Hindawi
Security and Communication Networks
Volume 2018, Article ID 7247095, 16 pages
https://doi.org/10.1155/2018/7247095

Research Article

Detecting Malware with an Ensemble Method Based on Deep Neural Network

Jinpei Yan, Yong Qi, and Qifan Rao

Department of Computer Science and Technology, Xi'an Jiaotong University, Xi'an, Shaanxi, China

Correspondence should be addressed to Yong Qi; qiy@xjtu.edu.cn

Received 18 August 2017; Revised 3 December 2017; Accepted 6 February 2018; Published 12 March 2018

Academic Editor: Zonghua Zhang

Malware detection plays a crucial role in computer security. Recent researches mainly use machine learning based methods heavily relying on domain knowledge for manually extracting malicious features. In this paper, we propose MalNet, a novel malware detection method that learns features automatically from the raw data. Concretely, we first generate a grayscale image from malware file, meanwhile extracting its opcode sequences with the decompilation tool IDA. Then MalNet uses CNN and LSTM networks to learn from grayscale image and opcode sequence, respectively, and takes a stacking ensemble for malware classification. We perform experiments on more than 40,000 samples including 20,650 benign files collected from online software providers and 21,736 malwares provided by Microsoft. The evaluation result shows that MalNet achieves 99.88% validation accuracy for malware detection. In addition, we also take malware family classification experiment on 9 malware families to compare MalNet with other related works, in which MalNet outperforms most of related works with 99.36% detection accuracy and achieves a considerable speed-up on detecting efficiency comparing with two state-of-the-art results on Microsoft malware dataset.

1. Introduction

Nowadays, various kinds of software provide wealth resources for users but also bring a certain potential danger; thus malware detection is always a highly concerned issue in computer security field. According to the recent study, the number of malicious samples is rapidly increasing. For instance, 69,277,289 kinds of malicious objects (scripts, exploits, executable files, etc.) are detected by Kaspersky Lab in 2016 [1]. The total number of malware samples increased 22% in the past four quarters to 670 million samples detected by McAfee Labs [2] in 2017. The number of samples is too large, requiring a highly effective way to detect malwares.

A large number of researches have studied methods for analyzing and detecting malware. Traditional commercial antivirus products usually rely on signature-based method, which needs a local signature database to store patterns extracted from malware by experts. However, this approach has great limitations since specific minor changes to malware can change the signature, so more and more malware could easily evade signature-based detection by encrypting, obfuscating, or packing. Hence, many different malware detection approaches with machine learning technology have been proposed in recent years, such as static analysis which learns statistical characteristics like API calls, N-grams, and so on [3, 4] or dynamic behavior analysis [5]. Though dynamic analysis does not require complex reverse engineering, it needs to simulate the operation environment for malwares, which is difficult to arouse all malware behaviors. At the same time, it is time-consuming for malware behavior monitoring since some malicious behaviors hide for a long time before attack. For static analysis, a great strength is that it can achieve rapid detection for massive malwares. However, various encryption and obfuscation techniques are the major issue for static analysis. Attackers can deliberately make various changes on malwares, hence static analysis is difficult to capture the characteristics of malware. Meanwhile, malware uses packing technologies to prevent reverse engineering which leads to high costs for static analysis.

At present, several machine learning methods [5–7] are paid the most attention for solving the above problems and have been applied to malware detection in the industry. However, many of them heavily rely on the relevant domain knowledge for malware analysis and artificial features extraction. These features are used to train a classification machine learning model and finally make the classification for a new file sample. But a serious problem is that malware is constantly being created, updated, and changed. To deal with this, a great deal of expert knowledge is required to catch up the changing malware environment, and the original well-designed features may not be applicable to a new malware family (a malware family refers to a malware variants group with homogeneous attack behaviors), resulting in heavy and inefficient feature engineering work. Thus, how to reduce the cost of artificial feature engineering and how to extract useful information from the raw data and let the model achieve features of self-learning to improve the accuracy and efficiency for malware detection are our main motivations.

In this paper, we present MalNet, a novel malware detection method for detecting whether a Windows executable file is malware. MalNet performs a comprehensive static analysis which includes two novel methods based on deep neural networks. One method is learning from grayscale images by Convolution Neural Network (CNN). The grayscale image is extracted from raw binary file in which CNN can get the structure features of a malware from its local image patterns. The other method is learning from opcode sequence by Long-Short Term Memory (LSTM). Opcode sequences are extracted by decompilation tool where LSTM can learn features about malicious code sequences and patterns. In reality, since malware often contains very long opcode sequences which cause the gradient vanishing problem of LSTM when training, we take truncated backpropagation algorithm based on subsequence to solve this problem in this paper which can also allow LSTM parallel computing on a bunch of subsequences to improve training efficiency. Meanwhile considering that malicious codes may be implanted into a normal file by attackers, in this case malicious features or behaviors only appear in some opcode subsequences; hence we come up with subsequence selection method to filter out benign subsequences of a malware which may mislead LSTM. Overall, MalNet uses these two networks to learn features from the raw data and then uses stacking ensemble to fuse two networks' discriminant result with extra metadata feature and finally generates a binary classification result for malware detection.

To verify the performance of MalNet, we perform evaluation experiments on a large dataset, which contains 21,736 malware samples from Microsoft and 20,650 benign samples collected by us. We choose 1/10 samples as validation dataset, where MalNet achieves detection accuracy of 99.88% and true positive rate of 99.14% with a false positive rate of 0.1%, much higher than the N-gram baseline result. Meanwhile we also make a malware family classification for 21,736 malware samples in 9 malware families to compare MalNet with other related works, where MalNet achieves 99.36% overall accuracy outperforming most of other methods. Besides, since rapid growing malware samples require a fast and efficient malware detection method, we evaluate the detection efficiency for MalNet. The result shows that since MalNet does not need to do special feature extraction, it only takes 0.03 s to give a prediction in detection phase which is superior comparing with two state-of-the-art methods only costing a little detection accuracy behind.

In summary, we make the following contributions in this paper:

(i) We propose a novel approach using deep neural networks for malware detection which takes CNN and LSTM networks to automatically learning features from the raw data to capture the malicious file structure patterns and code sequence patterns. It greatly reduces the cost of artificial features engineering.

(ii) We design and implement MalNet, a malware detection method, and solve practical problems such as grayscale image generation, very long sequences learning and gradient vanishing problem for LSTM, parallel computation for LSTM, and noise data processing. And we further use stacking ensemble for MalNet to combine networks' results to optimize the detection accuracy.

(iii) We make a series of evaluation experiments for MalNet including malware detection and malware family classification. The results show that MalNet outperforms most of other related approaches on malware detection accuracy and gets a superior detecting efficiency.

The rest of this paper is organized as follows. Related work is discussed in Section 2. MalNet detection methodology is introduced in Section 3. Experiments and analysis are presented in Section 4. Sections 5 and 6 discuss and conclude the paper.

2. Related Work

Malware detection has always been a concern area of research in recent years. Several methods and techniques have been proposed to counter the growing amount and sophistication of malware.

Static Analysis for Malware Detection. Static analysis often uses lexical analysis, parsing, control flow, and data flow analysis techniques [8] to mine the program. One common static malware detection method for the previous industry communities is signature-based method. For an unknown executable file, they can determine whether it is a known malware by searching whether there is a matching signature in the malicious code database. This detection method [9] generated a unique signature identifier for a malware based on some specific manually designed features. However, signature-based methods are limited to detect unknown malwares, since an unknown malware may contain new features not captured by signatures. In addition, these signatures present a series of fixed malicious characteristics. So if the malware passes through the some encryption or obfuscation operation, it will get a high probability to evade the signature-based detection.

The current situation has promoted the development of dynamic analysis. Moser et al. [8] explored the shortcomings of static analysis methods and introduced a code obfuscation scheme that make it harder to complete the detection relying solely on static analysis. Since the dynamic analysis is not susceptible to code obfuscation conversion, it is an important complement to static analysis.

Dynamic Analysis for Malware Detection. Dynamic analysis is used by running a malware in a controlled environment (virtual machine, simulator, emulator, sandbox, etc.) and analyzing the behavior of malicious code (interaction with the system) [10]. Before executing the malware sample, the corresponding monitoring tools are required to open first such as Process Monitor, Capture BAT (for monitoring file system and registry), Process Explorer, and Process Hacker-replace (for monitoring process), Wireshark (for monitoring network), and Regshot (for detecting system change).

In the process of dynamic analysis, malware detection result comes from behavior information (including system calls traces, network access, and file and memory modifications [5, 11]) collection and analysis from the operating system (the execution environment of the program) through software runtime. These techniques have been widely studied as malware detection solutions, but they have also been noted to be less robust when exposed to large dataset [12]. Since it is hard to simulate every situation that can arouse malware behavior, it is difficult to determine the effective time for monitoring malware activity and when to stop. Hence, simulating all malware behaviors needs continuous monitoring of malware behavior which results in colossal waste of computer resources, and it will be an arduous task when detecting mass malwares in present. However, antivirus engines today receive a flood of new malware samples each day, so an automated approach is needed to be fast and save the cost of extensive manual analysis. A variety of machine learning based techniques have been proposed and used for malware detection.

Machine Learning Based Malware Detection. Recently, machine learning methods (e.g., Support Vector Machines (SVM), Decision Trees (DT)) have been used to detect and classify unknown samples for malware family due to its scalability, rapidity, and flexibility. Schultz et al. [13] first proposed to apply the data mining method to detect malware and used three different types of static features, respectively, PE head, string sequence, and byte sequence. Then a rule-based algorithm called Ripper [14] is applied to DLL data mining and used naive Bayesian as a learning algorithm to find the character data and pattern feature information of byte sequence. It takes the malicious code data as input and obtains their best classification accuracy rate of 97.11%. Kolter and Maloof [15] then achieved a better result by using N-gram instead of nonoverlapping byte sequence features for data mining. Their conclusion suggests that the best decision can be obtained by using the boost decision tree.

