Journal of Ambient Intelligence and Humanized Computing
https://doi.org/10.1007/s12652-018-0803-6

ORIGINAL RESEARCH

Effective android malware detection with a hybrid model based on deep autoencoder and convolutional neural network

Wei Wang[1,2] · **Mengxue Zhao**[1] · **Jigang Wang**[3]

Received: 27 December 2017 / Accepted: 14 April 2018
© Springer-Verlag GmbH Germany, part of Springer Nature 2018

Abstract
Android security incidents occurred frequently in recent years. To improve the accuracy and efficiency of large-scale Android malware detection, in this work, we propose a hybrid model based on deep autoencoder (DAE) and convolutional neural network (CNN). First, to improve the accuracy of malware detection, we reconstruct the high-dimensional features of Android applications (apps) and employ multiple CNN to detect Android malware. In the serial convolutional neural network architecture (CNN-S), we use Relu, a non-linear function, as the activation function to increase sparseness and "dropout" to prevent over-fitting. The convolutional layer and pooling layer are combined with the full-connection layer to enhance feature extraction capability. Under these conditions, CNN-S shows powerful ability in feature extraction and malware detection. Second, to reduce the training time, we use deep autoencoder as a pre-training method of CNN. With the combination, deep autoencoder and CNN model (DAE-CNN) can learn more flexible patterns in a short time. We conduct experiments on 10,000 benign apps and 13,000 malicious apps. CNN-S demonstrates a significant improvement compared with traditional machine learning methods in Android malware detection. In details, compared with SVM, the accuracy with the CNN-S model is improved by 5%, while the training time using DAE-CNN model is reduced by 83% compared with CNN-S model.

Keywords Deep learning · Convolutional neural network · Autoencoder · Malware detection · Android applications

1 Introduction

Smart terminals have become fundamental personal equipment. More and more customers use smart phones to get a wealth of information and contact with each other. On the one hand, the openness of Android markets plays an important role in the popularity of Android applications (apps). On the other hand, Android apps have become the target of many attackers. According to the 2016 China Internet Security Report (China Internet Security 2016), a total of 14.03 million new malware samples on Android platform have been intercepted by 360 Internet Security Center. Ransomware started to break out on mobile phones. Throughout the year, 360 corporation intercepted 0.17 million new ransomware samples on mobile phones, attacking 1.70 million mobile phones (China Internet Security 2016). In 2017, the number of infections in Android platform is expected to grow ten times revealed by the security report. Nowadays, how to process the big data in security has become more and more important (Hamedani et al. 2018; Wu et al. 2016a, b; Atat et al. 2017). Research on modern cryptographic solutions for computer and cyber security is becoming increasingly popular (Ibtihal et al. 2017; Gupta et al. 2016). With the increasing threats of Android malware, it is urgent to develop effective malware detection methods that help to keep the threats out of individuals and the markets.

In recent years, machine learning models have been widely employed in malware detection. These models can learn the distinctions between malicious and benign apps. Deep learning is a relatively new area in machine learning research. In deep learning methods, the low-level-layers

✉ Wei Wang
wangwei1@bjtu.edu.cn

Mengxue Zhao
mxzhao@bjtu.edu.cn

Jigang Wang
wang.jigang@zte.com.cn

[1] Beijing Key Laboratory of Security and Privacy in Intelligent Transportation, Beijing Jiaotong University, Beijing, China

[2] Science and Technology on Electronic Information Control Laboratory, Chengdu, China

[3] ZTE Corporation, Chengdu, China

Published online: 28 April 2018

extract fine features, while high-level-layers exhibit higher-layered features. As one of the major models in deep learning, convolutional neural network (CNN) has been popularly used for image recognition (Li et al. 2018b) and shown promising performance in contextual categorization. In this work, we are motivated to detect Android malicious apps with two different structures of CNN. We also propose a new pre-training strategy DAE to learn more suitable feature representations. The proposed approach improves both the efficiency and accuracy of Android malware detection compared with basic CNN and machine learning methods.

We make the following contributions:

- We use CNN with different structures to improve the detection accuracy in Android malware detection. The experimental results show that CNN-S can reach 99.8% prediction accuracy that is 5% higher than SVM.
- We develop two different CNN architectures. The experimental results are different with different CNN architectures. We finally use the model CNN-S and the experimental results show that the detection accuracy with CNN-S is higher than that with CNN-P (99.80–99.82%) and is improved by 3% compared with basic CNN.
- We propose using DAE as a pre-training method of CNN-S to reduce the training time. We add the sparse rules to pre-train the models and use Relu, the non-linear function as the activation function, which can efficiently extract abstract features. Extensive experimental results demonstrate that DAE-CNN can reduce the training time by 83% compared with CNN-S.

The rest of this paper is organized as follows. Section 2 introduces related work on Android malware detection and Deep learning methods. Section 3 describes the architecture and theoretical background of DAE, CNN and DAE-CNN. Section 4 shows how CNN can be used to detect Android malware. The experimental results are also demonstrated in Sect. 4 and the conclusions follow in Sect. 5.

