Hindawi
Security and Communication Networks
Volume 2019, Article ID 4895984, 11 pages
https://doi.org/10.1155/2019/4895984

Research Article

MalDeep: A Deep Learning Classification Framework against Malware Variants Based on Texture Visualization

Yuntao Zhao,[1,2] **Chunyu Xu**,[1] **Bo Bo**,[1] **and Yongxin Feng**[1]

[1]*School of Information Science and Engineering, Shenyang Ligong University, Shenyang 110159, China*
[2]*College of Information Science and Engineering, Northeastern University, Shenyang 110819, China*

Correspondence should be addressed to Yongxin Feng; fengyongxin@263.net

Received 5 October 2018; Revised 5 February 2019; Accepted 5 March 2019; Published 1 April 2019

Guest Editor: Jin Wei

The increasing sophistication of malware variants such as encryption, polymorphism, and obfuscation calls for the new detection and classification technology. In this paper, MalDeep, a novel malware classification framework of deep learning based on texture visualization, is proposed against malicious variants. Through code mapping, texture partitioning, and texture extracting, we can study malware classification in a new feature space of image texture representation without decryption and disassembly. Furthermore, we built a malware classifier on convolutional neural network with two convolutional layers, two downsampling layers, and many full connection layers. We adopt the dataset, from Microsoft Malware Classification Challenge including 9 categories of malware families and 10868 variant samples, to train the model. The experiment results show that the established MalDeep has a higher accuracy rate for malware classification. In particular, for some backdoor families, the classification accuracy of the model reaches over 99%. Moreover, compared with other main antivirus software, MalDeep also outperforms others in the average accuracy for the variants from different families.

1. Introduction

Nowadays, with the advent of Internet of Things era, Cyberspace, known as the "fifth dimensional space", whose tentacles have extended to all aspects of life, is inextricably linked to everyone all over the world. Therefore, Cyberspace security is becoming more and more important and has been an issue which draws a lot of attention of researchers all along. According to the release of AV-Test [1], an internationally renowned security software evaluation agency, by the end of 2017 more than 600 million of the Windows malware and nearly 19 million of the malicious code variants for Android devices had been discovered. Symantec's report [2] states that in 2017 the mobile terminal malicious code family for mobile terminal grew by 54% over the same period in the last year. At the same time, the number of malicious code variants has risen sharply from 5:1 per family in 2011 to 38:1 in 2012 [3]. This suggests that malicious code programmers spend more time making minor changes or packaging to further propagate and escape detection. Therefore, the detection method against malicious code variants is the focus of current malware protection.

In order to protect legitimate users, the most effective way to defend against malware is antivirus software, which mainly adopts the signature-based detection method [4, 5]. Signature is a string of feature codes that is the only identity representation and can be distinguished from other software or program codes. However, malware makers can easily evade this detection method by variants generation techniques [6, 7]. There are two main kinds of the generation technology including share basic technology and obfuscation technology. The share basic technology is that a hacker implements malicious code variants by reusing the basic module. The obfuscation technology is developed for existing defense and detection technologies, which is now more widely used and divided into two categories according to its implementation principle. One is the interference confusion of reverse engineering (disassembly), which prevents reverse engineering from getting the correct analysis result. The other is the confusion of the instruction and control flow,

which is usually used to change the syntax characteristics of malicious code and hide its internal call logic by packing, inserting garbage code, replacing instruction equivalent, or redistributing register. For detecting malicious variants, the core technology is to extract and represent the essential features of the malicious behavior. So far the malware feature representations are mainly divided into two categories including the static and the dynamic representation. Based on the static representation, the feature is extracted from malicious code by analyzing the PE file structure, binary byte code, disassembled code, and system call after disassembly. However, the static representation is usually vulnerable to obfuscation technology. Different from the static representations, the main principle of dynamic representation is to place the target program or code under a virtual machine environment (sandbox or honeypot) and judge whether it is malicious by detecting the behavior of the running process. For example, the detection is performed by analyzing the sequence of API calls or sequences of instructions in terms of those behaviors. Compared with static representation, the dynamic one does not need to consider complicated reverse engineering such as disassembly and decryption. Though dynamic detection is more resilient to general obfuscation, it is a time intensive and resource consuming method, which means that it costs a lot of running time and storage space. Moreover, due to the fact that the dynamic execution conditions are sometimes not satisfied, some malicious codes and calls cannot be displayed, which will affect the discrimination for malicious codes.

At present, intelligent malware detection frameworks have been developed at the cloud or server side by applying big data and machine learning techniques [8–11]. Statistical machine learning methods, such as SVM, Naive Bayes, and decision trees, have been used for model construction to detect malicious codes [12, 13]. Currently visualization technology and machine learning are combined to detect malware variants. Nataraj et al. [14] propose an approach method for visualizing and classifying malware using image processing techniques. Malware binaries are visualized as Gray-Scale Images Texture (GSIT). The experimental results show that the images belonging to the same family appear very similar in layout and texture. Han et al. [15] propose a texture-fingerprint-based approach to extract or detect the feature from malware content. The malicious code is mapped to uncompressed gray-scale image, which is partitioned into blocks by the texture segmentation. The authors adopt gray level cooccurrence matrix (GLCM) to extract block feature and obtain texture fingerprints of malicious code. Machine learning is used for training and testing in both methods. But most of these methods belong to shallow learning architectures. The accuracy of the shallow learning algorithm depends on the feature representation, which needs to satisfy certain conditions or assumptions. For example, Naive Bayes requires the condition independence assumption that the attributes or characteristics of datasets are independent of each other. For liner SVM, it is necessary to meet the linear separable condition. These assumptions or conditions limited their real application. Moreover, the improper parameters or excessive training accuracy in shallow learning will lead to overfitting. Though they had been applied and succeeded in many ways, shallow learning architectures are still somewhat unsatisfactory in malware detection.