Saxe and Berlin [16] instead proposed a method to distinguish malware from benign one with a neural network. In their research, entropy histogram is calculated from binary data and the number of callings of the contextual byte data, and metadata of execution files and DLL import are extracted. Those four types of features are transformed to 256 dimensions vector one by one. Unknown samples are classified with feature vectors which are learned in a four-layer neural network. Their TPR result is 95.2% while FPR is 0.1%.

And there are some novel ideas for malware detection. Nataraj et al. [17] proposed a visualized malware classification approach through image processing. Specifically, the malware binary data is transformed into a grayscale image, and the classification was done by kNN model with Euclidean distance calculation. The experiments show that it is a fast malware detection method, but this method uses global image features so that attacker can use some local transformation for malware to evade. So in their follow-up paper [18], they compared two methods of image feature processing with dynamic analysis. The experimental results show that the method based on image feature is efficient and scalable and can obtain an accuracy closing to the dynamic analysis result. They also found that this improved method can perfectly deal with both packed and unpacked malware samples. Kong and Yan [19] proposed a framework for automatic malware classification based on unsupervised clustering learning by structured information (function call graphs). After extracting the fine-grained feature for function call graph of the malware, it will calculate the similarity of the malware by the distance matrix based discrimination learning method to cluster samples with the same malware family. After that, these pairs of malware distances are used for classification by an ensemble classifier. Santos et al. [20] proposed a method using N-gram features to distinguish malware from benignware. In their research, unknown malware is detected by k-nearest samples with most similar N-gram features. And there are more approaches with similar idea using N-gram based on byte, opcode, or API call frequency for identifying malware [15, 21].

Moreover, since it is difficult to accurately and efficiently complete malware detection from a single point of view of static or dynamic analysis, some studies have begun to integrate both dynamic and static features. Santos et al. [22] proposed a hybrid malware detection tool based on machine learning algorithms called OPEM that utilizes a set of features obtained from static and dynamic analysis of malicious code. Static features are obtained by mining opcodes from the executable files, and dynamic features are obtained by monitoring system calls, operations, and exceptions. The maximum accuracy of their malware detection rate is 96.60% with SVM classifier. The experiments proved that the hybrid method could get a better performance compared with running static or dynamic analysis separately.

The above machine learning based malware detection has achieved pretty good results; however, most of these methods rely heavily on expert knowledge for the design of features. At the same time, as the malware continues to grow and change dynamically, the human-designed features face many challenges which require a significant cost for manually updating features in response to new malware. Therefore, this paper tries to extract useful information from massive raw data and reduce the cost of artificial feature engineering by automatic feature learning characteristic of deep neural network.

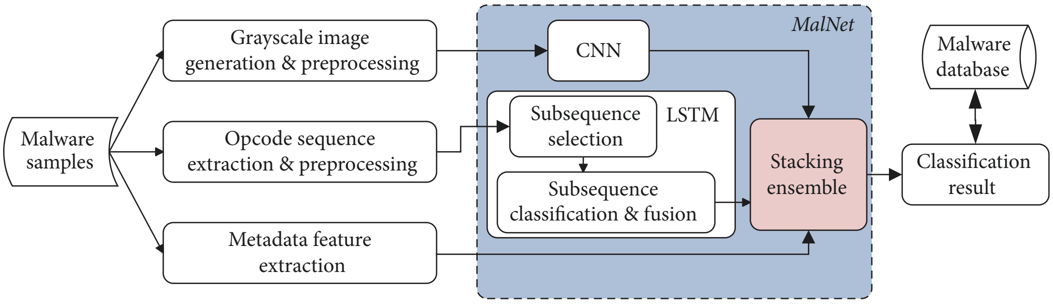

Malware samples
Grayscale image generation & preprocessing
Opcode sequence extraction & preprocessing
Metadata feature extraction
MalNet
CNN
Subsequence selection
LSTM
Subsequence classification & fusion
Stacking ensemble
Malware database
Classification result

Figure 1: The overview of our proposed malware detection process. MalNet is the core malware detection method.

3. Detection Methodology

In this section, we introduce the proposed method for malware detection and come up with a malware detection method called MalNet which uses CNN and LSTM networks. For malware detection, MalNet actually performs a binary classification task, receiving the raw file data as input, and outputs a discrimination probability indicating how likely it is a malware.

Concretely, the detection process by MalNet can be divided into two stages (see in Figure 1). The first stage is to preprocess malware sample data, it takes a binary form of a Windows executable file, generates a grayscale image from it, and extracts opcode sequence and metadata feature with decompilation tool. So this stage generates the appropriate data format as the input of the follow-up CNN and LSTM networks. The second stage applies the core process of MalNet, which takes CNN and LSTM networks, respectively, learning from the grayscale image and the opcode sequence. To optimize the detection performance, we use stacking ensemble to integrate two networks' output and metadata features and get final prediction result.

MalNet actually learns three different kinds of feature sets from the raw data; first MalNet learns malicious file structure features from the grayscale image by CNN and then learns malicious code pattern features from opcode sequence by LSTM. These two feature sets are reflecting the local pattern information; hence we add some simple metadata features as a description of the global information. The specific design of MalNet and the detection process are described in the following section.

3.1. Learning Malware Grayscale Image through CNN. In this section, we introduce CNN networks and describe how to construct malware structure feature by learning malware grayscale image through CNN. This method is inspired by Nataraj et al. [17], which visualizes malware binaries as grayscale images and these images can clearly reflect the structural characteristics of malware files.

3.1.1. Malware Grayscale Image Generation. In order to generate malware grayscale image, the raw data requires being processed and transformed into an image format. We take an executable file as input data and treat it as raw *.bytes* binary stream file. The binary stream file can be regarded as a hexadecimal stream file by converting every 4 bits into a hexadecimal number. Considering that the range of a hexadecimal number is exactly from 0 to 16, and we combine every two hexadecimal numbers exactly corresponding to the gray value of a 256-level image pixel. So the raw data can be converted to a grayscale image by this simple mapping transformation. The whole bunch of binary stream sequence is segmented for every 8 bits which corresponds to gray level of each pixel, and it is arranged sequentially to form the corresponding gray image. The generated grayscale images are shown in Figure 2(a).

Incidentally, we can use a similar method to generate grayscale images from decompiled files and only need to decompile the executable file first, obtaining *.asm* decompiled file, and then also treat it as binary stream for the same mapping transformation. However, in the actual process we found that the grayscale image generated by the decompiled file lost a lot of structure patterns (the generated grayscale images are seen in Figure 2(b)). Executable files with obvious differences still have very similar grayscale images presentations. The reason is that the decompiled file has a relatively fixed and organized structure since decompiled tool will output a normal format. For example, we use IDA Pro [23] which generates a decompiled file whose beginnings of each line are the PE segment name and the starting file address, followed by the decompile instruction. Therefore, all the decompiled files tend to have similar generated grayscale images. Thus, the grayscale image generated by the decompiled file is not suitable for further learning process.

3.1.2. Convolution Neural Network. MalNet takes a CNN to learn from grayscale images. As a typical deep neural network, CNN is widely used in computer vision area and image related tasks. The most notable characteristic of CNN is that it reduces a huge amount of calculation by the idea of weights sharing, local field, and subsample in space. It shares the same weight between a group of CNN neurons, mining patterns on local fields by convolution operation. CNN directly takes the raw image as input and outputs the classification or regression result with an end-to-end structure. And the neuron weights of CNN are trained by backpropagation algorithm. A typical application of CNN [24] is used for handwritten digital recognition through multiple convolution layers and pooling layers to handle the input data. Each convolution layer outputs a set of feature maps, while each feature map represents a high-level features extracted via one specific convolution filter. And the

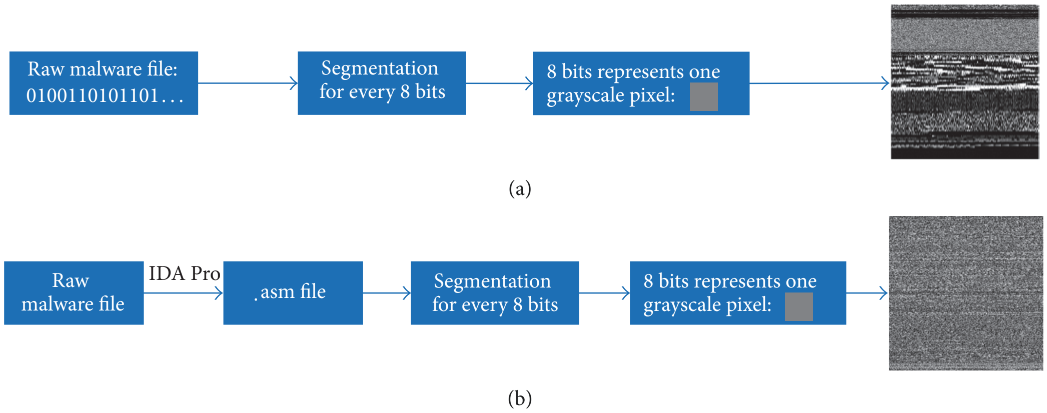

Raw malware file:
0100110101101...
Segmentation for every 8 bits
8 bits represents one grayscale pixel:
(a)
Raw malware file
IDA Pro
.asm file
Segmentation for every 8 bits
8 bits represents one grayscale pixel:
(b)

Figure 2: The generated grayscale images. (a) Grayscale images are generated from the raw binary file. (b) Grayscale images are generated from the decompiled file.

pooling layer mainly uses the principle of local correlation to complete the downsampling, so the subsequent convolution layer can extract features from a more global perspective. These greatly reduce the number of weight parameters and calculation for training a deep network.

