Android malicious code Classification using Deep Belief Network

Luo Shiqi[1], Tian Shengwei[1], Yu Long[2] Yu Jiong[1,3] and Sun Hua[1]
[1] School of Software, Xinjiang University
No.499, Xibei Road, Saybagh District, Urumqi, Xinjiang, 830008 - P.R. China
[e-mail: tianshengwei@163.com]
[2] Network Center, Xinjiang University
No.666, Shengli Road, Tianshan District, Urumqi, Xinjiang 830046 - P.R. China
[e-mail: yul_xju@163.com]
[3] School of Information Science and Engineering, Xinjiang University
No.666, Shengli Road, Tianshan District, Urumqi, Xinjiang 830046 - P.R. China
[e-mail: yujiong@xju.edu.cn]
*Corresponding author: Tian Shengwei

Received May 6, 2017; revised August 17, 2017; accepted September 15, 2017; published January 31, 2018

Abstract

This paper presents a novel Android malware classification model planned to classify and categorize Android malicious code at Drebin dataset. The amount of malicious mobile application targeting Android based smartphones has increased rapidly. In this paper, Restricted Boltzmann Machine and Deep Belief Network are used to classify malware into families of Android application. A texture-fingerprint based approach is proposed to extract or detect the feature of malware content. A malware has a unique "image texture" in feature spatial relations. The method uses information on texture image extracted from malicious or benign code, which are mapped to uncompressed gray-scale according to the texture image-based approach. By studying and extracting the implicit features of the API call from a large number of training samples, we get the original dynamic activity features sets. In order to improve the accuracy of classification algorithm on the features selection, on the basis of which, it combines the implicit features of the texture image and API call in malicious code, to train Restricted Boltzmann Machine and Back Propagation. In an evaluation with different malware and benign samples, the experimental results suggest that the usability of this method---using Deep Belief Network to classify Android malware by their texture images and API calls, it detects more than 94% of the malware with few false alarms. Which is higher than shallow machine learning algorithm clearly.

Keywords: malware classification, texture image, uncompressed gray-scale, Deep Belief Network

This research was partially supported by the Xinjiang Uygur Autonomous Region Science and Technology Personnel Training Project (QN2016YX0051). Research Innovation Project of Graduate Student in Xinjiang Uygur Autonomous Region(No. XJGRI2017007). The Project Of Cernet Next Generation Internet Technology Innovation Project(NGII20170420).

http://doi.org/10.3837/tiis.2018.01.022 **ISSN : 1976-7277**

1. Introduction

Android malware has become one of the major threat to network security. Analysis and detection of Android malicious code has been a vivid area of research in the recently year. Malware is a catch-all phrase used to refer to any program that is designed to "harm or subvert a system's intended functionality" [1] and falls under several categories including viruses, worms, and Trojans. At the same time, being driven by the economic benefits and applied to various new technology, the number of malicious code is growing exponentially, varieties of malware become the obstacle to the healthy development in the Internet.

In consequence, for the purpose of malware detection, various approaches have been proposed, and all the different research ventures can be categorized as static [2], dynamic [3] features analysis, graphs-based approach etc.

Dynamic(also name behavioral analysis), which can monitor the behavior of applications at run-time. It performed by observing the behavior of the malware while it is actually running on a host system. Such as, TaintDroid [4], DroidRanger [5] and DroidScope [6] are dynamic analysis method. But these algorithms spot whole malicious activities on the smart phone, which involve millions of smartphones at large scale in practice and takes a lot of spends and time.

Currently, static analysis is the conventional technique for malware detection ,which exhaustively examines all data flows and pinpoints problematic ones. The static analysis examines the binary code, analyzes all possible execution paths, and identifies malicious code without execution. Such as Kirin [7], Stowaway [8] and RiskRanker [9],are static analysis methods. In detail, Kirin [7] checks the permission of application for indications of malicious code activity. Stowaway [8] analyzes API calls to detect overprivileged applications and RiskRanker [9] identifies applications with different security risks statically. For characterizing Android applications' behaviors, static analysis takes into consideration some static information including permissions, deployment of components, intent message passing and Application Programming Interface (API) calls. There are many static analysis methods available for the examination of malware, such as the signature based [10], semantics-based [11], heuristic scanning technique [12] et al.

The signature based detection is a widely used method in static analysis. According to this method, the binary executables are transformed to represent hashes which are matched with a database of known malware samples [13] [14] [15] [16] [17], but it shows following weaknesses. The signature method requires continuous updates of signature and high maintenance cost. In addition, such method could be easily evaded by malware in polymorphic form. To avoid difficulties and complement to signature-based detection, heuristic-based scanning has been put forward. Heuristic-based scanning is generally based on sequences time-delay embedding approach [18], for threats with multiple anti-malware engines are needed to more effectively address modern risks. Usually static analysis methods induce only a small run-time overhead.

Although these approaches mentioned above, mainly built on manually crafted detection patterns, are efficient and scalable, generally speaking, they are not available for new malware instances. More and more malwares adopt measures such as: shell protection [19], polymorphism [20], encryption [21], or packing [22], which make the analysis and detection of Android malware rather tough.

The graphs-based approach includes Automated Behavioral Graph Matching [23], control-flow graphs [24], data dependency graphs [25], [26], permission event graphs [27],. But these graphs are checked against manually-crafted specifications to detect malware.

However, these detectors tend to seek an exact match for a given specification and therefore can potentially be evaded by polymorphic variants. Furthermore, the specifications used for detection are produced from known malware families and cannot be used to battle zero-day malware.

In classification system, the results of classification largely depend on the learned features. Therefore, it shows great significance in extracting the effective features for better detection or analysis. To overcome these limitations, solving the drawbacks above, many scholars use machine learning method to extract features. But most shallow machine learning algorithm, such as KNN [28], SVM [29], Naive Bayes [30], Random Forest [31], cannot improve accuracy greatly. But the shallow structure algorithm show the following weakness, which lies in limited samples, and its generalization ability for complex classification problems under certain constraints.

Deep learning (DL) is a recently developed field belonging to machine learning. It tries to mimic the human brain, which is capable of processing the complex input data fast, learning different knowledge intellectually, and solving different kinds of complicated human intelligence tasks well. It is widely applied to apply to image processing [32] [33] [34] et. DL is able to attaining distributed representation for input data by learning a deep nonlinear network structure. The last but not least, it shows strong concentration of the general learning essential characteristics of data sets from a few samples.

In computer vision processing area, the method of gray-scale image texture is widely use, for examples, detecting lung nodules with image texture features [35], texture classification with various levels of Gaussian noise [36], et. In this paper, the texture fingerprint [37] of a malware is the set of texture fingerprints for each uncompressed gray-scale image block.

Our approach is inspired by some success gained in the above methods. After fully considering combination of the texture images and API call. We introduce a novel approach to identify the Android malicious code in smart phone. The texture images, which mapped code to uncompressed gray-scale, obtain or detect the feature from malware content, API calls stand for the activity of malware or benign code. According the combination of the potential texture images and API call features gained from the code segment of APK file, using Deep Belief Network to classify malicious code into families is put forward.

2. Methodology

Correctly identifying the malicious code Android can be a tough work for security scholars. Our approach requires a comprehensive yet unique representation of Android apps that helps determine the typical indications of malicious activity.

To achieve this goal, our method spends a lot of time to execute a broad static analysis that extracts feature sets by using Deep Belief Networks. According to this method, it is necessary to extract feature sets from different sources (such as Class.dex, lib/*.so, Manifest.xml, API calls, etc).

