DETECTION OF MALICIOUS CODE VARIANTS BASED ON DEEP LEARNING

Seminar Report

Submitted in partial fulfillment of the requirements for the award of degree of

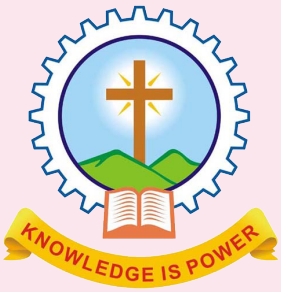
BACHELOR OF TECHNOLOGY
In
COMPUTER SCIENCE AND ENGINEERING

of

APJ ABDUL KALAM TECHNOLOGICAL UNIVERSITY

Submitted By

HARI SANKAR



Department of Computer Science & Engineering
Mar Athanasius College Of Engineering Kothamangalam

DETECTION OF MALICIOUS CODE VARIANTS BASED ON DEEP LEARNING

Seminar Report

Submitted in partial fulfillment of the requirements for the award of degree of

BACHELOR OF TECHNOLOGY

In

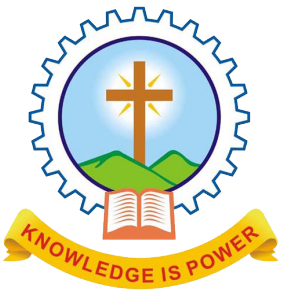
COMPUTER SCIENCE AND ENGINEERING

of

APJ ABDUL KALAM TECHNOLOGICAL UNIVERSITY

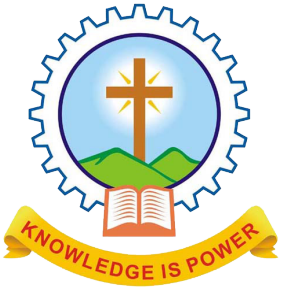
Submitted By

HARI SANKAR



Department of Computer Science & Engineering
Mar Athanasius College Of Engineering Kothamangalam

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
MAR ATHANASIUS COLLEGE OF ENGINEERING
KOTHAMANGALAM



CERTIFICATE

This is to certify that the report entitled **Detection of Malicious Code Variants Based on Deep Learning** *submitted by* ***Mr. HARI SANKAR****, Reg. No.* ***MAC15CS029*** *towards partial fulfilment of the requirement for the award of Degree of Bachelor of Technology in Computer science and Engineering from APJ Abdul Kalam Technological University for the year December 2018 is a bonafide record of the work carried out by him under our supervision and guidance.*

.....	.....	.....
Prof. Joby George	**Prof. Neethu Subash**	**Dr.Surekha Mariam Varghese**
Faculty Guide	***Faculty Guide***	***Head of the Department***

Date:

Dept. Seal

ACKNOWLEDGEMENT

First and foremost, I sincerely thank the God Almighty for his grace for the successful and timely completion of the seminar.

I express my sincere gratitude and thanks to Dr. Solly George, Principal and Dr. Surekha Mariam Varghese, Head Of the Department for providing the necessary facilities and their encouragement and support.

I owe special thanks to the staff-in-charge Prof. Joby George and Prof. Neetha Subash for their corrections, suggestions and sincere efforts to co-ordinate the seminar under a tight schedule.

I express my sincere thanks to staff members in the Department of Computer Science and Engineering who have taken sincere efforts in helping me to conduct this seminar.

Finally, I would like to acknowledge the heartfelt efforts, comments, criticisms, co-operation and tremendous support given to me by my dear friends during the preparation of the seminar and also during the presentation without whose support this work would have been all the more difficult to accomplish.

ABSTRACT

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

Contents

List of Figures

List of Abbreviation

CNN	Convolutional Neural Network
EMO	Evolutionary Multi-Objective Optimization
DRBA	Dynamic Resampling method based on the Bat algorithm
DBN	Deep Belief Network
IOT	Internet of Things
TP	True Positive
FP	False Positive
TN	True Negative
FN	False Negative

Introduction

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

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

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

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

Recently, rather than focusing on non-visible features for malware classification,proposed malware visualization, a new approach based on image processing techniques[1]. This work transformed the structure of packed binary samples into two-dimensional grayscale images. Thenthe image features were used for classification. Moreover, two different feature design strategies are studied comparatively for malware analysis. The results illustrated that the binary texture-based strategy provided performance roughly equivalent to the dynamic analysis techniques, but in less time.

Malware detection methods rely mainly on analysis of the features of malicious codes

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

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

Related works

The related research regarding malware, including malware detection based on feature analysis, malicious code visualization, image processing techniques for malware detection, and malware detection based on deep learning.

2.1 Malware detection based on feature analysis

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

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

The method includes, for each of a multiplicity of sample files, subdividing file code of the sample file into a plurality of code blocks and then removing duplicate code blocks to leave a sequence of unique code blocks as shown in fig 2.1. The sequence of unique code blocks is then compared with those obtained for other sample files in order to identify standard sections of code. The standard sections of code identified are then included within a database such that those sections of code can subsequently be disregarded when identifying features characteristic of malware.