2 Related work

Existing work on the detection of malicious apps mainly focuses on the analysis of static features (Rastogi et al. 2016; Sarma et al. 2012; Pandita et al. 2013; Lu et al. 2012; Klieber et al. 2014), or dynamic features (Enck et al. 2014; Wu and Hung 2014; Amos et al. 2013). Li and Li (2015) proposed an Android malware detection method based on characteristic trees. Yerima et al. (2014) developed machine-learning approaches based on Bayesian classification to detect uncovering unknown Android malware. Zhou and Jiang (2012) proposed a permission based scheme to detect new Android malware family samples and applied a heuristics-based filtering scheme to identify certain inherent behaviors of unknown malicious families. Shabtai et al. (2010) studied the techniques of static analysis to analyze Android source code. They also applied machine learning techniques to categorize games and tools with static features extracted from Android apps. There exists work on securing smartphone authentication (Shen et al. 2018a, b, c, d) or securing cloud authentication (Li et al. 2018a, b; Wang et al. 2018a, b, c, d; Shen et al. 2018a, b, c, d; Xie et al. 2018; Chen et al. 2015) and securing user authentication (Shen et al. 2017). In our previous work, we employed the ensemble of multiple classifiers, namely, Support Vector Machine (SVM), K-Nearest Neighbor (KNN), Naive Bayes (NB), Classification and Regression Tree (CART) and Random Forest (RF), to detect malicious apps and to categorize benign apps (Wang et al. 2018a, b, c, d). We also explored the permission-induced risk in Android apps on three levels in a systematic manner (Wang et al. 2014a, b). In addition, we gave insights regarding what discriminatory features are most effective to characterize malicious apps for building an effective and efficient malicious app detection system (Wang et al. 2017). We developed a tool called "SDFDroid" to identify the used sensors' types and to generate the sensor data propagation graphs in each app (Liu et al. 2018). Feature selection is one of the significant steps in classification (Zhang et al. 2017; Lee et al. 2017; Memos et al. 2018; Wang et al. 2015) or intrusion detection (Wang et al. 2008, 2014a, b, 2018a, b, c, d; Guan et al. 2009; Wang and Battiti 2006). Using machine learning methods can automatically analyze malicious behavior and detect malware effectively.

Deep learning is a new area of machine learning and has been widely applied to many scenes (Hamedani et al. 2018), such as event detection and analysis (Chang et al. 2017a, b; Chang and Yang 2017; Li et al. 2017a, b). It can also perform well in Android malware detection. Yuan et al. (2016) proposed to associate the features from the static analysis with features from dynamic analysis of Android apps and characterize malware using Deep Belief Network (DBN) (Bengio 2009; Hinton et al. 2014). CNN has shown popular performance in recognition and classification area, which is proposed by Lecun (Lecun et al. 1998) in 1998. The latest work employing CNN to Android malware detection has been found in 2017 (Huang et al. 2017; Nix and Zhang 2017). Different from existing work, we focus on the evaluation of different CNN structures and how to reduce the training time without changing the computing environments. To analyze the effect of different structures on detection accuracy, two different structures of CNN and one basic structure of CNN are employed in the same experimental environment. We also evaluate the influence of different activation functions. Due to the plenty of parameters, we use "dropout" (Hinton et al. 2012) technique to prevent

complex weights co-adaptations and reduce overfitting. In addition, we use the pre-training methods for the detection. Different from existing methods, we propose a hybrid model by combining DAE and CNN to reduce the training time.

3 Methods

3.1 Architecture of deep autoencoder (DAE)

Typical autoencoder (Hinton and Zemel 1994) is an unsupervised model that learns to reconstruct the input. Deep learning models are capable of learning complex hierarchical nonlinear features, which are considered as better representations for original data in many fields such as speech recognition and computer vision (Zhang et al. 2017). The basic structure of autoencoder contains encoder layer, hidden layer and decoder layer. The input of hidden layer is the output of encoder layer and the input of decoder layer is the output of hidden layer. The function of autoencoder is composed by the encoder and decoder with the symmetrical architecture to map $R^d \rightarrow R^d$. The encode layer parameterized by $\theta = \{W, b\}$ with input sequence $\boldsymbol{x}$ and output $\boldsymbol{y}$ is:

$$\boldsymbol{y} = f_\theta(x) = \sigma(W \times \boldsymbol{x} + b) \tag{1}$$

where $\sigma(x) = \max(0, x)$ is the activation function, Relu, of the encode layer. Compared with Sigmoid, Relu can usually eliminate the necessity of pre-training and make deep learning models converge to sometimes more discriminative solutions more quickly, while keeping the model sparse (Lecun et al. 1998; Huang et al. 2017). The reconstructed function between hidden layer and decode layer is:

$$z = f_{\theta'}(y) = \sigma(W' \times y + b') \tag{2}$$

The main target of autoencoder is to minimize the following function:

$$\theta, \theta' = \min_{\theta,\theta'} \sum_{i=1}^{d} L(x_i, z_i) \tag{3}$$

to ensure that the hidden layer can reconstruct the input layer.

In this work, we use DAE model, which has more than one hidden layer to extract features from training data. In addition, we extend DAE to complete the classification of Android apps. There are four layers in DAE model (as illustrated in Fig. 1), one encoding layer, two hidden layers and one classification layer. Using softmax as the activation function of classification layer and training data with labels, we achieved the goal of detecting Android malware.

