

Andro_MD: Android Malware Detection based on Convolutional Neural Networks

Nannan Xie^{a,b}, Xiaoqiang Di^{a,b,*}, Xing Wang^c, and Jianping Zhao^{a,b}

^a*School of Computer Science and Technology, Changchun University of Science and Technology, Changchun, 130022, China*

^b*Jilin Provincial Key Laboratory of Network and Information Security, Changchun, 130022, China*

^c*School of Computer Science and Information Technology, Beijing Jiaotong University, Beijing, 100044, China*

Abstract

Android OS maintains its dominance in smart terminal markets, which brings growing threats of malicious applications (apps). The research on Android malware detection has attracted attention from both academia and industry. How to improve the malware detection performance, what classifiers should be selected, and what features should be employed are all critical issues that need to be solved. Convolutional Neural Networks (CNN) is a typical deep learning technique that has achieved great performance in image and speech recognitions. In this work, we present an Android malware detection framework Andro_MD that can train and classify samples with a deep learning technique. The framework includes dataset construction and feature preprocessing, training and classification by CNN, and evaluation by experiments. First, an Android app dataset is constructed with 21,000 samples collected from third-party markets and 34,570 features of 7 categories. Second, we employ sequential and parallel models to train the extracted features and classify the malware apps. Finally, extensive experimental results show the effectiveness and feasibility of the proposed method. Comparisons with similar work and traditional classifiers show that Andro_MD has a better performance on malware detection, and its accuracy can achieve 99.25% with a FPR of 0.53%. The "request permissions" and "used permissions" feature categories have better performances with limited dimensions.

Keywords: android malware detection; convolutional neural networks; deep learning

(Submitted on December 20, 2017; Revised on January 21, 2018; Accepted on February 24, 2018)

© 2018 Totem Publisher, Inc. All rights reserved.

1. Introduction

Android is the most popular operating system of smart terminals. The IDC statistical report [32] showed that Android dominated 87.6% of the market shares in the second quarter of 2016. Android malware has become a serious threat of network security and privacy protection. 360 Internet Security Center published the "2016 Android malware special report", which showed that they have monitored more than 250 million mobile phones that have been infected by malicious programs. The new malicious behaviors are mainly charge consumption, malicious deduction and privacy theft. Malware on Android OS can be hidden in normal apps by camouflage, which makes malware detection more challenging. With the increasing number of Android apps on the markets, how to detect and reduce malware has become an emerging and critical issue.

Although there has been research that focused on Android malware detection, two major issues remain unresolved. First, which training or classification methods should be chosen to achieve better performance. Second, which features or feature categories have better classification ability with limited dimensions. Neither of them has clear conclusions currently.

Deep learning allows multiple-layer processing models to learn the data representations with multiple levels of abstraction [25]. The deep learning theory was proposed by Hinton and Salakhutdinov in 2006 [16], and is now a new research field in machine learning. Deep learning techniques construct high level abstract features that can be used to discover the distributed feature representations. The first typical multiple-layer learning structure is CNN, which was presented by Y. Lecun et al. [5]. It reduces parameters by the features' spatial relationship to improve training effectiveness. Besides, G. Hinton et al.

* Corresponding author.

E-mail address: dixiaoqiang@cust.edu.cn

implemented Deep Belief Network (DBN) [15] and Deep Boltzmann Machine (DBM) [30], which are also widely used deep learning algorithms. Deep learning has dramatically improved image recognition, speech recognition, visual object recognition and other domains.

(1) In this work, we are motivated to solve the above two issues with deep learning techniques. The major contributions of the work reported include: (1) An Android malware detection framework Andro_MD is presented, which employs CNN to train and classify Android apps. The features of the Android apps considered have functional relations, which is the theoretical basis of the employment. (2) An Android app dataset that includes 21,000 apps and 34,570 features is constructed. The samples are collected from popular third-party markets in China and are labeled as negative and malicious by VirusTotal [35]. The extracted features are divided into 7 categories in order to compare their classification abilities. (3) The classification abilities of different feature categories are evaluated and compared. Features in different categories are evaluated independently to find the balance of detection performance and feature dimension.

The rest of this paper is organized as follows: Section 2 introduces related work about Android malware detection and deep learning techniques; Section 3 describes the theory and development of CNN; Section 4 illustrates the proposed Andro_MD framework and feature preprocessing; Section 5 shows experimental results and Section 6 discusses conclusions.

2. Related Work

2.1. Android malware detection

According to the Android OS architecture, security problems of Android include not only the traditional threats related to hardware, software and applications, but also the particular vulnerabilities of the Android security mechanism. Zhou and Jiang [42] divided Android malware into four categories: malicious software installation, malicious software operation, malicious loading and malicious permission usage.

Android malware detection techniques include static analysis and dynamic analysis. The static method extracts features and detects malicious apps under non-running conditions and analyzes the source code by decompiling apk files. Androguard is an open source backward analysis tool that can be used in static feature extraction and analysis. Research about static analysis includes Kirin [11], SCanDroid [31], DroidChecker [7], Chex [22], and so on. Dynamic technique monitors the data flow and system state at runtime with listener mechanisms. DroidBox is a dynamic privacy leak detection tool based on TaintDroid and marks the taint source in the sandbox to analyze malicious behaviors. Related research includes Stowaway [13], Quire [9], Asdroid [18], XManDroid [39], and so on.