According to a first aspect of the invention, there is provided a method of identifying sections of code that can be disregarded when detecting features' that are characteristic of malware, which features are subsequently used for detecting malware. The method first comprises, for each of a multiplicity of sample files, subdividing file code of the file into a plurality of code blocks. Duplicate code blocks are then removed to leave a sequence of unique code blocks. The sequence of unique code blocks is then compared with those obtained for other sample files in order to identify standard sections of code. The standard sections of code identified in the comparison are then included in a database such that those sections of code can subsequently be

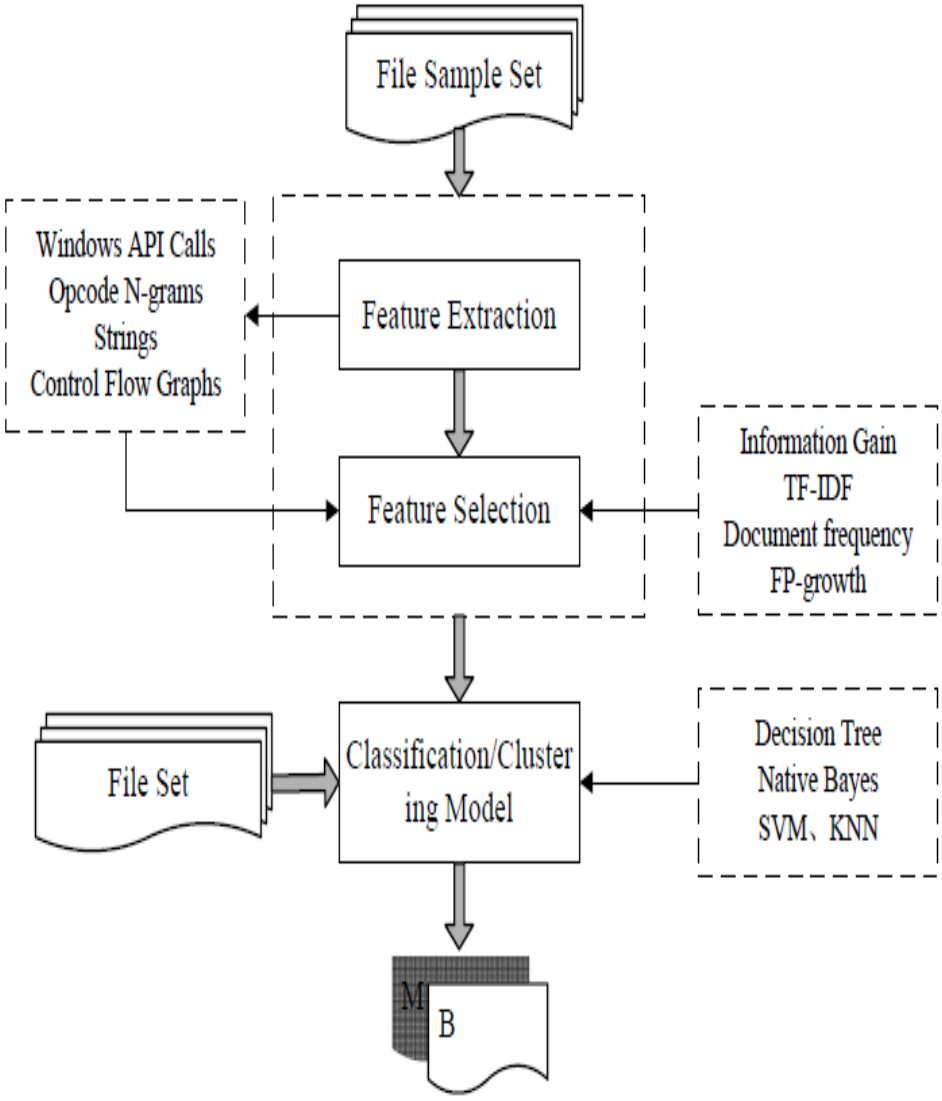


Fig. 2.1: Malicious code detection based on data mining

disregarded when identifying features characteristic of malware.

Embodiments of the present invention may provide a method that allows the elimination of features that are likely to result in false positives during malware scanning using a feature analysis approach. Embodiments of the present invention may also provide a faster method of scanning a computer for malware, and which may require less processing power than conventional scanning methods.

The step of subdividing file code into a plurality of code blocks may comprise subdividing the file code into blocks of one or more instructions, generating a signature for the instruction block, and using that signature as the code block. This signature may be a cyclic redundancy check value, and instruction parameters may be removed from an instruction block before generating a signature.

The step of comparing the sequence of unique code blocks with those obtained for other sample files in order to identify standard sections of code may comprise creating a sequence table for the file, a first row of the table including the first code block of the sequence of unique code blocks and each subsequent line adding on the next code block in the sequence, the code blocks within each row being reordered according to a predefined sorting procedure; and searching sequence tables of other files for matching rows

For each row in the sequence table, a count number may be included that indicates the number of times an identical row is found in the sequence tables for the other sample files.The step of identifying standard sections of code may comprise detecting distinct boundaries in the count numbers within a sequence table, and may further comprise ignoring false positive drops in the count numbers.

According to a second aspect of the invention, there is provided a method of identifying features that are characteristic of a malware sample, which features are subsequently used for detecting malware. The method first comprises identifying sections of malware sample code that match with entries in a database of known and trusted sections of code. The malware sample code is then executed in a sandbox and execution features are detected. Those features resulting from execution of the identified sections are rejected, and the remaining features, or a subset of those remaining features, are utilised to detect malware.

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

ciency when exposed to a large dataset.