3.1.3. Data Preprocessing for Grayscale Image. In order to meet CNN requirements for input data, we first preprocess the grayscale image. When CNN performs a task like image classification, it takes the input image data with the same sizes. Generally, the image data should have the same length and width (length to width ratio is 1 : 1). It is for the convenience of subsequent convolution operation. Since executive files have different file sizes, various grayscale image sizes also have big differences. In fact, a large grayscale image can reach 1.04 MB (2048 × 1036 pixels), while a small one is only 120 KB (512 × 472 pixels). So it is necessary to normalize all grayscale images.

We use bilinear interpolation algorithm, an image scaling method for normalization. It makes use of the four nearest pixels values in the original image to determine a virtual pixel value of the target image, which achieves better effect than the nearest neighbor interpolation. Also, the normalized size of grayscale image is a hyperparameter, which reflects the trade-off between classification accuracy and calculation cost. The larger the normalized image size, the richer the information received by the CNN input; then with more complex network structure better detection result will be obtained, but the corresponding cost is longer time-consuming for network training. To this end, we finally choose 64 × 64 as the normalized size of grayscale images.

3.2. Learning Opcode Sequence through LSTM. In this section, we introduce another important part of MalNet, which deals with opcode sequence with LSTM to learn malicious sequence features and patterns. Opcode sequences are extracted from decompiled files. These sequences actually reflect code logic and program execution logic of executive files. Hence, LSTM can mine malicious code sequence features corresponding to high-level malicious behavior from them.

3.2.1. Opcode Sequence Extraction. To learn from opcode sequence, first we need to extract opcode sequence from raw executive files. We decompile the executive file through IDA Pro which generates *.asm* format decompiled file. IDA Pro is a common decompilation and debugging tool that resolves malware into Intel x86 assembly instructions.

Then for the *.asm* file, we traverse all lines and slice sentences through space character as a delimiter to match each phrase to our predefined opcode set which contains all common Intel x86 assembly instructions. If the matching is successful, we retain the opcode; otherwise, we delete the phrase. During this process we find that there are a large number of duplicate opcode subsequences on decompiled files, such as $dd, dd, \ldots, dd$ or $db, db, db, \ldots, db$. So it is required to filter these duplicate subsequences by adding some rules. The pseudocode of our opcode sequence extraction algorithm is shown in Algorithm 1.

In the process of opcode sequence extraction, the size of the opcode set will affect the average length of opcode sequences. Larger opcode set will accept more kinds of opcodes. Since there are many noise data and too long opcode sequence will cause difficult learning problem with LSTM, we need to limit the size of the opcode set in a reasonable range so that it only contains the most valid information. So we treat all decompiled *.asm* files as text and instructions as vocabularies. Then we make frequency statistics and filter out the low frequency vocabularies. After that we use each vocabulary frequency as a feature and perform a classification by a random forests model, random forests can give a ranking for all features importance. We choose the vocabularies which give the best feature importance. Finally we get opcode set including 185 elements and extract opcode sequences with that. By now these opcode sequences should be digitized before being used as input of neural network; we use one-hot encoding which simply takes a mapping transformation to get a sparse vector like $[0, 0, 0, 1, 0, \ldots, 0]$ whose N binary status bits represent N states only containing one nonzero element. And each opcode gets a unique one-hot representation.

3.2.2. Very Long Sequence Learning by LSTM. We first briefly introduce LSTM network. As a deep neural network,

```
Input: Executive file
Output: Opcode sequence
files = get_files(); // Get all executive files;
for i in files;
    file = open(i.asm); // Open the corresponding IDA pro decompiled file;
    for line in file; // Read in line;
        words = line.split(" "); // Cut the line into phrases by space character;
            for word in words;
    //To judge each phrase, it requires to meet the following two points at the same time:
    (1) The current word belongs to opcode set opcode_set;
    (2) The last three words are not duplicated opcodes.
            if word in opcode_set and (word! = last_word and last_last_word! = last_word) :
            last_last_word = last_word;
            last_word = word;
            filter_words.add(word);
            end if
            end for
    end for
end for
```

Algorithm 1: Opcode sequence extraction algorithm for executive files.

Long-Short Term Memory (LSTM) [25] is widely used for processing time series data, which is an improved model based on Recurrent Neural Networks (RNNs). RNN uses an internal state to represent previous input values which allows it to capture temporal context. Based on this, LSTM uses the Constant Error Carousel (CEC) and well-designed "gate" structures to ease the vanishing gradient problem during errors backpropagation. So loss can flow backwards through longer timestep, which enables LSTM to learn long-term dependency and context. In brief, LSTM adds three gates (input gate, forget gate, and output gate) to decide and control the CEC state. Here MalNet uses LSTM network to learn from opcode sequence for malware detection.

However, one existing problem for LSTM is that it is difficult to effectively train when the input sequence is too long, though LSTM can capture longer time series context than RNN. In our work, if the size of executive file is very large, then the length of extracted opcode sequence is very long. For example, the average opcode sequence length of Ramnit malware family samples reaches 36,000. But the performance of LSTM is reduced fast when the length of input sequence exceeds 200. So how to process very long sequence with LSTM network is critical.

One simple strategy for very long sequences processing with LSTM network is Truncating And Padding (TAP). In particular, TAP first sets a fixed length N, truncates and discards the part of long sequences exceeding length N, and pads short sequences to length N with predefined identifier. It is convenient but it abandons a lot of information due to the truncation operation. Truncated Backpropagation Through Time (truncated BPTT) is another solution [26], which adds a time window constraint to limit the maximum distance for error backpropagation operation. So the error propagation and gradient calculation are only performed in the window, and the weights of nodes beyond the window are not updated. It enhances the computational efficiency by sacrificing a small part of the accuracy comparing with standard BPTT (or full BPTT) since standard BPTT calculation is less effective when backpropagation distance is too long. In addition, truncated BPTT is more suitable for online learning, as it can quickly adapt to the newly generated part of a very long sequence. Overall, truncated BPTT is reasonable and effective since it learns all sequence information comparing with TAP strategy.

In our scenario, we come up with a practical implement based on truncated BPTT algorithm for LSTM network. Since gradients are only propagated in the window, we first divide an opcode sequence into multiple subsequences, where the length of each subsequence equals the window length of truncated BPTT. Then for each subsequence we just do a full BPTT which equals doing the truncated BPTT for the whole sequence with no intersection window division. Most importantly, this allows LSTM to train in parallel on a bunch of subsequences. One of the major problems with LSTM is that the recurrent structure restricts it to train a sequence serially, which is inefficient. However, with these subsequences, our LSTM training process can be 3 times faster.

3.2.3. Subsequence Selection and Subsequence Fusion. With the above subsequence training strategy, subsequences become the input of LSTM network while the output is generated for each subsequence. It can be regarded as the discrimination result that LSTM thought how likely this subsequence is malicious. However, subsequences may contain a lot of noise especially for malware samples. Because, for many malware authors, they just implanted a malicious code snippet into a benignware, so many subsequences of this kind of sample are harmless. For this reason, it is crucial to take care of these subsequences in malware. Here we come up with a subsequence selection method for cleaning up

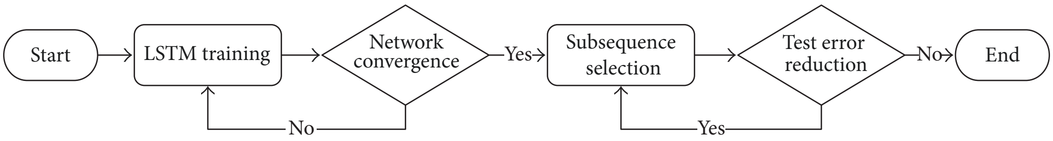


Figure 3: The process of subsequence selection in LSTM.

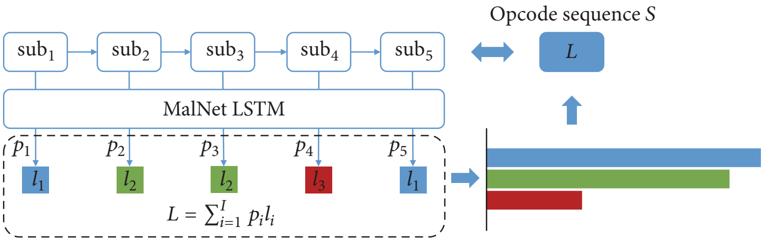


Figure 4: The process of subsequence fusion. Suppose that red, blue, and green represent three different prediction labels. By using relative majority weighted voting method, then final result follows the label l_i owning the most weighted votes.

these subsequences and providing a higher quality dataset to LSTM.

We utilize LSTM network itself without additional model to complete subsequence selection. In binary classification task (for malware detection), the output layer of LSTM network uses a logistic regression giving a probability value to identify negative class or positive class. This is similar to a recent work [27] using reconstruction error as the basis for subsequence anomaly detection by LSTM-based Encoder-Decoder. Specifically, the logistic regression layer takes the output of hidden layer and calculates the probability $p(y \mid x_j)$ of the current subsequence sample x_j. y is the corresponding labels where $y = 0$ presents negative class (benign label) and $y = 1$ presents positive class (malware label) and the actually logistic output can be presented as $p(y = 1 \mid x_j)$. The intuition is the higher or lower $p(y = 1 \mid x_j)$, the more confidence of LSTM for current sample. Since the benign subsequence in malware lacks malicious features but still with a malicious label, which will confuse LSTM, the confidence given by LSTM is relatively lower. Hence, we can use $p(y = 1 \mid x_j)$ to help subsequence selection.