In order to understand Android permissions, in our previous work we explored the permission-induced risk in Android apps on three levels in a systematic manner and used them in malware detection [38]. A system called Alde [23] was presented to analyze the privacy risk of analytic libraries in Android ecosystems. In addition to Android malware detection, we proposed a framework with different classifiers to automatically categorize benign apps [37].

2.2. Machine learning

Machine learning algorithm is often used in data processing and classification of Android malware detection. The features extracted from static and dynamic methods can be processed by machine learning algorithms. Different kinds of features and classifiers have been used in this field. D. Barrera et al. [4] used SOM to construct an Android malware permission model and detected malicious apps. A. Shabtai et al. [29] applied feature selection and classification algorithms to detect malicious samples. Wu et al. [36] conducted a thorough analysis to extract data flow-related API-level features and implemented the K-Nearest Neighbor classification model. They used 1,160 benign and 1,050 malicious samples to evaluate the presented methods, and the result achieved 97.66% accuracy. Baltaci and Kamil [3] chose Naive Bayesian as the supervised learning algorithm for the detection task. They used market metadata that included application category, download number, developer name and average rating as the processing features.

In order to compare different algorithms, Chang et al. [8] collected behaviors such as network activities, file read/write and permissions as features and used different machine algorithms to classify malware and evaluate their performances. Experiments showed that Random Forest could achieve the best accuracy in RF, J48, RandomCommittee, Bagging, and IBK (KNN). Combined with feature selection algorithms, MZ Masud et al. [27] showed the best overall result was obtained by MLP (Multi-Layer Perception). Although many researches are concerned about the detection features and classifiers, S. Qing [28] pointed out that there are still some problems with no perfect solutions, such as which features should be extracted, how many features should be enrolled, and which classifiers should be selected to achieve better performance.

2.3. Deep learning

As a new research field of traditional machine learning, deep learning is widely used in image and speech recognition, text classification and many other applications. Y. Sun et al. [33] proposed a hybrid ConvNet-RBM model for face verification, which used high-level relational visual features with rich identity similarity information. To characterize face similarities from different aspects, they concatenated the features extracted from different face region pairs by different deep ConvNets. WL. Hou et al. [19] investigated how to blindly evaluate the visual quality of an image by learning rules from linguistic descriptions. They proposed a blind IQA model that learned qualitative evaluations directly and output numerical scores for general utilization and fair comparison. Zhang et al. [40] proposed a deep ensemble method to address monaural speech separation as well as solve the problem where a single DNN with a given window length does not leverage contextual information sufficiently. Chen and Mak [5] proposed a Multitask-Learning (MTL) approach to improve low-resource automatic speech recognition using deep neural networks without requiring additional language resources. The proposed MTL obtained significant word recognition gains when compared with the single-task learning (STL) of the corresponding DNNs or ROVER.

In recent years, there has been research that employed deep learning in text and word processing. The usage of deep learning effectively improves emotional analysis, lexical analysis and word identification. I. Chaturvedi et al. [6] proposed a deep recurrent belief network with distributed time delays for learning multivariate Gaussians. It was possible to learn the long delays from short delays in a hierarchical manner. The proposed VBN framework was applied for modeling word dependencies in text and it can achieve over 30% improvement in accuracy on real-world scenarios compared to the baselines. A Chinese Emergency Event Recognition Model (CEERM) [41] based on deep learning achieves excellent recognition performance with a maximum F-measure value of 85.17%.

However, deep learning is not widely used in network security and mobile security, which is mainly due to the feature formats. The typical image data and text data have time or spatial relations, which is the foundation of deep learning employment. However, traditional security data, such as IP addresses, port numbers and time stamps, are independent of each other. But, there are still some attempts. In malware detection, David and Netanyahu [10] presented a novel method based on deep learning for automatic malware signature generation and classification. The results showed that signatures generated by the DBN allow for an accurate classification of new malware variants. Android malware features are different from traditional security features as they have functional level relations. DroidDetector [43] utilized 200 features extracted from Android apps for malware detection, and the results demonstrated DBN can achieve a high accuracy.

In this work, we are motivated to find out the effective classifiers and feature categories in Android malware detection and achieve better classification performance. Unlike existing work, we evaluate the effectiveness and feasibility of employing different CNN models in Android malware detection. We present the framework and implement CNN with parallel and sequential models. Then, we compare the classification abilities of 7 different feature categories. The selected CNN model and features can be used in real practice and relative work.

3. Convolutional Neural Networks

In the 1960s, works by Hubel and Wiesel [17] showed that cat and monkey visual cortexes contain neurons that individually respond to small regions of the "visual field". In 1980, Fukushima [12] introduced "neocognitron", which did not require units located at several network positions to have the same trainable weights. In 1989, back-propagation was applied to neocognitron-like, weight-sharing, convolutional neural layers with adaptive connections [24]. This work also introduced the MNIST dataset of handwritten digits, which has become the most famous benchmark of Machine Learning [20]. Then, in 1998, Y. LeCun et al. [26] improved the convolutional neural networks with gradient-based learning, which pushed the research of CNN to the climax.

CNN is a special deep-layer neural network and has the stability of deformation, such as translation, scaling and zooming. CNN reduces parameters and keeps stability by "Local receptive fields", "Weight sharing" and "Down-sampling".