2.2 Malicious code visualization

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

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

2.3 Image processing techniques for malware detection

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

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

These traditional approaches, however, were challenged by the high time cost required for complex image texture feature extraction. To address this challenge, we employed deep

learning to identify and classify images efficiently. In the next subsection, we present our research on malware detection based on deep learning.

2.4 Malware detection based on deep learning

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

Because of it powerful ability to learn features, many scholars have applied deep learning to malware detection. Using deep learning techniques, Yuan et al.[2] designed and implemented an online malware detection prototype system, named Droid-Sec. Their model achieved high accuracy by learning the features extracted from both static analysis and dynamic analysis of Android apps. David et al. presented a similar but more compelling method that did not need the type of malware behavior. Their work was based on a deep belief network (DBN) for automatic malware signature generation and classification. Compared with conventional signature methods for malware detection, their approach demonstrated increased accuracy for detecting new malware variants.

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

The proposed system

3.1 Formalized description

To address the above challenges, this paper offered the following contributions:

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

3.2 Major steps

It defines improved malicious code variant detection method based on a CNN, which included: (1) the malicious code mapping as the grayscale image; and (2) the CNN design for grayscale image detection. Fig. 2 presents an overview of these two processes. First, the binary files of malicious code are transformed into the grayscale images. Next, the convolution neural network is employed to identify and classify the images. According to the results of image classification, we realized the automatic recognition and classification of malicious software as shown in fig 3.1.

3.2.1 Binary malware to gray image

In general, there are several ways to transform binary code into images. In this paper, we used the visualization of executable malware binary files [1]. A malware binary bit string can be split into a number of substrings that are 8 bits in length. Each of these substrings can be seen as a pixel, because the 8 bits can be interpreted as an unsigned integer in the range 0 to 255. For example, if a bit string is 0110000010101100, the process is 0110000010101100 $\rightarrow 01100000, 10101100 \rightarrow 96, 172$.

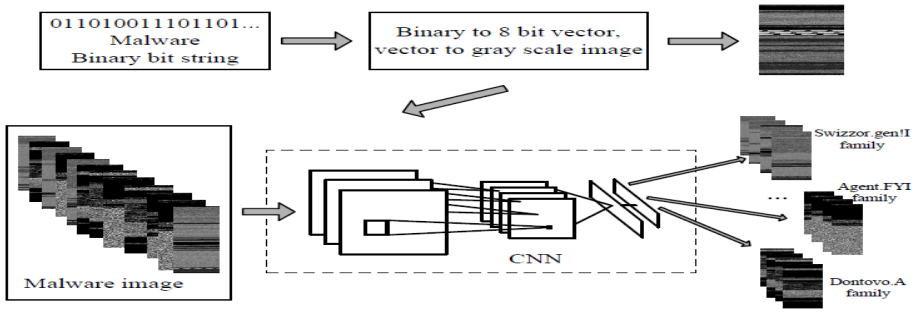


Fig. 3.1: overview of the method

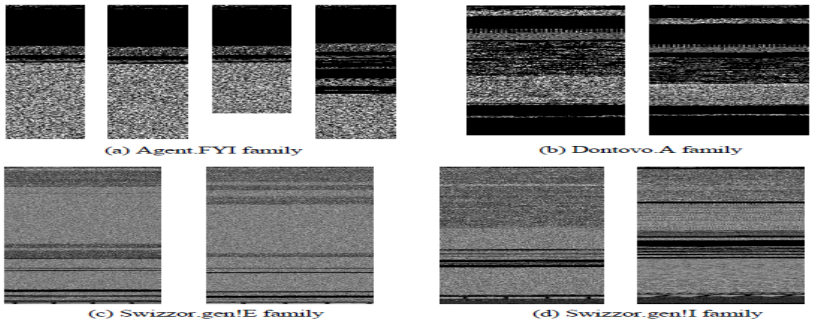


Fig. 3.2: Illustration of malware images

An eight-bit binary number $B = (b_7, b_6, b_5, b_4, b_3, b_2, b_1, b_0)$ can be converted into a decimal number I as follows:

$$I = b_0 * 2^0 + b_1 * 2^1 + b_2 * 2^3 + b_4 * 2^4 + b_5 * 2^6 + b_6 * 2^6 + b_7 * 2^7 \qquad \text{(Equ:3.1)}$$

After binary conversion, the binary malware bit string has been converted into a 1-D vector of decimal numbers. According to a specified width, this one-dimensional array can be treated as a 2-D matrix of a certain width. Finally, the malicious code matrix is interpreted as a grayscale image as shown in fig 3.2. For simplicity, the width of the image is fixed, and the height of the image varies depending on the size of the file. Table I, taken from [1], presents some recommended image widths for various file sizes, based on empirical observations.