In order to use the LSTM's output $p(y = 1 \mid x_j)$ to perform the subsequence selection for x_j, we set a threshold δ and compare it to the maximum likelihood probability. The formula is shown as follows:

$$\delta \geq \max \left\{ P\left(y = 1 \mid x_j\right), 1 - P\left(y = 1 \mid x_j\right) \right\}. \quad (1)$$

For current subsequence x_j, if the above formula holds, it means that LSTM sets a low confidence level that current subsequence belongs to any category, indicating that subsequence x_j cannot provide sufficient valid information for LSTM to judge or just with wrong label (benign subsequence of malware). So it can be seen as noisy subsequence and has been filtered out.

At the beginning of LSTM training process, since LSTM parameters are randomly initialized, the output of LSTM cannot be trusted so we use another threshold η to determine when the network has enough capacity to start subsequence selection after several iterations. Specifically we calculate the training error of a batch of inputs. If training errors of continuous M input batches are lower than the threshold η, then LSTM triggers subsequence selection, where M is a hyperparameter we set to 5. All these hyperparameters are chosen through experiment. Moreover, since training error will always decrease through subsequence selection, we divide a validation set and use validation error to find the right time to stop subsequence selection. Once validation error cannot reduce anymore, it is considered to be the appropriate time for ending subsequence selection (the whole process is seen in Figure 3).

By now LSTM output the classification results for subsequences; we need to use these results for completing the classification of original opcode sequence. We can use a simple fusion to solve it. One common fusion strategy is the voting method. In our scenario, we use the relative majority weighted voting method to fuse the discrimination result of subsequences.

$$H(x) = C_{\operatorname{argmax}_j} \sum_{i=1}^{T} w_i h_i^j(x), \quad (2)$$

where w_i represents the weight of subsequence discrimination result $h_i^j(x)$. Normally, $w_i \geq 0$ and $\sum_{i=1}^{T} w_i = 1$.

Considering that the classification output result of each subsequence is a nonnormalized probability given from the logistic regression layer of LSTM, here we can directly use these nonnormalized probabilities as weights w_i for each subsequence (see Figure 4).

Figure 4 shows how the weighted voting method is applied to subsequence fusion. Each of the original sequence is divided into subsequences, which are the input for trained LSTM network and get corresponding classification results. Finally, we use these subsequence results and corresponding weights w_i to get the prediction for the original opcode sequence.

3.2.4. Data Augmentation Strategy Based on Sliding Window. When dealing with classification tasks using neural networks, there is often a category imbalance problem in reality. For example, the number of benignware in reality is much more than malware and is easily accessible. While the distribution of different malware families is more uneven, those malware families which are widely spread have a bigger number of accessible samples and unpopular malware family only has few samples as dataset. In order to avoid these unpopular categories being ignored by the classifier resulting in poor recognition accuracy, data augmentation strategy is often used to solve category imbalance problem in the real world where oversamples on unpopular categories combined with negative sampling of popular categories to achieve a more balanced distribution of different category samples.

However, some data augmentation methods, such as mapping transformation (widely used in image data) and SMOTE algorithm (based on interpolation), are not suitable for sequence data. Here we propose a data augmentation strategy based on sliding window which is used for category imbalance problem on the subsequent malware family classification task. In the previous section, the way of subsequence segmentation for very long sequence has no intersections, which means there is no repetitive element between any two subsequences. For expanding data samples, we consider a segmentation method with intersection for very long sequence by using a sliding window. The window length is exactly equal to the window length of truncated BPTT, and then the number of generated subsequences is controlled by setting the step length of the sliding window.

So, the smaller the step length, the more the number of generated subsequences. To ensure that the original sequence of information is not lost, we add the constraint where the step length should be no more than the length of the sliding window. Suppose that the number of samples of the current category l_i is β_i; each sample length is α_{ij} where $j \in (0, 1, \ldots, \beta_i)$ and the length of the sliding window is set as τ. To expand the data sample for current class l_i to γ, we first calculate the total length of the sequence of category l_i, as follows:

$$\operatorname{len}(l_i) = \sum_{j=0}^{\beta_i} \alpha_{ij}. \tag{3}$$

For current category l_i, the step length d_i of the sliding window is calculated as follows:

$$d_i = \frac{\sum_{j=0}^{\beta_i} \alpha_{ij} - \tau}{\gamma}, \quad 1 \le d_i \le \tau. \tag{4}$$

Since the minimum length of step length is set to 1, there is a corresponding upper limit on the number of samples γ to be expanded:

$$\gamma \le \sum_{j=0}^{\beta_i} \alpha_{ij} - \tau. \tag{5}$$

In subsequent malware family classification task, this data augmentation strategy can achieve a relatively balanced distribution on the data sample numbers of different malware families.

3.3. Stacking Ensemble. MalNet also extracts some metadata features of the malware apart from using LSTM and CNN. The main reason is that LSTM and CNN capture local feature and metadata feature in contrast can get the global description for malware. And these metadata features are easy to obtain such as the size of malware source files, the starting address of the byte file, the size of decompiled file, number of rows, and the length of different PE segments.

Now MalNet has three parts of temp results which are LSTM discrimination result, CNN discrimination result, and metadata features. To achieve a final detection result, we integrate these three parts by stacking ensemble. A general procedure of a stacking ensemble method [28] involves a learner trained to combine other heterogeneous learners' results. Here learner usually means a machine learning model. The individual learners are called the first-level learners which gives a temp result, while the combiner is called the second-level learner to stack the output of first-level learners for making a better predictions. The basic idea is to train the first-level learners using the original training set and then use their output to generate a new dataset for training the second-level learner.

By now the above three parts are designed to obtain malware features from different perspectives, containing local level and global level. We use CNN network, LSTM network, and feature extraction as first-level learners and take a logistic regression as the second-level learner for stacking ensemble. The process of stacking ensemble can be seen in Figure 5.

And the objective function of logistic regression is defined as

$$P_\theta(x) = \frac{1}{1 + \exp(-\theta^T x)}, \tag{6}$$

where θ is the parameter. The range of $P_\theta(x)$ value is $(0, 1)$. For the training data $(x^{(j)}, y^{(j)} = i)$, the maximum likelihood probability is calculated and the loss is used to optimize θ through backpropagation algorithm. We use the Stochastic Gradient Decent (SGD) to train the logistic regression model as the second-level learner.

4. Experiments and Evaluations

In this section, we will describe the experiment environment and the concrete implementation of MalNet. To evaluate and optimize MalNet, we focus on four parts: performance of MalNet CNN, performance of MalNet LSTM, stacking ensemble result, and comparison with other works, respectively.

4.1. System Implementation. The core part of MalNet relies on deep neural networks. Table 1 summarizes the major hardware and software platforms environment for MalNet, which mainly contains a series of dependencies for CNN network and LSTM network.

Moreover, our datasets consist of 21,736 malware samples and 20,650 benignware samples. Herein, the malware dataset is provided by Microsoft [29], which contains 21,736 malware samples of 9 malware families on the Windows operating

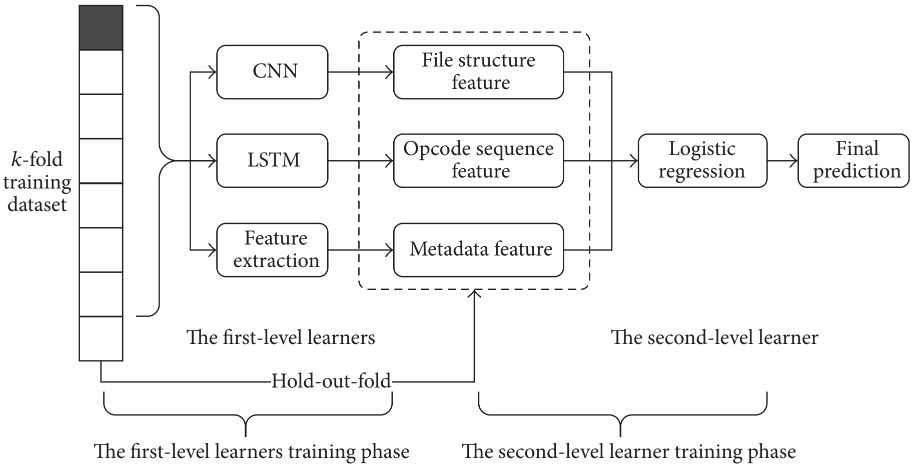


Figure 5: The learning process of stacking ensemble model.

Table 1: The platforms environment for MalNet.

Platforms	Content
Hardware dependencies	NVIDIA GPU Maxwell-based GTX980 16 GB of memory
Software dependencies	Ubuntu 14.04 LTS Python 2.7.6 Numpy 1.8.2 Scipy 0.13.3 Tensorflow 0.7.0 Theano 0.9.0.dev Lasagne 0.1 Nolearn 0.6.0.dev Scikitlearn 0.15.2 Pandas 0.15.2 Mahotas 1.2.4
GPU components	NVIDIA GPU driver CUDA 7.5 cuDNN V4

system taking more than 500 GB space. For each sample, two file formats are provided, the malware source files (binary stream files without the PE head) and the corresponding decompiled files by IDA Pro, respectively. And the benignware source files are collected by us from some software providers, such as Cnet [30] and Baidu Software Center [31].

In evaluation experiments, we measure the following performance metrics: accuracy, true positive rate (TPR), false positive rate (FPR), equal error rate (EER), and receiver operating characteristic (ROC). EER is the same as FPR value when operating threshold is adjusted such that FPR and false reject rate (FRR) become equal. The main metrics are defined as follows:

(i) TPR is defined as the probability that the current malicious sample is correctly identified:

$$\text{TPR} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}. \tag{7}$$

(ii) FPR is defined as the probability that the benign sample is wrongly identified as malware:

$$\text{FPR} = \frac{\text{False Positives}}{\text{False Positives} + \text{True Negatives}}. \tag{8}$$

Note that TPR reflects the usability of MalNet while FPR shows the security. Here, due to the bias for security considerations, we evaluate MalNet TPR value when FPR equals to 0.1%, which keeps a very low FPR as a prerequisite. In addition, to determine the detection efficiency of MalNet, we calculate the time consumptions on training phase and detecting phase.