(1) Local receptive fields. A neuron gets input from the "local accepted domain" of the upper layer to extract the "local domain features". When a new feature is extracted, it maintains the relative position with other features. Thus, it keeps the image deformation invariance.

(2) Weight sharing. Each filter covers the whole visual field, which is also called the convolution kernel. All sharing units of a filter form a feature map. Weight sharing is reasonable because the repeat units recognize features without considering the position. What's more, it improves the feature extraction efficiency with reduction of variables.

(3) Down-sampling. A convolutional layer is followed by a down-sampling layer to reduce the resolution of mapping, which reduces the deformation sensitivity and the parameter numbers.

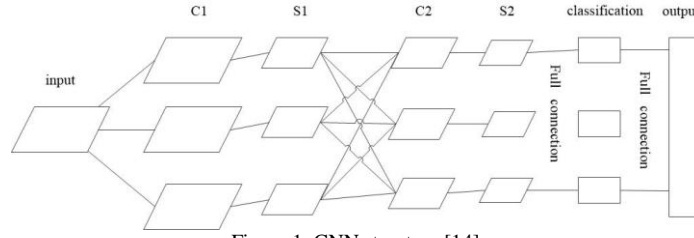


Figure 1. CNN structure [14]

A typical two-layer CNN structure is shown in Figure 1. Input is the original image or the source dataset. C1 is the first following convolution layer. It uses an activation function, such as Sigmoid, Tanh or ReLu, to realize the first feature extraction. There are 3 convolutional kernels, and then it gets three feature maps in C1. Next, S1 is the sampling layer that diminishes the feature maps and reduces parameters. Then, the process is iterated with C2 and S2. The output of S2 is flattened by a full connection, and finally, we get the results by the classification algorithm. Set x as the input vector, $J * I$ is the kernel, b is the bias, f is an activation function, then output $m * n$ is calculated by equation (1):

$$y_{mn} = f\left(\sum_{j=0}^{J-1} \sum_{i=0}^{I-1} x_{m+i, n+j} w_{ij} + b\right), (0 \leq m \leq M, 0 \leq n \leq N) \quad (1)$$

The pooling layer takes samples from the upper feature map to reduce its size. The pooling equation with $S_1 * S_2$ is:

$$y_{mn} = \frac{1}{S_1 S_2} \sum_{j=0}^{S_2-1} \sum_{i=0}^{S_1-1} x_{(m \times S_1 + i, n \times S_2 + j)} \quad (2)$$

In equation (2), x is the input and y is the pooling output. The pooling layer reduces calculation complexity and improves the model's robustness in order to keep the invariance of deformation.

The basic training process of CNN is described as above. There are different techniques to implement the structure in real practice. LeNet-5 and ImageNet [21] are the widely used models. They employ flexible layers, parameters and activation functions, such as ReLu, LRN and Softmax, to realize recognition and classification in different fields.

4. Framework of Andro_MD

4.1. Overview of the framework

By employing CNN in Android malware detection, we try to extract features that can give better representation of the samples and have better classification abilities. The proposed Andro_MD framework is shown in Figure 2. It includes three parts: dataset construction, classification and evaluation.

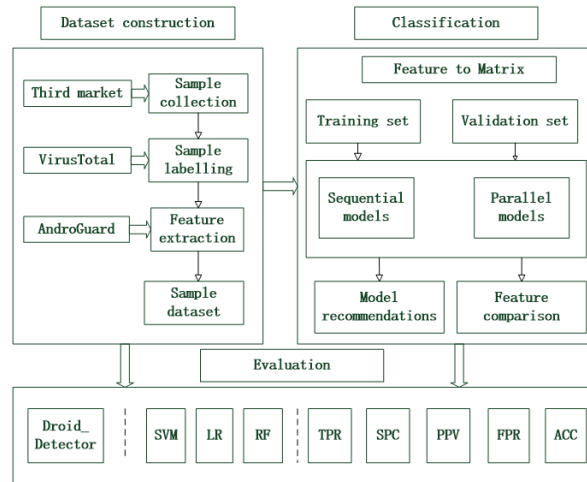


Figure 2. Andro_MD Framework

(1) Dataset construction. The dataset construction process includes samples collected from the third-party markets, samples labeling and features extraction. We use VirusTotal to scan the apps and label them as malicious and negative. The apps' static features are extracted by AndroGuard and divided into 7 categories.

(2) Classification. Android features have to be preprocessed to fit the convolutional process. The feature vectors are reshaped to matrices by embedding, and then the dataset is divided into a training set and validation set. Different CNN parallel and sequential models are employed to train the features, and classifiers are used to detect malware.

(3) Evaluation. The CNN results are compared with similar work "DroidDetector", which is another Android malware detection practice using DBN. Then, we conduct comparisons with traditional algorithms including SVM, LR and RF. Indexes such as Accuracy (ACC), True Positive Rate (TPR) and False Positive Rate (FPR), are used to give precise evaluations. The effectiveness and feasibility of the presented work are also evaluated.

4.2. Feature extraction

Android apps' features can be divided into platform-based and customized. Platform-based features are provided officially and can be applied by developers to implement some common functions. Customized features are sample-dependent since developers can define different features for different apps. In this work, we extract platform-based features with the consideration of reducing the dependency of samples in order to get conclusions. The 7 feature categories are described as follows.