It shows examples of malware images from different families . As we can see, the images of the same malware family are visually similar, and they differ noticeably from images belonging to another family. It shows four variants of the malware Agent, called the Agent.FYI family. The size of each member is different, but they still have similarities because new malware is often created from old software. In addition, when families are similar, images can display their differences clearly. For example, compared with the Swizzor.gen!I family

in Fig., the Swizzor.gen!E family in Fig. has black stripes. Inspired by the visual similarity of malware images, we can classify and detect malicious software by using image recognition methods. In our current work, we used a convolutional neural network (CNN) to recognize malware images

3.2.2 Malware image classification based on convolutional neural network

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

In this paper, we developed a CNN to classify malware. The structure of the CNN for grayscale image recognition consists of several components, as shown in Fig.. First is the input layer, which brings the training images into the neural network. Next are the convolution and sub-sampling layers. The former layer can enhance signal characteristics and reduce noise. The latter can reduce the amount of data processing while retaining useful information. Then there are several fully connected layers that convert a two-dimensional feature into a one-dimensional feature that conforms to the classifier criteria. Finally, the classifier identifies and sorts the malware images into different families according to their characteristics as shown in fig 3.3. The detailed iterative formula of each layer can be given as follows:

Convolution layers

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

$$x_j^l = f(\sum_{i \in M_j} x_j^{l-1} * k_{ij}^l + b_j^l) \qquad \text{(Equ:3.2)}$$

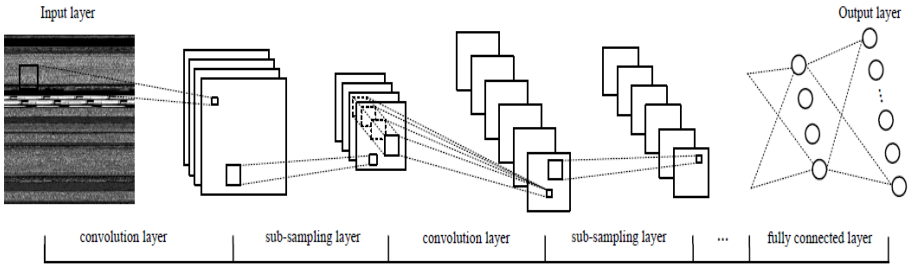


Fig. 3.3: Illustration of malware classification based on a CNN

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

$$\delta_{j}^{i} = \delta_{j}^{i+1} \circ f'(u^{l}) = \beta_{j}^{l+1} up(\delta_{j}^{l+1}) \circ f'(u^{l}) \quad \text{(Equ:3.3)}$$

where l+1 layer is the sampling layer, the weight W represents a convolution kernel, and its value $beta_{j}^{l+1}$ is an upsampling operation

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

$$\frac{\partial E}{\partial b_{j}} = \sum_{u,v} (\delta_{j}^{l})_{u,v} \quad \text{(Equ:3.4)}$$

$$\frac{\partial E}{\partial k_{ij}^{l}} = \sum_{u,v} (\delta_{j}^{i})_{u,v} (p_{i}^{l-1})_{u,v} \quad \text{(Equ:3.5)}$$

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

Sub-sampling layers

The sub-sampling layer is also called the pooling layer.

Generally, its convolution kernels are the maximum or mean of patch (maximum pooling, average pooling), and are not modified by backward propagation. This layer can weaken the influence of image deformation (e.g., translation, scale,rotation). It reduces the dimension of

the feature map, improves the models accuracy, and avoids overfitting. In CNN, for each output of the sampling layer, the feature map is given as follows:

$$x_j^l = f(down(x_j^{l-1} + b_j^l) \quad \text{(Equ:3.6)}$$

where down(.) represents a sub-sampling function, and b is bias. The sensitivity is calculated as shown:

$$\delta_j^l = \delta_j^{l+1} W_j^{l+1} \circ f'(u^l) \quad \text{(Equ:3.7)}$$

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

To solve this problem, in this paper, we reshaped the malware image to a fixed size square image (e.g., 48*48, 96*96) as shown in fig 3.4. In this way, the malware images were normalized and could be entered directly into the CNN network for classification. The advantage of normalization is that it reduces the dimensionality of the image effectively, and is more conducive to the training of the model. In the dimensionality reduction process, inevitably some characteristic information is lost.

For most images in our malware dataset, texture features can be preserved effectively whether there is an enlargement or reduction. An example can be found in the variants of Dontovo.A family shown in Fig. The original image size was 64*257. After compression (24*24 and 48*48) or expansion (96*96 and 192*192), the texture feature remained sharp (the upper part is black, and the bottom is gray). However, for those big images with small textures feature, the reshape operation may cause the loss of critical information. For example, the textures feature (black dots at the bottom)of a variant from the Swizzor.gen!E family was damaged when it was compressed too much (from 512*718 to 24*24 or 48*48).

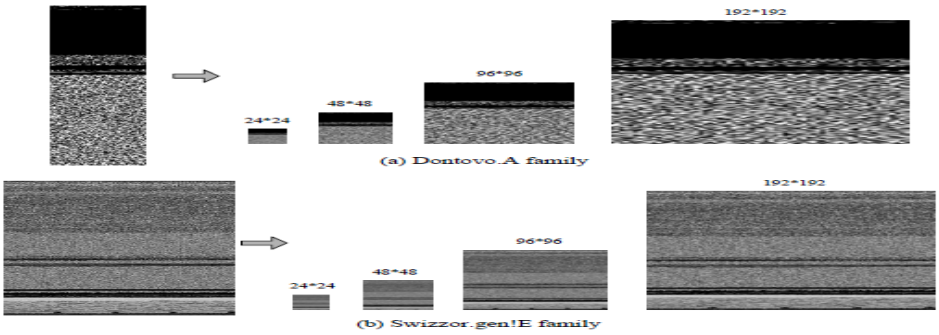


Fig. 3.4: Reshape the malware image to a fixed size square image

3.3 Malware image data equilibrium

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

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

3.3.1 Image data augmentation technology

Intra-domain spoofing means both attacker and destination host are in the same domain, so the checkpoint in the domain border cannot effect and filter forged packets. Taking the scenario in Fig 3.5 as example, the spoofing host 10.192.123.2 pretends to be the host 10.192.123.10 and conduct an attack to victim host 10.192.123.11. If the router has not deployed any anti-spoofing measures, the attack could be harmed to destination host and pretending host both. Thus, this threat reminds us that anti-spoofing measures have to impose to intra-domain area, instead of domain border only.

