that is, $Accuracy = 97.73\%$. This superior performance is seen for *Precision*, *Recall* and $F1-score$ as well.

Figure 7(f) compares the *Accuracy* of the classifier with *n-grams* in different machine learning algorithms. From Figure 7(f) and Table 2, the *Accuracy* of the classifiers in RF and Adaboost algorithm is highest while that of the classifiers in NB algorithm is lowest. Although the *Accuracy* of the classifier in SVM algorithm is similar to RF and Adaboost algorithm, the *Accuracy* of the classifier in SVM algorithm

TABLE 2. The Mean and Std of different machine learning algorithm for Phase 2 in Malscore. Num represents the number of features.

Algorithm	Precision	Recall	F1-score	Accuracy
SVM (Num=16)	97.86% (±0.15%)	96.32% (±0.38%)	97.08% (±0.18%)	97.55% (±0.13%)
RF(Phase 2) (Num=13)	**97.37% (±0.52%)**	**97.53% (±0.31%)**	**97.19% (±0.32%)**	**97.73% (±0.29%)**
Adaboost (Num=13)	96.64% (±0.33%)	97.82% (±0.19%)	97.23% (±0.36%)	97.74% (±0.21%)
NB (Num=20)	76.47% (±5.29%)	83.55% (±2.64%)	79.85% (±3.01%)	79.69% (±3.29%)
KNN (Num=11)	88.91% (±0.70%)	85.65% (±0.46%)	87.25% (±0.59%)	87.14% (±0.77%)
[45]	-	-	-	93% (-)

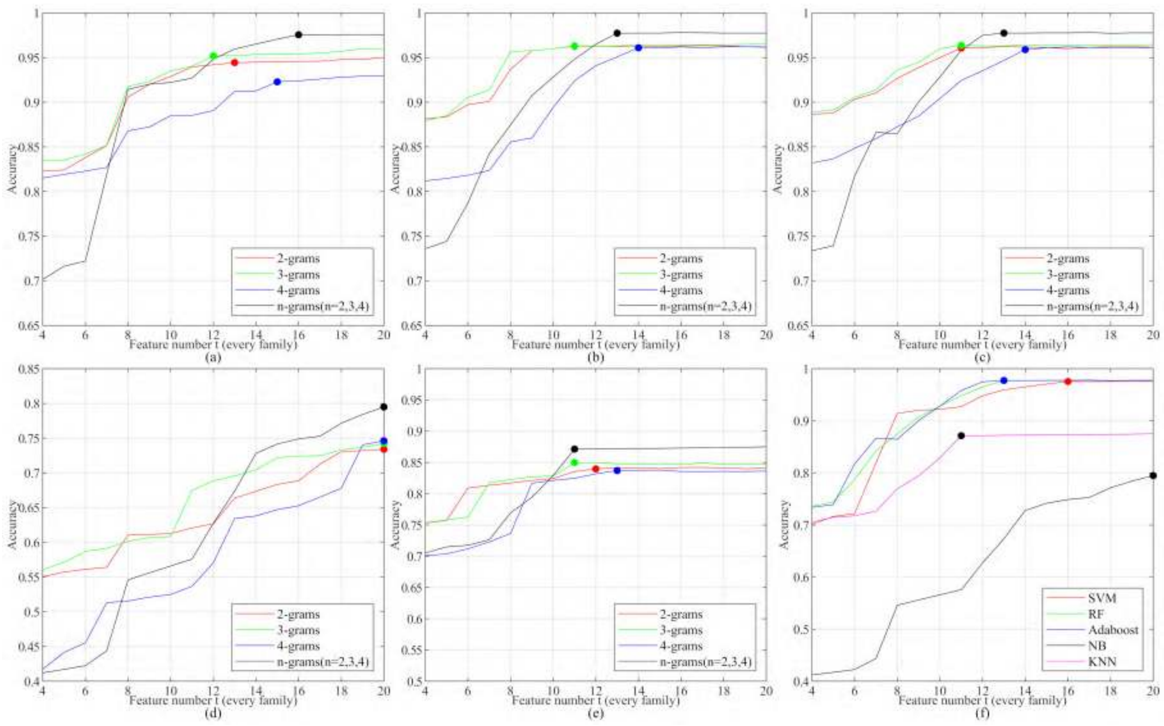

FIGURE 7. Evaluation of *n-grams* and machine learning. (a) *n-grams* with different *n* in Support Vector Machine (SVM); (b) *n-grams* with different *n* in Random Forest (RF); (c) *n-grams* with different *n* in Adaboost; (d) *n-grams* with different *n* in Naive Bayes (NB); (e) *n-grams* with different *n* in K-Nearest Neighbor (KNN); (f)different machine learning with variable *n-grams*.

TABLE 3. The comparison of time cost between RF algorithm and Adaboost algorithm.

Algorithm	Training time	Testing time	Accuracy (t=13)	Best Accuracy
RF	$18s481ms$	$4.5ms$	97.73%	97.80%(t=16)
Adaboost	$27s31ms$	$6.9ms$	97.74%	97.80%(t=17)
Comparison	+31.63%	+34.78%	+0.01%	+0%

cannot reach stability until the number of features is equal to 16.

To further verify performance of the classifiers in RF and Adaboost algorithms, we analyzed the time cost of both. The comparison results are shown in Table 3. From Table 3, the training and testing time of the classifier in Adaboost algorithm are 31.63% and 34.78% longer than that of the classifier in RF algorithm while the *Accuracy* is only 0.01% higher than that in RF algorithm. Therefore, we chose the classifier in RF algorithm as classifier D.

D. EVALUATION OF PROBABILITY THRESHOLD IN PHASE 1

In order to maximize classification advantage of Phase 1 and Phase 2, the PT was used to concatenate trained classifier S based CNN algorithm and classifier D based RF algorithm. A suitable PT can not only improve the *Accuracy* of the classification, but also minimize the time cost of Malscore. Figure 8 shows the trend of the number of samples inputted into the Phase 2 and the classification *Accuracy* as the PT increases. From Figure 8, the number of samples inputted into the Phase 2 is small, and the *Accuracy* of the Malscore is similar to the *Accuracy* of the classifier S (the green dot in Figure 8 represents the *Accuracy* of the classifier S) when the PT is lower than 0.4. So the dynamic analysis in Phase 2 will be meaningless, because most malware can be filtered. When the PT is greater than 0.89, as the number of samples inputted into the Phase 2 increases, the classification performance of the Phase 1 becomes weaker. We needed to determine PT by weighing classification accuracy and time cost. These malware samples with low credibility (possibly classified correctly, possibly incorrectly) in Phase 1 were further submitted to Phase 2 for analysis and judgment by the PT. And the suitable PT will bring the advantages of phase 1 (static analysis) and phase 2 (dynamic analysis) into full play.

