Journal of Systems Engineering and Electronics
Vol. 35, No. 1, February 2024, pp.163 – 177

DCEL: classifier fusion model for Android malware detection

XU Xiaolong[1,*], JIANG Shuai[2], ZHAO Jinbo[2], and WANG Xinheng[3]

1. Jiangsu Key Laboratory of Big Data Security & Intelligent Processing, Nanjing University of Posts and Telecommunications, Nanjing 210023, China; 2. School of Computer Science, Nanjing University of Posts and Telecommunications, Nanjing 210023, China; 3. School of Computing and Engineering, University of West London, London W5 5RF, UK

Abstract: The rapid growth of mobile applications, the popularity of the Android system and its openness have attracted many hackers and even criminals, who are creating lots of Android malware. However, the current methods of Android malware detection need a lot of time in the feature engineering phase. Furthermore, these models have the defects of low detection rate, high complexity, and poor practicability, etc. We analyze the Android malware samples, and the distribution of malware and benign software in application programming interface (API) calls, permissions, and other attributes. We classify the software's threat levels based on the correlation of features. Then, we propose deep neural networks and convolutional neural networks with ensemble learning (DCEL), a new classifier fusion model for Android malware detection. First, DCEL preprocesses the malware data to remove redundant data, and converts the one-dimensional data into a two-dimensional gray image. Then, the ensemble learning approach is used to combine the deep neural network with the convolutional neural network, and the final classification results are obtained by voting on the prediction of each single classifier. Experiments based on the Drebin and Malgenome datasets show that compared with current state-of-art models, the proposed DCEL has a higher detection rate, higher recall rate, and lower computational cost.

Keywords: Android malware detection, deep learning, ensemble learning, model fusion.

DOI: 10.23919/JSEE.2024.000018

1. Introduction

With the rapid development and popularization of mobile Internet [1], smartphones and other mobile smart terminals [2] have become indispensable tools in people's daily works and lives. According to the Internet data center survey of International Data Corporation (IDC), as of the third quarter of 2020, the market share of Android operating system is 85.0% of the total mobile operating system [3]. In 2018, Wang et al. [4] implemented a large-scale analysis of six million Apps in 16 Chinese Android App markets and the Google Play market. Approximately 12.30% of Apps in Chinese Android App markets were reported as malicious Apps based on at least 10 antivirus engines. In 2019, the 360 Security Brain intercepted about 1.809 million new malicious Apps on mobile terminals and intercepted about 5 000 new mobile malicious programs every day [5]. In April 2020, the survey of Atlas virtual private network (VPN)'s global malware infections in the past 30 days showed that about 404 million pieces of malware has been found worldwide [6], which brings huge security risks to Android mobile devices, including tariff consumption, privacy stealing, and remote control. Tariff consumption is mainly for the tariffs of mobile phone users, forcing customized services and profiting from them. Privacy stealing mainly implements collecting users' private text messages, address books, call records, social data, etc. Remote control mainly uses the hypertext transfer protocol (HTTP) to receive control commands from command-and-control (C&C) server to realize remoting manipulation [7].

Android malware detection methods are mainly divided into two categories, including static detection and dynamic detection [8]. With the static detection method, there is no need to run the application [9]. However, the static detection method has two main shortcomings. On the one hand, the static detection method is easily affected by obfuscation techniques. For example, renaming package names, method names, or reordering instance variables and local codes, may cause data obfuscation [10], and moving method call location may cause control flow obfuscation without changing semantics [11]. On the other hand, injection of non-Java code, network activity, and runtime modification of objects (such as reflection) are not within the scope of static analysis because they are only in the program runtime visible. Unlike static analysis that directly parses Android application package (APK) files, dynamic analysis requires executing applications, observing and collecting runtime data. Dynamic detection needs to execute the application to obtain operating information, which can capture the runtime state of the application, but the dynamic detection technology has a large cost in time and resource consumption, and the

Manuscript received December 16, 2021.
*Corresponding author.
This work was supported by the National Natural Science Foundation of China (62072255).

extracted feature information is not stable, and the complexity of dynamic analysis of the application is high.