In deep learning, to avoid overfitting, we usually need to enter sufficient data to train the model. If the data sample is relatively small, we can use data augmentation to increase the sample, thereby restraining the influence of imbalanced data. The appropriate data augmentation method can avoid overfitting problems effectively, and improve the robustness of the model.

Generally, transformation of the original image data (changing the location of image

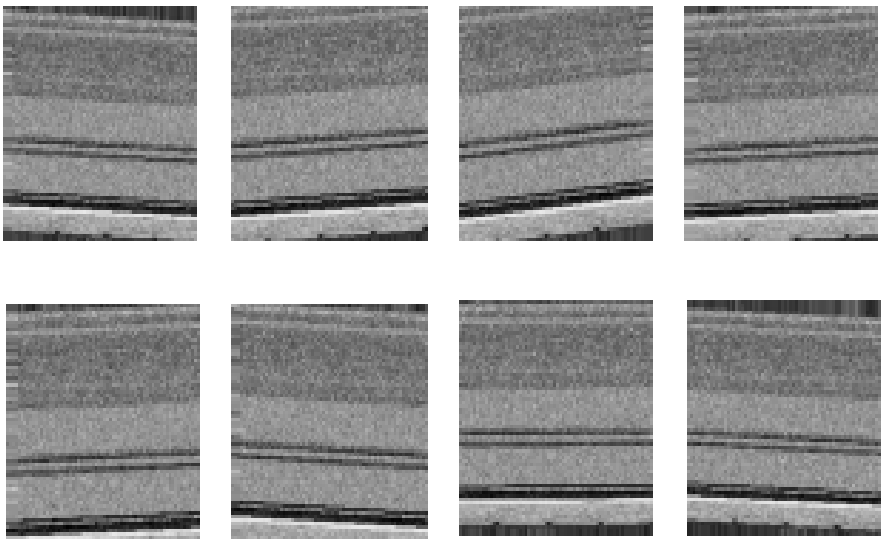


Fig. 3.5: Images generated by data augmentation for Swizzor.gen!E family

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

Let A = [a1, a2, ..., a8] be a collection of these techniques. Mi=am, ..., an is the sequence of operations, and i is the length of M. M_2 = a_1, a_2 represents data augmentation by using rotation transformation and flip transformation. Typically, each augmentation technique has a weight $_i$,so the sequence of operations of weighted data augmentation techniques can be given by

$$M_i = \lambda_m a_m, ..., \lambda_n a_n \qquad \text{(Equ:3.8)}$$

where $_m$ is the weight of the augmentation technology, and a_m subset of A. Usually it is necessary to perform multiple data augmentations.s Some images generated by data augmentation for the malware Swizzor.gen!E family. Clearly, most of the newly generated images retained the original texture features. However, a few images lost some information through inappropriate transition (i.e., overrotation). The images viewed in the lower-left corner lack a texture feature (black dots at the bottom) of the Swizzor.gen!E family.

3.3.2 Data equalization based on a bat algorithm

Imbalanced data is a common problem in machine learning, and it can affect the accuracy of the model in classification problems. In this paper, the malware image data was very uneven, as shown in Fig. 7. The maximum ratio was about 36:1. For example, the Allaple.A family had 2,949 images, while the Skintrim.N family had 80. Such differences have a very big impact on a CNNs performance in image classification. The classifying model trained according to the data of the two families may achieve 97the Skintrim.N family are not identified. However, it is not a reasonable classifier because it can only identify some data.

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

For instance, take the Allaple.A family (2,949 images) and the Skintrim.N family (80 images). If the training set for the former consists of 1,000 samples, we can choose 500 of them. For the latter, because of the insufficient number of data, we must repeat sampling until the training sample reaches 500. Because the training sample is less than the total sample, the trained classification model for the Skintrim.N family will suffer from overfitting. Therefore, the proper proportion of samples is important for the training set. Unfortunately, it is difficult to find an optimal proportion due to the complex weight combinations with dozens of families

A swarm intelligent algorithm is an effective solution for complex optimization problems. Many researchers have made several in-depth studies about swarm intelligence and its applications. Wang et al. used the evolutionary multi-objective optimization (EMO) algorithms to deal with the floorplan problem in cyber-physical social system. Cui et al. applied a modified cuckoo search algorithm to improve the performance of DV-Hop.