2.1 Deep Belief Networks (DBN)

2.1.1 Deep Belief Networks (DBN)

In 2006, Hinton, Osindero, and Teh [38] introduced a greedy layer-wise unsupervised learning algorithm for Deep Belief Networks (DBN), shows in **Fig. 1** The training strategy for such networks may hold great promise as a principle to help address the problem of training deep networks. Upper layers of a DBN are supposed to represent more "abstract" concepts that

explain the input data whereas lower layers extract "low-level features" from the data. As an unsupervised learning in deep architectures, DBN is a multi-layered probabilistic generative model. Deep Belief Network can be defined as a stack of Restricted Boltzmann machines with a Back Propagation(BP) to fine tuning. Previously, Deep Belief networks has been successfully employed in recognize, cluster and generate images, video sequences and motion-capture data. [39] [40]

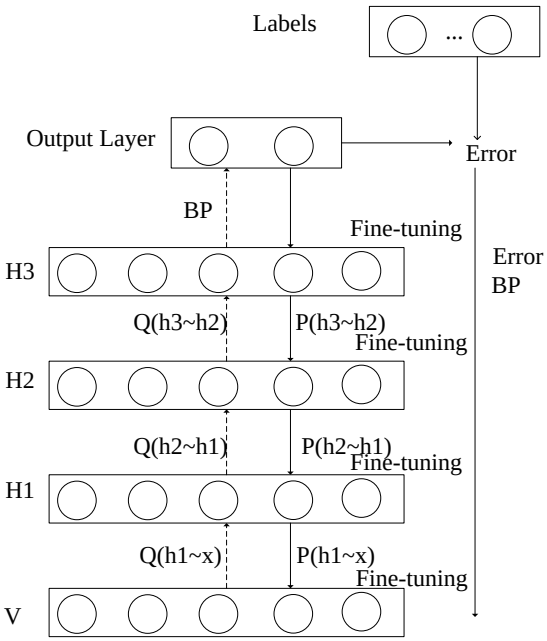


Fig. 1. DBN structure

2.1.2 Restricted Boltzmann Machine (RBM)

Restricted Boltzmann Machine (RBM) is a undirected graphical model of. There are no links between units of the same layer, only between input (or visible) units x_j and hidden (also invisible) units h_i. The difference between standard Boltzmann machines and RBM is that in the restricted model units within the same layer are not connected. which makes inference and learning within this graphical model tractable. As **Fig. 2** shows it.

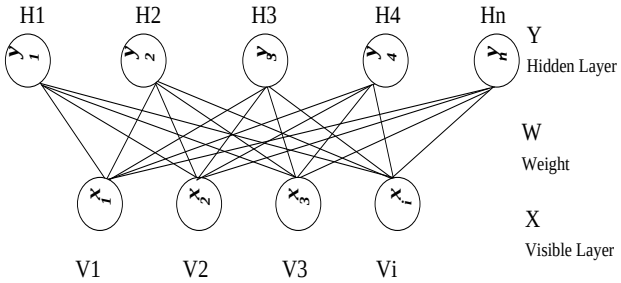


Fig. 2. RBM structure

RBM is a model of Energy Based Model(EBM),defined via the energy function, It consists of m visible units $V=\{v_1,v_2,,,,,,,v_m\}$ to represent observable data and n hidden units $H=\{h_1,h_2,,,,,,,h_n\}$ to capture dependencies between observed variables. And the joint probability distribution under the model is given by the Gibbs $p(v,h\mid\theta)$ with the energy function $E_\theta(v,h\mid\theta)$

$$p(v,h\mid\theta)=\frac{1}{\sum_{v,h}e^{-E(v,h|\theta)}}e^{-E(v,h|\theta)} \qquad (1)$$

$$E_\theta(v,h\mid\theta)=-\sum_{i\in visible}x_iv_i-\sum_{j\in hidden}y_jh_j-\sum_{i,j}v_ih_jw_{ij} \qquad (2)$$

For all i $\in$ {1,real,value} and j $\in$ {1,real,value}, w_{ij} is a real valued weight associated with the edge between units V_j and H_i and b_j and c_i are real valued bias terms associated with the jth visible and the ith hidden variable, respectively.

In terms of probability this means that the hidden variables are independent given the state of the visible variables and vice versa:

$$p(h\mid v)=\prod_{i=1}^{n}p(h_i\mid v)\quad and\quad p(v\mid h)=\prod_{i=1}^{m}p(v_i\mid h) \qquad (3)$$

The absence of connections between hidden variables makes the marginal distribution of the visible variables easy to calculate:

$$p(v,h\mid\theta)=\frac{1}{\sum_{v,h}e^{-E(v,h|\theta)}}e^{-E(v,h|\theta)} \qquad (4)$$

This equation shows why a (marginalized) RBM can be regarded as a product of experts model, in which a number of “experts” for individual components of the observations are combined multiplicatively.

The RBM can be interpreted as a stochastic neural network, where nodes and edges correspond to neurons and synaptic connections, respectively. The conditional probability of a single variable being one can be interpreted as the firing rate of a (stochastic) neuron with sigmoid activation function $f(x)$, where $f(x)$ is the sigmoid function, and the units will switch to state 0 otherwise.

$$P(V_i=1\mid H;\theta)=f(\sum_i w_{ij}h_j+x_i) \qquad (5)$$

$$P(H_j=1\mid V;\theta)=f(\sum_i w_{ij}v_i+y_j) \qquad (6)$$

$$f(x)=\frac{1}{1+e^{-x}} \qquad (7)$$

The independence between the variables in one layer makes Gibbs sampling especially easy: Instead of sampling new values for all variables subsequently, the states of all variables in one layer can be sampled jointly. Thus, Gibbs sampling can be performed in just two sub steps: sampling a new state h for the hidden neurons based on $p(h\mid v)$ and sampling a state v for the visible layer based on $p(v\mid h)$ This is also referred to as block Gibbs sampling.

Based on the algorithm above, RBM uses iteration between layers to training, and access to learning parameters $\theta=(w_{ij},x_i,y_j)$ ultimately.

$$\theta^* = \arg \max L(\theta) = \arg \max_{\theta} \sum_{t=1}^{T} \log P\left(v^{(t)} \mid \theta\right) \tag{8}$$

2.1.3 The Contrastive Divergence Fast learning algorithm

The goal of RBM training is to make marginal probability distribution $p(v)$ fit probability distribution of training samples based on justifying the parameters of model. To achieve this, we use k-steps contrastive divergence learning algorithm [41] to train RBMs which is a standard way to train RBMs. The idea of CD-k is quite simple: the chain is run for only k steps, starting from an example $v^{(0)}$ of the training set and yielding the sample $v^{(k)}$. Each step t consists of sampling $h(t)$ from $p(h \mid v^{(t)})$ and sampling $v^{(t+1)}$ from $p(v \mid h^{(t)})$ subsequently. The gradient in equation (2) with respect to θ of the log-likelihood for one training pattern $v^{(0)}$ is then approximated by equation (9),

$$CD_k(\theta, v^{(0)}) = -\sum_h p(h \mid v^{(0)}) \frac{\partial \varepsilon(v^{(0)}, h)}{\partial \theta} + \sum_h p(h \mid v^{(k)}) \frac{\partial \varepsilon(v^{(k)}, h)}{\partial \theta} \tag{9}$$

In the following, we restrict our considerations to RBMs with binary units for which $E_{p(h_i|v)}[h_i] = sigmoid(c_i + \sum_{j=1}^{m} w_{ij} v_j)$

2.1.4 Updating parameters

All common training algorithms for RBMs approximate the log-likelihood gradient given some data and perform gradient ascent on these approximations.

$$\begin{aligned} \Delta W_{ij} &= (learning \quad rate\langle V_i H_j \rangle_{data} - \langle V_i H_j \rangle_{recon}) \\ \Delta B_i &= (learning \quad rate\langle V_i \rangle_{data} - \langle V_i \rangle_{recon}) \\ \Delta C_i &= (learning \quad rate\langle H_i \rangle_{data} - \langle H_i \rangle_{recon}) \end{aligned} \tag{10}$$

Where in the above equation, $\langle * \rangle_{data}$ represents the Mathematical expectation of training data, $\langle * \rangle_{recon}$ represents the Mathematical expectation of reconstructed model.

2.1.5 Back Propagation(BP)

BP is a common method of training artificial neural networks and used in conjunction with an optimization method such as gradient descent. The algorithm repeats a two phase cycle, propagation and weight update. When an input vector is presented to the network, it is propagated forward through the network, layer by layer, until it reaches the output layer. The output of the network is then compared to the desired output, using a loss function, and an error value is calculated for each of the neurons in the output layer. The error values are then propagated backwards, starting from the output, until each neuron has an associated error value which roughly represents its contribution to the original output. BP is described in **Fig. 3**.