As a branch of machine learning, deep learning, characterization of learning a deep nonlinear network, achieves the approximation of complex function and shows the great power in extracting the intrinsic feature of the training data. By learning and training on the deep nonlinear network structure from a large number of hidden layers, the feature representation of samples in the original space is transformed to the new feature space step by step. Compared with other methods of machine learning, deep learning is an end-to-end system, in which there is no need for human participation and prior knowledge. The best advantage of depth learning is that it can automatically learn features and extract features.

In this paper, MalDeep, a novel deep learning classification framework for texture visualization of gray image, is proposed against malware variants. The framework is based on convolutional neural network supporting the weight update from cloud side to device ends. Through three processes, namely, code mapping, texture partitioning, and texture extracting, binary files are converted to gray images without decryption and disassembly. The image texture representation for malware features is easy to get, which provides a new feature space to study malware classification. Furthermore, we built a malware classifier on convolutional neural network of ten hidden layers. We adopt the dataset [16], from Microsoft Malware Classification Challenge including 9 categories of malware and 10868 variant samples, to train the model. With learning and training on the real sample collection, the experiment results show that MalDeep outperforms main antivirus software in the detection of the malware from different families.

The rest of this paper is organized as follows. Section 2 presents the overview of our MalDeep framework. Section 3 introduces our proposed method in detail. In Section 4, based on the real sample collection from Microsoft Malware Protection Center, we systematically evaluate the performance of our MalDeep which integrates our constructed CNN model, in comparison with other antivirus software (e.g., Kingsoft, Qihoo-360). Finally, Section 5 is conclusion and future work.

2. MalDeep Framework

MalDeep, our developed detection system of deep learning against malware, is performed on binary file, which is more likely to be achieved than disassembly file. Moreover, there is the richest and most primitive and lossless information in binary file. Any re-coding and pre-feature extraction will result in potential loss of useful information. It is very suitable for depth learning in binary files. Furthermore, the deep learning model has more advantages in image feature extraction with many successful cases. Different from single client detection system, we separate the training model from the detection model. The model training process that consumes computing and storing resources is deployed at cloud side, and the testing and detection process is at device (client) side. The learning model parameters $[W_{M\times N}]$ at the

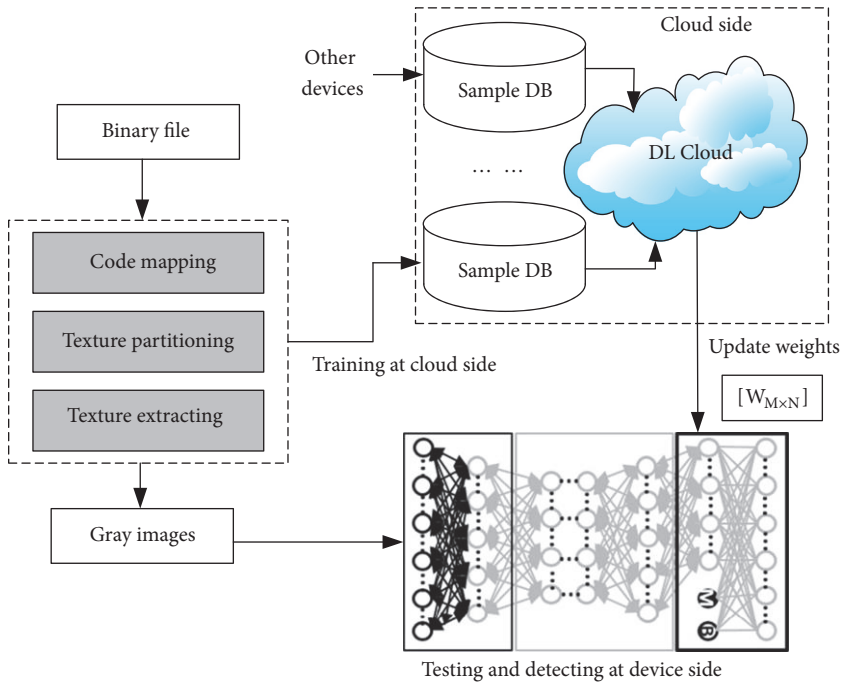


Figure 1: System framework of MalDeep.

device side are updated periodically from the cloud, which greatly reduces the computing and storage resources at the device side. The accuracy of the deep learning model relies on the cloud by training through a large number of data samples. The overall system framework of MalDeep is as shown Figure 1.