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

Bat Algorithm

Objective function $f(\mathbf{x})$, $\mathbf{x} = (x_1, ..., x_d)^T$
Initialize the bat population $\mathbf{x}_i$ $(i = 1, 2, ..., n)$ *and* $\mathbf{v}_i$
Define pulse frequency f_i *at* $\mathbf{x}_i$
Initialize pulse rates r_i *and the loudness* A_i
while *(t <Max number of iterations)*
Generate new solutions by adjusting frequency,
and updating velocities and locations/solutions [equations (2) to (4)]
  if $(rand > r_i)$
  Select a solution among the best solutions
  Generate a local solution around the selected best solution
  end if
  Generate a new solution by flying randomly
  if $(rand < A_i \ \& \ f(\mathbf{x}_i) < f(\mathbf{x}_*))$
  Accept the new solutions
  Increase r_i *and reduce* A_i
  end if
Rank the bats and find the current best $\mathbf{x}_*$
end while
Postprocess results and visualization

Fig. 3.6: Pseudo code of the bat algorithm

is a combination of sampling weights that can be given by the following equation:

$$position = w_1, ..., w_n \quad \text{(Equ:3.9)}$$

Bat algorithm (BA) is a novel population-based evolutionary algorithm inspired by echolocation behavior. Due to its simple concept, BA has been widely applied to various engineering applications. As an optimization approach, the global search characteristics determine the optimization performance and convergence speed. In BA, the global search capability is dominated by the velocity updating. How to update the velocity of bats may seriously affect the performance of BA.

Due to its simple concept, BA has been widely applied to many problems. Networks in real world more or less have overlapping community structure , however, Ma et al. found that traditional community detection algorithms assumed that one vertex can only belong to one community. To solve this problem, Hassan et al. designed a discrete bat algorithm to solve it, and experiments on real life networks show the ability of the proposed algorithm. Satellite images are a reliable source for investigating the temporal changes in crop cultivated areas, Senthilnath et al. proposed a novel bat algorithm (BA)-based clustering approach for solving crop type classification problems using a multispectral satellite image. There are still many applications, such as visual tracking , distribution feeder reconfiguration, and redundancy allocation problem , due to the page limitation, please refer to the corresponding references.

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

3.4 Experimental evaluation

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

Algorithm 1: DRBA($dataset$) operation

Input: $dataset$: training set for CNN; $bats$: bat population;

Output: $trainset$: training set for each epoch;

$(p, v) \leftarrow$ **getState**$(bats)$;
while $t < maxIterations$ **do**
  for $bat_i \subset bats$ **do**
    $(p_i^{t+1}, v_i^{t+1}) \leftarrow$ **updateState**(p_i^t, v_i^t);
    $templeset \leftarrow$ **reSample**$(dataset, p_i^{t+1})$;
    $accuracy_i^{t+1} \leftarrow$ **trainModel**$(templeset)$;
  end
  $bats \leftarrow$ **localSearch**$(bats)$;
  $p^* \leftarrow$ **getBest**$(bats)$;
  $t = t + 1$;
end
$trainset \leftarrow$ **reSample**$(dataset)$;
Return$(trainset)$;

Fig. 3.7: Bat Algorithm

Dataset and experimental setting

The dataset consisted of 9,342 grayscale images of 25 malware families. The number of samples in each family was different. As shown, there was a great deal of imbalance in the number of raw data samples. Therefore, it was necessary to use a data balancing method to balance the number of training samples.

First, we used data augmentation techniques to improve the quality of small data samples. The expansion of small data samples can solve problems (e.g., overfitting) caused by the unbalanced training data, effectively.

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

For our model, we designed different architectures of the CNN for malware images with different sizes. For the 24*24 inputs, our model had seven layers, including five hidden layers. The detailed structure was as follows: C1:8*20*20, S2:8*10*10, C3:16*8*8, S4:16*4*4, and C5:80*1*1. Each type of convolution kernel corresponded to a type of feature map. The maps per layer referred to the number of feature maps per layer. For the input of other sizes, as the size increased, the number of layers increased (e.g., double size meant adding two layers). Furthermore, we used the imresize function with interpolation method bicubic (default) in MATLAB R2017a to resize/normalize the images. The initial learning rate was set to 0.01, and the decay policy was inv. The maximum number of iterations was 10000. Finally, the training was carried out using the GPU. Moreover, in the image data balancing method DRBA, for BA, the population size was 20, other parameters were the same.

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

$$Precision = \frac{TP}{TP + FP} \quad \text{(Equ:3.10)}$$

$$Recall = \frac{TP}{TP + FN} \quad \text{(Equ:3.11)}$$

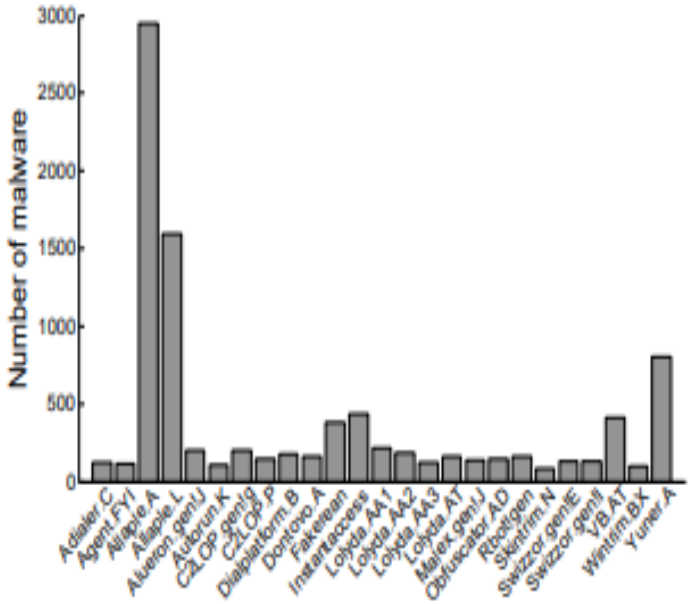


Fig. 3.8: Number of malware images in different families

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \quad \text{(Equ:3.12)}$$

where True Positive (TP) and False Positive (FP) are the number of file samples correctly and wrongly classified as malicious, respectively. Similarly, True Negative (TN) and False Negative (FN) are the number of file samples correctly and wrongly classified as benign, respectively.