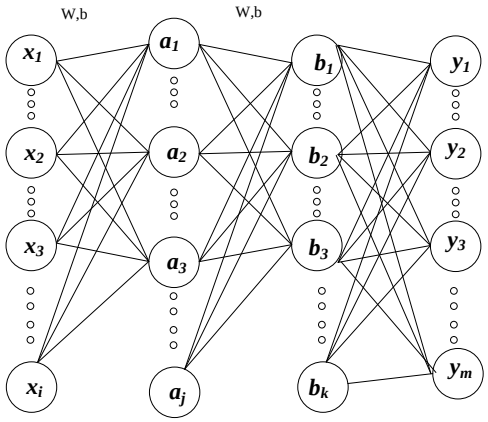


Fig. 3. BP structure

2.2 Android malware analysis

It is well known that Android applications (which are written in Java language) are packed inside a APK file. On desktop and server environments, a Java program is compiled to Java bytecodes, which is an intermediate representation (IR) for Java Virtual Machine (JVM). A Java bytecode is based on a stack-based instruction set (hence the term “stack-based Java bytecode”) and has object-oriented features. On the other hand, Android applications are run on Dalvik Virtual Machine (DVM). Which is an optimized virtual machine for the Android platform. However, on resource-limited mobile devices, since the performance of stack-based Java bytecode is not well due to slow interpretation, Android scholars have created a new bytecode set for DVM (Dalvik bytecode) to improve the performance of Android applications. In contrast to the Java bytecode, the Dalvik bytecode is based on a register-based instruction set. Therefore, it can reduce the code size and running time.

APK is just a ZIP file containing among a compact Dalvik Executable (.dex) file. In terms of static analysis, all we need is the .apk file of an Android app. By unzipping the APK, we extract the potential feature of code segment in APK file, and get the "classes.dex" file. Then we use the tool dex2jar to convert the classes.dex file to Java .class files. Meanwhile, we obtain the file "classes.dex.dex2jar.jar", now we use the Java decompiler JD-GUI to extract the source code. At last, Java source, which contains all the decompiled, code has been got.

Every application developed for Android must include a manifest file called Androidmanifest.xml, it provides data supporting the installation and later execution.

Android malicious code analysis is the study aiming to develop, examine and explore approaches and techniques to classify machine executables into either harmful or not [42]. Classification techniques plays a critical role in the Android malware analysis. To choice of potential explicit feature has enormous influence on the performance of deep learning-based malware detection, we need to select a sets of features.

In addition, we extract different feature sets from the application’s .dex, lib/*.so, androidmanifest.xml and other code. In this paper, we focus on the Image texture and five species of API call commonly used.

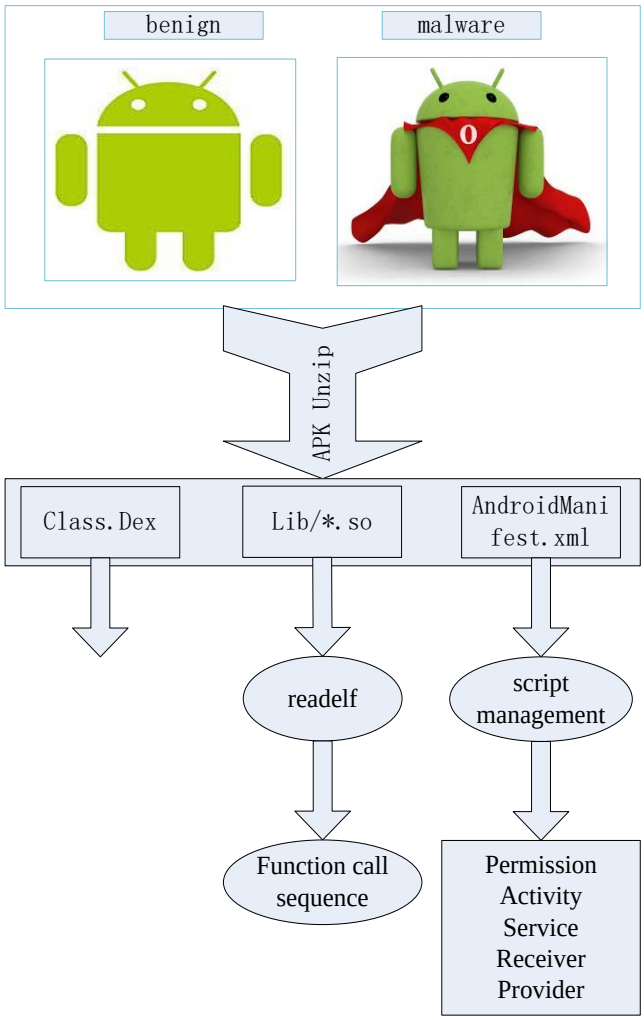


Fig. 4. Experiment Design

2.3 Image texture analysis

2.3.1 Image texture analysis

Visualization is a useful technique widely used in computer security like computer forensics, Network space security, image analysis, image classification and large-scale image search, to name a few. Recently, image texture-based classification was used to classify malware [43] [44]. Image texture is a block of pixels which contains variations of intensities arising from repeated patterns.

In this work, we propose an image texture based analysis to analyze malicious code. First the executable is converted into byteplot image. Later analysis is performed on this byteplot. Our method is more robust to code obfuscation and does not require unpacking or decryption. Other advantage of our approach is that we can apply widely used image processing techniques like textures analysis, near duplicate detection techniques to malware detection.

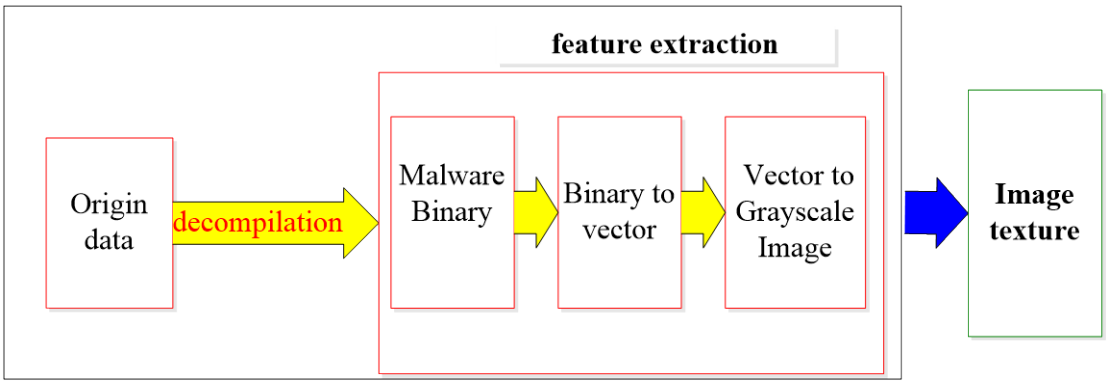


Fig. 5. Feature extraction of image texture

Table 1. Feature extraction of image texture

```
def getMatrixfrom_bin(filename, width = 512, oneRow = False):
    with open(filename, 'rb') as f:
        content = f.read()
    hexst = binascii.hexlify(content)
    fh = numpy.array([int(hexst[i:i+2],16) for i in range(0, len(hexst), 2)])
    if oneRow is False:
        rn = len(fh)/width
        fh = numpy.reshape(fh[:rn*width],(-1,width))
    fh = numpy.uint8(fh)
    return fh
def getMatrixfrom_asm(filename, startindex = 0, pixnum = ??): #?? Represent the pix num
    with open(filename, 'rb') as f:
        f.seek(startindex, 0)
        content = f.read(pixnum)
    hexst = binascii.hexlify(content)
    fh = numpy.array([int(hexst[i:i+2],16) for i in range(0, len(hexst), 2)])
    fh = numpy.uint8(fh)
        return fh
for sid in subtrain.Id:
    i += 1
    print "dealing with {0}th file...".format(str(i))
    filename =basepath + sid + ".txt"
    im = getMatrixfrom_asm(filename, startindex = 0, pixnum =??)#?? Represent the pix num
    mAPImg[sid] = im
```

2.3.2 Image texture analysis based on Deep Belief Network

To demonstrate where the performance gains are produced, we use Deep Belief network to extracted Image texture from the step mentioned above. Hinton et. show that RBMs can be stacked and trained in a greedy manner to form so-call Deep Belief Networks, as it is illustrated in **Fig. 1.**

Compared with conventional neural network algorithm such as BP algorithm, DBN offer a better performance. DBN can solve the problems of falling into the local minimum point easily, which is realized by two key training steps: pre-training and fine-tuning.