Today's malware detection methods have the following three main challenges:

(i) Diversified forms of malicious code. Malicious code creators continuously modify the source code of malicious software to create many types of variants. The increasingly mature obfuscation technology also makes the obfuscated malicious code change greatly while retaining the normal execution of malicious behaviors, which makes the detection of malware more and more difficult [12].

(ii) The number of samples to be analyzed is large. In reality, the safety inspection system is faced with thousands of samples to be analyzed every day, and analyzing each sample in detail is time-consuming and laborious work. In recent years, there has also been a malware factory, which automatically morphs malware so that it can generate numerous malware variants, which has led to a dramatic increase in malware.

(iii) It is difficult to label sample data. Most malware detection methods build classifiers based on a large number of labeled sample data to classify unlabeled samples. However, it is a very time-consuming and laborious task to obtain a large number of samples with labeled information. Moreover, in actual situations, some newly intercepted malicious samples belong to unknown families. At this time, the family identification method based on supervised learning cannot handle this type of sample.

In response to the above-mentioned main challenges, effective malware detection techniques need to meet three points, namely, high detection rate, low runtime overhead, and a small number of labeled samples. However, the existing methods can only meet one or two of them. According to the different application scenarios and technical means, we divide the existing malware detection methods into three categories:

(i) Based on the signature method: the detection purpose is achieved by matching the signature of the software to be detected with the signature of the existing malware, so its runtime overhead is small, and it can provide certain evidence (the detected malicious signature) to explain. However, this kind of method is easy to be bypassed by deformed malicious code, so it cannot effectively deal with the first challenge.

(ii) Behavior-based method: a customized detection method is proposed for specific malicious behaviors, which can effectively explain the detected malware behavior. However, they often use high-overhead data flow analysis, natural language processing and other technologies, so they cannot handle large-scale sample sets and cannot effectively deal with the second challenge.

(iii) Based on the machine learning method: through feature engineering, the program is converted into feature vectors for representation, and then a classifier is constructed to detect and classify malware quickly and accurately. This kind of method can deal with different malicious code variants to a certain extent, and it requires less time overhead. However, this type of method requires a large number of labeled samples for learning, and a small number of samples will lead to low detection accuracy and cannot effectively meet the third challenge.

Therefore, in this article, we propose a new classifier fusion mechanism deep neural networks and convolutional neural networks with ensemble learning (DCEL) based on deep learning, which uses model fusion methods to improve the predictive ability of deep learning algorithms. In order to prove the effectiveness of the method, we use the dataset provided by Yerima et al. [13], which is based on the Drebin [14] and Malgenome [15] datasets to optimize the original data. Through experiments, we can observe some application programing interface (API) attributes frequently used by malware. The fusion of deep learning methods can effectively improve the classification accuracy of the model. The contributions of this article are the followings:

(i) We analyze and explore the attributes of the malware dataset. Due to the existing decision-making results, the security analyst cannot provide sufficient effective information to explain the malicious behavior of the sample. Therefore, we use the chi-square test feature selection algorithm to rank the static attributes of the malware dataset, explore the proportion of the features with the highest ranking in the malware samples, and classify the software threat level. According to the ranking results, the top 20% of the attributes are classified as high risk A, 20% to 40% are classified as risk B, 40% to 60% are classified as mild risk, and 60% to 80% are classified as grade D normal, the remaining features are classified as E-level security. The experimental results show that the API call signature category features used by high-risk software account for a relatively high proportion, while the API call signature category features used by ordinary software account for a low proportion, and the Manifest Permission category feature accounts for a high proportion.

(ii) A new classifier fusion classification mechanism based on deep learning is proposed. This mechanism first performs data preprocessing on the original data, removes some redundant data, converts the one-dimensional data into a two-dimensional gray image, and uses the convolutional neural network (CNN) algorithm [16] to train the gray image to generate model 1. At the same time, different levels of deep neural networks (DNN) [17] are used to train one-dimensional data to generate model 2 and model 3. Then use the ensemble learning method based on the majority voting mechanism to fuse all the models to get the final classifier.