As a result, the classification *Accuracy* of Malscore was only 0.14% higher than that of the classifier D (the red dot in Figure 8 represents the *Accuracy* of the classifier D). In Figure 8, the *Accuracy* of Malscore tends to be stable when the PT is between 0.79 and 0.89. When the number of samples inputted into the Phase 2 increases from 8,128 to 17,878, the *Accuracy* increases only from 98.77% to 98.89%. According to Figure 8, we chose $PT = 0.84$. When $PT = 0.84$, the classification accuracy of Malscore can achieve 98.82% while minimizing the number of samples submitted to Phase 2. When Malscore classified malware, PT was fixed as Malscore's system parameter.

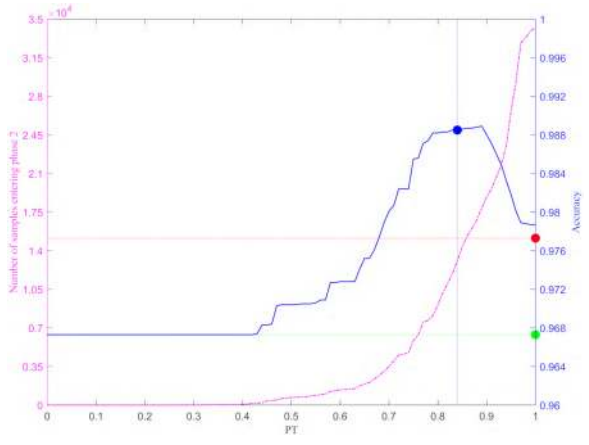

FIGURE 8. **Evaluation of probability threshold (PT) in Phase 1.**

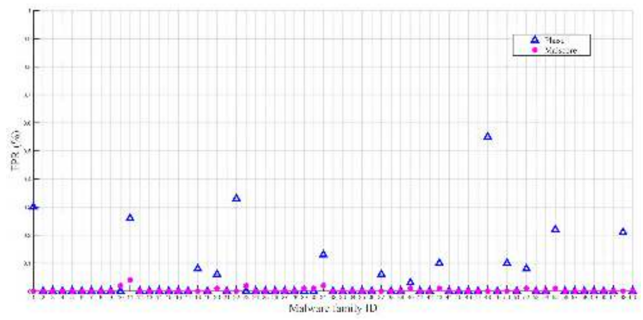

FIGURE 9. **Average FPR (false positive rate) of different malware family.**

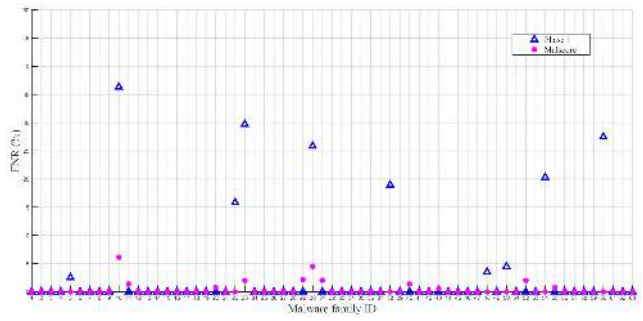

FIGURE 10. **Average FNR (false negative rate) of different malware family.**

E. EXPERIMENTAL RESULTS FOR MULTI-CLASSIFICATION

According to the evaluation in Sections VI-C and VI-D, Malscore set the probability threshold in Phase 1 is 0.84 and the machine learning algorithm in Phase 2 uses the RF algorithm. We verified the classifier S, the classifier D, the ensemble learning (both the classifier S and the classifier D) and the Malscore, and the verification results are shown in Table 4. From Table 4, the classifier D is better in some families than the classifier S, for example, the 18th family and the 22nd family. The classification accuracy of Malscore is 98.82%, which is higher than that of classifier S and classifier D, and is similar to ensemble learning.

Figures 9 and 10 show the FPR and FNR of Phase 1 and Malscore. As shown in Figures 9 and 10, Malscore has lower FPR and FNR than Phase 1 (Static Analysis), and FPR and FNR are 0% in 52 families. Because the dataset contained some malware samples using packing/encryption technology, which makes static analysis difficult. Malscore used dynamic analysis to compensate for shortcomings of static analysis, so that the classification system can recognize these samples. However, from Figures 9 and 10, Malscore cannot eliminate FPR/FNR. There were two main reasons. (1) To avoid the detection of dynamic analysis, some malware may detect the execution environment. If virtual machine environment is detected, these malware will resist running. Then, dynamic analysis still has a problem that it cannot traversed all execution paths. When malicious execution path(s) cannot be triggered, malware will be judged as ''benign''. (2) The threshold (probability scoring) we set was to minimize FPR/FNR of phase 1 (static analysis) in Malscore, and cannot guarantee that classification accuracy of Phase 1 was absolutely 100%. Even though FPR/FNR of Phase 1 was lower, it will still make a ''contribution'' to Malscore's FPR/FNR.

While Malscore reduced FPR/FNR, the decision whether or not to omit the costly dynamic analysis depending on a probability (credibility of static analysis results) for every malware sample can help us solve the problem of costly dynamic analysis. Combining with the classification results in Table 4, it is found that the recall of packed malware samples in Phase 1 is lower than Malscore.

F. CALCULATION OF TIME COST

Tables 5 and 6 show the time cost of classifier S and classifier D. From Tables 5 and 6, generation of grayscale images and extraction of *n-grams* features account for more than 99% of the time cost in the total learning process. The classifier D belonged to the dynamic analysis so that it was also much more time cost than the classifier S belonging to the static analysis. At the same time, Tables 5 and 6 also show some interesting information. For example, the test time of the classifier in RF algorithm is more than that of the classifier in CNN algorithm, but the training time is less.

Tables 7 and 8 show the time cost of the ensemble learning and the Malscore. This ensemble learning algorithm used ensemble learning idea to integrate Phase 1 and Phase 2, and was used to compare with Malscore to illustrate the advantages of Malscore. Ensemble learning was used to identify malware by integrating the results of multiple component learners (Different types of individual learners), thereby improving the final accuracy and reducing the FPR. We used the idea of bagging [46] to construct ensemble learning method, and bagging uses the playback sampling to select the training dataset of each learner, which can not only ensure that the trained learners are different, but also ensure the classification performance of component learners. In this paper, we trained two component learners h_S and h_D, h_S was classifier S based on CNN algorithm and h_D was classifier D based on RF algorithm. For each malware x, a label c_j was predicted by the learners from label set $\{c_1, c_2, \ldots, c_j, \ldots, c_{63}\}$, and we used plurality voting as a combination strategy.