The process of training DBN is as follows:

Step 1: Pre-train the single layer of RBM. We can use thorough dataset as training set in this part, for it is an unsupervised learning. After certain epochs of training, the cost function will change very little when updates using Eq.(11). Constrastive divergence (CD) is developed by Geoffrey Hinton for training RBM model.

The data negative log-likelihood gradient for an RBM with binary units is:

$$
\begin{aligned}
-\frac{\partial \log p(\theta)}{\partial W_{ij}} &= E_v[p(h_i \mid v) \cdot v_j] - v_j^{(i)} \sigma(W_{ij} v^{(i)} + b_i) \\
-\frac{\partial \log p(\theta)}{\partial b_i} &= E_v[p(h_i \mid v)] - \sigma(W_{ij} v^{(i)}) \\
-\frac{\partial \log p(\theta)}{\partial a_j} &= E_v[p(h_i \mid v)] - v^{(i)}
\end{aligned}
\tag{11}
$$

Step 2: Use that first layer to obtain a representation of the input that will be used as input data for the second layer. Train the second layer as an RBM, taking the transformed data (samples or mean activations) as training examples (for the visible layer of that RBM).

Step 3: Iterate 1 and 2 for the desired number of layers, each time propagating upward either samples or mean values.

Step 4: Fine-tune all the parameters of this deep architecture with a supervised training criterion. Labeled data samples are used in this step.

In this paper, the importance of using the texture image information becomes obvious and deep networks are powerful enough to extract image texture information from the input image.

Suppose ***T1*** is the input image with size of ***n*** pix (also parameters of image texture features). On the other hand, DBN standard topology utilizes only ***T1*** as input which is vector. More precisely, ***T1*** is converted to 1-D vector and DBN standard topology uses this vector in first layer as input.

In the proposed DBN topology, both inputs layer are vector. Extracting image texture features in first and second layer of DBN multi-input topology improves DBN power to images texture more accurate.

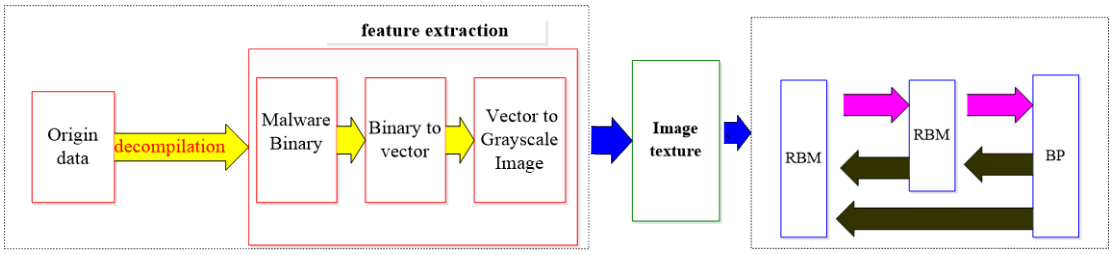


Fig. 6. Image texture analysis based on Deep Belief network

2.4 API Call analysis

2.4.1 API Call analysis

Application Programming Interface (API) calls for characterizing Android applications' behaviors. In order to acquire certain tasks on the devices, such as sending a message, each application has to request permission or call the corresponding API from the user during the installation or using. In our work, frequency of occurrence listed below have been counted.

a. API calls: This kind of feature consists of two parts, namely restricted API call and suspicious API call. The Android permission system restricts access to a series of critical API calls. As these calls can particularly lead to malicious behavior.

b. Used permissions: Whenever an API call is invoked during the execution of an application, the Android platform will verify if the API call is permission-protected before proceeding to execute the call; such permissions are referred to as used permissions.

c. Url: all URLs found in the disassembled code are included in the last set of features. Some of these addresses might be involved in botnets and thus present in several malware samples, which can help to improve the learning of detection patterns.

d. Intenstion: Inter-process and intra-process communication on Android is mainly performed through intents: passive data structures exchanged as asynchronous messages and allowing information about events to be shared between different components and applications. We collect all intents listed in the manifest as another feature set, as malicious application often listens to specific intents. A typical example of an intent message involved in malicious is BOOT_COMPLETED, which is used to trigger malicious activity directly after rebooting the phone.

e. Activity: Malicious activity is usually reflected in specific patterns and combinations of the extracted features. For example, a malware sending premiums SMS messages might contain the permission SEND_SMS , and the hardware component android.hardware.telephony

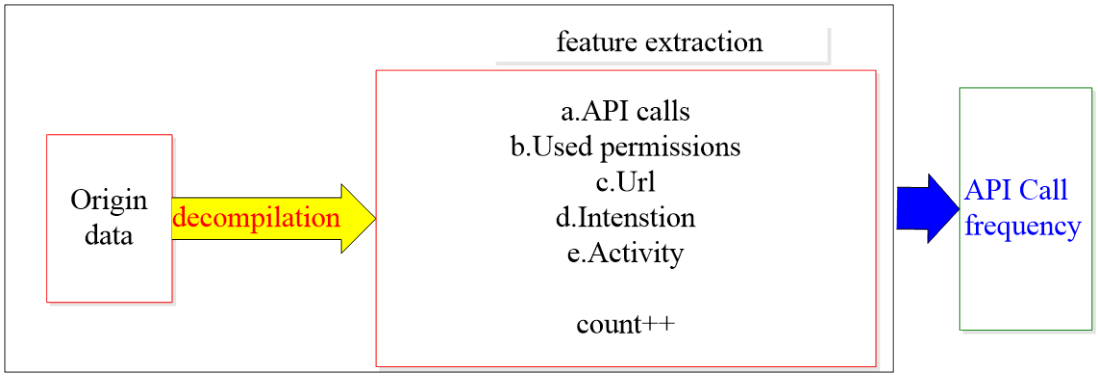


Fig. 7. Feature extraction of API Call

2.4.2 API Call analysis based on Deep Belief network

Similarly, in this paper, Deep Belief network are used to extract API call from the step mentioned above.

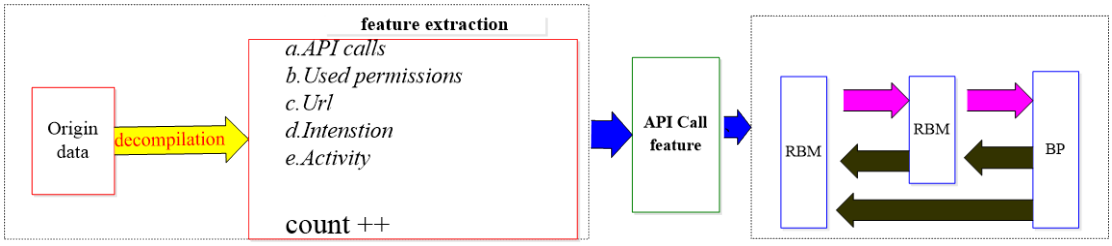


Fig. 8. API Call analysis analysis based on Deep Belief network

3. Experiment environment

For our experiment, we set up a appropriate experiment environment to detect and analyze Android malicious code.

Table 2. Experimental environment

Parameter	Value
OS	Windows 7 64 bit platform
CPU	Intel(R) Core(TM) i5-4200M CPU
RAM	8G/ DDR3 /1333MHz
Hard disk	120G SSD+2T HDD
Python	2.7.3

It is essential to constructs an experiment environment of Python 2.7.3 with suitable and available module. I've listed it here.