4.1.1. MalNet CNN. We build two CNN network structures, called BaseNet and VGGNet. For BaseNet, we use the 5 × 5 convolution kernel and stride size 1. Also, to simplify the design of CNN network, the edges of the image are padded so that the output feature map size of the convolution operation is consistent with the input image. At the same time, BaseNet uses 2 × 2 max pooling operation and stride size 2, so the length and the width of sampled images are reduced to half of the original. BaseNet totally uses 2 convolution layers, 2 max-pooling layers, and 1 fully connected layer and adds Dropout layer [32] for each pooling layer and fully connected layer to prevent overfitting.

Since BaseNet uses a large convolution window, it cannot support deeper network structure when the input image size is small. To this end, we build VGGNet according to Simonyan and Zisserman work [33], which uses small window convolution filter to apply a deeper network. Specifically VGGNet uses 3 × 3 convolution kernel, stride size 1, and 1-pixel edge padding. Also, 3 × 3 max-pooling operation is used and stride size is set to 2. So it can deal with input images with larger size under the same size of feature map, only requiring a small amount of additional calculation. VGGNet has deeper network structure, a total of 3 convolution layers, 2 pooling layers, and 2 fully connected layers. Also, we use Leaky ReLU activation function [34], uniform distribution weight initialization, and batch normalization [35] which

Table 2: The list of two CNN network structures parameters.

	Network layer type	Size	Output dimension
BaseNet	Input layer	-	(1, 1, 64, 64)
	Convolutional Layer	32 5 × 5 Convolution kernel	(1, 32, 64, 64)
	Max pooling layer	2 × 2, stride 1	(1, 32, 32, 32)
	Dropout layer	-	(1, 32, 32, 32)
	Convolutional layer	64 5 × 5 Convolution kernel	(1, 64, 32, 32)
	Max pooling layer	2 × 2, stride 1	(1, 64, 16, 16)
	Dropout layer	-	(1, 64, 16, 16)
	Fully connected layer	Logistic regression	(1024, 1)
	Dropout layer	-	(1024, 1)
	Output layer	-	1
VGGNet	Input layer	-	(1, 1, 64, 64)
	Convolutional layer	32 3 × 3 Convolution kernel	(1, 32, 64, 64)
	Convolutional layer	16 3 × 3 Convolution kernel	(1, 16, 64, 64)
	Max pooling layer	2 × 2, stride 1	(1, 16, 32, 32)
	Dropout layer	-	(1, 16, 32, 32)
	Convolutional layer	32 3 × 3 Convolution kernel	(1, 32, 32, 32)
	Max pooling layer	2 × 2, stride 1	(1, 32, 16, 16)
	Dropout layer	-	(1, 32, 16, 16)
	Fully connected layer	512 maxout unit	(32, 512)
	Dropout layer	-	(32, 512)
	Fully connected layer	Logistic regression	(32, 1)
	Dropout layer	-	(32, 1)
	Output layer	-	1

enhance CNN network convergence performance. The specificity of BaseNet and VGGNet structures and parameters is concluded in Table 2.

4.1.2. MalNet LSTM. LSTM network consists of two layers, and each layer has 185 neuron nodes. We also use Dropout as regularization mechanism to reduce overfitting. We add Dropout layer for two hidden layers of LSTM and set the probability value ρ (the selection of ρ determines the intensity of Dropout) of 0.5. Besides, Dropout mechanism only takes effect in the training phase. It can be regarded that Dropout helps training many subnetworks during the training phase and the predicting phase; it combines all subnetworks to make an ensemble prediction.

Moreover, we use SGD and set the batch size (=30) for training. And Adam optimization algorithm [36] is used as the optimizer; it combines momentum factor with Adagrad optimization algorithm, which provides a fine-grained control of the learning rate decay. Adam optimizer only needs an initial learning rate (=$2e-3$) as a hyperparameter, and the learning rate during the training process can be adjusted adaptively without manually setting weight decay. The summary settings and parameters for LSTM networks are listed in Table 3.

Table 3: The list of LSTM network parameters.

Model parameters	LSTM
Maximum iterations	$6.40E+04$
Weights initialization	[−0.04, +0.04]
Truncated BPTT length	120
Batch training samples	30
Initial learning rate	$2.00E-03$
Dropout probability	0.5
Gradient regularization factor	10
Activation function	Tanh
Optimization algorithm	Adam
Propagation direction of time series	One-way
Hidden layers	2
Hidden nodes	185

4.2. Performance of MalNet CNN. The corresponding grayscale images of 21,736 malware samples and 20,650 benignware samples are first generated and normalized to the size of 64 × 64. And we use 6-fold cross validation to evaluate two CNN networks, BaseNet and VGGNet.

For the above two CNN network structures we do a comparative experiment to evaluate their detection performance. We use their output discriminant result for grayscale images as corresponding malware detection result. Apart from network structure, the rest of the basic settings are consistent for BaseNet and VGGNet, such as the Leaky ReLU activation function, Adam optimization algorithm, and the initial learning rate. Figures 6 and 7 represent ROC curves of validation result for two networks and the list of 6 groups of experiment results corresponding to 6-fold cross validation, respectively. The results show that VGGNet achieves an average AUC of 0.99952 and average

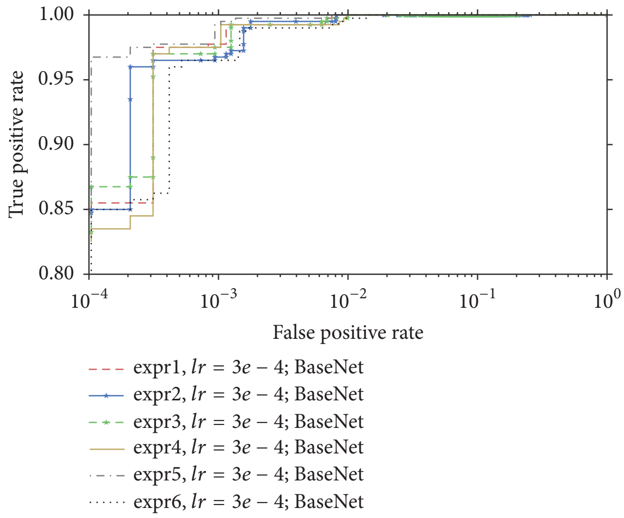


FIGURE 6: The ROC curve of BaseNet classification result.

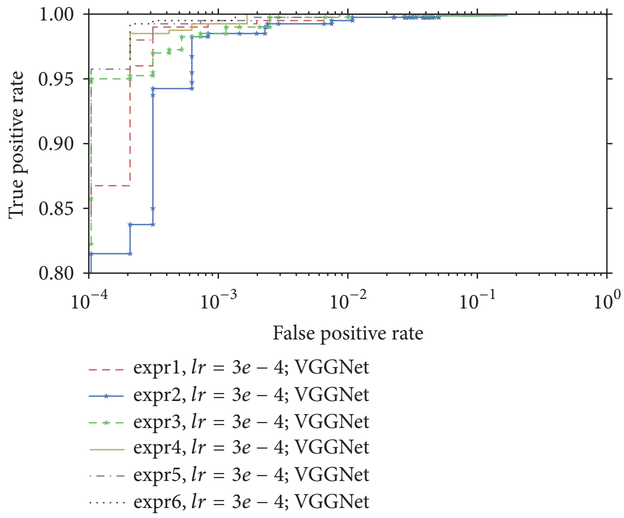


FIGURE 7: The ROC curve of VGGNet classification result.

detection accuracy of 98.14%, which are better than the average AUC of 0.99896 and average classification accuracy of 96.82% for BaseNet. Since the grayscale image contains a wealth of local slight information, we consider that VGGNet can achieve a better performance due to its deeper network structure which can capture more localized image association. And although deeper network often requires longer training time-consuming, VGGNet greatly reduces the number of nodes required for the fully connected layer by using maxout mechanism, resulting in no increase of network parameters and no decrease on training efficiency. Hence VGGNet is used as the CNN network part for MalNet.

4.3. Performance of MalNet LSTM. In data preprocessing phase, we first define an opcode set containing 185 candidate opcodes and use this to extract opcode sequences from the decompiled files. And then we divide all opcode sequences into multiple subsequences for subsequent LSTM training with truncated BPTT.

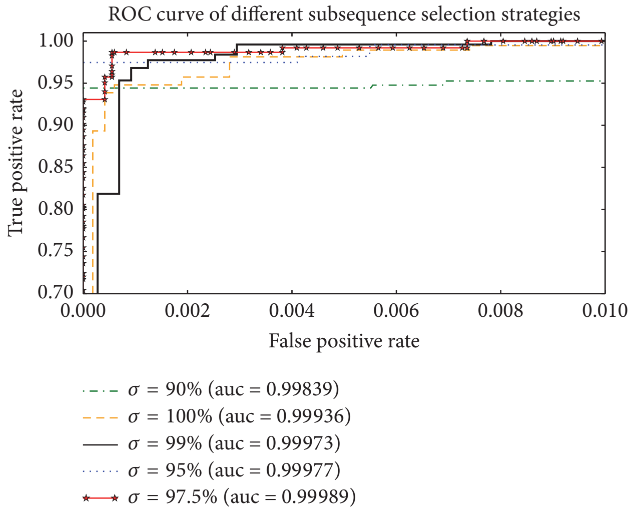


FIGURE 8: The classification results for different subsequence selection rates.