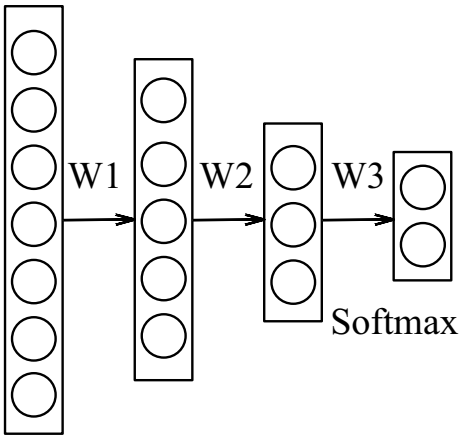


Fig. 1 Deep autoencoder model

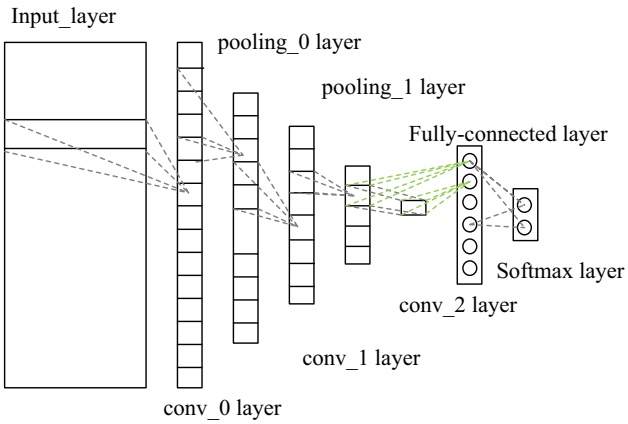


Fig. 2 CNN-S model

3.2 CNN with different architectures

The CNN-S model architecture, shown in Fig. 2, is a slight variant of basic CNN architecture.

We reconstruct the extracted features of each app as the input of the convolutional layer. Given $\boldsymbol{x_i} \in R^k$ as the k-dimensional feature vector corresponding to the i-th feature in the feature codes of each Android app, the Android app of length n (padded where necessary) can be represented as:

$$\boldsymbol{x}_{1:\,n} = \boldsymbol{x}_1 \oplus \boldsymbol{x}_2 \oplus \ldots \oplus \boldsymbol{x}_n \tag{4}$$

The input data are convoluted by the kernels and learnable filters. A filter applied to a window of m features to produce a new feature. For example, the feature y_i is generated from a window of features $\boldsymbol{x}_{i:\,i+m-1}$ filtered with $W \in R^{m*k}$ by:

$$\boldsymbol{y}_i = f(W * \boldsymbol{x}_{i:i+m-1} + b) \tag{5}$$

where $f(x) = \max(0, x)$ is a nonlinear activation function Relu, $b \in R$ is a bias term. The filter $W \in R^{m*k}$ is applied to each possible window of features in $\{x_{1:m}, x_{2:m+1}, \ldots, x_{n-m+1:n}\}$ to produce a feature map $\boldsymbol{y} \in R^{(n-m+1)*1}$. The feature maps are imported to the max-pooling layer, taking the maximum value $Y = \max\{\boldsymbol{y}\}$, capturing the most important features and reducing the feature dimension for each feature map.

The CNN-S model consists of three convolutional layers with max-pooling layers located between two of them. The activation function of each convolutional layer is Relu. A small and non-zero gradient is obtained with the help of Relu and the accuracy of the CNN increases. Different from the basic CNN, the output of the third convolutional layer is imported to the fully connected layer together with the second max-pooling layer to maximize the extraction of features. In this layer, the neurons are fully connected to all activations in the former layers.

After that, we add a dropout layer to the fully connected layer to prevent co-adaptation of hidden neurons. The dropped-out neurons do nothing to the forward pass and only the neurons without dropout participate in back-propagation. Therefore, this layer helps to reduce the complex of co-adaptation of neurons. Without dropout, the experimental result exhibits substantial overfitting.

At the end of CNN-S model, the softmax layer is employed to do classification whose output is the probability distribution over labels.

The CNN-P architecture, illustrated in Fig. 3, is another variant of basic CNN architecture. The CNN-P model uses three multiple filters with different window sizes to extract multiple features. The penultimate layer is formed with these features. Thus, the features are imported to a fully connected layer. In both the two models proposed, the error between the actual output and network output are computed and minimized by being back propagated. The weights of the CNN are then further adjusted to fine-tune them.

Time complexity determines the model's training and testing time. If the complexity is too high, it will take a great deal of time to train and test a model. Due to high complexity, a model cannot be evaluated quickly or make a quick prediction. The factors that affect the time complexity of a convolutional layer include the sizes of input feature maps, the sizes of kernels, the quantities of input channels and output channels. Given D as the depth of CNN model, l as the name of a convolutional layer, M_l as the sizes of feature maps, K_l as the sizes of kernels, C_{in} and C_{out} as the quantities of input channels and output channels, the time complexity of a CNN model can be represented as:

$$Time \sim O\left(\sum_{l=1}^{D} M_l^2 * K_l^2 * C_{l-1} * C_l\right) \tag{6}$$

Training time can be reduced if we can reduce the dimension of input data according to the equation.

3.3 Architecture of proposed DAE-CNN

Due to the high dimensional features of Android apps, it is too expensive to train the deep neural networks. To reduce the training time and take advantage of the power of CNN, we combine DAE with CNN. Using DAE as a pre-training method can capture the essensial features of Android apps efficiently. We extract the output of hidden layer in DAE and add sparse rules to make it available for CNN. Considering the time complexity mentioned in Sect. 3.2, DAE-CNN, shown in Fig. 4, can learn more flexible patterns of training data in a short time.

4 Malware detection with CNN

Figure 5 illustrates the proposed DAE-CNN model. As described in the figure, we extract features from 23,000 apps collected from various app stores and process the data to adapt into deep learning models. We use keras, the python deep learning library, to implement the deep learning models. In order to demonstrate that DAE-CNN can improve the detection accuracy, experiments with DAE-CNN and with