(1) Suspicious API calls. They describe 15 kinds of suspicious operations that an app may execute. These API calls are sensitive operations in smartphones, such as "access device ID ()", "sending SMS ()" and "receiving SMS ()".

(2) Code related patterns. These 5 features are referred when an app dynamically loads external executable files, and it can be used to detect whether an app has some defined behaviors. The 5 code related patterns are: whether to load .dex files, whether to load Linux native codes, whether to execute shell commands, whether to employ Java reflection techniques and whether to invoke cryptographic functions.

(3) Hardware features. They are extracted in manifest files with <user-feature> elements and show the hardware that one app employs, such as "bluetooth", "touchscreen" and "camera".

(4) Filtered intents. Android uses "intent" as a message object, and the intent is used to request an action or process from other apps. For instance, an app may declare the intent "BOOT_COMPLETED" to activate a specific system event.

(5) Requested permissions. These permissions are requested by apps from the Android platform, and they can characterize the resource accessing of apps. For example, "receive_SMS", "send_SMS", and "read_SMS" are the typical permissions that can reflect behaviors related with a short message.

(6) Restricted API calls. Combined with used permissions, they can indicate the resources that apps actually access. The restricted API calls are collected by scanning the disassembled code for apk files, and whether they invoke the protected API calls or not are recorded as features.

(7) Used permissions. They are obtained by the API-permission mapping provided by PScout [2]. Used permissions are the employed permissions of the apps and have differences with the requested permissions, which may reflect some information of the malicious behaviors.

4.3. Feature conversion

Considering the training process, CNN and traditional Neural Network algorithms have different demands on feature relations and feature formats. The dataset to be trained by CNN needs to have relations, which is the foundation of "Local Receptive Fields" to maintain deformation invariance. In text dataset, the extracted features have a semantic relation. There are some functional relations between Android features. For instance, if an app applies the permission "wifi" and "network", it may also employ the hardware feature "bluetooth" and API call "send_SMS ()". These relations between features are the

foundation of CNN employment. Android features are processed by three steps: padding feature sentences, creating feature vectors and constructing feature matrices.

(1) Padding feature sentences. In order to make features into the same length, we calculate the highest dimension of the features and take it as the standard length. We then fill other feature sequences at the end with padding words. Features with the same length are used to create feature vectors.

(2) Creating feature vectors. We make statistics of all the features, remove duplicates, and build a feature alphabet. Thus, every feature item possesses a serial number. Then, the features are mapped to their numbers in the alphabet, and the numerical feature vectors are created.

(3) Constructing feature matrices. By the embedding technique, the feature vectors are converted to feature matrices that fit the deep training process.

After constructing the dataset, we conduct the feature extraction and feature conversion to make the features matrices. The dataset includes benign and malicious samples, and the features are divided into 7 categories. In the experiment section, we evaluate the CNN performance when using it in Android malware detection and compare the classification abilities of different feature categories.

5. Experiments

5.1. Experiment description

(1) Machine environment. HP ProLiant DL380 with 64GB memory is used to run the experiments, and Keras (v1.1.0) is used as the deep learning platform to implement the experiments.

(2) Dataset. The benign apps are collected from six Chinese third-party markets include AnZhi, AppChina, MyApp, LenovoMarket, GFan and NDuo. The samples are scanned by VirusTotal and labeled as benign if there is no alarm. The malicious samples are collected from the Android Malware Genome Project (AMGP) [37], malware repository VirusShare [34], Drebin dataset [1] and antivirus companies. These malicious samples are also scanned and confirmed by VirusTotal and labeled as malicious. With the extracted features by AndroGuard, we construct the dataset with 20,000 benign apps, 1,000 malicious apps and 34,570 features.

(3) Evaluation indexes. We use 5 indexes to evaluate the work, which include: True Positive Rate (TPR), False Positive Rate (FPR), Accuracy (ACC), Specificity (SPC), Positive Predictive Value (PPV).

5.2. Experiments

(1) Android malware classification

The typical CNN structure is sequential as in Figure 1 of Section 3. In that structure, the output of an upper layer is the input of the current layer, and the output of the current layer is the input of the following layer. There is a parallel structure where each training layer receives the same input but processes using different kernels, and the final output is combined by the individual outputs. The sequential and parallel structures with three layers are shown in Figure 3.

In this experiment, we use 1,000 benign and 1,000 malicious samples where 20% is for validation to compare the performances of sequential and parallel models. The kernel in sequential model is 5×5 , and in the parallel model is $n \times 256$. The results are in Table 1 and the ROC curves are in Figure 4 and Figure 5.

The 9 groups in Table 1 show the 5 evaluation indexes of the sequential and parallel models. "S" indicates "Sequential" and "P" indicates "Parallel". S_L1 means a sequential model with 1 layer. S_L5 achieves the best classification result with ACC as 99.25% and FPR as 0.53%. P_L4 follows with the ACC as 98.5%.