One important task is to evaluate and optimize our proposed subsequence selection strategy. The main function of subsequence selection is to filter noisy subsequences, especially the benign part of a malware to ensure high quality input data for LSTM network. Here a quantitative indicator for subsequence selection is required to identify the degree of tolerance for noise in opcode sequences. We define a hyperparameter called selection rate σ whose value refers to the proportion of training subsequences pass through subsequence selection. The choice for σ in fact reflects the trade-off between MalNet generalization performance and data quality. In our experiment, we compare with five different selection rates σ (=100%, 99%, 97.5%, 95%, 90%).

In Figure 8, we can see that the AUC value is the lowest when σ = 90% and the highest when σ = 97.5%. The AUC values of the rest subsequence selection rate are better than the benchmark results without subsequence selection (σ = 100%), which proves that subsequence selection strategy is effective. However, the detection performance declines rapidly when σ decreases to 90%. We think the reason is that LSTM network usually has a certain antinoise ability, which means the input data with certain noise (not too much) only has little interference on the discriminant result of LSTM and LSTM can automatically perceive and resist noise from the data. In contrast, filtering out too many subsequences will cause LSTM to lack enough data for learning. So we choose σ = 97.5% as optimized hyperparameter for our subsequence selection strategy.

In the later experiment, we further compare the detection performance of TAP strategy and truncated BPTT on different BPTT length α, where TAP strategy just truncates the part of sequence exceeding α before training and truncated BPTT divides opcode sequence into subsequence with length α for training. The experiment result (seen in Figure 9) shows that truncated BPTT is superior to TAP strategy in

Table 4: The experiment results for different LSTM networks.

Strategy	Models	Accuracy (%)	AUC	TPR (FPR = 0.1%)	EER (%)	Training times (h)
TAP	LSTM ($\alpha = 30$)	71.43	0.8863	52.03	-	**0.41**
	LSTM ($\alpha = 60$)	87.13	0.9791	81.67	6.45	0.54
	LSTM ($\alpha = 80$)	91.56	0.9854	85.39	4.88	0.89
	LSTM ($\alpha = 120$)	94.86	0.9931	91.53	3.17	-
Truncated BPTT	LSTM ($\alpha = 30$)	94.37	0.9928	91.11	3.10	-
	LSTM ($\alpha = 60$)	96.83	0.9950	93.37	-	-
	LSTM ($\alpha = 80$)	98.08	0.9989	95.13	-	1.24
	LSTM ($\alpha = 120$)	98.47	0.9993	96.81	-	1.36
	LSTM ($\alpha = 180$)	97.82	0.9987	95.42	-	1.53
Truncated BPTT + subsequence selection	LSTM ($\alpha = 120, \sigma = 90\%$)	97.98	0.9988	95.01	1.34	1.16
	LSTM ($\alpha = 120, \sigma = 95\%$)	98.83	0.9997	97.22	0.84	-
	LSTM ($\alpha = 120, \sigma = 99\%$)	98.66	0.9996	96.69	0.92	-
	LSTM ($\alpha = 120, \sigma = 97.5\%$)	**99.13**	**0.9999**	**98.69**	**0.54**	**1.34**

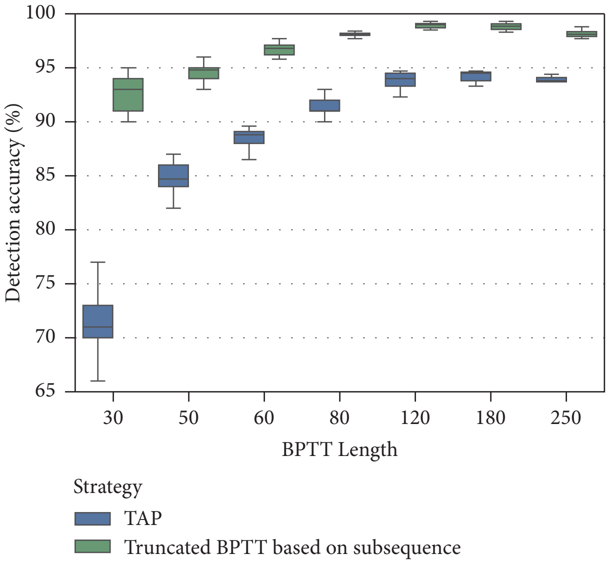


Figure 9: The detection results for different strategies in different BPTT lengths.

all length α. For a small BPTT length α, TAP strategy loses a large amount of sequence information due to truncating operation and this defect can be compensated when α gets larger. But we can see when α is more than 120, the performance of TAP strategy can no longer grow up due to the gradient vanishing. In contrast, truncated BPTT does not discard any sequence segment but splits sequence into subsequences, training and predicting them, respectively, and then makes a fusion prediction. So even if α is small, it can still get considerable detection performance. But we can see its performance only gets slight improvement when α increases and even begins to decrease when α is over 180; this is because when α is too large, truncated BPTT on each subsequence will encounter gradient vanishing problem and in this case the subsequence should be further divided and trained separately. So we finally choose $\alpha = 120$ for our MalNet LSTM.

Finally, we conclude all experiments about LSTM network in Table 4. The optimal LSTM network setting for MalNet is containing 2 hidden layers of 185 neurons nodes combined with truncated BPTT and subsequence selection strategies, where the truncated BPTT length α is 120 and subsequence selection rate σ is 97.5%. The training process for this takes 1.34 hours with GPU and the detection result achieves 99.13% accuracy and TPR reaches to 98.69% when FPR is 0.1%.

4.4. Stacking Ensemble Result. MalNet fuses CNN network and LSTM network results by stacking ensemble. Concretely, stacking ensemble combines three parts (CNN discriminant result, LSTM discriminant result, and metadata features) to train a second-level learner, which is a logistic regression classifier.

Besides, we reproduce an N-gram based malware detection method according to relevant studies [15, 20, 21] and use it as a baseline in our dataset. It extracts 1-gram, 2-gram, and 3-gram features on both bytes and opcodes, discards low frequency features, makes a further feature selection according to the feature importance from random forest, and finally constructs a 12,834-dimensional feature vector for each sample and trains a SVM classifier for malware detection.

We make a malware detection experiment using 21,736 malware samples and 20,650 benignware samples. After training all models to make a prediction whether it is a malware on test set containing 2,000 malware samples and 2,000 benignware samples, the comparison results are seen in Table 5 (MF represents metadata feature).

4.5. Comparison with Other Works. Due to the sensitivity of malware data, many datasets used by other related works are not public, which increased the difficulty for comparison of different malware detection methods. Fortunately, our paper

Table 5: The malware detection experiment results for different methods.

Models	Accuracy (%)	AUC	TPR (%)	FPR (%)	EER (%)	Training time (h)	Detection time (ms)
N-gram	93.21	0.9864	89.22	0.1	3.94	4.18	2304
LSTM	99.13	**0.9999**	98.69	0.1	0.54	**1.34**	**13.62**
CNN	98.14	0.9989	96.92	0.1	1.55	1.46	15.97
LSTM + CNN + MF (MalNet)	**99.88**	**0.9999**	**99.14**	0.1	**0.37**	2.91	30.33

Table 6: The comparison with other works.

Methods	Accuracy (%)	Training time (h)	Detection time (s)
Kaggle Winner Solution [37]	**99.83**	72	13.33
Novel Features [38]	99.77	21.86	4
Linear kNN [39]	96.6	-	-
Random Forest [40]	95.62	-	-
One-class SVM [41]	92	-	-
tGAN [42]	96.39	-	-
Strand Gene Sequence [43]	98.59	**0.75**	0.3
MalNet	99.36	2.91	**0.03**

uses a publicly available malware dataset from Microsoft released in 2015. This dataset is for a Kaggle competition and so far some works have done their experiments on this dataset, making it easy to do a convincing comparison for malware detection. Since this dataset was originally designed for malware family classification task, here we need to make minor changes to MalNet to complete malware family classification. Specifically, since the only difference is that malware detection is a binary classification problem and malware family classification is a multiclass classification problem, we simply change MalNet's output layer from logistic regressions to softmax regressions. On the other hand, Microsoft provided nine malware families (Ramnit, Lollipop, Kelihos_ver3, Vundo, Simda, Tracur, Kelihos_ver1, Obfuscator. ACY, and Gatak) in this dataset. However, the samples of different malware families are unevenly distributed. One malware family has fewer than 100 samples, while the largest one contains nearly 6,000 samples. Such unbalanced sample size will have a negative impact on the multiclass classification results, so we use the data augmentation strategy of LSTM mentioned earlier to mitigate category imbalance problem.

For CNN this grayscale image does not have rotational invariance, since the pixels of the grayscale image are extracted from the binary stream line by line, and the rotate transformation would break these textures. Similarly, grayscale image does not support tilt transformation of a certain angle. Hence we customize some mapping transformations for data augmentation including horizontal rollover, horizontal shift (randomly shift −10 to 10 pixels to the right), and longitudinal stretch (randomly cut into 3 to N parts with vertical, each part stretches in ratio with 1/1.3 to 1.3).

So, we conduct the same malware family classification experiment with MalNet using the same Microsoft malware dataset as other related works; the results are summarized as shown in Table 6. It can be seen that MalNet achieves 99.36% classification accuracy and outperforms most of related works, which closes to Kaggle Winner Solution [37] with 99.83% accuracy. It is noteworthy that although there are two approaches [37, 38] having slightly better detection accuracy over MalNet, both approaches rely on a large number of feature engineering works. And the biggest problem with this is the potential inefficiency performance on both training and detecting phase from the experiment results they claimed.

As we can see Kaggle Winner Solution takes almost 3 days to train their model with a relatively good hardware environment including Google Compute Engine with 16 CPUs, 104 GB RAM, and 1 TB of disk space. And the authors claimed that the real model training time is only 1 hour and the remaining time is fully used for feature engineering. This is because they extracted massive features (around 70 K original features) to make up for the lack of expert knowledge in this area. The work of Novel Feature [38] performs feature engineering more effectively through a certain expert knowledge and avoids huge time-consuming calculation like 3-gram and 4-gram feature extraction. However, it still takes approximately 1 day to extract features from the training data. More seriously, this has a greater impact on the efficiency of the detection phase. Novel Features take about 4 seconds to predict a sample, while Kaggle Winner Solution takes around 13 seconds. These seem difficult to satisfy the efficiency for a real antivirus scenario.