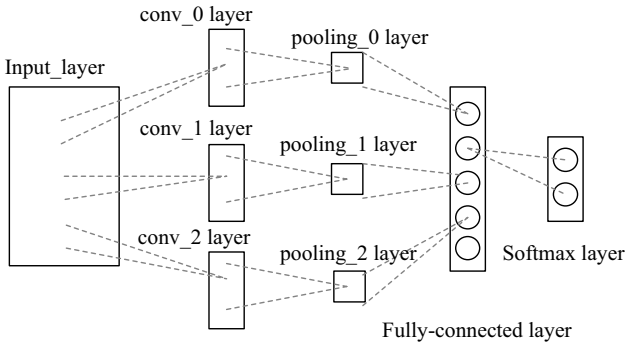


Fig. 3 CNN-P model

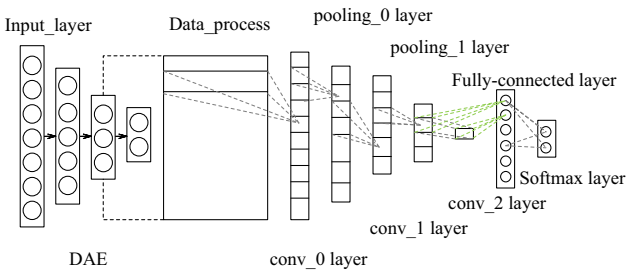


Fig. 4 DAE-CNN-S model

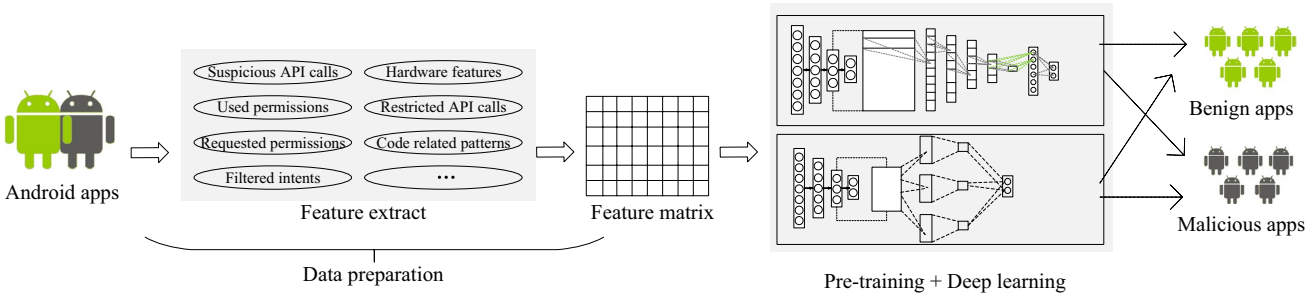


Fig. 5 Framework of DAE-CNN models

other traditional machine learning methods are conducted. To analyze the effect of CNN models under different parameters, CNN models with different structures are applied for malware detection. We also compare DAE-CNN models with CNN models to verify that using DAE-CNN can reduce training time.

4.1 Data preparation

We crawl 10,000 apps from Anzhi play store. Scanned by Virustotal, all the 10,000 apps are confirmed as benign apps. We collect 13,000 malicious apps from VirusShare. With 23,000 apps, we thoroughly train and test various models.

In this work, we are motivated to conduct experiments on high dimensional features of large scale Android apps. Compared with dynamic analyses, static analyses cost less in time and complexity. Therefore, we conduct static analyses to extract features from each app by Androguard and Android SDK Tools. In this way, we obtain a total of 34,570 features for each app. The seven categories of static features are permissions, requested permissions, filtered intents, restricted API calls, hardware features, code related patterns, and suspicious API calls. The number of features is too large to be processed efficiently. We reconstruct the structure of features. In detail, we encode all the features and use the feature code to indicate each app and pad where necessary with zero. Therefore, the dimension of dataset is reduced from 34,570 to 413.

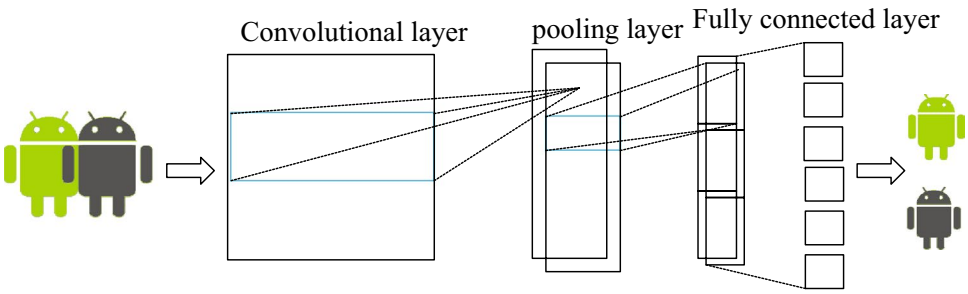
To make it available for CNN model and improve the accuracy, we use $\boldsymbol{x}_i \in R^{256}$ to signify the 256-dimensional feature vector corresponding to the i-th feature in the feature codes of each Android app. Therefore, each app is indicated as a matrix with size 413×256. Dataset is randomly divided into training data and testing data by 4:1.

4.2 Experimental setup

We conduct experiments with several variants of deep learning models and other machine learning models. All the experiments are conducted in the same circumstance with the same dataset. Keras is a high-level neural networks API. It was written in Python and is capable of running on top of TensorFlow or Theano. Deep learning models are trained and tested by calling Keras functional API.

- *Basic CNN (CNN-0)* The baseline model consists of one convolutional layer, one max-pooling layer and one fully connected layer (shown in Fig. 6). The activation of convolutional layer is relu and the fully connected layer is sofmax.
- *CNN-S* The structure of CNN-S model is illustrated in Fig. 2. CNN-S is trained from scratch. Kernel size and filter size are the main factors affecting the accracy of the training model. According to the dimension of input data and output data, we design the CNN-S model. The filter window of the first convolutional layer is set as 4×256 with 50 kernels. The pool size of the first max-pooling

Fig. 6 Traditional CNN model



layer is 10×1. The filter window of the second convolutional layer is set as 6×1 with 50 kernels. The pool size of the second max-pooling layer is 6×1. The filter window of the third convolutional layer is set as 6×1. For each convolutional layer, we apply Relu, the nonlinear activation function, to achieve scale invariance. We adopt two fully connected layers to aggregate the features learned from the second pooling layer and the third convolutional layer to do classification. The detailed parameters are shown in Table 1.