TABLE 4. ***Precision*****,** ***Recall*****,** ***F*****1*****−score*** **and** ***Accuracy*** **of the four classification methods.**

ID	Family	No.	Classifier S			Classifier D			Ensemble learning			Malscore		
			recall	precision	F1-score	recall	precision	F1-score	recall	precision	F1-score	recall	precision	F1-score
1	Backdoor.Win32.Agent	4451	1	0.8981	0.9463	1	1	1	1	1	1	1	1	1
2	Backdoor.Win32.Bifrose	2852	1	1	1	1	1	1	1	1	1	1	1	1
3	Backdoor.Win32.Ceckno	806	1	1	1	1	1	1	1	1	1	1	1	1
4	Backdoor.Win32.Delf	1635	1	1	1	1	1	1	1	1	1	1	1	1
5	Backdoor.Win32.Hupigon	16472	0.9748	1	0.9872	1	1	1	1	1	1	1	1	1
6	Backdoor.Win32.IRCBot	1024	1	1	1	1	1	1	1	1	1	1	1	1
7	Backdoor.Win32.PcClient	2878	1	1	1	1	1	1	1	1	1	1	1	1
8	Backdoor.Win32.Popwin	897	1	1	1	1	0.8606	0.9251	1	1	1	1	1	1
9	Backdoor.Win32.Rbot	2161	1	1	1	1	1	1	1	1	1	1	1	1
10	Backdoor.Win32.sdBot	1225	0.6367	1	0.7781	0.8776	0.7288	0.7963	0.9673	0.9916	0.9793	0.9388	0.9746	0.9563
11	Backdoor.Win32.small	2246	1	0.8315	0.908	0.8909	1	0.9423	0.9955	0.9824	0.9889	0.9866	0.9672	0.9768
12	Backdoor.Win32.VB	1210	1	1	1	1	1	1	1	1	1	1	1	1
13	Rootkit.Win32.Agent	1555	1	1	1	1	1	1	1	1	1	1	1	1
14	Trojan.Win32.Agent	8500	1	1	1	0.9612	0.9808	0.9709	1	1	1	1	1	1
15	Trojan.Win32.BHO	1135	1	1	1	1	1	1	1	1	1	1	1	1
16	Trojan.Win32.Buzus	4010	1	1	1	1	1	1	1	1	1	1	1	1
17	Trojan.Win32.Delf	1512	1	1	1	1	1	1	1	1	1	1	1	1
18	Trojan.Win32.Dialer	1320	1	0.9041	0.9496	1	1	1	1	1	1	1	1	1
19	Trojan.Win32.DNSChanger	889	1	1	1	1	0.8683	0.9295	1	1	1	1	1	1
20	Trojan.Win32.Inject	2003	1	0.9525	0.9757	0.8978	0.9231	0.9102	0.9875	0.9875	0.9875	0.9925	0.99	0.9913
21	Trojan.Win32.Midgare	1398	1	1	1	1	1	1	1	1	1	1	1	1
22	Trojan.Win32.Monder	1676	0.8418	0.7139	0.7726	1	1	1	1	1	1	1	1	1
23	Trojan.Win32.Monderb	1011	0.703	1	0.8256	0.9406	0.8962	0.9179	0.9901	0.9662	0.978	0.9802	0.9659	0.973
24	Trojan.Win32.Obfuscated	3161	1	1	1	1	1	1	1	1	1	1	1	1
25	Trojan.Win32.Pakes	1625	1	1	1	1	1	1	1	1	1	1	1	1
26	Trojan.Win32.StartPage	904	1	1	1	1	1	1	1	1	1	1	1	1
27	Trojan.Win32.Vaklik	1150	1	1	1	1	1	1	1	1	1	1	1	1
28	Trojan.Win32.Vapsup	3502	1	1	1	1	1	1	1	1	1	1	1	1
29	Trojan.Win32.VB	1626	1	0.9969	0.9985	0.9231	0.7833	0.8475	0.9754	0.9906	0.9829	0.9785	0.9907	0.9845
30	Trojan-banker.Win32.Bancos	892	0.7416	1	0.8516	0.8596	1	0.9245	1	1	1	0.9551	0.9714	0.9632
31	Trojan-banker.Win32.Banbra	1223	1	0.8419	0.9142	0.9796	1	0.9897	1	1	1	0.9796	0.9677	0.9736
32	Trojan-banker.Win32.Banker	4728	1	1	1	1	1	1	1	1	1	1	1	1
33	Trojan-clicker.Win32.Agent	903	1	1	1	1	1	1	1	1	1	1	1	1
34	Trojan-downloder.Win32.Agent	8459	1	1	1	1	1	1	1	1	1	1	1	1
35	Trojan-downloder.Win32.Banload	5253	1	1	1	1	1	1	1	1	1	1	1	1
36	Trojan-downloder.Win32.Dadobra	651	1	1	1	1	1	1	1	1	1	1	1	1
37	Trojan-downloder.Win32.Delf	3074	1	0.967	0.9832	1	1	1	1	1	1	1	1	1
38	Trojan-downloder.Win32.Fraudload	1691	0.8107	1	0.8954	1	1	1	1	1	1	1	1	1
39	Trojan-downloder.Win32.Hmir	2605	1	1	1	1	1	1	1	1	1	1	1	1
40	Trojan-downloder.Win32.Obfuscated	1079	1	0.96	0.9796	1	0.871	0.931	1	0.9774	0.9886	0.9861	0.9771	0.9816
41	Trojan-downloder.Win32.Small	4080	1	1	1	0.9252	0.9219	0.9235	1	1	1	1	1	1
42	Trojan-downloder.Win32.VB	2091	1	1	1	1	1	1	1	1	1	1	1	1
43	Trojan-downloder.Win32.Zlob	4692	1	0.965	0.9822	0.9595	0.9967	0.9777	0.9947	1	0.9973	0.9947	0.9968	0.9957
44	Trojan-dropper.Win32.Agent	3357	1	1	1	1	1	1	1	1	1	1	1	1
45	Trojan-dropper.Win32.Small	727	1	1	1	1	1	1	1	1	1	1	1	1
46	Trojan-dropper.Win32.VB	1184	1	1	1	1	1	1	1	1	1	1	1	1
47	Trojan-gamethief.Win32.Lmir	1107	1	1	1	1	1	1	1	1	1	1	1	1
48	Trojan-gamethief.Win32.Magania	4573	0.965	0.8252	0.8897	1	1	1	1	1	1	1	1	1
49	Trojan-gamethief.Win32.Nilage	957	1	1	1	1	1	1	1	1	1	1	1	1
50	Trojan-gamethief.Win32.OnLineGames	20862	0.9552	0.992	0.9733	1	1	1	1	1	1	1	1	1
51	Trojan-gamethief.Win32.WOW	1239	1	1	1	1	1	1	1	1	1	1	1	1
52	Trojan-PSW.Win32.Agent	750	1	0.8427	0.9146	1	0.9804	0.9901	1	0.9868	0.9934	0.98	0.9545	0.9671
53	Trojan-PSW.Win32.Delf	793	1	1	1	1	1	1	1	1	1	1	1	1
54	Trojan-PSW.Win32.LdPinch	2517	0.7972	1	0.8872	1	0.9921	0.996	1	1	1	1	1	1
55	Trojan-PSW.Win32.Magania	2586	1	0.8748	0.9332	0.9671	0.9542	0.9606	0.9961	1	0.9981	0.9923	0.9942	0.9932
56	Trojan-PSW.Win32.OnLineGames	5466	1	1	1	1	1	1	1	1	1	1	1	1
57	Trojan-PSW.Win32.QQPass	1308	1	1	1	0.8092	0.9099	0.8566	1	1	1	1	1	1
58	Trojan-Spy.Win32.Agent	1216	1	1	1	1	1	1	1	1	1	1	1	1
59	Trojan-Spy.Win32.Banker	1084	1	1	1	1	1	1	1	1	1	1	1	1
60	Trojan-Spy.Win32.Delf	1365	0.7253	1	0.8408	0.9158	0.8681	0.8913	1	1	1	1	1	1
61	Trojan-Spy.Win32.Pophot	1464	1	1	1	1	1	1	1	1	1	1	1	1
62	Trojan-Spy.Win32.Zbot	2089	1	0.8496	0.9187	1	1	1	1	1	1	1	1	1
63	Worm.Win32.AutoRun	3708	1	0.9987	0.9993	1	1	1	1	1	1	1	1	1
Accuracy				0.9673			0.9773			0.9891			0.9882	