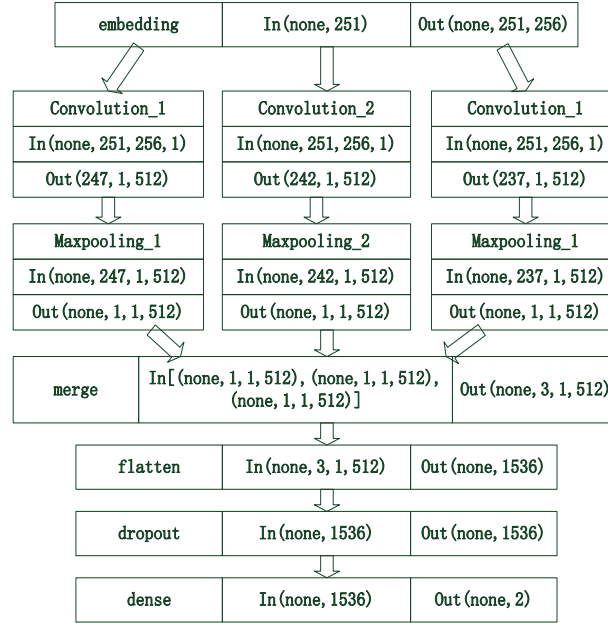


Figure 3(a). Parallel model of CNN



Figure 3(b). Sequential model of CNN

Table 1. Comparison of sequential and parallel models

No.	Name	Kernels	FPR (%)	TPR (%)	ACC (%)	SPC (%)	PPV (%)	PredictT (s)
1	S_L1	5*5	4.21	96.67	96.25	95.79	96.21	13.23
2	S_L2	5*5	2.63	97.62	97.50	97.37	97.62	45.17
3	S_L3	5*5	1.58	97.62	98.00	98.42	98.56	52.70
4	S_L4	5*5	1.58	99.05	98.75	98.42	98.58	54.31
5	S_L5	5*5	0.53	99.05	99.25	99.47	99.52	54.28
6	P_L2	(5*256),(10*256)	1.05	97.62	98.25	98.95	99.03	0.73
7	P_L3	(5*256),(10*256), (15*256)	1.05	97.62	98.25	98.95	99.03	1.28
8	P_L4	(5*256),(10*256), (15*256),(20*256)	1.05	98.10	98.50	98.95	99.04	2.04
9	P_L5	(5*256),(10*256), (15*256),(20*256), (25*256)	2.11	97.14	97.50	97.89	98.08	2.93

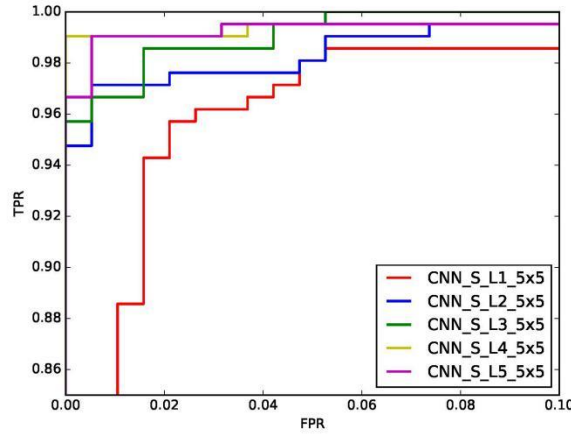


Figure 4. The ROC of sequential model

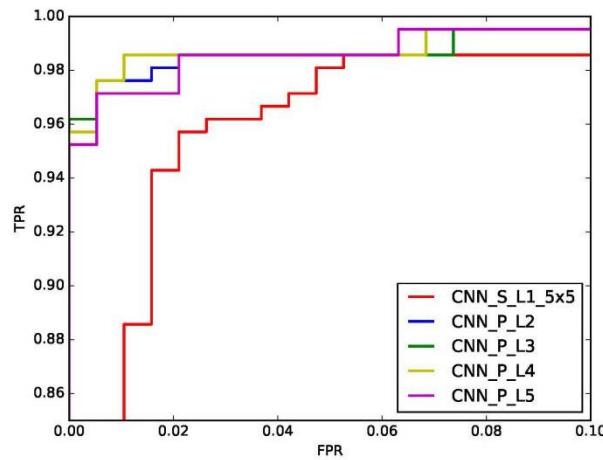


Figure 5. The ROC of parallel model

For a certain FPR, a higher TPR means better classification results. Thus, in Figure 4, S_L4 and S_L5 have better performances. In Figure 5, except the 1-layer group, the ROC curves of others are similar. P_L3 has a higher TPR with a lower FPR. With the rise of FPR, P_L4 shows a better overall performance. When FPR is between 0.02 and 0.06, the performance of P_L2, P_L3, P_L4 and P_L5 tend to be similar. With overall consideration, S_L5 and P_L4 have better performances.

Time consumption is also an important issue to be considered. The training time and prediction time of deep learning algorithms are related to many factors such as implementation platforms, parameters and hardware environments. Since the training process does not need to be executed every time, we just compare the prediction time in this work. The "PredictT" column in Table 1 shows the prediction time under this experimental environment. The parallel models consume less time than the sequential models because of the different amounts of parameters to be calculated. The average time of parallel models is 1.75 seconds, while the average time of sequential models is 43.84 seconds. Although the sequential models have a better overall TPR and ACC, in real practice, the parallel models are also a good choice with the consideration of time consumption.

(2) Comparison with similar work

In the real practice of Android malware detection, it is difficult to collect adequate malicious samples for training. In this step, we compare the classification results with different malicious and benign ratios. The experiment refers to a similar work "DroidDetector" [44], in which DBN is used to implement the deep learning process. CNN_P_L4 is used as a classifier, and the comparison results are shown in Table 2.