Table 3. The Python module for the experiment

name	introduction	version
numpy	The fundamental package for scientific computing with Python	1.9.2
pandas	Data structures and data analysis tools	0.16.0rc1(0.11.1)
PIL	Python Imaging Library	1.1.7
Scikit-learn	Python module for machine learning	0.16.1
scipy	Python-based ecosystem of open-source software for mathematics	0.15.1
twisted	An asynchronous networking framework	15.4.0
six	Compatibility library	1.7.3
pip	The PyPA recommended tool for installing Python packages.	8.1.2
wheel	A built-package format for Python	0.29.0
datutil	Extensions to the standard Python datetime module	2.2
pyparsing	A general parsing module for Python	2.0.1
setuptools	Package development process library	0.16.1
pytz	Brings the Olson tz database into Python	2016.6.1
nolearn	Python maching learning module	0.6.0
theano	Python deep learning module	0.8.2
gdbn	Python deep learning DBN moudle	0.2

4. Experimental Classification Results

The samples was collected from August 2010 to October 2012. The dataset contains 5560 malicious and 123453 benign application.

In order to guarantee the efficient stringency of experiments data applied to the model, this paper selects 1550,2620,5825,6965 samples from the dataset.

To begin with, we read a lot of files from the features set, as it shows in **Table 4.**

Table 4. File reading

```
1. subtrainLabel = pd.read_csv('label.csv')
2. subtrainfeature1 = pd.read_csv("imgfeature.csv")
3. subtrainfeature2 = pd.read_csv("call.csv")
4. subtrain = pd.merge(subtrainfeature1,subtrainfeature2,on='Id')
5. subtrain = pd.merge(subtrain,subtrainLabel,on='Id')
6. labels = subtrain.Class
7. subtrain.drop(["Class","Id"], axis=1, inplace=True)
8. subtrain = subtrain.as_matrix()
```

In addition, in order to promote the accuracy of classification algorithm on feature selection, on the basis of that, we amplify the implicit features of texture image and API call in malicious code, to train Restricted Boltzmann Machine and Back Propagation.

Parameter tuning of DBN will affect the performance of the model. The appropriate parameters are essential to optimal recognized. After, a plenty of repeated experiment have been done. The parameters are shown in **Table 5.**

Table 5. The parameters of DBN model

parameters	values
optimize	BP
loss	log-likelihood gradient
test_size	0.4
learn_rates	0.9
learn_rate_decays	0.5
epochs	300

Table 6. The parameters of DBN model

```
1. (trainX, testX,trainY,testY)=cross_validation.train_test_split(subtrain,labels,test_size=0.4)
2. dbn=DBN([trainX.shape[1],2],learn_rates=0.9,learn_rate_decays=0.5,epochs=300,verbose=1)
3. dbn.fit(trainX,trainY)
4. preds=dbn.predict(testX)
```

To guarantee the efficient stringency of experiments data applied to the model, 60% of experimental data are applied for DBN training. The rest were applied as testing data for validating the DBN's predictability.

In order to have a fair comparison, the *correct classification rate* is performed as the following equation (11).

$$correct\ classification\ rate=(TP+TN)/(P+N) \tag{11}$$

Where *TP*, *TN*, *P*, and *N* denoted *True Positive*, *True Negative*, *Positive instances* and *Negative instances*, respectively.

4.1 The effect of Image Texture features on experimental results

In order to guarantee the quality of experimental data, this paper selects 1550, 2620 samples from the dataset to indicate the effect of Image Texture features on experimental

results, in thousands of trials have been done. Pix num ,also named parameters of image texture features, with different pix num. The effect of Image Texture features on experimental results are in the following table and graph.

Table 7. The effect of Image Texture features on experimental results

	500	1000	2000	2500
1550	94.0	94.1	**94.8**	94.3
2620	94.08	94.9	**95.1**	94.6

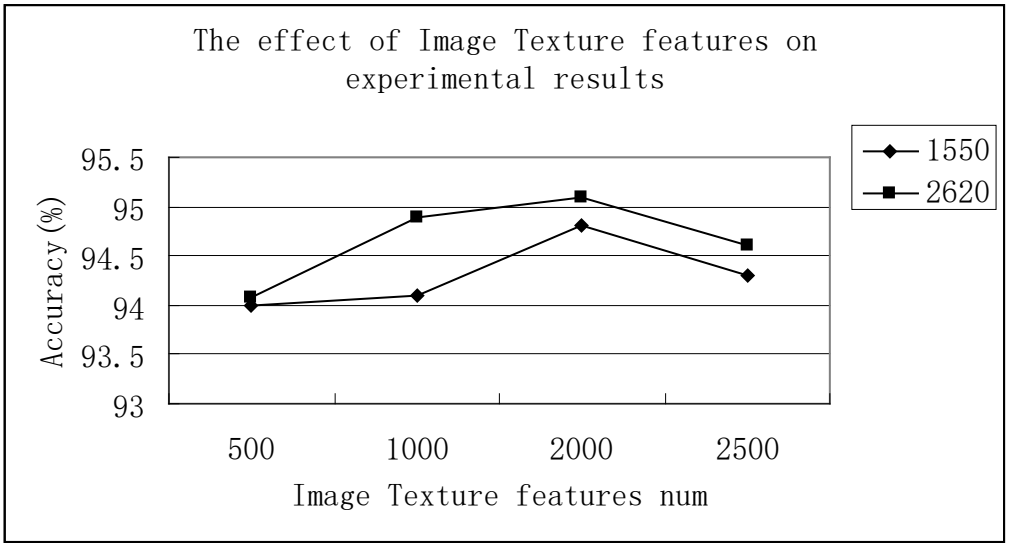


Fig. 9. The effect of Image Texture features on experimental results

4.2 The effect of DBN Layer on experimental results

For proving the effect of DBN layers to experimental results, this paper makes a large number of experimental.

Table 8. The effect of DBN Layer on experimental results

	2	3	4
1550	**94.8**	94.6	94.1
2620	**95.1**	94.8	94.0

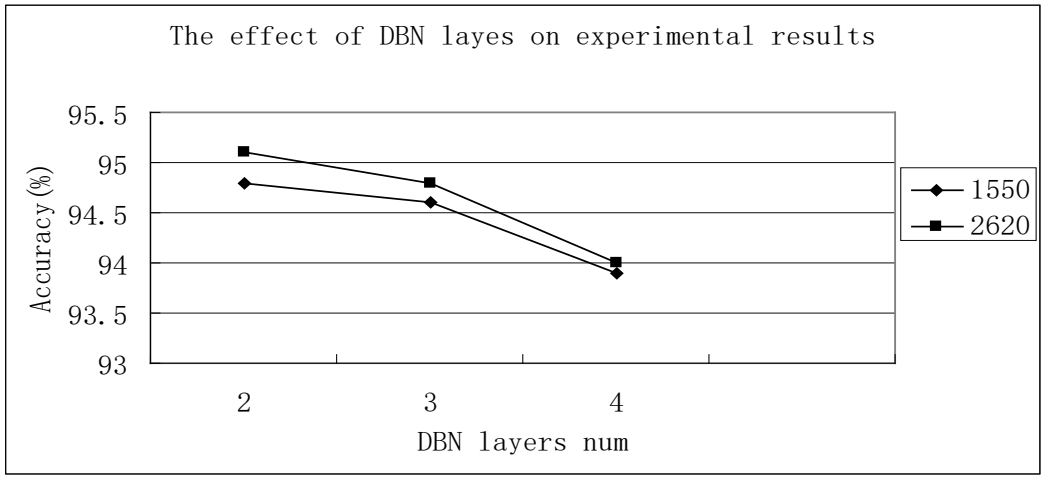


Fig. 10. The effect of DBN Layer on experimental results

According to the result above, we adopt the best parameters of Image Texture features, DBN Layers, Learning Rates, Learning Rates Decays et.

In our work, the Image Texture features is set to 2000, DBN Layers is set to 2.

4.3 DBN model with image texture feature or not

It is generally accepted that a good feature is crucial to make a powerful detection accuracy at high level. To verify image texture is one of the essential feature for Android malicious code classification, 1550, 2620, 5825, 6965 samples from the dataset has been screened for the experiment.