We represented the predicted output of h_S and h_D on sample x as two 63-dimensional vector $(h_S^1(x), h_S^2(x), \ldots, h_S^j(x), \ldots, h_S^{63}(x))$ and $(h_D^1(x), h_D^2(x), \ldots, h_D^j(x), \ldots, h_D^{63}(x))$, where h^j is a class probability representing the output of learners on the class label c_j. Equation 11 can calculate the voting result $H(x)$, which is the predictive label c_j of malware sample x.

$$H(x) = c_{\underset{j}{\arg\max}\ [h_S^j(x)+h_D^j(x)]} \tag{11}$$

$H(x)$ is the highest vote label. If there are multiple labels with the highest vote, one of them will be selected randomly as the label of sample x.

Table 7 shows the time cost of the total training and testing phase, and Table 8 shows the sample average of the time cost. From Tables 7 and 8, compared with the ensemble learning in testing, the preprocessing and test time of Malscore represent a reduction of 59.58% and 61.70%, respectively, when the

TABLE 5. The time cost of the classifier S (total sample).

Phase	Training		Testing	
	Preprocessing	Training	Preprocessing	Testing
Time	$5h16min$	$3h10min$	$1h47min$	$44ms$

Accuracy represents a reduction of 0.09%. Because the probability threshold of Phase 1 was set to 0.84 in Section VI-D, the number of samples inputted into Phase 2 was only 13,087, which was 21,833 less than the number of samples in the Classifier D, thus the time cost in testing was reduced.

After analysis in subsections VI-E and VI-F, Malscore's classification performance was better than static analysis, dynamic analysis and ensemble learning in terms of cost-accuracy tradeoff. With the maturity of malware technology, more and more problems will bring troubles to detection methods, such as traversing execution paths, concept drift [16], and obfuscated system calls [27], [28]. Malscore was only starting, and needed further optimization in future research work to deal with various detection difficulties.

G. CONFUSION MATRIX

Figure 11 shows the Confusion Matrix of the 63 families in the experiments. The ID of each malware family is shown in Table 4. Each column of the Confusion Matrix represents the predicted classification of the sample, and each row represents the real classification of the sample. It is evident from Figure 11 that the Malscore has the best classification capability for 52 families, in which families, the Accuracy can achieve 100%.

H. COMPARISON WITH OTHER SIMILAR METHODS

The *Accuracy* and classification cost of malware classification are the key to identifying malware classification methods. By probability scoring and machine learning, the *Accuracy* of malware classification of Malscore

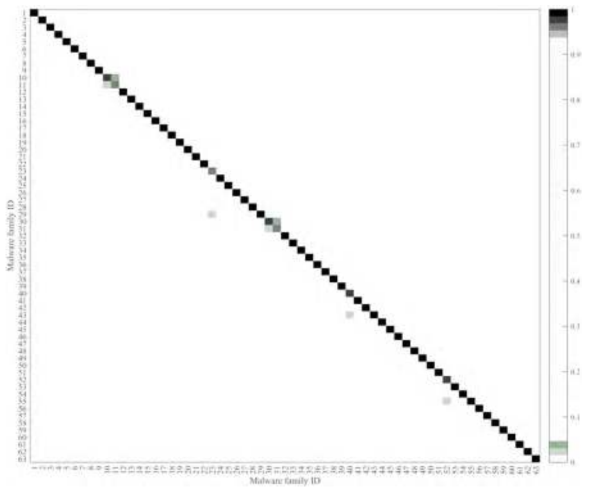

FIGURE 11. Confusion Matrix related to the malware classification.

reached 98.82%. Firstly, Malscore fused CNN network and RF algorithm results by concatenating ensemble. Concretely, concatenating ensemble combined two phases (CNN discriminant result and RF discriminant result) to train a malware classifier. Table 9 provides more inside into Malscore. Table 9 shows Malscore yields much better performance compared with Phase 1 or Phase 2 only.