(iii) Applying DCEL to Android malware detection, the model does not need a large number of samples or require actual application running with static detection.

The model has high detection accuracy and recall rate, and low computational cost. Compared with traditional machine learning fusion models, this model eliminates the time and computational costs of designing the feature selection method, and can quickly analyze thousands of Android software daily. It perfectly solves the three major challenges of high detection rate, low runtime cost, and small number of labeled samples required.

The rest of this article is organized as follows: Section 2 briefly discusses related work. Section 3 introduces the DCEL mechanism. Section 4 presents experimental setup and result analysis. Section 5 provides conclusions and future work.

2. Related work

In recent years, the rapid development of mobile malware detection technology has quickly become a hot issue in the academic software security field, and a large number of excellent related research results have been published in international conferences and journals. In this part, we divide these methods into three categories according to the different feature extraction methods: static analysis, dynamic analysis, and hybrid analysis.

Firstly, static analysis disassembles the source code of Android application software and then analyzes the source code to identify whether the application is malicious software. The static analysis mainly includes signature-based, permission-based, and Dalvik virtual machine bytecode detection methods. Signature-based detection methods [18] extract specific semantic modules from the malware dataset and then generate the signature for malware detection. The signatures of all the extracted malware are stored to generate a malware signature database. However, this signature-based detection method is easily limited by code obfuscation, encryption, etc. It can only detect discovered malware, but not new variants of malware or 0-day malware, and it requires long-term and timely updates to the malware dataset to maintain detection accuracy. Zheng et al. [19] proposed Droid Analytics, which extracts information and generates signatures from three different code granularities of the software. However, many functional methods are used not only by malware but also by benign methods, which leads to a relatively high false alarm rate.

The permission-based detection method [20] is based on the analysis of the AndroidManifest.xml file. Sato et al. [21] proposed a lightweight malware detection mechanism. It only needs to analyze the AndroidManifest.xml file, and extract the permission application, classification, priority, process name and other information to analyze whether an Android application has malicious behavior. However, their detection accuracy is low, the false alarm rate is high, and it takes a lot of time and expense to calculate the judgment threshold.

The bytecode detection method based on the Dalvik virtual machine [22] is to detect dangerous operations performed by malware by analyzing the bytecode control flow and data flow. Zhang et al. [23] proposed a graph-based Dalvik bytecode analysis method, using graph theory and information theory to implement a lightweight Android malware detection framework. This method extracts the features of malware by topological feature extraction and the disadvantage is that it takes a lot of time to generate the topological graph. The topological feature update of malicious samples is the biggest bottleneck of this method, which requires high-performance support.

Secondly, dynamic analysis is the detection of software during software execution for code obfuscation and malicious software after code encryption. Dynamic analysis is mainly based on abnormal dynamic detection. Anomaly-based dynamic analysis detection is a commonly used detection technique in Android malware detection. Burguera et al. [24] proposed a method called CrowDroid to dynamically detect the behavior of Android software. CrowDroid records the system call information generated by the Android application software, and generates a log file, then sends it to the server. Finally, the server classifies the detected software according to log files. However, since this method requires the in-depth analysis of the software, a lot of resource support is needed. At the same time, when benign software generates too many system calls, this method is easy to misjudge the software as malicious software.