- *CNN-P* CNN-P is pre-trained on CNN-multichannel (Kim 2014) and then fine-tuned. Finally, the hyberparameters in CNN-P model are set as follows: three filter windows of 3×256, 4×256, 5×256 with 80 kernels each, the nonlinear activition function Relu, pool sizes in maxpooling layers of 411×1, $410\times1409\times1$, dropout rate of 0.5, the interation number of 50. Based on the hyberparameters, we get 240 feature maps which are the most optimal features for classfication. The detailed parameters are shown in Table 2.
- *DAE* We conduct four different DAE structures including 413-200-100-2, 413-200-2, 413-100-20-2, 413-200-100-20-2 on the dataset. For each structure, we apply Relu as the activation function on each layer. Threre are two keys in DAE training, one is the structure, the other is the number of iterations. Due to overfitting, a model does not always perform better as the number of interations increases. We try different numbers of interations in 413-200-100-20-2. With the increasing numbers of interation, the performance of DAE keeps (Table 3). Based on the results of DAE model, the structure of 413-200-100-20 is chosen for pretraining of CNN.
- *DAE-CNN* As the layers of deep learning model increase, the number of feature maps increase exponentially. The training time grows rapidly as well. It is necessary to build a layer to reduce the dimension of input data. We use the output of the third layer of DAE (413-200-100-20-2) as the input of the first convolutional layer in CNN-S and the input of the CNN-P as well to compensate for CNN's limitation and reduce the training time. We add the sparse rules (Glorot et al. 2011) to the 20 features of each app extracted from DAE model to apply CNN-S and CNN-P. The hyberparameters of DAE-CNN-S are set as follows: the filter window of convolutional layers are 3×256, 3×1, 2×1, the pool size of the pooling layers are 3×1, 3×1.The parameters of DAE-CNN-S are set as follows: the pool size of the pooling layers are 18×1, 17×1, 16×1.
- *Other traditional machine learning methods* We compare the proposed methods with some machine learning methods mentioned in Sect. 2. We employ scikit-learn (Pedregosa et al. 2011; Glorot et al. 2011) packages written in Python as the machine learning tools in the experiments. The methods using the same training set and testing set are given as follows: *SVM, Decision Tree, Random Forest (RF), K-Nearest Neighbor (KNN).*

Table 1 Parameters of CNN-S model

Layer	Output shape	Parameters
input_1	(None, 413)	0
embedding_1	(None, 413, 256)	540,928
reshape_1	(None, 413, 256, 1)	0
conv2d_1	(None, 410, 1, 50)	51,250
max_pooling2d_1	(None, 41, 1, 50)	0
conv2d_2	(None, 36, 1, 50)	15,050
max_pooling2d_2	(None, 6, 1, 50)	0
conv2d_3	(None, 1, 1, 50)	15,050
flatten_1	(None, 300)	0
dropout_1	(None, 300)	0
flatten_2	(None, 50)	0
merge_1	(None, 350)	0
dense_1	(None, 2)	702

Table 2 Parameters of CNN-P model

Layer	Output shape	Parameters
input_1	(None, 413)	0
input_1	(None, 413)	0
embedding_1	(None, 413, 256)	540,928
reshape_1	(None, 413, 256, 1)	0
conv2d_1	(None, 411, 1, 80)	61,520
conv2d_2	(None, 410, 1, 80)	82,000
conv2d_3	(None, 409, 1, 80)	102,480
max_pooling2d_1	(None, 1, 1, 80)	0
max_pooling2d_2	(None, 1, 1, 80)	0
max_pooling2d_3	(None, 1, 1, 80)	0
merge_1	(None, 3, 1, 80)	0
flatten_1	(None, 240)	0
dropout_1	(None, 240)	0

Table 3 Detection results with DAE model

Iteration	Structure	FPR	TPR	ACC
20	431-200-100-2	7.6	96.5	94.7
100	431-200-100-2	4.9	98.7	97.1
50	431-200-100-2	2.9	98.7	**98.1**
50	**431-200-100-20-2**	**2.1**	**98.1**	**98.0**
50	431-100-20-2	2.9	97.2	97.1
50	431-200-2	9.1	85.3	88.1

Bold values indicate the best performance (with pre-set parameters or models)

Table 4 Detection results

	FPR (%)	TPR (%)	ACC (%)	Recall (%)	PPV (%)	FScore (%)	Time (mins)
CNN-0	0.49	97.87	97.80	97.87	97.78	97.82	311.9
CNN-S	**0.21**	99.89	**99.82**	99.89	**99.84**	**99.86**	332.9
DAE-CNN-S	1.82	98.65	98.60	98.65	98.73	98.69	*59.6*
CNN-P	0.33	**99.91**	99.80	**99.91**	99.74	99.82	1101.9
DAE-CNN-P	1.67	98.57	98.50	98.57	98.71	98.64	*183.1*
DAE	7.62	96.53	94.72	96.53	94.27	95.78	3.6
SVM	5.06	95.56	95.29	95.56	96.09	95.82	6.5
Decision tree	1.84	97.85	97.54	97.85	97.58	97.71	0.43
RF	1.69	97.91	97.65	97.91	97.70	97.80	0.3
K-NN	15.98	93.60	89.01	93.60	88.38	90.92	0.73

Bold values indicate the best performance (with pre-set parameters or models)

This part of our paper mainly investigates the methods DAE-CNN-S and DAE-CNN-P and their efficiency. Thus they are shown with italics values for significance