In Table 2, "Ma/Be" indicates the sample ratio of malicious to benign. The training and validation data account for 50%. For example, the first group has 200 malicious and 200 normal samples, in which 100 malicious ones and 100 negative ones are used for training. We compare the evaluation indexes of P_L4 of Andro_MD and DBN of DroidDetector.

From Table 2, P_L4 achieves a better overall ACC and TPR than DroidDetector. Its best accuracy achieves 97.10%, which shows the proposed method has an improved Android malware detection.

The best TPR appears in the third group where 2,000 samples are enrolled in this step and the ratio is 1:1. This is also the case for DroidDetector because the benign and malicious samples are trained sufficiently. But in real practice, it is difficult to collect as many malicious samples as benign ones. When the ratio is 1:100, Andro_MD can achieve 81% TPR and the classification accuracy is 99.80%.

Table 2. Results of Andro_MD and DroidDetector

Po/Be	Dataset	ACC (%)		TPR (%)		FPR (%)		PPV (%)	
	Ma/Be	Andro_MD	Droid-Detector	Andro_MD	Droid-Detector	Andro_MD	Droid-Detector	Andro_MD	Droid-Detector
1:1	200/200	95.50	94.50	95.00	94.00	4.00	5.00	95.96	94.95
1:1	400/400	96.25	95.75	96.50	95.50	4.00	4.00	96.02	95.98
1:1	1000/1000	97.10	96.60	97.00	95.60	2.80	2.40	97.19	97.55
1:2	200/400	96.67	95.33	94.00	90.00	2.00	2.00	95.92	95.74
1:5	200/1000	98.17	97.17	90.00	90.00	0.20	1.40	98.90	92.78
1:10	200/2000	98.55	97.82	86.00	78.00	0.20	0.20	97.72	97.50
1:20	200/4000	99.19	97.86	83.00	59.00	0.0	0.20	100.0	93.65
1:50	200/10000	99.61	99.24	84.00	80.00	0.08	0.38	95.45	80.81
1:100	200/20000	99.80	99.54	81.00	62.00	0.01	0.08	98.78	88.57

(3) Comparison with other classifiers

In order to compare CNN with traditional machine learning algorithms, we compare S_L5, P_L3 with linear SVM, Logistic Regression (LR) and Random Forest (RF). These algorithms are typical classifiers and are widely used in many fields. The dataset used is the same as in Step 1. Results are shown in Table 3 and Figure 6.

Table 3. Results of five algorithm

Name	FPR (%)	TPR (%)	ACC (%)	SPC (%)	PPV (%)
S_L5	0.53	99.05	99.25	99.47	99.52
P_L3	1.05	97.62	98.25	98.95	99.03
SVM	1.40	97.80	98.20	98.60	98.59
LR	0.70	95.80	97.55	99.30	99.27
RF	2.98	97.31	97.15	97.02	97.02

In Table 3, the indexes are calculated when the classification threshold is 0.5. From the overall accuracy, S_L5 still achieves the best at 99.25%, while SVM can reach closely at 98.2% but with a higher FPR. These results show that CNN has advantages in classification performance when being used in Android malware detection. Figure 6 shows ROC curves of the five algorithms. Two deep learning algorithms achieve better TPR with lower FPR. When FPR is higher than 2%, the five TPRs tend to be similar.

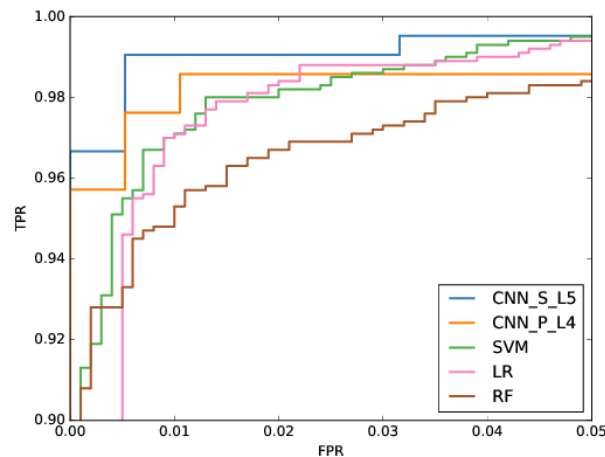


Figure 6. The ROC of five algorithms

(4) Comparison of feature categories

In real practice, it is very complex if we have to extract the total 34,570 features every time. We want to find out which feature categories have good classification ability with limited dimensions. In this step, we compare 7 feature categories with 5 evaluation indexes. The samples in Step 1 and P_L3 are used in this step. Results are shown in Table 4.

Table 4. Results of 7 feature categories

No.	Features	Dimension	FPR (%)	TPR (%)	ACC (%)	SPC (%)	PPV (%)
FS_1	Suspicious API calls	15	7.37	88.57	90.50	92.63	0.93
FS_2	Code related patterns	5	26.32	68.57	71.00	73.68	0.74
FS_3	Hardware features	41	16.32	75.24	79.25	83.68	0.84
FS_4	Filtered intents	132	14.74	73.33	79.00	85.26	0.85
FS_5	Request permissions	93	3.68	96.19	96.25	96.32	0.97
FS_6	Restricted API calls	34188	1.58	97.14	97.75	98.42	0.99
FS_7	Used permissions	96	5.79	94.76	94.50	94.21	0.95

The best result of 97.75% ACC is obtained by FS_6. But, Restricted API calls include 34,188 dimensions, which is more than 98% of the total. Besides, "requested permissions (FS_5)" and "used permissions (FS_7)" can achieve the 96.25% and 94.5% ACC separately, and they contain 93 and 96 features respectively. Therefore, considering the feature dimensions and classification results, "requested permissions" and "used permissions" are recommended since they can achieve satisfying results with limited dimensions.