Table 9. DBN model with different feature

	dbn（image texture+API call）	dbn（API call only）
1550	**94.8**	94.6
2620	**95.1**	95.0
5825	**95.7**	93.5
6956	**95.6**	94.7

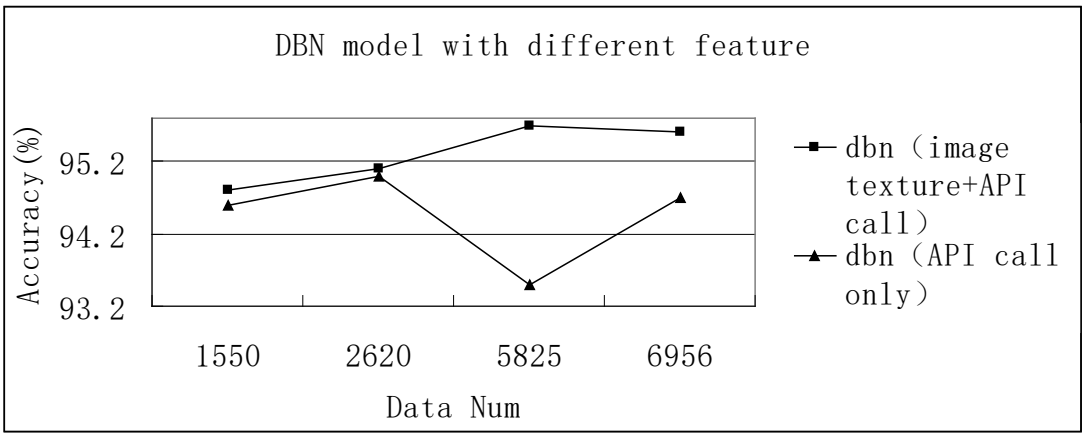


Fig. 11. DBN model with image texture feature or not

In other words, it shows that our method which utilized the information of image texture can improve forecast accuracy by an average of 0.85%. This indicates that image texture can effectively promote the classification accuracy. Next step, we consider the combined features of image texture and API call to test and verify our experiment.

4.4 DBN compared to shallow machine learning

In order to confirm our algorithm is more suitable for Android malicious code classification, 1550,2620,5825,6965 samples from the dataset has been screened for the experiment. The selected samples' forecast accuracy in each case are shown in **Table 10**.

Table 10. DBN compared to shallow machine learning

	dbn (image texture+API call)	svm (image texture+API call)	knn (image texture+API call)	ann (image texture+API call)
1550	**94.8**	92.7	94.5	93.6
2620	**95.1**	94.2	95.0	95.0
5825	**95.7**	94.5	95.1	95.2
6956	**95.6**	94.8	95.4	95.0

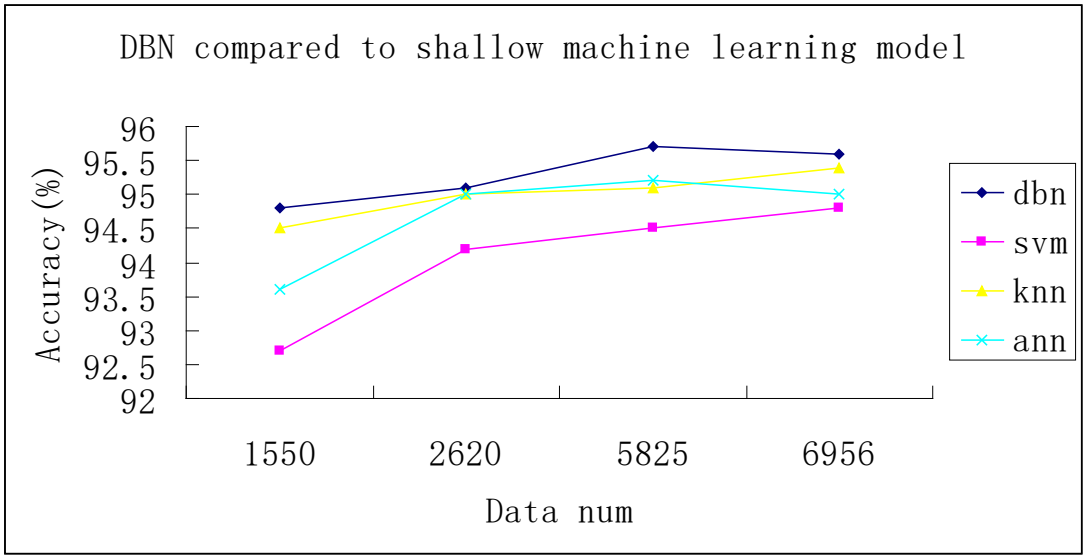


Fig. 12. DBN compared to shallow machine learning

Fig. 12 demonstrates in that our method, which used fusion features of image texture and API call, achieves the accuracy more than 94% contrast to shallow machine learning method.

As we can see, comparing with KNN ,SVM and ANN model, DBN model plays a great promotion in accuracy. The results demonstrate that the proposed model solved the problems with high detection. This is not surprising because as shallow machine learning, SVM, KNN are unable to extract feature automatically and have low fitting ability, while Deep Belief Network, as a multilayer deep learning model, has strong depicting ability through multilayer iterative algorithm.

In order to confirm the availability and versatility of the algorithm of DBN to malware classification with the combined features of texture image and .asm call, we apply DBN to

classify malicious code on the dataset of Microsoft Malware Classification Challenge (BIG 2015)|Kaggle. Comparing with KNN and ANN, DBN also shows a great promotion in accuracy with different samples of 1000, 1405, 2000, 3178, 3626. The results have been showed in the my previous research.

Table 11. Classification accuracy comparison

Data Num	dbn	knn	ann
1000	**0.9225**	0.9000	0.9103
1405	**0.9608**	0.9480	0.9520
2000	**0.9610**	0.9562	0.9572
3178	**0.9716**	0.9583	0.9600
3626	**0.9724**	0.9648	0.9705
average	**0.95766**	0.94546	0.9500

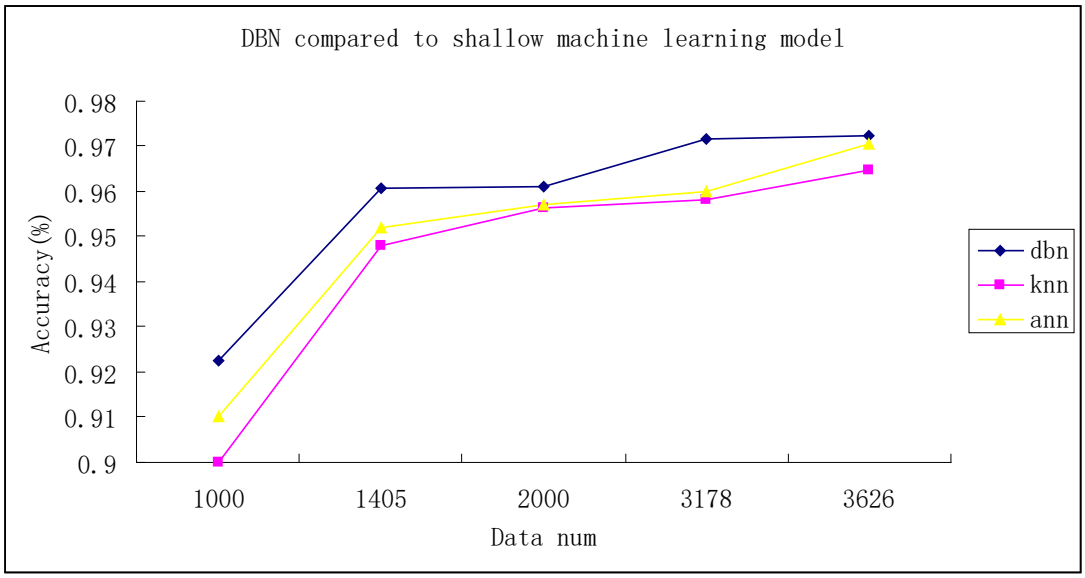


Fig. 13. DBN compared to shallow machine learning with Microsoft dataset

5. Experimental Analysis

The existing works for feature learning aiming at Android malicious code detection employed features based on features frequency [45], or only relied on statically or dynamic analysis.