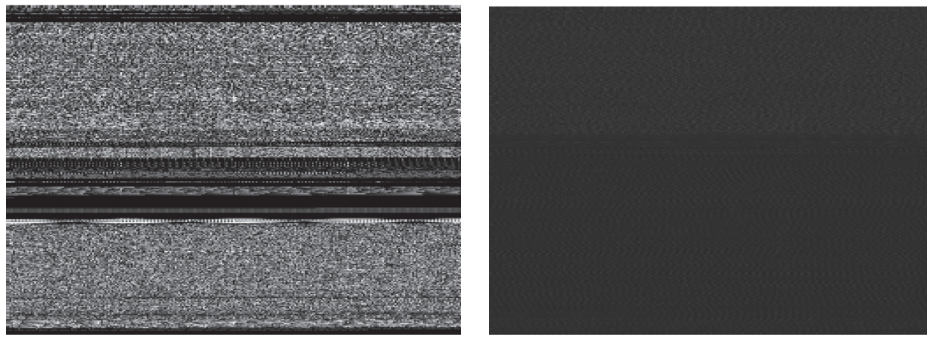
MalDeep is aimed at malware variants based on machine learning of gray images from binary file. First of all, the binary file of malware is converted into gray images through three processes, namely, code mapping, texture partitioning, and texture extracting. We read an 8-bit unsigned integer for a given malicious code as a pixel, set a width of image, e.g., 256, as a vector, and finally generate a two-dimensional array of the whole file. Texture is the regional feature of the image, which is the description of the spatial distribution of the pixel gray level of the image. Through texture analysis, important description information can be obtained from images. Texture partitioning can determine the boundaries of the texture by calculating the significant changes of the features. Moreover, texture partitioning can separate noise texture features and nonsignificant features from image features. Based on texture partitioning, texture extracting is the process through which the main features are achieved and nonsignificant features are excluded. Of course, in order to preserve the full information of the malware file, we can do only code mapping, not the last two steps. After the gray images are generated, the images are put into deep learning model for classifying malware variants. Considering the excellent performance of convolutional neural network (CNN) in image processing, we use CNN as the model. The model parameters are from Deep Learning Cloud (DL Cloud), which collects the malware samples, trains the model, generates the parameters, and updates the parameters to the device side.

3. Proposed Method

3.1. Binary File to Gray Images. Binary file of malware is easier to be achieved than disassembly file, especially for the client device. In addition, binary files have better visualization. Using IDA Pro [8] to disassemble the malicious code, we can get the ASM (file format) assembly file, and we get the binary file of malware at the same time. In order to compare between the ASM file and the binary file, we visualize the same malware and get Figure 2.

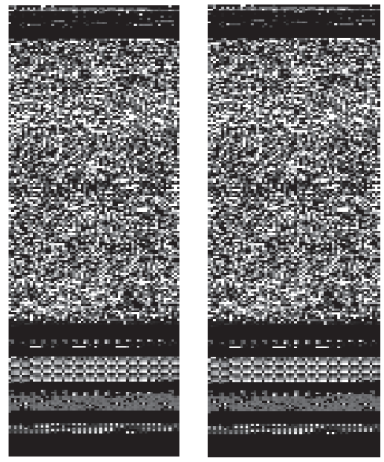
From Figure 2, we can see that the texture of the binary file is clearer, which makes it easier to extract features from them. Assembly files rely on fixed syntax structure, which makes the gray difference of images less obvious. It is especially critical for the construction of subsequent neural networks. Because the core idea of the neural network is to excavate the hidden features in the image, the texture and the representation of the file must be obvious. The fewer the layers needed to build a neural network, the shorter the time required to build the classification model.

On the other hand, most malware variants occur in non-critical areas, such as adding only null instruction (NOP) to the contents of malicious code files. Feature 3 is for two malware variants with B269894F434657DB2B15949641A67532 (MD5) and 049436bb90f71cf38549817d9b90e2da (MD5). The similarity of two images is as high as 99.9473%. They



(a) Image of binary file of malware (b) Image of ASM assembly file of malware

Figure 2: Comparison between binary file and ASM file.



(a) MD5-∗532 malware (b) MD5-∗2da malware

Figure 3: Image comparison between two malware variants.

almost have the same texture. The image comparison between two malware variants is shown in Figure 3.

In order to verify their homology, the two malicious samples were submitted to VirusTotal website [8]. After 71 antivirus engines scanned two files, 68 of them antivirus software engines identified the two samples as the same malware family. Table 1 shows some of results.

Although they are simple variants from the same malware family, the MD5 codes of the two samples are entirely different. In fact, there are only 7 bits in all 13284 bits. It means that every tiny change in malware variants will produce new signature-based codes. If the virus library is not updated and preserves the new code, the antivirus software will not be able to detect this variant. Accordingly, the feature representation method based on image can better match the variants from the same family.

In MalDeep, through three processes, code mapping, texture partitioning, and texture extracting, binary files of malware are converted into gray images. Firstly, we read an 8-bit unsigned integer for a given malicious code as a pixel, which is a gray level from 0 to 255. Secondly, we set a width of image, e.g., 256. Then we convert a row of 8∗256 bits to a vector such as $[x_1, x_2, \cdots, x_{256}]$. The height of the image is equal to file size divided by 8∗256. Finally, we generate a two-dimensional array of the whole file. In texture partitioning, the boundaries of the texture are determined by calculating the significant changes of entropy H of each row of images. For example, let $H = -\sum_{i=1}^{256} p_i \log p_i$, and $p_i = pixel_i/256$. A region with no significant change in entropy of continuous n rows is taken as the boundary of texture partitioning. In texture extracting, nonsignificant regions of the image are excluded and main features are achieved. Of course, in order to preserve the full information of the malware file, we can do only code mapping, not the last two steps. The whole process is as shown Figure 4.