Then we compared Malscore with other similar classification methods. Figures 12 and 13 provide a comparison result. Figure 12 shows the *Accuracy* and number of malware of other similar classification methods, including static ensemble learning with grayscale images and opcode sequence [10], [12], static and dynamic analysis with API calls or opcode sequence [29], [31], only static analysis with grayscale images or API calls [11], [20], and only dynamic analysis with FCG or registry [47], [48]. From Figure 12, the accuracy of methods 5 and 6 is slightly higher than that of Malscore. By making analysis the articles of methods 5 and 6 in depth, we found that the data sets of these two methods

TABLE 6. The time cost of the classifier D (total sample).

Phase	Training				Testing			
	Preprocessing	selecting n-grams	Calculating TF	Training	Preprocessing	selecting n-grams	Calculating TF	Testing
Time	$62h39min$	$27h45min$	$20h3min$	$19min19s$	$21h18min$	$9h26min$	$6h49min$	$3s217ms$

TABLE 7. The comparison of time cost between Ensemble learning and Malscore.

	Training		Testing		Accuracy
	Preprocessing	Training	Preprocessing	Testing	
Ensemble learning	$115h43min$	$3h30min$	$38h21min$	$3s261ms$	98.91%
Malscore	$115h43min$	$3h30min$	$15h30min$	$1s249ms$	98.82%
Comparison	-0%	-0%	-59.58%	-61.70%	-0.09%

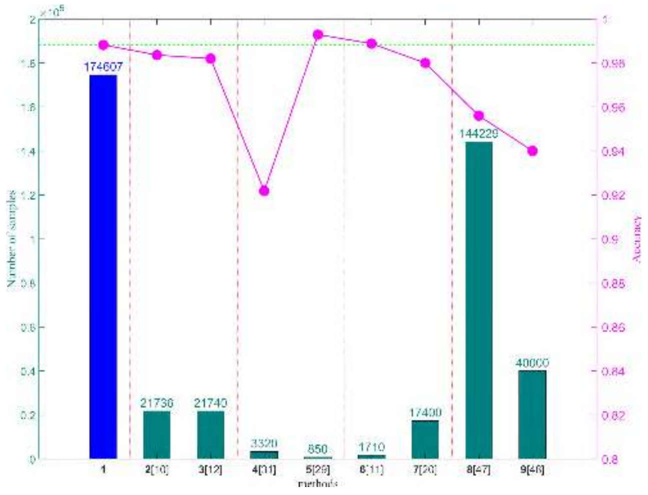

FIGURE 12. Comparison with other similar methods (From the perspective of dynamic analysis and static analysis). (1)Method 1 represents the Malscore; (2)Methods 2 and 3 represent static ensemble learning; (3)Methods 4 and 5 represent static and dynamic analysis; (4) Methods 6 and 7 only represent static analysis; (5) Methods 8 and 9only represent dynamic analysis.

are too small to cover the features contained in the malware family as much as possible, and the samples selected from each malware family have relatively high similarity. Moreover, method 5 uses a traditional graph distance similarity algorithm that can take more time cost, and method 6 is vulnerable to packing or obfuscation techniques. In order to make up the problems of methods 5 and 6, Malscore concatenated static analysis (deep learning) and dynamic analysis (machine learning) to classify malware. Although our accuracy was theoretically lower than methods 5 and 6, it can be seen from Table 10 that Malscore has a better accuracy-cost tradeoff in the same dataset. Although method 5 is 0.29% higher than Malscore, method 5 costed an average of 53.72*ms* to detect a malware sample while Malscore only costed 0.04*ms*. For our testing set (34,920 samples), method 5 costed about 31.27 minutes, while Malscore only costed 1.25*s*. For millions of malware, the detection time of method 5 took days to complete. Method 6 had the advantage of static analysis, and the preprocessing time and testing time were lower than Malscore, but the accuracy of method 6 on our testing set was only 90.45%, not 98.88% mentioned in the original article. This is because our testing set contains about 2,500 packed/encrypted malware, and static analysis cannot extract such malware features. The accuracy of method 6 also decreased with the increase of packed malware, which is the limitation of static analysis. The average preprocessing time and testing time of Malscore were only 0.95*s* and 0.02*ms* more than the static analysis method 6, but 2.25*s* and 0.05*ms* less than ensemble learning. As shown in Table 10, Malscore is better in the accuracy-cost tradeoff. Malscore not only had the ability to handle packed malware, but it also had lower time cost. In practice, Malscore focused on static analysis and supplements dynamic analysis. In Table 10, Malscore's average time cost is 1.6*s*. In fact, 1.6*s* is the average cost of

TABLE 8. The comparison of time cost between Ensemble learning and Malscore (Sample average).

	Training		Testing	
	Preprocessing	Training	Preprocessing	Testing
Ensemble learning	$2.98s$	$360ms$	$3.95s$	$0.09ms$
Malscore	$2.98s$	$360ms$	$1.60s$	$0.04ms$

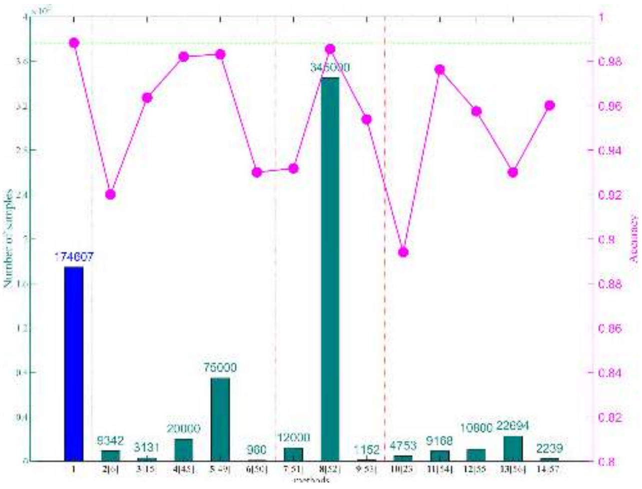

FIGURE 13. Comparison with other similar methods (From the perspective of machine learning and deep learning).

static analysis and dynamic analysis. Because of the high cost of dynamic analysis, the average value is higher. We deployed Malscore on our server. In our dataset, about 75% of the malware was an average of 185*ms* instead of 1.6*s*.

In Figure 12, we compared Malscore-like classification methods from the perspective of dynamic analysis and static analysis. Besides these eight malware classification methods in Figure 12, we further compared Malscore-like classification methods from the perspective of machine learning and deep learning. As shown in Figure 13, we classified the classification methods into three categories. The first is that grayscale images and API sequences are learned by learning algorithms different from CNN and RF [6], [15], [45], [49], [50]. The second is that malware features different from gray image and API sequence are learned by CNN and RF algorithms [51]–[53]. The last is that learning algorithms and malware features different from those of Malscore [23], [54]–[57]. From Figure 13, (1) Compared with other machine learning or deep learning algorithms, the CNN and RF algorithms in Malscore had better classification performance when learning grayscale images and API sequences. (2) Compared with other malware features, the grayscale images and API sequences proposed in Malscore can be better learned in CNN algorithm and RF algorithms. (3) The Malscore had better classification performance both in feature selection and classification algorithm selection.