Thirdly, due to the strong recognition and classification capabilities of machine learning, researchers often combine static and dynamic analysis and machine learning for Android malware detection. Singh et al. [25] used a potential semantic indexing technology to construct a low-dimensional representation of the opcode while retaining the semantic knowledge originally contained in the operator, thereby building a lightweight detection system. Roy et al. [26] performed static analysis to map each API to certain functions and then aggregated these functions to find out the frequency of each function. This method has better robustness, scalability, and higher receiver operating characteristic (ROC)-area under curve (AUC) value. The model proposed by Unver et al. [27] is based on converting some files in the Android application source into grayscale images. The method extracts feature from the images, and is used to train multiple machine learning classifiers. The method has a high classification accuracy of 98.75% and a typical computation time of no more than 0.018 s per sample. Copty et al. [28] achieved high accuracy by exploring the path in the system that only approximated behavior in the real system, aggregating features from multiple paths, and using a funnel-like machine learning classifier configuration. However, since tens of thousands of samples are processed every day, and each percentage point is very important, it takes a lot of time for model training and

high computational overhead. Machiry et al. [29] proposed a model LoopMC. The model locates loops in the application without the need for source code, extracts a set of tags for each loop, and then treats each unique combination of these tags as different features to construct a feature space and perform classification. However, the system cannot use bytecode encryption or dynamic code loading and is easily evaded by simple malware without loops. Kim et al. [30] used various functions to reflect the attributes of Android applications from various aspects, used similarity-based feature extraction methods to refine these features, and proposed a multimodal deep learning method as a malware detection model. However, this method designs multiple models and outputs the results, which leads to high model complexity and large computational overhead.

3. Detection mechanism

Android malware detection methods are mainly divided into static detection and dynamic detection. In practical applications, there are many applications on the Android application market every day. The dynamic detection technology has a large cost in time and resource consumption, and the extracted feature information is unstable. The dynamic analysis of the application is complicated. The malware program on the platform was detected in a short time. The static detection technology balances efficiency and cost well, and obtains a higher detection accuracy at the cost of lower time and resources, which is suitable for the needs of the Android application market. Therefore, this article finally chooses static detection technology for Android malware detection.

3.1 DCEL architecture

Aiming at the problem of simply using sensitive API features to cause normal software to be falsely detected, we propose DCEL based on deep learning. Fig. 1 is a framework diagram of DCEL.