3.2. CNN for Classification of Malware Based on Gray Images. As a feed-forward neural network, convolutional neural network (CNN) can respond to a part of the surrounding units in the coverage area, which has excellent performance in texture processing of a large number of malicious code images. In this paper, we adopt CNN to model malicious code texture. There are two main advantages: Firstly, CNN

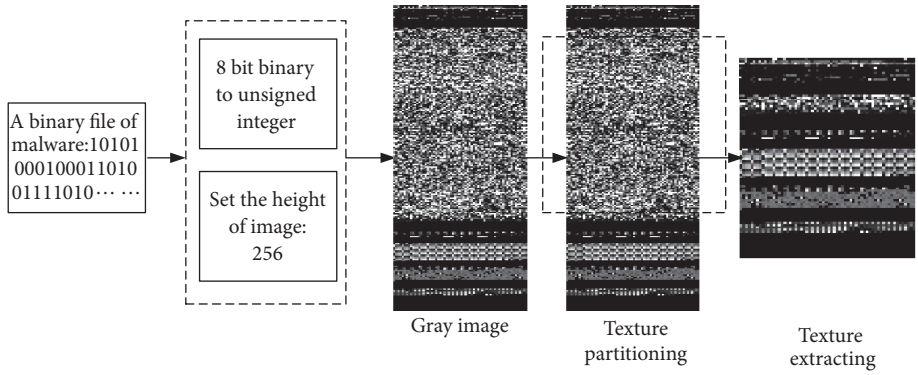


Figure 4: Binary file to gray images.

Table 1: Family of two malicious samples.

antivirus software	*family of MD5-∗532*	*family of MD5-∗2da*
Ad-Aware	Backdoor.Regin.C	Backdoor.Regin.C
AegisLab	Troj.W32.Regin.gen!c	Troj.W32.Regin!c
AhnLab-V3	Trojan/Win32.Regin.C302029	Trojan/Win32.Regin.C302029
ALYac	Trojan.Regin	Backdoor.Regin.C
Antiy-AVL	Trojan/Win32.Unknown	Trojan/Win32.Unknown
Arcabit	Backdoor.Regin.C	Backdoor.Regin.C
Avast	Win32:Regin-A [Rtk]	Win32:Regin-A [Rtk]
Kaspersky	Trojan (004b14761)	Trojan (0040f9a61)
Kingsoft	HEUR:Trojan.Win32.Regin.gen	HEUR:Trojan.Win32.Regin.gen
MAX	Backdoor.Agent.RE	Backdoor.Agent.RE

can learn and extract the deep-seated nonlinear features, which are difficult to find and understand by traditional feature extraction methods. Secondly, the training process of CNN needs a lot of calculation. Once trained, CNN is quite lightweight and can run with very small computing resources, which makes it very suitable for cloud-device framework.

The final structure of the model consists of 10 layers. For simplicity, only the fully connected layer, the added Dropout layer, and the fully connected loss layer are shown in Figure 5.

The loss function of full connection layer depends on the maximum regression and cross-entropy loss of full connection layer, whose objective function is expressed as follows.

$$J_0 = -\frac{1}{M}\left\{\sum_{i=1}^{M}\sum_{j=0}^{K} I\left(y_i = c_j\right)\log p_j^{(i)}\right\} \tag{1}$$

$I(y_i = c_j)$ indicates a function, which is defined as follows.

$$I(x) = \begin{cases} 1, & x = true \\ 0, & x = flase \end{cases} \tag{2}$$

Thereinto, M is the edge length of output feature map of Convolution Kernel. K is the edge length of Convolution Kernel. Therefore, Time Complexity of the model is as the follows.

$$Time \sim O\left(\sum_{l=1}^{D} M_l^2 \cdot K_l^2 \cdot C_{l-1}C_l\right) \tag{3}$$

Thereinto, D is the number of layers of CNN and l is the l-th convolution layer of CNN. C_l is the number of output channels C_{out} in the l-th convolution layer. Therefore, the number of input channels C_{in} of l-th convolution layer is the number of output channels C_{out} of the $(l-1)$-th convolution layer.

Space Complexity of the model is as follows.

$$Space \sim O\left(\sum_{l=1}^{D} K_l^2 \cdot C_{l-1}C_l\right) \tag{4}$$

Space Complexity of the model is only related to the size K of Convolution Kernel, the number C of channels, and the depth D of the model. It has nothing to do with the size of the input data in the process of calculation.

3.2.1. Construction of the Convolutional Layer. The first hidden layer of convolutional neural network for MalDeep is expressed as H_1, also known as convolutional layer. H_1 is made up of many convolutional planes, i.e., $H_1 = (h_{1,\alpha})$.

$$h_{1,a} = f\left(x \overset{\cup}{*} W^{1,\alpha} + b^{1,\alpha}\right) \tag{5}$$