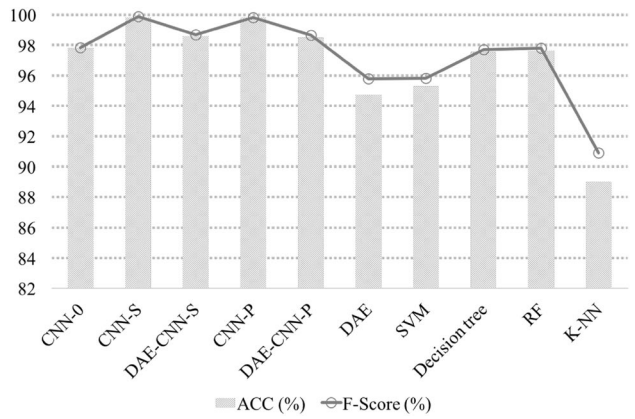


Fig. 7 Comparison of ACC and F-Score in ten different models

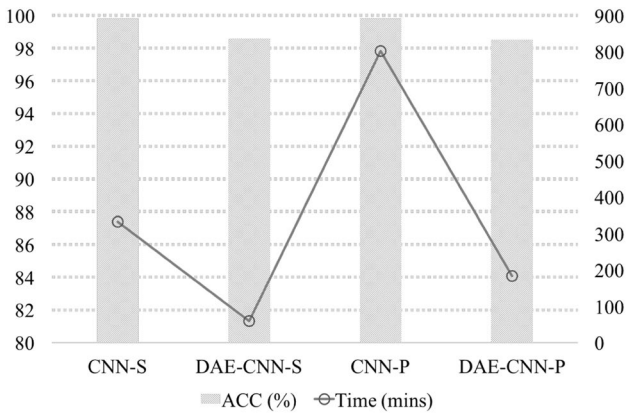


Fig. 8 Comparation of Time and ACC between CNN models with and without DAE

4.3 Results

The results of ten experiments are illustrated in Table 4. The operating characteristics to evaluate the performance of the structures are set as follows: FPR (false positive rate), TPR (true positive rate), ACC (accuracy), Recall, PPV (positive predict value), FSCORE (the harmonic mean of precision and sensitivity: $FSCORE = 2TP/2TP + FP + FN$), Training Time.

4.4 Evaluations

- *Effect of CNN structures*: All the experiments are conducted in the same environment. Figure 7 shows the ACC and FPR of the proposed algorithms. It can be seen from Fig. 7 that different CNN structures have different performance. The CNN-S achieves better accuracy and F-score than basic CNN and CNN-P. Compared with traditional machine learning methods, training with CNN can improve the accuracy apparently. Compared with SVM, the accuracy with the CNN-S model is improved by 5%. The CNN models have disadvantages either. Traditional machine learning methods only need to store a matrix for prediction, while CNN has to store the whole model. The parameters in training a CNN model are thousands mutiples of parameters in traditional machine learning. Thus training machine learning models takes short time and little space compared with CNN models.
- *Effect of pre-training methods*: To reduce the training time, the DAE-CNN models are proposed. As shown in Fig. 8, models with the DAE pre-training method consume less time than CNN models without DAE. The efficiency of detection with DAE-CNN-S is improved by 83% compared with CNN-S model. Although the accuracy of DAE-CNN is a little lower than CNN models, it is still higher than traditonal machine learning methods. In general, taking time and other evaluation index into consideration, DAE-CNN-S has more advantages than other methods mentioned in this paper in Android mal-

Table 5 Detection results with DAE model

Structure	FPR (%)	TPR (%)	ACC (%)
CNN-0 (Relu)	0.49	96.5	94.7
CNN-0 (Sigmoid)	2.57	96.28	95.91

ware detection. We believe that the combination of DAE and CNN has potential ability for high accuracy when the dataset is large enough.

- *Other hyberparameters in CNN models*: Using Relu (Nair and Hinton 2010) as the activation function can avoid the gradient from nonsaturate in the positive region and increase the accuracy of the CNN. We also conduct experiments using sigmoid and the results shown in Table 5 prove that Relu can improve the behavior of CNN. Dropout can be used as an efficient way to prevent overfitting.

5 Conclusion

In this work, we propose a hybrid model for Android malware detection with DAE and CNN to improve the detection accuracy and reduce the training time. The CNN-S and CNN-P structures are employed in the training process, during which the "dropout" technique is used to prevent overfitting. Experimental results demonstrate that the proposed model is effective for large-scale Android malware detection. Compared with SVM, the accuracy of CNN-S model is improved by 5%. The time consumption for training with DAE-CNN is reduced by 83% compared with CNN-S model. In future work, we will extract more fine-grained features from Android apps and explore more effective algorithms to analyze and detect more sophisticated Android malware.

Acknowledgements The work reported in this paper was supported in part by National Key R&D Program of China, under Grant 2017YFB0802805, in part by Shanghai Key Laboratory of Integrated Administration Technologies for Information Security, under Grant AGK2015002, in part by ZTE Corporation Foundation, under Grant K17L00190, in part by funds of Science and Technology on Electronic Information Control Laboratory, under Grant K16GY00040, in part by the Fundamental Research funds for the central Universities of China, under Grant K17JB00060 and K17JB00020, and in part by Natural Science Foundation of China, under Grant U1736114 and 61672092.