In Figure 13, Method 1 represents the Malscore; Methods 2 to 4 learn grayscale images using RF, Artificial

TABLE 9. The Mean and Std of Malscore on the Phase 1, Phase 2, and concatenating Phase 1 and Phase 2.

Algorithm	Precision	Recall	F1-score	Accuracy	Training time	Testing time
Phase 1: CNN+ grayscale image	**96.98% (±0.32%)**	96.07% (±0.43%)	96.52% (±0.39%)	96.73% (±0.21%)	$3h10min$	$44ms$
Phase 2: RF+API	97.37% (±0.52%)	**97.53% (±0.31%)**	97.19% (±0.32%)	97.73% (±0.29%)	$19min18s$	$3s217ms$
Malscore: RF+CNN	98.59% (±0.09%)	99.03% (±0.23%)	98.81% (±0.11%)	**98.82% (±0.12%)**	$3h30min$	$1s249ms$

TABLE 10. The accuracy and time cost of comparative experiments (sample average).

Methods	Preprocessing	Testing	Accuracy
Malscore	$1.60s$	$0.04ms$	98.82%
RGB image+Similarity[29]	$1.86s\uparrow$	$53.72ms\uparrow$	99.11%$\uparrow$
Grayscale image+SVM[11]	$0.65s\downarrow$	$0.02ms\downarrow$	90.45%$\downarrow$
Ensemble Learning	$3.95s\uparrow$	$0.09ms\uparrow$	98.91%$\uparrow$

Neural Network and ensemble learning algorithms, respectively; Methods 5 and 6 learns API sequences using RNN and CNN algorithms, respectively; Method 7 learns Local binary pattern features of grayscale images using CNN algorithm; Methods 8 and 9 learn self-organizing feature maps and opcode using RF algorithm, respectively; Method 10 learns system call sequence using a neural network based on convolutional and recurrent network layers; Method 11 learns image from opcode using SVM algorithm; Method 12 learns malware feature maps generated by using deep autoencoder using transfer-deep convolutional generation adversarial networks; Method 13 learns metadata, input functions and opcode of PE files by combining CNN algorithm with feed-forward Neural Network algorithm; Method 14 learns machine activity data by using RNN algorithm. The captured metrics of active data are: system CPU usage, user CPU use, packet send, packets received, et al.

VII. LIMITATIONS

In this paper, Malscore used probabilistic scoring to concatenate static analysis and dynamic analysis. Because of the high cost of dynamic analysis, Malscore didn't use dynamic analysis as the main detection method, but used dynamic analysis as a complementary method of static analysis for the robustness of packed/obfuscated malware. Therefore, Malscore's detection cost was lower than that of dynamic analysis, and its detection accuracy was higher than that of static analysis and dynamic analysis.

Over time, the performance of malware detection methods will naturally degrade with the appearance of concept drift problems [16]. Malscore doesn't take into account concept drift, and we don't provide any incremental learning strategies to fortify Malscore's re-learning ability. At the same time, with the development of malware detection methods, more and more malware producers use a variety of technologies to resist dynamic and static analysis. For example, these producers obfuscate not only raw malware, but also system calls of malware in dynamic analysis. This kind of malware will make our static analysis and dynamic analysis lose ''eye''. Therefore, Malscore also lacks the ability to detect such malware.

Although Malscore has achieved some success in malware detection, there are still some areas for improvement. Our future work on Malscore will focus on the following:

1) Although concept drift is a well-known and troublesome thing in machine learning, we can use MaMaDroid [58]–[60] and incremental learning algorithms to reduce the impact of concept drift on Malscore. In this environment, additional experiments are needed to evaluate the impact of concept drift on Malscore.
2) Although dynamic analysis of obfuscation features is a problem, recently some methods have been proposed to solve this problem [27], [28]. We hope to enhance the robustness of dynamic analysis in Malscore, not only as a supplement to static analysis, but also as a major contribution to Malscore detection.

VIII. CONCLUSION

In this paper, the main goal was to present a system Malscore for malware classification, which was based on probability scoring and machine learning. In the Malscore, the convolutional neural networks with Spatial Pyramid Pooling was used to analyze grayscale image (static feature) in Phase 1, and the variable *n-grams* and machine learning were used to analyze native API call sequence (dynamic features) in Phase 2. To concatenate Phase 1 and Phase 2, a probability scoring was proposed to identify the credibility of the classification results in Phase 1, and Phase 2 further analyzed these malware with lower credibility. We performed experiments on 174,607 malware samples from 63 malware families. The result showed Malscore achieved 98.82% accuracy for malware classification, and the recall and precision of 100% in 52 families. Compared with the ensemble learning in testing, the preprocessing and test time of Malscore represented a reduction of 59.58% and 61.70%, respectively. This showed that the Malscore had a lower time cost. Malscore took only 0.04*ms* to detect a packed malware sample and 0.001*ms* to detect a unpacked malware sample in an operating system with 16G RAM and 3.6G CPU. Through this system, not only the performance of static analysis for static classification is utilized, but also the universality of dynamic analysis for packing and obfuscation techniques is fully utilized.

REFERENCES

[1] AO Kaspersky Lab. *Kaspersky Security Bulletin Overall Statistics for 2017*. Accessed: Jul. 22, 2018. [Online]. Available: https://securelist.com/ksb-overall-statistics-2017/83453/

[2] B. Christiaan, D. Taylor, G. Steve, K. Mary, M. Niamh, and P. Chris. (Jun. 2018). McAfee Labs Threats Reports. McAfee. Accessed: Aug. 2, 2018. [Online]. Available: https://www.mcafee.com/enterprise/en-us/assets/reports/rp-quarterly-threats-jun-2018.pdf

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

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

[5] K. He, X. Zhang, S. Ren, and J. Sun, "Spatial pyramid pooling in deep convolutional networks for visual recognition," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 37, no. 9, pp. 1904–1916, Sep. 2015.

[6] K. Kosmidis and C. Kalloniatis, "Machine learning and images for malware detection and classification," in *Proc. ACM 21st Pan-Hellenic Conf. Informat.*, 2017, pp. 1–7.