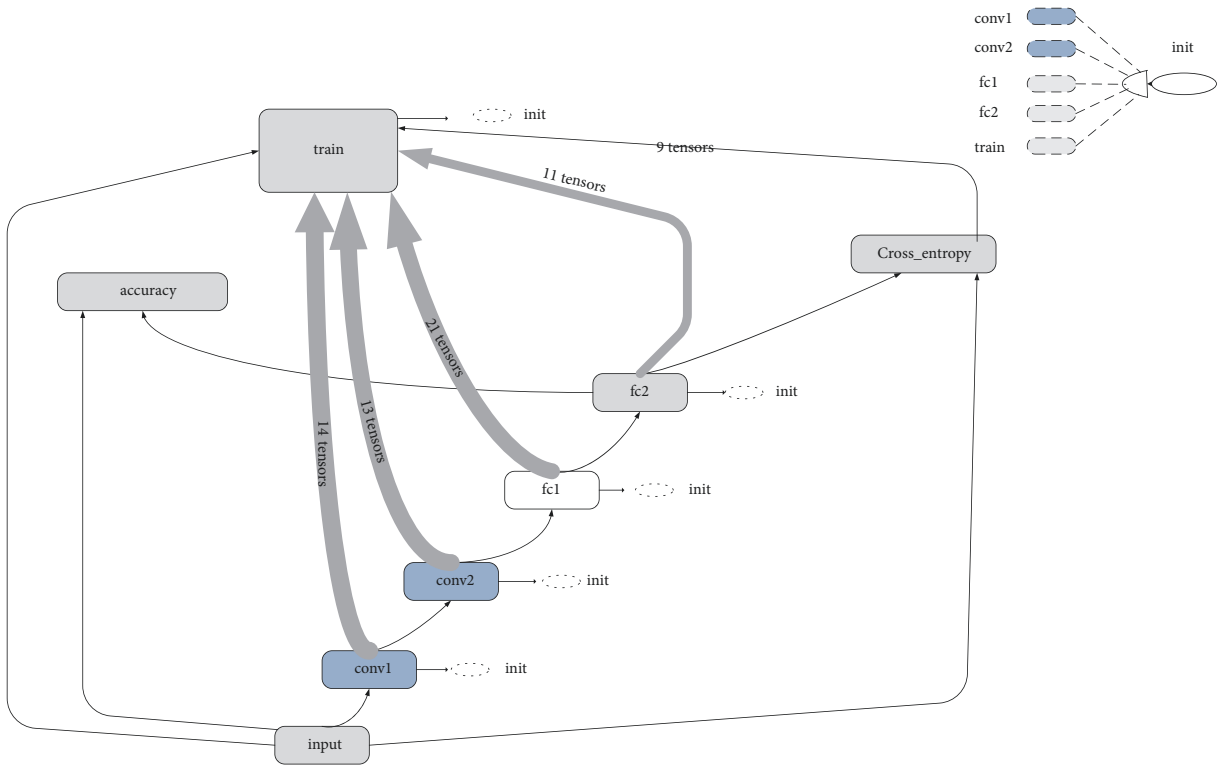


Figure 5: Overall logic diagram of the model.

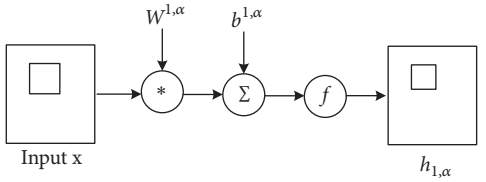


Figure 6: Connection from input layer to convolutional layer.

The connection and computation from input layer to convolutional layer are shown in Figure 6, where $h_{1,\alpha}$ is the convolutional plane α of H_1. $W^{1,\alpha}$ is the convolutional kernel α of H_1. $(\overset{\cup}{*})$ is inner convolution. Given two matrices A and B, sizes of $M \times N$ and $m \times n$, respectively, and $M \geq m, N \geq n$, the element of $A \overset{\cup}{*} B$ is expressed as follows.

$$c_{ij} = \sum_{s=1}^{m} \sum_{i=1}^{n} a_{i+m-s,j+n-i} \cdot b_{st}, \quad 1 \leq i \leq M - m + 1,\ 1 \leq j \leq N - n + 1 \tag{6}$$

3.2.2. Construction of the Downsampling Layer. The second hidden layer of convolutional neural network for MalDeep is expressed as H_2, which is used to downsample from H_1, so known as downsampling layer. H_2 is made up of many downsampling planes, i.e., $H_2 = (h_{2,\alpha})$.

$$h_{2,a} = g\left(\beta_2 down_{\lambda_2,\tau_2}\left(h_{1,\alpha}\right) + \gamma_2\right) \tag{7}$$

The connection and computation from convolutional layer to downsampling layer are shown in Figure 7, where the weight β_2 is usually set to 1, γ_2 is usually set to 0, and $down_{\lambda,\tau}()$ is downsampling operation.

$$down_{\lambda,\tau}(A) = \frac{1}{\lambda \times \tau} \sum_{s=(i-1)\times\lambda+1}^{i\times\lambda} \sum_{t=(j-1)\times\tau+1}^{j\times\tau} a_{st} \tag{8}$$

3.2.3. Construction of the Third Hidden Layer. The third hidden layer of convolutional neural network for MalDeep is expressed as H_3, which is computed through many downsampling planes and convolutional kernels from H_2. The r downsampling planes are expressed as h_{2,α_i} $(1 \leq i \leq r)$, and convolutional kernels are expressed as $W_{\alpha_i}^{3,(\alpha_1,\alpha_2,\cdots,\alpha_r)}$. H_3 is made up of many convolutional planes, i.e., $H_3 = (h_{3,\alpha})$.

$$h_{3,\omega} = f\left(\sum_{i=1}^{r} h_{2,a_i} \overset{\cup}{*} W_{a_i}^{3,\omega} + b^{3,\omega}\right) \tag{9}$$

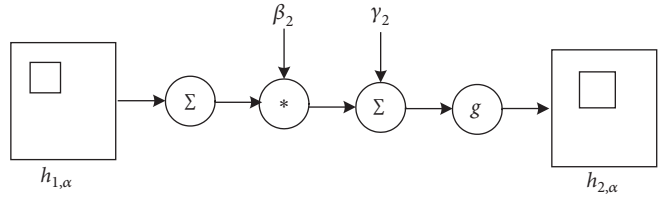


Figure 7: Connection from convolutional layer to downsampling layer.