On detecting phase, the model prediction is often very fast and makes the slowness of feature extraction more obvious. MalNet is able to detect one sample in 0.03 seconds (note that the computation time for decompiling is not taken into account as this step is required by all methods), which is faster than other related works, especially comparing with Novel Features and Kaggle Winner Solution. This is because the most time-consuming part for deep neural network is in the training phase in which backpropagation needs to calculate plenty of gradients to complete the training of the mass parameters. However, the prediction phase only needs one forward propagation and it is even more efficient with GPU acceleration. Actually, since the training phase is offline,

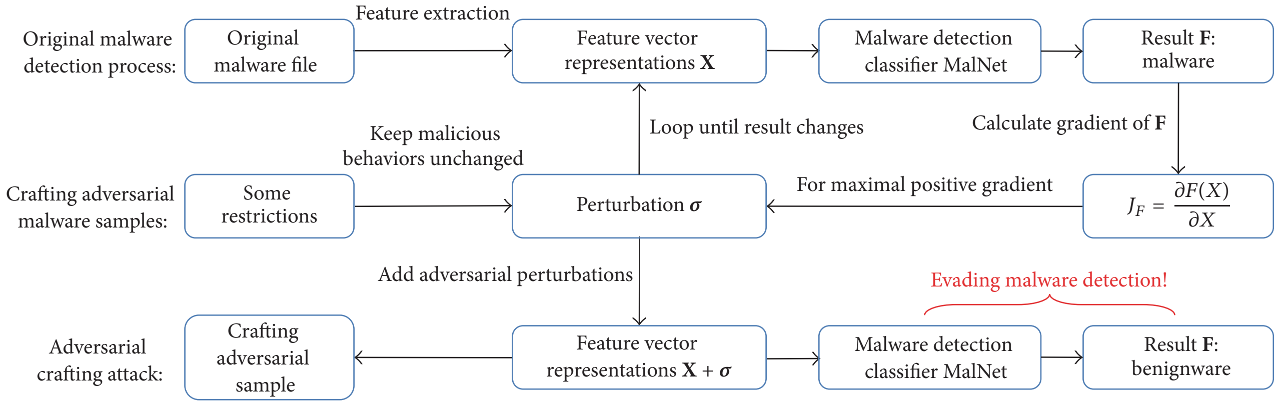

Original malware detection process:
Original malware file
Feature extraction
Feature vector representations **X**
Malware detection classifier MalNet
Result **F**: malware
Calculate gradient of **F**
$J_F = \frac{\partial F(X)}{\partial X}$
For maximal positive gradient
Loop until result changes
Keep malicious behaviors unchanged
Crafting adversarial malware samples:
Some restrictions
Perturbation σ
Add adversarial perturbations
Evading malware detection!
Adversarial crafting attack:
Crafting adversarial sample
Feature vector representations $\mathbf{X} + \sigma$
Malware detection classifier MalNet
Result **F**: benignware

Figure 10: The adversarial crafting attack on malware detection.

the key factor is detection efficiency rather than training efficiency. Nowadays, the emergence of massive amounts of malware per day is a challenge to the malware detection efficiency. Therefore, although MalNet is slightly behind the best detection accuracy, its detection efficiency gets greatly enhanced in return, more suitable for the real-life application scenarios.

5. Discussion

Some recent works try to evade the detection of machine learning based malware classifiers by adversarial learning [44, 45]. Their experiments show that it is possible to generate adversarial samples based on a trained machine learning classifier. The core of adversarial sample crafting is to find a small perturbation σ on feature vectors X of original malware sample to change the classification results F to benign. Formally, they compute the gradient of F with respect to X to estimate the direction in which a perturbation σ in X would maximally change F's output. The basic idea is shown in Figure 10.

This attack scene is mainly caused by the characteristics of discriminative model and lacking of sufficient data. When dealing with a classification task with discriminative model, since it is almost impossible to have enough data to help model make decision in whole feature space, discriminative model will try to expand the distance between samples and decision boundary for better classification result and, meanwhile, expand the area of each category in feature space. The benefit of this is to make the classification easier, but the downside is that it also includes a lot of feature spaces that do not clearly belong to current category, which enables attackers generating adversarial samples from this feature space.

The earliest work of this topic came from Nguyen et al. [46] which found that a slight change in the image could trick the image classifier, and then it has been introduced into the security area in recent years to attack security systems that rely on machine learning model. According to the conclusions of some of these related works, we make several changes to MalNet to prevent adversarial crafting attacks as much as possible. First is to add regularization so that the model does not get too overfitting to the training set and promote enclosure of the feature space of benign category. Here we add $L2$ regularization to MalNet which keeps a conservative discrimination result to unknown feature space to prevent adversarial samples using these feature spaces fooling malware detection classifier. Second, we try adversarial training which crafts adversarial samples in advance and let MalNet train these samples by online learning which enhances MalNet from resisting adversarial crafting attack. Third, as Biggio et al. [45] discovered that ensemble learning with different classifiers can generate a more robust classifier for adversarial crafting attack, here we use a stacking ensemble for MalNet. Fourth, Biggio et al. believed some of features are not easily evaded, such as N-gram; here we use LSTM to mine features like N-gram from raw opcode sequences. Finally we take the idea of "gradient masking" [47] in our real system, which let model output hard decision (the predicted target category) rather than probabilities of different categories, so it is hard for the attackers to obtain a useful gradient to build adversarial samples (a minor change cannot affect the output result).

Nevertheless, there is no detailed analysis in this paper of whether MalNet is susceptible to adversarial crafting attacks or a quantitative assessment of the effects of above changes we take. Besides, some defenses to adversarial attack are claimed not robust enough in last few years [48, 49] and other methods came up [50]. In general, adversarial crafting attack is a big, important, and popular topic; we have not given a complete analysis to MalNet for adversarial attack, and in the future we will consider conducting some exploration and detailed analysis with relevant experiments for evaluation.

6. Conclusion

In this paper, we propose a malware detection method called MalNet, which uses two deep neural networks CNN and LSTM to, respectively, learn from grayscale image and opcode sequence extracted from raw binary executive files, followed by a stacking ensemble to fuse them. We use MalNet to complete a malware detection experiment for 42,386 samples (1/10 samples for validation) and it achieves 99.88% accuracy and 99.14% of TPR with FPR of 0.1%. We also make a malware family classification experiment for comparison to other related works, and MalNet outperforms most of other works with 99.36% accuracy and raises detecting efficiency a lot comparing with two state-of-the-art results on Microsoft malware dataset.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This work is partially supported by the National Natural Science Foundation of China under Grant no. 61672421.

References

[1] "Kaspersky Security Bulletin 2016. Overall statistics for 2016," https://securelist.com/kaspersky-security-bulletin-2016-executive-summary/76858/.

[2] "McAfee Labs Threats Report in June 2017," https://www.mcafee.com/us/resources/reports/rp-quarterly-threats-jun-2017.pdf.

[3] D. K. S. Reddy and A. K. Pujari, "N-gram analysis for computer virus detection," *Journal of Computer Virology and Hacking Techniques*, vol. 2, no. 3, pp. 231–239, 2006.

[4] M. Narouei, M. Ahmadi, G. Giacinto, H. Takabi, and A. Sami, "DLLMiner: Structural mining for malware detection," *Security and Communication Networks*, vol. 8, no. 18, pp. 3311–3322, 2015.

[5] G. Willems, T. Holz, and F. Freiling, "Toward automated dynamic malware analysis using CWSandbox," *IEEE Security and Privacy*, vol. 5, no. 2, pp. 32–39, 2007.

[6] K. Rieck, P. Trinius, C. Willems, and T. Holz, "Automatic analysis of malware behavior using machine learning," Technical Report, University of Mannheim, 2009.

[7] M. Zakeri, F. Faraji Daneshgar, and M. Abbaspour, "A static heuristic approach to detecting malware targets," *Security and Communication Networks*, vol. 8, no. 17, pp. 3015–3027, 2015.

[8] A. Moser, C. Kruegel, and E. Kirda, "Limits of static analysis for malware detection," in *Proceedings of the 23rd Annual Computer Security Applications Conference (ACSAC '07)*, pp. 421–430, December 2007.

[9] C. Wressnegger, K. Freeman, F. Yamaguchi, and K. Rieck, "Automatically inferring malware signatures for anti-virus assisted attacks," in *Proceedings of the 2017 ACM Asia Conference on Computer and Communications Security, (ASIA CCS '17)*, pp. 587–598, UAE, April 2017.

[10] L. Martignoni, E. Stinson, M. Fredrikson, S. Jha, and J. Mitchell, "A layered architecture for detecting malicious behaviors," in *Proceeding of the International Symposium on Recent Advances in Intrusion Detection (RAID '08)*, 2008.

[11] W. Lee and S. J. Stolfo, "A framework for constructing features and models for intrusion detection systems," *ACM Transactions on Information and System Security*, vol. 3, no. 4, pp. 227–261, 2000.

[12] P. Li, L. Liu, D. Gao, and M. K. Reiter, "On challenges in evaluating malware clustering," in *Proceedings of the International Symposium on Recent Advances in Intrusion Detection (RAID '10)*, vol. 6307, 2010.

[13] M. G. Schultz, E. Eskin, E. Zadok, and S. J. Stolfo, "Data mining methods for detection of new malicious executables," in *Proceedings of the IEEE Symposium on Security and Privacy (S & P)*, pp. 38–49, May 2001.