[7] K. Kancherla and S. Mukkamala, "Image visualization based malware detection," in *Proc. IEEE Comput. Intell. Cyber Secur.*, Apr. 2013, pp. 40–44.

[8] X. G. Han, Q. U. Wu, X. X. Yao, C. Y. Guo, and Z. Fang, "Research on malicious code variants detection based on texture fingerprint," *J. Commun.*, vol. 35, no. 8, pp. 125–136, 2014.

[9] H. Yan, H. Zhou, and H. Zhang, "Automatic malware classification via PRICoLBP," *Chin. J. Electron.*, vol. 27, no. 4, pp. 852–859, 2018.

[10] L. Liu, B.-S. Wang, B. Yu, and Q.-X. Zhong, "Automatic malware classification and new malware detection using machine learning," *Frontiers Inf. Technol. Electron. Eng.*, vol. 18, no. 9, pp. 1336–1347, Sep. 2017.

[11] A. Makandar and A. Patrot, "Malware class recognition using image processing techniques," in *Proc. Int. Conf. Data Manage., Anal. Innov.*, 2017, pp. 76–80.

[12] J. Yan, Y. Qi, and Q. Rao, "Detecting malware with an ensemble method based on deep neural network," *Security Commun. Netw.*, vol. 2018, Mar. 2018, Art. no. 7247095.

[13] M. Kalash, M. Rochan, N. Mohammed, N. D. B. Bruce, Y. Wang, and F. Iqbal, "Malware classification with deep convolutional neural networks," in *Proc. IEEE IFIP Int. Conf. New Technol., Mobility Secur.*, Feb. 2018, pp. 1–5.

[14] S. Tobiyama, Y. Yamaguchi, H. Shimada, T. Ikuse, and T. Yagi, "Malware detection with deep neural network using process behavior," in *Proc. IEEE 40th Annu. Comput. Softw. Appl. Conf.*, Jun. 2016, pp. 577–582.

[15] A. Makandar and A. Patrot, "Malware analysis and classification using artificial neural network," in *Proc. IEEE Int. Conf. Trends Autom., Commun. Comput. Technol.*, Dec. 2016, pp. 1–6.

[16] L. Onwuzurike, E. Mariconti, P. Andriotis, E. De Cristofaro, G. Ross, and G. Stringhini, "MaMaDroid: Detecting Android malware by building Markov chains of behavioral models (extended version)," *ACM Trans. Privacy Secur.*, vol. 22, no. 2, pp. 1–34, 2019.

[17] G. Suarez-Tangil, S. K. Dash, M. Ahmadi, J. Kinder, G. Giacinto, and L. Cavallaro, "DroidSieve: Fast and accurate classification of obfuscated Android malware," in *Proc. 7th ACM Conf. Data Appl. Secur. Privacy*, 2017, pp. 309–320.

[18] V. Avdiienko, K. Kuznetsov, A. Gorla, A. Zeller, S. Arzt, S. Rasthofer, and E. Bodden, "Mining apps for abnormal usage of sensitive data," in *Proc. 37th IEEE Int. Conf. Softw. Eng.*, May 2015, pp. 426–436.

[19] Y. Ki, E. Kim, and H. K. Kim, "A novel approach to detect malware based on api call sequence analysis," *Int. J. Distrib. Sensor Netw.*, vol. 11, no. 6, 2015, Art. no. 659101.

[20] A. Pekta and T. Acarman, "Malware classification based on API calls and behaviour analysis," *IET Inf. Secur.*, vol. 12, no. 2, pp. 107–117, 2018.

[21] T. Lee, B. Choi, Y. Shin, and K. Jin, "Automatic malware mutant detection and group classification based on the n-Gram and clustering coefficient," *J. Supercomput.*, vol. 74, no. 8, pp. 3489–3503, 2015.

[22] P. Natani and D. Vidyarthi, "Malware detection using API function frequency with ensemble based classifier," in *Proc. Int. Symp. Secur. Comput. Commun.*, vol. 377, Nov. 2013, pp. 378–388.

[23] B. Kolosnjaji, A. Zarras, G. Webster, and C. Eckert, "Deep learning for classification of malware system call sequences," in *Proc. Australas. Joint Conf. Artif. Intell.* Cham, Switzerland: Springer, 2016, pp. 137–149.

[24] L. Onwuzurike, M. Almeida, E. Mariconti, J. Blackburn, G. Stringhini, and E. D. Cristofaro, "A family of droids-Android malware detection via behavioral modeling: Static vs dynamic analysis," in *Proc. IEEE 16th Annu. Conf. Privacy, Secur. Trust (PST)*, Aug. 2018, pp. 1–10.

[25] V. M. Afonso, M. F. de Amorim, A. R. A. Grégio, G. B. Junquera, and P. L. de Geus, "Identifying Android malware using dynamically obtained features," *J. Comput. Virol. Hacking Techn.*, vol. 11, no. 1, pp. 9–17, 2015.

[26] S. K. Dash, G. Suarez-Tangil, S. Khan, K. Tam, M. Ahmadi, and J. Kinder, "DroidScribe: Classifying Android malware based on runtime behavior," in *Proc. IEEE Secur. Privacy Workshops (SPW)*, 2016, pp. 252–261.

[27] H. Cai, N. Meng, B. Ryder, and D. Yao, "DroidCat: Unified dynamic detection of Android malware," Ph.D. dissertation, Dept. Comput. Sci., Virginia Polytechnic Inst. State Univ., Blacksburg, VA, USA, 2016.

[28] H. Cai, N. Meng, B. G. Ryder, and D. Yao, "DroidCat: Effective Android malware detection and categorization via app-level profiling," *IEEE Trans. Inf. Forensics Security*, vol. 14, no. 6, pp. 1455–1470, Jun. 2019.

[29] K. Han, B. Kang, and E. G. Im, "Malware analysis using visualized image matrices," *Sci. World J.*, vol. 2014, Jul. 2014, Art. no. 132713.

[30] P. V. Shijo and A. Salim, "Integrated static and dynamic analysis for malware detection," *Procedia Comput. Sci.*, vol. 46, pp. 804–811, Feb. 2015.

[31] S. Kilgallon, L. De La Rosa, and J. Cavazos, "Improving the effectiveness and efficiency of dynamic malware analysis with machine learning," in *Proc. IEEE Resilience Week*, Sep. 2017, pp. 30–36.

[32] S. Chen, M. Xue, Z. Tang, L. Xu, and H. Zhu, "StormDroid: A streaminglized machine learning-based system for detecting Android malware," in *Proc. 11th ACM Asia Conf. Comput. Commun. Secur.*, 2016, pp. 377–388.