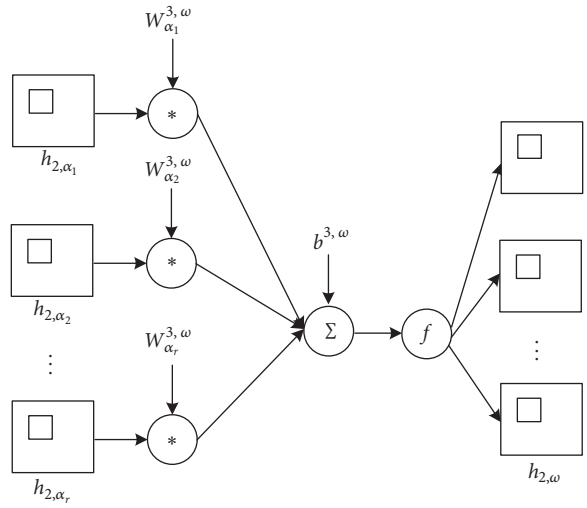


Figure 8: Connection from the second hidden layer to the third hidden layer.

Here $\omega = (\alpha_1, \alpha_2, \cdots, \alpha_r)$. The connection from the second hidden layer to the third hidden layer is shown in Figure 8.

3.2.4. Construction of the Fourth Hidden Layer. The fourth hidden layer of convolutional neural network for MalDeep is expressed as H_4, which is used to downsample from H_3. H_4 is made up of many downsampling planes, i.e., $H_4 = (h_{4,\omega})$.

$$h_{4,\omega} = g\left(\beta_4 down_{\lambda_4,\tau_4}\left(h_{3,\omega}\right) + \gamma_4\right) \tag{10}$$

The connection and computation from convolutional layer to downsampling layer are shown in Figure 9, where the weight β_4 is usually set to 1 and γ_4 is usually set to 0.

3.2.5. Full Connection Layer. The fifth to the tenth hidden layers of CNN for MalDeep are expressed as $H_5 \sim H_{10}$, which is used to classify malware. In full connection layer the activation function is usually *sigmoid*. In the output layer H_{10} the activation function is replaced by *softmax*.

4. Experiment and Result Analysis

4.1. Dataset and Performance Indices. The dataset is from Microsoft Malware Classification Challenge (BIG 2015) [16]. This competition is hosted by WWW 2015/BIG 2015 and the following Microsoft groups: Microsoft Malware Protection Center, Microsoft Azure Machine Learning, and Microsoft Talent Management. The malicious code was collected from 9 different malware families, including worms, malicious advertisements, Trojan horses, homepage hanging horses, fuzzy malware, and the back door of 4 family members. 80% of them will be used as training set and the remaining 20% as test set. The description of the dataset is shown in Table 2.

We evaluate the malware detection performance of different methods using the measures shown in Table 3.

4.2. Experimental Results. It can be seen from Figure 10 that the cross entropy of cost function of test loss and training loss is constantly decreasing and converging gradually with the increase of the number of iterations, indicating the availability of the established model.

The accuracy of the model by 50 iterations is shown in Figure 11. It can be seen from the figure that the accuracy increases with the number of iterations. Finally, the accuracy is around 92% after 50 iterations.

The confusion matrix of 9 categories of malware after classification of MalDeep is shown in Table 4.

The classification accuracy of various malware variants based on MalDeep is shown in Figure 12. And the accuracy of Obfuscator.ACY family is the lowest, which is due to the diversification of its malicious codes. On the other hand, the metamorphism and packers from Obfuscator.ACY family interfere with the detection model, which may affect texture in malware images and result in the incorrect classification for malware. For all that, the accuracy of Obfuscator.ACY family is still over 80%.

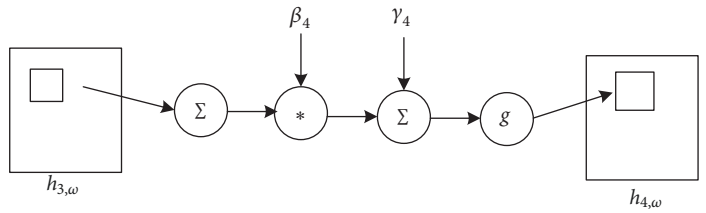


Figure 9: Connection from the third hidden layer to the fourth hidden layer.

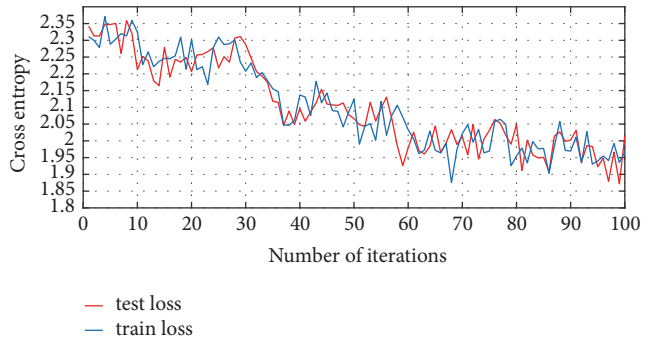


Figure 10: Analysis of the cross entropy of cost function of test loss and training loss.

Table 2: Description of the dataset.

Malware family name	*Number of samples*	*Malware category*
RAmnit	1541	Worm
Lollipop	2478	Adware
Kelihos_ver3	2942	Backdoor
Vundo	475	Trojan
Simda	42	Backdoor
Tracur	751	TrojanDownloader
Kelihos_ver1	398	Backdoor
Obfuscator.ACY	1228	obfuscated malware
Gatak	1013	Backdoor

Table 3: Performance indices of malware detection.