[14] W. Cohen, "Fast effective rule induction," in *Proceeding of the 12th International Conference on Machine Learning*, 1995.

[15] J. Z. Kolter and M. A. Maloof, "Learning to detect and classify malicious executables in the wild," *Journal of Machine Learning Research*, vol. 7, pp. 2721–2744, 2004.

[16] J. Saxe and K. Berlin, "Deep neural network based malware detection using two dimensional binary program features," in *Proceedings of the 10th International Conference on Malicious and Unwanted Software, (MALWARE '15)*, pp. 11–20, USA, October 2015.

[17] 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.

[18] L. Nataraj, V. Yegneswaran, P. Porras, and J. Zhang, "A comparative assessment of malware classification using binary texture analysis and dynamic analysis," in *Proceedings of the 4th ACM Workshop on Security and Artificial Intelligence (AISec '11)*, pp. 21–30, 2011.

[19] D. Kong and G. Yan, "Discriminant malware distance learning on structural information for automated malware classification," in *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, (KDD '13)*, pp. 1357–1365, USA, August 2013.

[20] I. Santos, Y. K. Penya, J. Devesa, and P. G. Bringas, "N-grams-based file signatures for malware detection," in *Proceedings of the ICEIS 2009 - 11th International Confeence on Enterprise Information Systems*, pp. 317–320, May 2009.

[21] A. Shabtai, R. Moskovitch, C. Feher, and etal., "Detecting unknown malicious code by applying classification techniques on opcode patterns," *Security Informatics*, vol. 1, no. 1, 2012.

[22] I. Santos, J. Devesa, F. Brezo, J. Nieves, and P. G. Bringas, "OPEM: A static-dynamic approach for machine-learning-based malware detection," in *Proceedings of the International Joint Conference CISIS'12-ICEUTE12-SOCO12 Special Sessions, 2013*, vol. 189, pp. 271–280.

[23] IDA Pro., http://www.hexrays.com/products/ida/support/download_freeware.shtml.

[24] A. Graves and J. Schmidhuber, "Framewise phoneme classification with bidirectional LSTM and other neural network architectures," *Neural Networks*, vol. 18, no. 5-6, pp. 602–610, 2005.

[25] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[26] R. J. Williams and J. Peng, "An efficient gradient-based algorithm for on-line training of recurrent network trajectories," *Neural Computation*, vol. 2, no. 4, pp. 490–501, 1990.

[27] P. Malhotra, A. Ramakrishnan, and G. Anand, "LSTM-based encoder-decoder for multi-sensor anomaly detection," https://arxiv.org/abs/1607.00148.

[28] Z.-H. Zhou, *Ensemble methods foundations and algorithms. Machine Learning & Pattern Recognition*, Taylor & Francis, London, UK, 2012.

[29] Microsoft Malware, https://www.kaggle.com/c/malware-classification.

[30] PC. Windows Software, http://download.cnet.com/windows/.

[31] Baidu Software Center, http://rj.baidu.com/.

[32] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," https://arxiv.org/abs/1207.0580.

[33] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *Proceedings of the 3rd International Conference for Learning Representations (ICLR '15)*, 2015.

[34] A. L. Maas, A. Y. Hannun, and A. Y. Ng, "Rectifier nonlinearities, improve neural network acoustic models," in *In Proceeding of the 30th International Conference on Machine Learning (ICML '13)*, 2013.

[35] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proceedings of the 32nd International Conference on Machine Learning (ICML '15)*, pp. 448–456, July 2015.

[36] D. Kingma and J. B. Adam, "A method for stochastic optimization," in *Proceedings of the 3rd International Conference for Learning Representations (ICLR '2015)*, 2015.

[37] L. Wang, "Microsoft Malware Classification Challenge (BIG 2015) First Place Team: Say No To Overfitting," https://github.com/xiaozhouwang/kaggle_Microsoft_Malware/blob/master/Saynotooverfitting.pdf.

[38] M. Ahmadi, D. Ulyanov, S. Semenov, M. Trofimov, and G. Giacinto, "Novel feature extraction, selection and fusion for effective malware family classification," in *Proceedings of the 6th ACM Conference on Data and Application Security and Privacy, (CODASPY '16)*, pp. 183–194, USA, March 2016.

[39] B. N. Narayanan, O. Djaneye-Boundjou, and T. M. Kebede, "Performance analysis of machine learning and pattern recognition algorithms for Malware classification," in *Proceedings of the 2016 IEEE National Aerospace and Electronics Conference and Ohio Innovation Summit, (NAECON-OIS '16)*, pp. 338–342, USA, July 2016.

[40] F. C. Garcia and F. P. Muga, "Random forest for malware classification," https://arxiv.org/abs/1609.07770.

[41] E. Burnaev and D. Smolyakov, "One-class SVM with privileged information and its application to malware detection," https://arxiv.org/abs/1609.08039.

[42] J. Kim, S. Bu, and S. Cho, "Malware detection using deep transferred generative adversarial networks," in *Proceedings of the International Conference on Neural Information Processing (ICONIP)*, 2017.

[43] J. Drew, M. Hahsler, and T. Moore, "Polymorphic malware detection using sequence classification methods and ensembles: BioSTAR 2016 Recommended Submission - EURASIP Journal on Information Security," *EURASIP Journal on Information Security*, vol. 2017, no. 1, article no. 2, 2017.

[44] K. Grosse, N. Papernot, P. Manoharan, M. Backes, and P. McDaniel, "Adversarial perturbations against deep neural networks for malware classification," https://arxiv.org/abs/1606.04435.

[45] B. Biggio, I. Corona, D. Maiorca et al., "Evasion attacks against machine learning at test time," in *Proceedings of the Joint European Conference on Machine Learning and Knowledge Discovery in Databases (ECML PKDD)*, 2013.

[46] A. Nguyen, J. Yosinski, and J. Clune, "Deep neural networks are easily fooled: High confidence predictions for unrecognizable images," https://arxiv.org/abs/1412.1897.

[47] N. Papernot, P. McDaniel, I. Goodfellow, S. Jha, Z. B. Celik, and A. Swami, "Practical black-box attacks against machine learning," in *Proceedings of the 2017 ACM Asia Conference on Computer and Communications Security, (ASIA CCS '17)*, pp. 506–519, April 2017.

[48] N. Carlini and D. Wagner, "Defensive distillation is not robust to adversarial examples," https://arxiv.org/abs/1607.04311.

[49] N. Carlini and D. Wagner, "Adversarial examples are not easily detected: Bypassing ten detection methods," https://arxiv.org/abs/1705.07263.

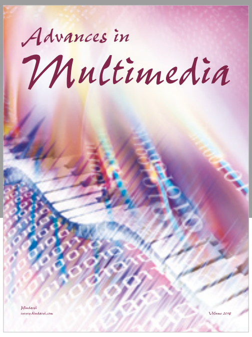
[50] F. Liao, M. Liang, and Y. Dong, "Defense against adversarial attacks using high-level representation guided denoiser," https://arxiv.org/abs/1712.02976.

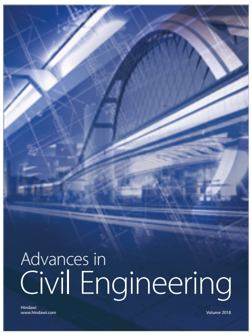
Journal of
Engineering

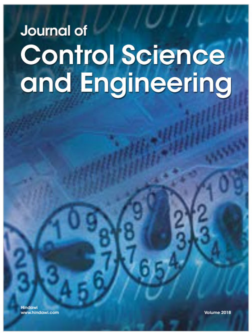
The Scientific World Journal

International Journal of
Rotating Machinery

Journal of
Sensors

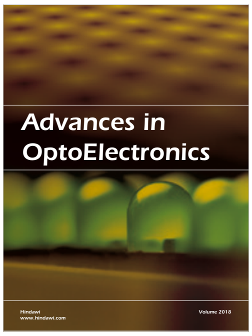

Advances in
Multimedia


Advances in
Civil Engineering


Journal of
Control Science and Engineering

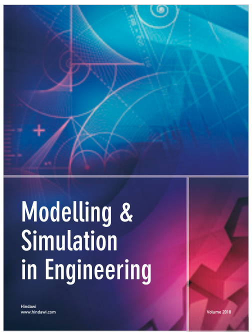
Journal of
Robotics

Journal of
Electrical and Computer Engineering

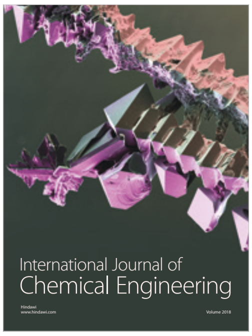

Advances in
OptoElectronics

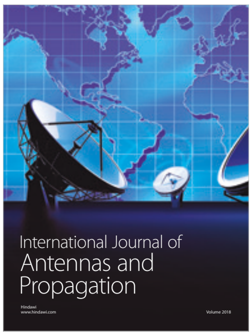
VLSI Design

International Journal of
Navigation and Observation

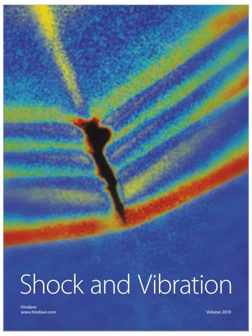

Modelling & Simulation in Engineering

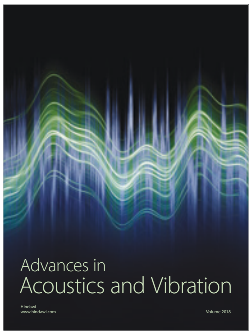
International Journal of
Aerospace Engineering


International Journal of
Chemical Engineering


International Journal of
Antennas and Propagation

Active and Passive
Electronic Components


Shock and Vibration


Advances in
Acoustics and Vibration