Analyzing the code using static analysis is a widely addressed technique for detecting unknown malicious codes. [46], the static malware analysis is not need to execute the malware, but it can not deal with the Shell code, polymorphism, metamorphism. Although the dynamic malware analysis avoids the drawbacks above, but it needs to execute the malware in run-time to avoid them, the dynamic analysis of malware detection monitors malicious activity on the smart phone, it involved millions of smartphones in practice Paranoid Android at large scale is technically not feasible. And the existing methods do not consider the combination of texture image and API call. The texture image can represents the static features of malware and the

API signals malware activity.

According to the previous work [47] ,the malicious code has a unique "image texture" in feature spatial relations, and the same type of malicious code have similar "image texture" feature. Also, to perform certain tasks on the devices, such as sending a message, each application has to request permission or call the corresponding API from the user during the installation or using.

So, in this letter we access information of texture image from malicious or benign code, which is with size of n pix. The last but not least, in order to improve the efficient of detection, our work combines texture image and API call frequency.

Moreover, the method to detect Android malicious code employed traditional machine learning algorithms which have shallow architectures. In contrast, in this paper, our work adopts Deep learning algorithm named Deep belief Network which can learn high level representations by associating features obtain from static analysis, which has more possible to character Android detection.

Contrast to basic learning method for shallow structure algorithm, the limitation lies in limited samples and cell cases of complex function said ability is limited, its generalization ability for complex classification problems under certain constraints. [48] Deep learning method plays a better generalization performance of the classification and can learn more about cell cases of complex function.

As it illustrates in **Fig. 11** with different samples of malicious code or benign, when we use deep belief network combined with different feature, it is clearly to see the gap between the use of image texture or not.

As it is shown in **Fig. 12** with different samples of malicious code or benign, the method of deep belief network plays a strong advantages contrast to the shallow machine learning method(knn, svm, ann) in classification accuracy, also in **Fig. 13** The reason of this phenomenon is the fusion features, particularly the image texture image, which contains strong depicting ability.

To make a long story short, the method of image texture-based to classify malware is a very essential factor to the improvement of accuracy, which can extract or detect the feature more from malware content. At the same time, the algorithm of Deep Belief Network is also applied to combine with image texture and API call. The Deep Belief Network learns more about of the malware samples with the features of texture image and API call on account of its deep structure. It plays well generalization and expression ability in samples of malicious code. The results show that the way put in our work, the accuracy to detect malware can reach 95.3% on average contrast to shallow machine learning method and without the image texture.

6. Conclusion

In this research, the image texture and deep learning method used in Android malware are discussed and initially applied on feature extraction and classification of data.

The dataset (Drebin) used for these experiments consisted of 5560 malicious and 123453 benign application. In this paper, we select part of them. You can get it from http://user.cs.uni-goettingen.de/˜ darp/ drebin.

As it is shown in **Fig. 11, Fig. 12** and **Fig. 13**, the feature of image texture can well retain the information contained the origin data. What is more, during the tuning of parameters, it depicts that DBN shows great depicting ability of input data.

Further researches on malware detection and analysis, will apply the semantic information and access the text embedding for profound malware semantic mining.

Acknowledgements

We would like to thank all the participants in our study that provided useful and detailed feedback. Meanwhile, I would thank all my tutor for the research.

This research was partially supported by the Xinjiang Uygur Autonomous Region Science and Technology Personnel Training Project (QN2016YX0051). Research Innovation Project of Graduate Student in Xinjiang Uygur Autonomous Region(No. XJGRI2017007). The Project Of Cernet Next Generation Internet Technology Innovation Project(NGII20170420).

References

[1] G. McGraw and G. Morisett, "Attacking Malicious Code: A Report to the Infosec Research Council," *IEEE Software*, vol. 17, no. 5, pp. 33-41, Sep/Oct 2000. Article (CrossRef Link)

[2] Seshagiri P,Vazhayil A,Sriram P., "AMA: Static Code Analysis of Web Page for the Detection of Malicious Scripts," *Procedia Computer Science*, 93:768-773, 2016. Article (CrossRef Link)

[3] Willems C, Holz T, Freiling F., "Toward Automated Dynamic Malware Analysis Using CWSandbox," *IEEE Security & Privacy Magazine*, 5(2):32-39, 2007. Article (CrossRef Link)

[4] W.Enck, P.Gilbert, B.gon Chun, L.P.Cox, J. Jung, P. McDaniel, and A. Sheth, "Taintdroid: An information-flow tracking system for realtime privacy monitoring on smartphones," in *Proc. of USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pp. 393–407, 2010. Article (CrossRef Link)

[5] Y. Zhou, Z. Wang, W. Zhou, and X. Jiang, "Hey, you, get off of my market: Detecting malicious apps in official and alternative android markets," in *Proc. of Network and Distributed System Security Symposium (NDSS)*, 2012. Article (CrossRef Link)

[6] L.-K. Yan and H. Yin, "Droidscope: Seamlessly recon- structing os and dalvik semantic views for dynamic android malware analysis," in *Proc. of USENIX Security Symposium*, 2012. Article (CrossRef Link)

[7] Enck, William, Ongtang, et al., "On lightweight mobile phone application certification," in *Proc. of ACM Conference on Computer and Communications Security (CCS)*, pp. 235–245, 2009. Article (CrossRef Link)

[8] A. P. Felt, E. Chin, S. Hanna, D. Song, and D. Wagner, "Android permissions demystified," in *Proc. of ACM Conference on Computer and Communications Security (CCS)*, pp. 627–638, 2011. Article (CrossRef Link)

[9] M. Grace, Y. Zhou, Q. Zhang, S. Zou, and X. Jiang, "Riskranker: scalable and accurate zero-day android malware detection," in *Proc. of International Conference on Mobile Systems, Applications, and Services (MOBISYS)*, pages 281–294, 2012. Article (CrossRef Link)

[10] Filiol E, Jacob G, Liard M L., "Evaluation methodology and theoretical model for antiviral behavioural detection strategies," *Journal of Computer Virology and Hacking Techniques*, 3(1):23-37, 2007. Article (CrossRef Link)

[11] Venkitaraman R, Gupta G. "Static program analysis of embedded executable assembly code[C]," in *Proc. of International Conference on Compilers, Architecture, and Synthesis for Embedded Systems, CASES 2004*, Washington Dc, Usa, 157-166, September, 2004. Article (CrossRef Link)

[12] Zhang B, Li Q, Ma Y., "Research on dynamic heuristic scanning technique and the application of the malicious code detection model," *Information Processing Letters*, 117:19-24, 2017. Article (CrossRef Link)

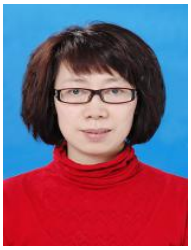
[13] Caballero J, Grier C, Kreibich C, et al., "Measuring Pay-per-Install: The Commoditization of Malware Distribution," in *Proc. of Usenix Conference on Security*, USENIX Association, 2011. Article (CrossRef Link)