Indices	Description
TP	of samples correctly classified as malicious
TN	of samples correctly classified as benign
FP	of samples mistakenly classified as malicious
FN	of samples mistakenly classified as benign
TP rate (TPR)	TP/(TP + FP)
FP rate (FPR)	TP/(TN + FP)
Accuracy (ACY)	(TP + TN)/(TP + TN + FP + FN)

Furthermore, we compare MalDeep with GIST [14] and GLCM [15]. Both GIST and GLCM are methods for visualizing and classifying malware using image processing techniques. In GSIT malware binaries are visualized as Gray-Scale Images Texture (GSIT). The experimental results show that the images belonging to the same family appear very similar in layout and texture. In GLCM authors propose a texture-fingerprint-based approach to extract or detect the feature from malware content. The malicious code is mapped to uncompressed gray-scale image, which is partitioned into blocks by the texture segmentation. The authors adopt gray level cooccurrence matrix (GLCM) to extract block feature

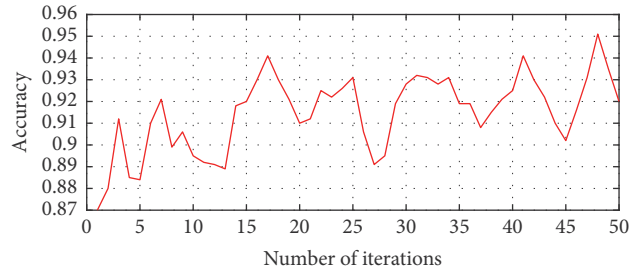


Figure 11: Accuracy of the model by 50 iterations.

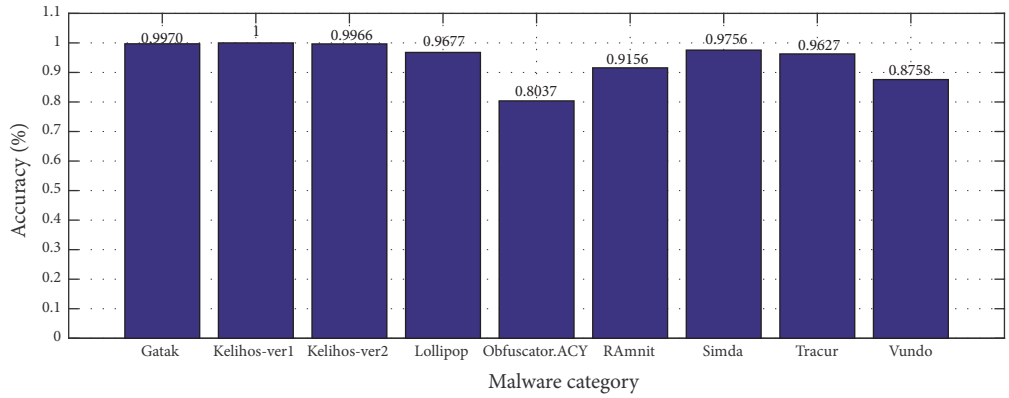


Figure 12: Classification accuracy of various malware variants based on MalDeep.

and obtain texture fingerprints of malicious code. In both methods the shallow learning is used for classification on training and testing.

The experimental results of malware detection method against malware variants are shown in Table 5.

At last, we compare the average accuracy for malware classification between main antivirus software and MalDeep based on Microsoft dataset. Table 6 shows detection results from MalDeep and other antivirus software. We can see that MalDeep outperforms others in the detection of the malware from different families.

5. Conclusion and Future Works

In this paper, MalDeep, a novel malware classification framework of deep learning based on texture visualization, is proposed against malware variants. Through conversion from binary file to gray images including code mapping, texture partitioning, and texture extracting, we adopt the texture representation for malware features. On this foundation, we analyze the advantages of binary images relative to assembly images and then compare the similarity between malware variants in the same family. Furthermore, we built a malware classification model of convolutional neural network based on texture visualization. Moreover, we adopt the dataset [16], from Microsoft Malware Classification Challenge, to train the model. The malicious variants were collected from 9 different malware families, including worms, malicious advertisements, Trojan horses, homepage hanging horses, fuzzy malware, and the back door of 4 family members. The experiment results show that the model has good convergence performance on cost function, cross entropy, test loss, and training loss. Meanwhile, the established model has a higher accuracy rate for malware classification. In particular, for some backdoor families, the classification accuracy of the model reaches over 99%. Compared with other main antivirus software, MalDeep outperforms others in the detection of the malware from different families. In future, focusing more closely on cloud-device framework, we plan to carry out performance analysis of and comparison between a single device and cloud-device framework. Limited by experimental conditions, there is no quantitative analysis on the model parameters of MalDeep transmitted between the cloud and device ends. With improving experimental conditions, in subsequent studies we will further analyze the influence of the model parameters (format, simplification, update time, etc.) on the performance of the model. As future work, we also plan to extend the detection capabilities of MalDeep, e.g., by supporting more complex and efficient algorithm for images, which may speed up the computation and classification of MalDeep.

Data Availability

The dataset [16] is from Microsoft Malware Classification Challenge (BIG 2015). This competition is hosted by WWW 2015/BIG 2015 and the following Microsoft groups: Microsoft Malware Protection Center, Microsoft Azure Machine Learning, and Microsoft Talent Management. The dataset is real and available.