6. Conclusions

When traditional machine learning algorithms are used in Android malware detection, the main issues are which classifiers and features should be selected to achieve better performances. In this work, we design Andro_MD, which employs different CNN models in Android malware detection. We implement it with dataset construction and feature preprocessing, training and classification, experimental comparison and evaluation. We construct a labeled dataset that includes 21,000 samples and 34,570 features. In order to make the feature format fit to CNN, the feature preprocessing includes padding feature sequences, creating feature vectors and constructing feature matrices.

Experimental results show the effectiveness and feasibility of the proposed Andro_MD. By comparison of CNN parallel and sequential models, we find that CNN_S_L5 achieves the best 99.25% accuracy on our dataset. The proposed work is compared with DroidDetector and traditional machine learning algorithms SVM, LR and RF, which show that CNN obtains the best overall performance. Considering the classification accuracy and feature dimensions, we find that "requested permissions" and "used permissions" have better performances with 96.25% and 94.5% accuracy individually respectively.

The presented framework and implementation techniques in this work can also be used in Android malware family recognition and Android app classification. However, there are still some defects in this work. Sufficient comparisons about different parameters such as convolution kernel sizes and different activate functions should be conducted. We will research these aspects in the future.

Acknowledgements

This work was partly financially supported through grants from the Project Development Plan of Science and Technology in Jilin Province (No. 20150204081GX).

References

1. D. Arp, M. Spreitzenbarth, M. Hbner, et al. "Drebin: Effective and Explainable Detection of Android Malware in Your Pocket," in: Proc. NDSS'14, 2014.
2. K. Au, Y. Zhou, Z. Huang, et al. "PScout: Analyzing the Android Permission Specification," in: Proc. of CCS'12, pp. 217-228, 2012
3. A. N. Baltaci, A. Kamil. "AntiWare: An Automated Android Malware Detection Tool Based on Machine Learning Approach and Official Market Metadata," in: Proc. IEEE UEMCON, pp.20-22, 2016
4. D. Barrera, H. Kayacik, P. Oorschot, et al. "A Methodology for Empirical Analysis of Permission-based Security Models and Its Application to Android," in: Proc. CCS 10, pp. 73-84, 2010
5. D. P. Chen, B. Mak. "Multitask Learning of Deep Neural Networks for Low-Resource Speech Recognition," IEEE-ACM Trans. on Audio Speech and Language Processing, vol. 23, no. 7, pp.1172-1183, 2015
6. I. Chaturvedi, Y. S. Ong, I. W. Tsang, et al. "Learning Word Dependencies in Text by Means of A Deep Recurrent Belief Network,"