Efficiency of malware image data equalization

To verify the validity of our proposed data equalization method, including the image data augmentation technique (IDA) and DRBA, we compared the results when no equalization method was used (Nothing) with models using IDA, DRBA, and IDA+DRBA. The performance of the different classification models on the training set is shown . As seen all methods achieved a good result with respect to accuracy, even if the data set was not processed. The Nothing method may have demonstrated overfitting. It shows the Loss curve, for which the results of the four methods were similar.

Because of the imbalance of the samples, it was easy to cause over-fitting, so the Nothing method appeared to show a high level of accuracy. The accuracy of the DRBA method was the

best. The accuracy rate increased continuously with the increase of the epoch. The accuracy of the IDA method was not as good as its performance on the training set, perhaps because some features of the image were lost in the image transformation process. There were some differences between training data and test data. As seen from Fig. 9(b), the loss curve of DRBA and IDA+DRBA was decreasing, while the curve of Nothing was oscillating within a certain range, which meant it was overfitting.

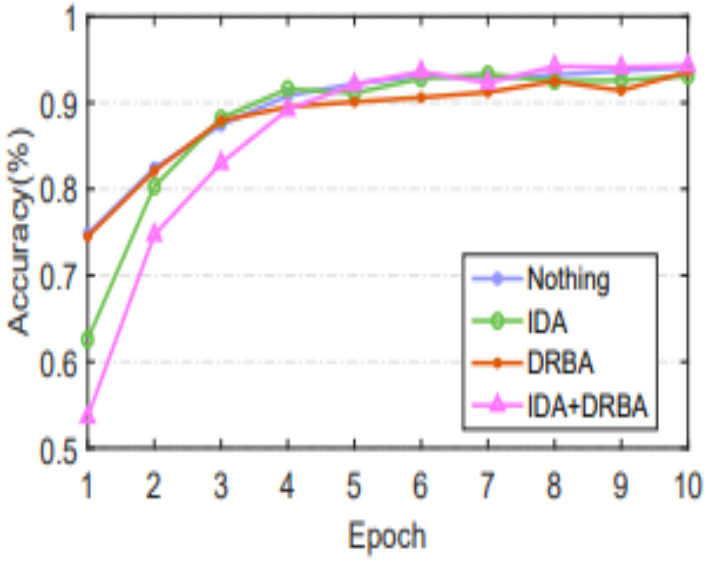
Model performance with different malware image size

The CNN framework required all input images to have a fixed size. Therefore, we reshaped the malware image to a fixed size square image. The smaller the transformed image, the more obvious the texture features were destroyed. In contrast, the larger images had good fidelity, and the texture features were clear. We conducted experiments setting the malware image at 24*24, 48*48, 96*96 and 192*192, respectively.

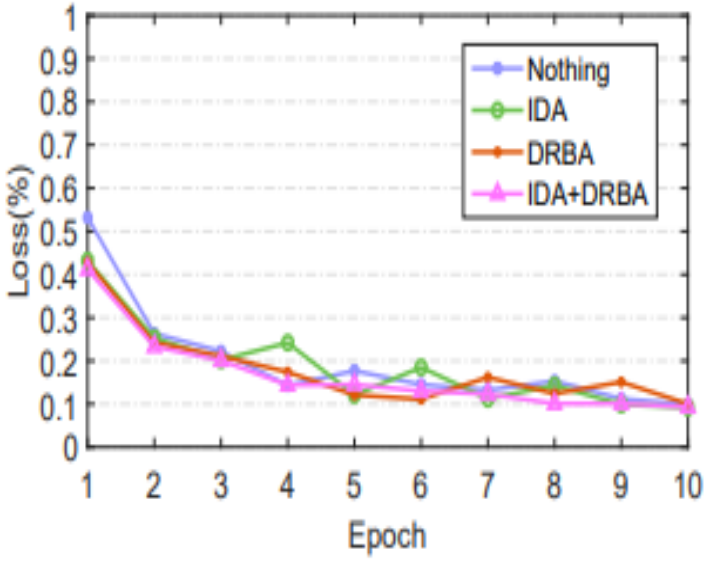
From the results we can see that our model performed better as the image became larger as shown in fig 3.9 and 3.10. However, a bigger image requires more training time. We noted that the performances at 96*96 and 192*192 were similar .Since the classification model would take more time to train using the larger image, the 96*96 image was a good choice for training on this malware dataset.