Table 4: Confusion matrix of 9 categories of malware.

real	prediction									
	RAmnit	Lollipop	Kelihos_ver3	Vundo	Simda	Tracur	Kelihos_ver1	Obfuscator.ACY	Gatak	number
RAmnit	1411	5	6	15	0	0	16	87	1	1541
Lollipop	11	2398	0	10	0	0	0	56	3	2478
Kelihos_ver3	1	0	2932	0	0	0	0	9	0	2942
Vundo	7	13	0	416	1	6	0	32	0	475
Simda	0	0	0	0	40	0	0	1	0	41
Tracur	0	12	0	0	0	723	1	15	0	751
Kelihos_ver1	0	0	0	0	0	0	384	0	0	384
Obfuscator.ACY	12	5	32	91	25	16	54	987	6	1228
Gatak	0	0	0	0	0	1	0	2	1010	1013

Table 5: Experimental results of malware detection method against malware variants.

Method	TPR/%	FPR/%	ACY/%	AUC
GIST	83.9	12.8	82.3	0.826
GLCM	90.3	10.8	86.6	0.850
MalDeep	97.2	9.4	92.9	0.983

Table 6: Comparison between main antivirus software and MalDeep.

Antivirus software	*Kingsoft*	*Qihoo-360*	*Microsoft*	*Ad-Aware*	*McAfee*	***MalDeep***
Average accuracy	81.3%	91.2%	88.9%	86.3%	90.9%	**92.5**%

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this article.

Authors' Contributions

Yuntao Zhao is the main contributor of this work, given that he originated the idea, provided the general design, and wrote most of the paper. Chunyu Xu and others contributed to the implementation and testing of the experiment. All authors read and approved the final manuscript.

Acknowledgments

This work was supported by China Postdoctoral Science Foundation (2016M590234), General Project of Liaoning Provincial Department of Education (LG201611), Postdoctoral fund of Shenyang Ligong University, Project of Applied Basic Research of Shenyang (18-013-0-32), 2017 Distinguished Professor Project, Liaoning Nature Foundation (20180551066), Program for Liaoning Distinguished Professor.

References

[1] Av test, "Facts and figures - security report 2016/2017," https://www.av-test.org/fileadmin/pdf/security_report/AV-TEST_Security_Report_2016-2017.pdf, 2017.

[2] Internet Security Threat Report (ISTR), https://www.symantec.com/security-center/threat-report, 2018.

[3] C. Wueest, "Symantec security response: financial threats 2015," http://www.syman-tec.com/content/en/us/enterprise/media/security_response/whitepapers/financial-threats-2015.pdf, 2016.

[4] E. Filiol, "Malware pattern scanning schemes secure against black-box analysis," *Journal of Computer Virology and Hacking Techniques*, vol. 2, no. 1, pp. 35–50, 2006.

[5] E. Filiol, G. Jacob, and M. L. Liard, "Evaluation methodology and theoretical model for antiviral behavioural detection strategies," *Journal of Computer Virology and Hacking Techniques*, vol. 3, no. 1, pp. 23–37, 2007.

[6] A. H. Sung, J. Xu, P. Chavez, and S. Mukkamala, "Static analyzer of vicious executables (SAVE)," in *Proceedings of the 20th Annual Computer Security Applications Conference (ACSAC '04)*, pp. 326–334, December 2004.

[7] E. Mariconti, L. Onwuzurike, P. Andriotis, E. De Cristofaro, G. Ross, and G. Stringhini, "MaMaDroid: detecting android malware by building markov chains of behavioral models," in *Proceedings of the ISOC Network and Distributed Systems Security Symposiym (NDSS)*, pp. 1–15, 2017.

[8] Y. Ye, D. Wang, T. Li, D. Ye, and Q. Jiang, "An intelligent PE-malware detection system based on association mining," *Journal of Computer Virology and Hacking Techniques*, vol. 4, no. 4, pp. 323–334, 2008.

[9] M. Bailey, J. Oberheide, J. Andersen, Z. Mao, F. Ahanian, and J. Nazario, "Automated classification and analysis of internet malware," in *Proceedings of the 10th International Symposium on Research in Attacks, Intrusions and Defenses (RAID)*, pp. 178–197, 2007.

[10] M. M. Masud, T. M. Al-Khateeb, K. W. Hamlen et al., "Cloud-based malware detection for evolving data streams," *ACM Transactions on Management Information Systems (TMIS)*, vol. 2, no. 3, pp. 1–27, 2011.

[11] Y. Ye, D. Wang, T. Li, and D. Ye, "IMDS: intelligent malware detection system," in *Proceedings of the 13th ACM SIGKDD*, pp. 1043–1047, August 2007.

[12] R. A. Dunne, *A Statistical Approach to Neural Networks for Pattern Recognition*, Wiley Press Group, Hoboken, NJ, USA, 2007.

[13] J. Kolter and M. Maloof, "Learning to detect and classify malicious executables in the wild," in *Proceedings of the Tenth ACMSIGKDD International Conference on Knowledge Discovery and Data Mining (ACM SIGKDD'04)*, vol. 7, pp. 470–478, 2004.

[14] L. Nataraj, S. Karthikeyan, G. Jacob, and B. S. Manjunath, "Malware images: visualization and automatic classification," in *Proceedings of the 8th International Symposium on Visualization for Cyber Security, (VizSec '11)*, USA, July 2011.

[15] X. G. Han, W. Qu, and X. X. Yao et al., "Research on malicious code variants detection based on texture fingerprint," *Journal on Communications*, vol. 35, no. 8, pp. 126–136, 2014.

[16] Microsoft Malware Classification Challenge (BIG 2015), https://www.kaggle.com/c/malware-classification/data, 2015.