[14] Chen P S, Lin S C, Sun C H., "Simple and effective method for detecting abnormal internet behaviors of mobile devices," *Information Sciences*, 321:193-204, 2015. Article (CrossRef Link)
[15] Coron J S., "On the Exact Security of Full Domain Hash," *Advances in Cryptology — CRYPTO 2000*, Springer Berlin Heidelberg, 229-235, 2004. Article (CrossRef Link)
[16] Griffin K, Schneider S, Hu X, et al., "Automatic Generation of String Signatures for Malware Detection," in *Proc. of Recent Advances in Intrusion Detection, International Symposium, RAID 2009*, Saint-Malo, France, Proceedings. DBLP, 101-120, September 23-25, 2009. Article (CrossRef Link)
[17] Mohaisen A, Alrawi O, Larson M, et al., "Towards a Methodical Evaluation of Antivirus Scans and Labels," *Revised Selected Papers of the, International Workshop on Information Security Applications*, Springer-Verlag New York, Inc., 231-241, 2003. Article (CrossRef Link)
[18] Mehdi B, Ahmed F, Khayyam S A, et al., "Towards a Theory of Generalizing System Call Representation for In-Execution Malware Detection," in *Proc. of IEEE International Conference on Communications*, IEEE, 1-5, 2010. Article (CrossRef Link)
[19] Xie P D, Li M J, Wang Y J, et al., "Unpacking Techniques and Tools in Malware Analysis," *Applied Mechanics & Materials*, 198-199:343-350, 2012. Article (CrossRef Link)
[20] Cowen B, Shafi K., "Fractal methods for the representation and analysis of polymorphism in malware," in *Proc. of Military Communications and Information Systems Conference*, IEEE, 1-5, 2013. Article (CrossRef Link)
[21] Ozsoy M, Khasawneh K N, Donovick C, et al., "Hardware-based Malware Detection using Low level Architectural Features," *IEEE Transactions on Computers*, pp. 3332-3344, 2016. Article (CrossRef Link)
[22] J Šťastná, M Tomášek, "The Problem of Malware Packing and its Occurrence in Harmless Software," *Acta Electrotechnica et Informatica*, 16(3): 41–47, 2016. Article (CrossRef Link)
[23] Park Y, Reeves D, Mulukutla V, et al., "Fast malware classification by automated behavioral graph matching," in *Proc. of CSIIRW '10 Proceedings of the Sixth Annual Workshop on Cyber Security and Information Intelligence Research*, 1-4, 2010. Article (CrossRef Link)
[24] Christodorescu M, Jha S, Seshia S A, et al., "Semantics-aware malware detection," in *Proc. of Security and Privacy, 2005 IEEE Symposium on*, 32-46, 2005. Article (CrossRef Link)
[25] Fredrikson M, Jha S, Christodorescu M, et al., "Synthesizing Near-Optimal Malware Specifications from Suspicious Behaviors," *IEEE Symposium on Security and Privacy. IEEE Computer Society*, 45-60, 2010. Article (CrossRef Link)
[26] Kolbitsch C, Comparetti P M, Kruegel C, et al., "Effective and efficient malware detection at the end host," in *Proc. of 18th Usenix Security Symposium*, 351-366, Montreal, Canada, August 10-14, 2009. Article (CrossRef Link)
[27] Chen K Z, Johnson N, D'Silva V, et al., "Contextual Policy Enforcement in Android Applications with Permission Event Graphs," *Heredity*, 110(6):586, 2013. Article (CrossRef Link)
[28] Schultz M G, Eskin E, Zadok F, et al., "Data mining methods for detection of new malicious executables," in *Proc. of Security and Privacy, 2001. S&P 2001. Proceedings. 2001 IEEE Symposium on. IEEE*, 38-49, 2001. Article (CrossRef Link)
[29] Arp D, Spreitzenbarth M, Hübner M, et al., "DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket," in *Proc. of Network and Distributed System Security Symposium*, 2014. Article (CrossRef Link)
[30] Schultz M G, Eskin E, Zadok F, et al., "Data mining methods for detection of new malicious executables," in *Proc. of Security and Privacy, 2001. S&P 2001. Proceedings. 2001 IEEE Symposium on. IEEE*, 38-49, 2001. Article (CrossRef Link)
[31] Alam M S, Vuong S T., "Random Forest Classification for Detecting Android Malware," in *Proc.* of Green Computing and Communications (GreenCom), 2013 IEEE and Internet of Things (iThings/CPSCom), IEEE International Conference on and IEEE Cyber, Physical and Social Computing, 663-669, 2013. Article (CrossRef Link)
[32] Shen F, Shen C, Zhou X, et al., "Face image classification by pooling raw features," *Pattern Recognition*, 54(C):94-103, 2016. Article (CrossRef Link)

[33] Shen F, Zhou X, Yang Y, et al., "A Fast Optimization Method for General Binary Code Learning," *IEEE Transactions on Image Processing A Publication of the IEEE Signal Processing Society*, 25(12):5610-5621, 2016. Article (CrossRef Link)
[34] Shen F, Yang Y, Liu L, et al., "Asymmetric Binary Coding for Image Search," *IEEE Transactions on Multimedia*, 19(9), 2022-2032, 2017. Article (CrossRef Link)
[35] Shi C Z, Zhao Q, Luo L P., "Application of Gray-Scale Texture Feature in the Diagnosis of Pulmonary Nodules," *Applied Mechanics & Materials*, 140:34-37, 2012. Article (CrossRef Link)
[36] Hadizadeh H., "Multi-resolution local Gabor wavelets binary patterns for gray-scale texture description," *Pattern Recognition Letters*, 65(C):163-169, 2015. Article (CrossRef Link)
[37] Han X G, Qu W, Yao X X, et al., "Research on malicious code variants detection based on texture fingerprint," *Journal on Communications*, 2014. Article (CrossRef Link)
[38] Geoffery E. Hinton, Salakhutdinov RR., "Reducing the dimensionality of data with neural networks," *Science*, 313(5786), 504-7, Jul 28 2006. Article (CrossRef Link)
[39] Ch'Ng S I, Seng K P, Ang L M, et al., "Block-based Deep Belief Networks for face recognition," *International Journal of Biometrics*, 4(2), 130-143, 2012. Article (CrossRef Link)
[40] Yu D, Deng L., "Deep Learning and Its Applications to Signal and Information Processing [Exploratory DSP]," *IEEE Signal Processing Magazine*, 28(1), 145-154, 2011. Article (CrossRef Link)
[41] Geoffrey E. Hinton, "Training products of experts by minimizing contrastive divergence," *Neural Comput.*, 14(8), 1771–1800, August 2002. Article (CrossRef Link)
[42] Yin H, Song D, Egele M, et al., "Panorama:capturing system-wide information flow for malware detection and analysis," in *Proc. of ACM Conference on Computer and Communications Security, CCS 2007*, Alexandria, Virginia, Usa, DBLP, 116-127, October 2007. Article (CrossRef Link)
[43] L. Nataraj, S. Karthikeyan, G. Jacob, and B. Manjunath, "Malware images: Visualization and autmoatic classification," in *Proc. of VizSec '11 the 8th International Symposium on Visualization for Cyber Security*, 2011. Article (CrossRef Link)
[44] Kancherla K, Mukkamala S., "Image visualization based malware detection," *Computational Intelligence in Cyber Security*, IEEE, 40-44, 2013. Article (CrossRef Link)
[45] Zhao K, Zhang D, Su X, et al., "Fest: A feature extraction and selection tool for Android malware detection," in *Proc. of Computers and Communication (ISCC), 2015 IEEE Symposium on*, 714-720, 2015. Article (CrossRef Link)
[46] Louk M, Lim H, Lee H J, et al., "An effective framework of behavior detection-advanced static analysis for malware detection," in *Proc. of International Symposium on Communications and Information Technologies*, IEEE, 361-365, 2014. Article (CrossRef Link)
[47] Peng L, Wang R, Wu A., "Research on Unknown Malicious Code Automatic Detection Based on Space Relevance Features," *Journal of Computer Research & Development*, 49(5):949-957, 2012. Article (CrossRef Link)
[48] Bengio Y., "Learning deep architectures for AI," *Foundations and trends in machine learning*, 2(1):1-127, 2009. Article (CrossRef Link)

Luo Shiqi, born in 1993. Postgraduate student in the School of Software, Xinjiang University. His main research interests include Information Security.

Tian Shengwei, born in 1973. PhD. Professor in the School of Software, Xinjiang University. His main research interests include Intelligence Computing.



Yu long, born in 1974. Professor in the Xinjiang University. She received the M.S. degree in Xinjiang university. Her research interests include Intelligence Technology.

Yu Jiong, born in 1964. Professor and PhD supervisor in computer science at the School of Information Science and Engineering, Xinjiang University. His main research interests include on grid computing, parallel computing, etc.

Sun Hua, born in 1974. She received the PHD degree in East China University of Science and Technology. Her research interests include Information Security.