Comparison of our method with other malicious code detection method

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

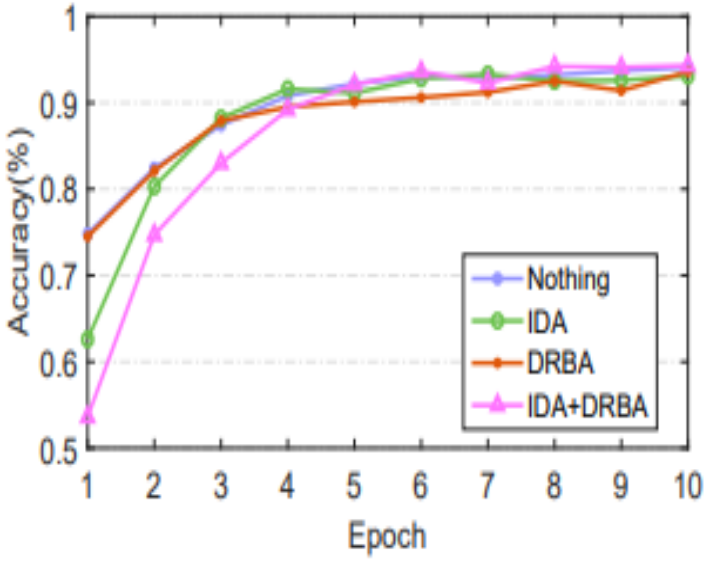


(a) *Accuracy*

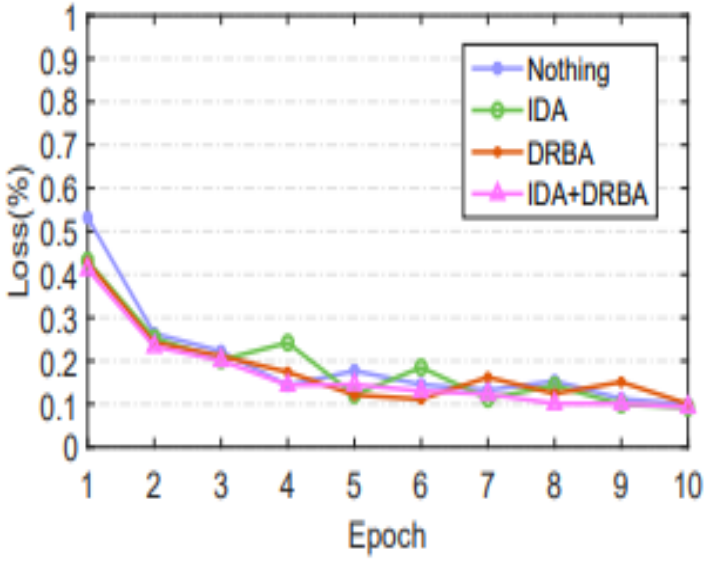


(b) *Loss*

Fig. 3.9: Performance of different classification models on the training set.



(a) *Accuracy*



(b) *Loss*

Fig. 3.10: Performance of different classification models on the validation set.

Conclusion

This paper proposed a novel method to improve the detection of malware variants through the application of deep learning. First, this method transformed the malicious code into grayscale images. Next, the images were identified and classified by a convolutional neural network that could extract the features of the malware images automatically. Because of the effectiveness and efficiency of the CNN for identifying malware images, the detection speed of our model was significantly faster than speeds achieved by other approaches. In addition, our model provided an effective solution for the data imbalance problem among different malware families. Our experimental results on 9,342 grayscale images of 25 malware families showed that the proposed approach achieved 94.5accuracy with good detection speed.In this study, the CNN framework required all input images to have a fixed size, which limited our model. In future work, we would like to use the SPP-net model to allow images of any size to be used as input. The SPP-net can extract features at variable scales thanks to a spatial pyramid pooling layer. We can introduce that layer into our model between the last subsampling layer and the fully connected layer to improve our models flexibility. In addition, the transformation of malicious code into color images would be a good topic for future research.

REFERENCES

[1] . Bouvrie. Notes on convolutional neural networks. 2006

[2] . Cai, H. Wang, Z. Cui, J. Cai, Y. Xue, and L. Wang. Bat algorithm with triangle-flipping strategy for numerical optimization. International Journal of Machine Learning and Cybernetics, 9(2):199215, 2018.

[3] . Christodorescu, S. Jha, S. A. Seshia, D. Song, and R. E. Bryant. Semantics-aware malware detection. In 2005 IEEE Symposium on Security and Privacy, pages 3246. IEEE, 2005.

[4] . Cui, Y. Cao, X. Cai, J. Cai, and J. Chen. Optimal leach protocol with modified bat algorithm for big data sensing systems in internet of things. Journal of Parallel and Distributed Computing, 2018. (doi:10.1016/j.jpdc.2017.12.014).

[5] . Cui, B. Sun, G. Wang, Y. Xue, and J. Chen. A novel oriented cuckoo search algorithm to improve dv-hop performance for cyberphysical systems. Journal of Parallel and Distributed Computing, 103:4252, 2017.

[6] . E. David and N. S. Netanyahu. Deepsign: Deep learning for automatic malware signature generation and classification. In Neural Networks (IJCNN), 2015 International Joint Conference on, pages 18. IEEE, 2015.

[7] . Daz-Pernil, A. Berciano, F. Pena-Cantillana, and M. A. Guti errez- Naranjo. Bioinspired parallel computing of representative geometrical objects of holes of binary 2d-images. International Journal of BioInspired Computation, 9(2):7792, 2017.

[8] . Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. In Proceedings of the 22nd ACM international conference on Multimedia, pages 675678. ACM, 2014.