References

Amos B, Turner H, White J (2013) Applying machine learning classifiers to dynamic android malware detection at scale. In: 9th international wireless communications and mobile computing conference (IWCMC), July 1–5, 2013, Sardinia, Italy, pp 1666–1671

Atat R, Liu L, Chen H, Wu J, Li H, Yi Y (2017) Enabling cyber-physical communication in 5G cellular networks: challenges, spatial spectrum sensing, and cyber-security. IET Cyber-Phys Syst Theory Appl 2(1):49–54

Bengio Y (2009) Learning deep architectures for AI. Foundations & Trends®. Mach Learn 2(1):1–127

Chang X, Yang Y (2017) Semisupervised feature analysis by mining correlations among multiple tasks. IEEE Trans Neural Netw Learn Syst 28(10):2294–2305

Chang X, Ma Z, Yang Y, Zeng Z, Hauptmann AG (2017a) Bi-level semantic representation analysis for multimedia event detection. IEEE Trans Cybern 47(5):1180–1197

Chang X, Yu YL, Yang Y, Xing EP (2017b) Semantic pooling for complex event analysis in untrimmed videos. IEEE Trans Pattern Anal Mach Intell 39(8):1617–1632

Chen X, Li J, Huang X, Ma J, Lou W (2015) New publicly verifiable databases with efficient updates. IEEE Trans Dependable Secure Comput 12(5):546–556

China Internet Security (2016) Research report. http://zt.360.cn/1101061855.php?dtid=1101061451&did=490301065. Accessed Dec 2017

Enck W, Gilbert P, Han S, Tendulkar V, Chun B, Cox LP, Jung J, McDaniel P, Sheth AN (2014) TaintDroid: an information-flow tracking system for realtime privacy monitoring on smartphones. Acm Trans Comput Syst (TOCS) 32(2):1–29

Glorot X, Bordes A, Bengio Y (2011) Deep sparse rectifier neural networks. In: Proceedings of the 14th international conference on artificial intelligence and statistics (AISTATS), April 11–13, 2011, Ft. Lauderdale, FL, USA, pp 315–323

Guan X, Wang X, Zhang X (2009) Fast intrusion detection based on a non-negative matrix factorization model. J Netw Comput Appl 32(1):31–44

Gupta BB, Agrawal DP, Yamaguchi S (2016) Handbook of research on modern cryptographic solutions for computer and cyber Security, 1st edn. IGI Publishing, Hershey, PA. ISBN:1522501053 9781522501053

Hamedani K, Liu L, Atat R, Wu J, Yi Y (2018) Reservoir computing meets smart grids: attack detection using delayed feedback networks. IEEE Trans Ind Inf 14(2):734–743

Hinton GE, Zemel RS (1994) Autoencoders, minimum description length and Helmholtz free energy. Adv Neural Inf Process Syst (NIPS) 6:3–10

Hinton GE, Srivastava N, Krizhevsky A, Sutskever I, Salakhutdinov RR (2012) Improving neural networks by preventing co-adaptation of feature detectors. CoRR 1207:0580

Hinton GE, Osindero S, Teh YW (2014) A fast learning algorithm for deep belief nets. Neural Comput 18(7):1527–1554

Huang HD, Yu CM, Kao HY (2017) R2-D2: color-inspired convolutional neural network (cnn)-based android malware detections. CoRR 1705:04448

Ibtihal M, Driss EO, Hassan N (2017) Homomorphic encryption as a service for outsourced images in mobile cloud computing environment. Int J Cloud Appl Comput 7(2):27–40

Kim Y (2014) Convolutional neural networks for sentence classification. In: Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP), October 25–29, 2014 Doha, Qatar, pp 1746–1751

Klieber W, Flynn L, Bhosale A, Jia L, Bauer L (2014) Android taint flow analysis for app sets. In: Proceedings of the 3rd ACM SIGPLAN international workshop on the state of the art in Java program analysis, June 9–11, 2014, Edinburgh, United Kingdom, pp 1–6

LeCun Y, Bottou L, Bengio Y, Haffner P (1998) Gradient-based learning applied to document recognition. Proc IEEE 86(11):2278–2324

Lee K, Choi HO, Min SD, Lee J, Gupta B, Nam Y (2017) A comparative evaluation of atrial fibrillation detection methods in