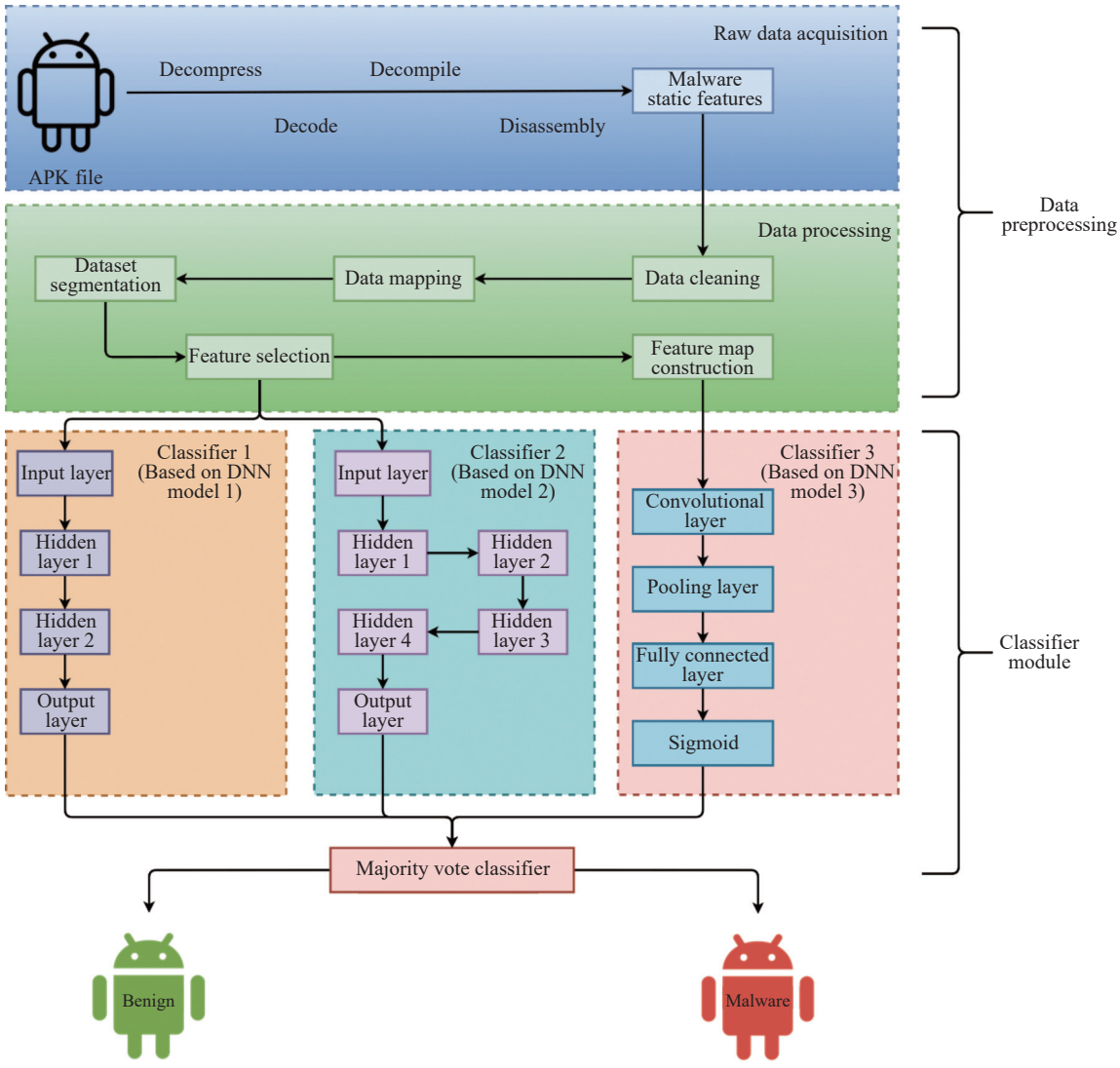

Fig. 1 DCEL overall framework

DCEL mainly includes the following two stages:

In the preprocessing stage, the APK file (Android application package) is first decompiled through the disassembly tool to obtain the static characteristics of the malware we need. If it is included in our pre-defined static feature set, we will use the static feature corresponding to the sample. The feature is recorded as 1, otherwise, it is recorded as 0. After analyzing a large number of APKs, we can get an initial malware static characteristic dataset, and mark the label of malware as 0 and the label of benign software as 1. Then there are four steps including data cleaning, data mapping and dataset segmentation, feature selection, and feature map construction.

In the classification stage, we use a deep learning fusion mechanism to classify malware. First, the processed dataset is directly trained using different levels of DNN to obtain two DNN models. On the other hand, it is converted into a two-dimensional gray image to construct an API feature map, and then a CNN classification model is used to train the API feature map. Finally, using the ensemble learning method, each classifier votes for a certain category, and the category with the majority of votes is used as the overall output of the ensemble model.

The next two parts will respectively elaborate on the specific scheme of data preprocessing and the classification model of the classification stage.

3.2 Data preprocessing

3.2.1 Raw data acquisition

The raw data acquisition is the first step in the analysis of the Android application software APK file. By decompressing the APK, you can get the AndroidManifest.xml file and the classes.dex file, and decode, decompile and disassemble the two types of files respectively. Convert it into a file type that is more readable and easier to understand. Here we use APKTool to decode the AndroidManifest.xml file and decompile the classes.dex file. Feature extraction from the converted file includes four types of feature information, including permission feature, method API feature, intention component feature, and Android system signature instruction feature, as shown in Table 1. These features are existential variables, either exist or do not exist, and you can intuitively confirm whether they appear in the Android application software. Then we combine these discrete features into the original data feature vector set we need.