- Knowledge-based Systems, vol. 108, no. SI, pp. 144-154, 2016
7. P. Chan, L. Hui, S. Yiu. "Droidchecker: Analyzing Android Applications for Capability Leak," in: Proc. WISEC'12, pp. 125-136, 2012
 8. W. Chang, H. Sun, W. Wu. "An Android Behavior-based Malware Detection Method using Machine Learning," in: Proc. IEEE ICSPCC, pp.5-8, 2016
 9. M. Dietz, S. Shekhar, Y. Pisetsky et al. "Quire: Lightweight Provenance for Smart Phone Operating Systems," in: Proc. USENIX Security Symp., pp.23-23, 2011
 10. O. E. David, N. S. Netanyahu. "DeepSign: Deep Learning for Automatic Malware Signature Generation and Classification," in: Proc. IJCNN, pp.1-8, 2015
 11. W. Enck, M. Ongtang, P. McDaniel. "On Lightweight Mobile Phone Application Certification," in: Proc. CCS 09, pp. 235-245, 2009
 12. K. Fukushima. "Neocognitron: A Self-organizing Neural Network Model for A Mechanism of Pattern Recognition Unaffected by Shift in Position," Biological Cybernetics, vol. 36, no. 4, pp. 193-202, 1980
 13. P. Felt, E. Chin, S. Hanna, et al. "Android Permissions Demystified," in: Proc. CCS 11, pp. 627-638, 2011
 14. C. Gao, Y. Cong, Y. Zheng, et al. "Recording Equipment Identifying Research Based on Convolution Neural Networks," Informatization Research, vol. 42, no. 2, pp. 51-54, 2016
 15. G. Hinton, S. Osindero, YW. Teh. "A Fast Learning Algorithm for Deep Belief Nets," Neural Computation, vol. 18, no. 7, pp. 1527-1554, 2006
 16. G. Hinton, R. Salakhutdinov. "Reducing the Dimensionality of Data with Neural Networks," Science, vol. 313, no. 5786, pp. 504, 2006
 17. H. Hubel, N. Wiesel. "Receptive Fields and Functional Architecture of Monkey Striate Cortex," The Journal of Physiology, vol. 195, no. 1, pp. 215-243, 1968
 18. J. Huang, X. Zhang, L. Yan, et al. "AsDroid: Detecting Stealthy Behaviors in Android Applications by User Interface and Program Behaviors Contradiction," in: Proc. ICSE, pp. 1036-1046, 2014
 19. W. Hou, X. Gao, D. Tao, et al. "Blind Image Quality Assessment via Deep Learning," IEEE Trans. on Neural Networks and Learning Systems, vol. 26, no. 6, pp.1275-1286, 2015
 20. S. Jurgen. "Deep Learning in Neural Networks: An Overview," Neural Networks, vol. 61, pp. 85-117, 2015
 21. A. Krizhevsky, I. Sutskever, G. Hinton. "Imagenet Classification with Deep Convolutional Neural Networks," in: Proc. NIPS, pp. 1097-1105, 2012
 22. L. Lu, Z. Li, Z. Wu, et al. "Chex: Statically Vetting Android Apps for Component Hijacking Vulnerabilities," in: Proc. CCS 12, pp.229-240, 2012
 23. X. Liu, S. Zhu, W. Wang, et al. "Alde: Privacy Risk Analysis of Analytics Libraries in the Android Ecosystem," in: Proc. 12th SecureComm., pp.10-12, 2016
 24. Y. LeCun, B. Boser, J. S. Denker, et al. "Back-propagation Applied to Handwritten Zip Code Recognition," Neural Computation, vol. 1, no. 4, pp. 541-551, 1989
 25. Y. LeCun, Y. Bengio, G. Hinton. "Deep Learning," Nature, vol. 521, no. 7553, pp. 436-444, 2015
 26. Y. LeCun, B. Leon, B. Yoshua, et al. "Gradient-based Learning Applied to Document Recognition," in: Proc. of the IEEE, vol. 86, no. 11, pp. 2278-2324, 1998
 27. M. Masud, S. Sahib, M. Abdollah, et al. "Analysis of Features Selection and Machine Learning Classifier in Android Malware Detection," in: Proc. ICISA, pp.1-5, 2014
 28. S. Qing. "Research Progress on Android Security," Journal of Software, vol. 27, no.1, pp. 45-71, 2016
 29. A. Shabtai, U. Kanonov, Y. Elovici, et al. "Andromaly: A Behavioral Malware Detection Framework for Android Devices," Journal of Intelligent Information Systems, vol.38, no. 1, pp. 161-190, 2012
 30. R. Salakhutdinov, G. Hinton. "Deep Boltzmann Machines," in: Proc. AISTATS, pp. 448-455, 2009
 31. "SCanDroid: Automated Security Certification of Android Applications," Available at https://www.researchgate.net/publication/228847936_SCanDroid_Automated_security_certification_of_Android_applications, Last accessed on August 7, 2017
 32. "Smartphone OS Market Share," Available at <http://www.idc.com/prodserv/smartphone-os-market-share.jsp>, Last accessed on August 1, 2016
 33. Y. Sun, G. Wang, O. Tang. "Hybrid Deep Learning for Face Verification," IEEE Trans. on Pattern Analysis and Machine Intelligence, vol. 38, no. 10, pp. 1997-2009, 2016
 34. "VirusShare," Available at <http://virusshare.com>, Last accessed on April 4, 2016
 35. "VirusTotal," Available at <https://www.virustotal.com/zh-cn/>, Last accessed on January 1, 2016
 36. S. Wu, P. Wang, Y. Zhang. "Effective Detection of Android Malware Based on the Usage of Data Flow APIs and Machine Learning," Information & Software Technology, vol.75 , pp.17-25, 2016
 37. W. Wang, Y. Li, X. Wang, et al. "Detecting Android Malicious Apps and Categorizing Benign Apps with Ensemble of Classifiers," Future Generation Computer Systems (online first), 2017
 38. W. Wang, X. Wang, D. Feng, et al. "Exploring Permission-induced Risk in Android Applications for Malicious Application Detection," in: Proc. IEEE Trans. on Information Forensics and Security, pp.1869-1882, 2014
 39. "Xmandroid: A New Android Evolution to Mitigate Privilege Escalation Attacks. Ruhr-University Bochum," Available at https://www.researchgate.net/publication/228960321_XManDroid_A_New_Android_Evolution_to_Mitigate_Privilege_Escalation_Attacks, Last accessed on October 5, 2017
 40. X. L. Zhang, D. L. Wang. "A Deep Ensemble Learning Method for Monaural Speech Separation," IEEE-ACM Trans. on Audio Speech and Language Processing, vol. 24, no. 5, pp. 967-977, 2016

41. Y. Zhang, Z. Liu, W. Zhou. "Event Recognition Based on Deep Learning in Chinese Texts," PLOS ONE, vol.11, no. 8, 2016.
42. Y. Zhou, X. Jiang. "Detecting Android Malware: Characterization and Evolution," in: Proc. of IEEE Symposium on SP, pp. 95-109, 2012
43. Z. L. Yuan, Y. Q. Lu, Z. G. Wang, et al. "Droid-Sec: Deep Learning in Android Malware Detection," ACM Sigcomm Computer Communication Review, vol. 44, no. 4, pp. 371-372, 2014
44. Z. L. Yuan, Y. Q. Lu, Y. B. Xue. "DroidDetector: Android Malware Characterization and Detection Using Deep Learning," Tsinghua Science and Technology, vol.21, no.1, pp.114-123, 2016