koreans based on optical recordings using a smartphone. IEEE Access 5:11437–11443
Li Q, Li X (2015) Android malware detection based on static analysis of characteristic tree. In: 2015 International conference on cyber-enabled distributed computing and knowledge discovery (CyberC), September 17–19, 2015, Xi'an, China, pp 84–91
Li P, Li J, Huang Z, Gao CZ, Chen WB, Chen K (2017a) Privacy-preserving outsourced classification in cloud computing. Cluster Comput. https://doi.org/10.1007/s10586-017-0849-9
Li Z, Nie F, Chang X, Yang Y (2017b) Beyond trace ratio: weighted harmonic mean of trace ratios for multiclass discriminant analysis. IEEE Trans Knowl Data Eng 29(10):2100–2110
Li J, Sun L, Yan Q, Li Z, Srisa-an W, Ye H (2018a) Significant permission identification for machine learning based android malware detection. IEEE Trans Industr Inf PP(99):1–1
Li Y, Wang G, Nie L, Wang Q (2018b) Distance metric optimization driven convolutional neural network for age invariant face recognition. Pattern Recogn 75C:51–62
Liu X, Liu J, Wang W, He Y, Zhang X (2018) Discovering and understanding android sensor usage behaviors with data flow analysis. World Wide Web 21(1):105–126
Lu L, Li Z, Wu Z, Lee W, Jiang G (2012) CHEX: statically vetting android apps for component hijacking vulnerabilities. In: ACM conference on computer and communications security, October 16–18, 2012, Raleigh, NC, USA, pp 229–240
Memos VA, Psannis KE, Ishibashi Y, Kim BG, Gupta B (2018) An efficient algorithm for media-based surveillance system (EAMSuS) in IoT smart city framework. Future Gener Comput Syst 83:619–628
Nair V, Hinton GE (2010) Rectified linear units improve restricted Boltzmann machines. In: Proceedings of the 27th international conference on machine learning (ICML-10), June 21–24, 2010, Haifa, Israel, pp 807–814
Nix R, Zhang J (2017) Classification of Android apps and malware using deep neural networks. Int Jt Conf Neural Netw (IJCNN) May 14–19, 2017, Alaska, USA, pp 1871–1878
Pandita R, Xiao X, Yang W, Enck W, Xie T (2013) WHYPER: towards automating risk assessment of mobile applications. Usenix Conf Secur 2013:527–542
Pedregosa F, Gramfort A, Michel V, Thirion B, Grisel O, Blondel M, Prettenhofer P, Weiss R, Dubourg V (2011) Scikit-learn: machine learning in Python. Mach Learn Res 12(10):2825–2830
Rastogi S, Bhushan K, Gupta BB (2016) Android applications repackaging detection techniques for smartphone devices. Procedia Comput Sci 78:26–32
Sarma BP, Li N, Gates C, Potharaju R, Nita-Rotaru C, Molloy I (2012) Android permissions: a perspective combining risks and benefits. In: ACM symposium on access control models and technologies. June 21–23, 2017, Indianapolis, IN, USA, pp 13–22
Shabtai A, Fledel Y, Elovici Y (2010) Automated static code analysis for classifying android applications using machine learning. In: 2010 international conference on computational intelligence and security (CIS), December 11–14, 2010, Nanning, Guangxi, China, pp 329–333
Shen C, Chen Y, Guan X, Maxion Y (2017) Pattern-growth based mining mouse-interaction behavior for an active user authentication system. IEEE Trans Dependable Secur Comput PP(99):1–1
Shen J, Zhou T, Chen X, Li J, Susilo W (2018a) Anonymous and traceable group data sharing in cloud computing. IEEE Trans Inf Forensics Secur 13(4):912–925
Shen C, Li Y, Chen Y, Guan X, Maxion R (2018b) Performance analysis of multi-motion sensor behavior for active smartphone authentication. IEEE Trans Inf Forensics Secur 13(1):48–62
Shen J, Gui Z, Ji S, Shen J, Tan H, Tang Y (2018c) Cloud-aided lightweight certificateless authentication protocol with anonymity for wireless body area networks. Netw Comput Appl 106:117–123
Shen C, Chen Y, Guan X (2018d) Performance evaluation of implicit smartphones authentication via sensor-behavior analysis. Inf Sci 430:538–553
Wang W, Battiti R (2006) Identifying intrusions in computer networks with principal component analysis. ARES 2006:270–279
Wang W, Guan X, Zhang X (2008) Processing of massive audit data streams for real-time anomaly intrusion detection. Comput Commun 31(1):58–72
Wang W, Guyet T, Quiniou R, Cordier M, Masseglia F, Zhang X (2014a) Autonomic intrusion detection: adaptively detecting anomalies over unlabeled audit data streams in computer networks. Knowl Based Syst 70:103–117
Wang W, Wang X, Feng D, Liu J, Han Z, Zhang X (2014b) Exploring permission-induced risk in android applications for malicious application detection. IEEE Trans Inf Forensics Secur 9(11):1869–1882
Wang W, He Y, Liu J, Gombault S (2015) Constructing important features from massive network traffic for lightweight intrusion detection. IET Inf Secur 9(6):374–379
Wang X, Wang W, He Y, Liu J, Han Z, Zhang X (2017) Characterizing android apps' behavior for effective detection of malapps at large scale. Future Gener Comput Syst 75:30–45
Wang Y, Zhu G, Shi Y (2018a) Transportation spherical watermarking. IEEE Trans Image Process 27(4):2063–2077
Wang H, Wang W, Cui Z, Zhou X, Zhao J, Li Y (2018b) A new dynamic firefly algorithm for demand estimation of water resources. Inf Sci 438:95–106
Wang W, Li Y, Wang X, Liu J, Zhang X (2018c) Detecting android malicious apps and categorizing benign apps with ensemble of classifiers. Future Gener Comput Syst 78:987–994
Wang W, Liu J, Pitsilis G, Zhang X (2018d) Abstracting massive data for lightweight intrusion detection in computer networks. Inf Sci 433–434:417–430
Wu WC, Hung SH (2014) DroidDolphin: a dynamic android malware detection framework using big data and machine learning. In: Proceedings of the 2014 conference on research in adaptive and convergent systems (RACS), pp 247–252
Wu J, Guo S, Li J, Zeng D (2016a) Big data meet green challenges: greening big data. IEEE Syst J 10(3):873–887
Wu J, Guo S, Li J, Zeng D (2016b) Big data meet green challenges: big data toward green applications. IEEE Syst J 10(3):888–900
Xie D, Lai X, Lei X, Fan L (2018) Cognitive multiuser energy harvesting decode-and-forward relaying system with direct links. IEEE Access 6:5596–5606
Yerima SY, Sezer S, McWilliams G (2014) Analysis of Bayesian classification-based approaches for android malware detection. IET Inf Secur 8(1):25–36
Yuan Z, Lu Y, Xue Y (2016) Droiddetector: android malware characterization and detection using deep learning. Tsinghua Sci Technol 21(1):114–123
Zhang C, Liu C, Zhang X Almpanidis G (2017) An up-to-date comparison of state-of-the-art classification algorithms. Expert Syst Appl 82:128–150
Zhou Y, Jiang X (2012) Dissecting android malware: characterization and evolution. In: IEEE symposium on security and privacy (SP), May 20–23, 2012, San Francisco, USA, pp 95–109

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.