[33] R. Lyda and J. Hamrock, "Using entropy analysis to find encrypted and packed malware," *IEEE Security Privacy*, vol. 5, no. 2, pp. 40–45, Mar./Apr. 2007.

[34] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *Proc. Int. Conf. Learn. Represent.*, 2014. [Online]. Available: https://arxiv.org/abs/1409.1556

[35] A. L. Maas, A. Y. Hannun, and A. Y. Ng, "Rectifier nonlinearities improve neural network acoustic models," in *Proc. 30th Int. Conf. Mach. Learn.*, vol. 28, 2013, p. 3.

[36] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proc. Int. Conf. Mach. Learn.*, vol. 37, Jul. 2015, pp. 448–456.

[37] *Cuckoo Sandbox*. Accessed: Oct. 19, 2018. [Online]. Available: http://www.
cuckoosandbox.org/

[38] P. F. Brown, P. V. deSouza, R. L. Mercer, V. J. D. Pietra, and J. C. Lai, "Class-based n-Gram models of natural language," *J. Comput. Linguistics*, vol. 18, no. 4, pp. 467–479, 1992.

[39] R. G. Brereton and G. R. Lloyd, "Support Vector Machines for classification and regression," *Analyst*, vol. 135, no. 2, pp. 230–267, 2010.

[40] M. Hassen, M. M. Carvalho, and P. K. Chan, "Malware classification using static analysis based features," in *Proc. 32th Int. Conf. Mach. Learn.*, vol. 37, 2015, pp. 448–456.

[41] J. Cao, L. Drabeck, and R. He, "Statistical network behavior based threat detection," in *Proc. Comput. Commun. Workshops*, 2017, pp. 420–425.

[42] C. Ravi and R. Manoharan, "Malware detection using windows api sequence and machine learning," *Int. J. Comput. Appl.*, vol. 43, no. 17, pp. 12–16, 2012.

[43] M. Ni, T. Li, Q. Li, H. Zhang, and Y. Ye, "FindMal: A file-to-file social network based malware detection framework," *Knowl.-Based Syst.*, vol. 112, pp. 142–151, Nov. 2016.

[44] *Vxheavens*. Accessed: Aug. 21, 2018. [Online]. Available: https://83.133.184.251/virensimulation.org/index.html

[45] B. L. Zhao, X. Meng, J. Han, J. Wang, and F. D. Liu, "Homology analysis of malware based on graph," *J. Commun.*, vol. 38, no. S2, pp. 86–93, Nov. 2017.

[46] L. Breiman, "Bagging predictors," *Mach. Learn.*, vol. 24, no. 2, pp. 123–140, 1996.

[47] J. Stiborek, T. Pevný, and M. Rehák, "Multiple instance learning for malware classification," *Expert Syst. Appl.*, vol. 93, pp. 346–357, Mar. 2018.

[48] H. H. Jazi and A. A. Ghorbani, "Dynamic graph-based malware classifier," in *Proc. Privacy, Secur. Trust*, 2017, pp. 112–120.

[49] L. Liu and B. Wang, "Malware classification using gray-scale images and ensemble learning," in *Proc. Int. Conf. Syst. Inform.*, 2017, pp. 1018–1022.

[50] R. Pascanu, J. W. Stokes, H. Sanossian, M. Marinescu, and A. Thomas, "Malware classification with recurrent networks," in *Proc. IEEE Int. Conf. Acoust.*, Apr. 2015, pp. 1916–1920.

[51] J.-S. Luo and D. C.-T. Lo, "Binary malware image classification using machine learning with local binary pattern," in *Proc. IEEE Int. Conf. Big Data*, Dec. 2017, pp. 4664–4667.
[52] P. Burnap, R. French, F. Turner, and K. Jones, "Malware classification using self organising feature maps and machine activity data," *Comput. Secur.*, vol. 73, pp. 399–410, Mar. 2018.
[53] A. Shabtai, R. Moskovitch, C. Feher, S. Dolev, and Y. Elovici, "Detecting unknown malicious code by applying classification techniques on OpCode patterns," *Secur. Informat.*, vol. 1, p. 1, Dec. 2012.
[54] T. Wang and X. Ning, "Malware variants detection based on opcode image recognition in small training set," in *Proc. IEEE Int. Conf. Cloud Comput. Big Data Anal.*, Apr. 2017, pp. 328–332.
[55] J.-Y. Kim, S.-J. Bu, and S.-B. Cho, "Zero-day malware detection using transferred generative adversarial networks based on deep autoencoders," *Inf. Sci.*, vols. 460–461, pp. 83–102, Sep. 2018.
[56] B. Kolosnjaji, G. Eraisha, G. Webster, A. Zarras, and C. Eckert, "Empowering convolutional networks for malware classification and analysis," in *Proc. Int. Joint Conf. Neural Netw.*, 2017, pp. 3838–3845.
[57] M. Rhode, P. Burnap, and K. Jones, "Early-stage malware prediction using recurrent neural networks," *Comput. Secur.*, vol. 77, pp. 578–594, Aug. 2018.
[58] X. Fu and H. Cai, "On the deterioration of learning-based malware detectors for Android," in *Proc. IEEE 41st Int. Conf. Softw. Eng., Companion*, 2019, pp. 350–351.
[59] H. Cai and J. John, "Towards sustainable Android malware detection," in *Proc. ACM 40th Int. Conf. Softw. Eng., Companion*, 2018, pp. 350–351.
[60] H. Cai, "A preliminary study on the sustainability of Android malware detection," ACM, New York, NY, USA, Tech. Rep., Jul. 2018. [Online]. Available: https://arxiv.org/abs/1807.08221

DI XUE is currently pursuing the Ph.D. degree with the College of Computer Science and Technology, Harbin Engineering University. His research interests include big data, and networks and information security.

JINGMEI LI received the M.S. and Ph.D. degrees from Harbin Engineering University, Harbin, China. She is currently a Professor with the College of Computer Science and Technology, Harbin Engineering University. Her research interests include computer architecture performance optimization, big data and cloud computing, networks and information security, and embedded technology.

TU LV is currently pursuing the master's degree with the College of Computer Science and Technology, Harbin Engineering University.

WEIFEI WU received the M.S. degree from the College of Computer Science and Technology, Harbin Engineering University, where he is currently pursuing the Ph.D. degree. His research interests include cloud computing, and networks and information security.

JIAXIANG WANG is currently a Professor with the College of Computer Science and Technology, Harbin Engineering University. His research interests include data security and access control, and networks and information security.