Table 1 Some static characteristics after APK file characteristics

Static characteristics	Category
SEND_SMS	Manifest permission
READ_PHONE_STATE	Manifest permission
Ljava.net.URLDecoder	API call signature
Android.content.pm.Signature	API call signature
Android.intent.action.PACKAGE_REPLACED	Intent
Android.intent.action.SEND_MULTIPLE	Intent
remount	Commands signature
/system/app	Commands signature

3.2.2 Data cleaning

Generally speaking, data cleaning is the process of streamlining the database to remove duplicate records and converting the remaining part into a standard acceptable format. The data cleaning standard model is to input data to the data cleaning processor, "clean up" the data through a series of steps and then output the cleaned data in the desired format. Data cleaning deals with data loss, out-of-bounds value, inconsistent code, duplicate data and other issues from the aspects of data accuracy, completeness, consistency, uniqueness, timeliness, and validity.

Data cleaning is generally for specific applications, so it is difficult to summarize unified methods and steps, but corresponding data cleaning methods can be given according to different data.

(i) Methods to solve incomplete data (i.e., missing values)

In most cases, missing values must be filled in manually (i.e., manually cleaned up). Of course, some missing values can be derived from this data source or other data sources, which can replace missing values with average, maximum, minimum or more complex probability estimates to achieve the purpose of cleaning up.

(ii) Error value detection and solutions

Use statistical analysis methods to identify possible

error values or outliers, such as deviation analysis, identify values that do not comply with distribution or regression equations, or use simple rule libraries (common sense rules, business specific rules, etc.) to check data values, or use constraints between different attributes, external data to detect and clean up data.

(iii) Methods of detecting and eliminating duplicate records

Records with the same attribute value in the database are considered to be duplicate records. The records are checked for equality by judging whether the attribute values between the records are equal. The equal records are merged into one record (i.e., merge/clear). Combining/clearing is the basic method to eliminate weight.

3.2.3 Data mapping and dataset segmentation

After the above steps, the data in the standard format is obtained. Since the label column in the sample data is composed of letters, in order to eliminate its influence on the algorithm, we need to convert the letters into numerical values. The class tag consists of two types of data, namely S and B, where B stands for Benign and S stands for Malware. We use 0 and 1 for replacement, which makes these unavailable features available. Finally, we randomly divide the data into the training set and test set according to 0.8:0.2, and then randomly use 0.2 ratio of data in the training set as the verification set.

3.2.4 Feature selection

Feature selection is an important process in data preprocessing, which refers to the process of selecting relevant features from a given dataset. By screening important feature subsets, data dimensions are reduced, the time for classifier modeling is reduced, and the accuracy of intrusion detection is improved. In fact, when the number of features exceeds a certain limit, the results of data modeling will deteriorate. Some features in the dataset do not contain or contain very little information, which has little impact on modeling.

Chi-square test is a commonly used hypothesis testing method based on χ^2 distribution. Its null hypothesis H_0 is: there is no difference between the observed frequency and the expected frequency. The basic idea of the test is: first assume that H_0 is established, and calculate the χ^2 value based on this premise, which represents the degree of deviation between the observed value and the theoretical value. According to the χ^2 distribution and the degree of freedom, the probability P of obtaining the current statistics and more extreme cases can be determined when the H_0 hypothesis is established. If the current statistic is greater than the P value, it indicates that the observed value deviates too much from the theoretical value. The invalid hypothesis should be rejected, indicating that there is a significant difference between the comparative data; otherwise, the invalid hypothesis cannot be rejected, and the actual situation represented by the sample cannot be considered as the theoretical assumptions are different.

The idea of chi-square test is to judge the correct rate of the theoretical value through the deviation between the observed value and the theoretical value. The basic formula of the chi-square test, which is the calculation formula of χ^2, is the deviation between the observed value and the theoretical value:

$$\chi^2 = \sum \frac{(A-E)^2}{E} = \sum_{i=1}^{k} \frac{(A_i - E_i)^2}{E_i} = \sum_{i=1}^{k} \frac{(A_i - np_i)^2}{np_i} \quad (1)$$

where A represents the observation frequency (observation value), E represents the expected frequency (theoretical value) calculated based on H_0, and k is the number of observation values. A_i is the observation frequency at level i, E_i is the expected frequency at level i, n is the total frequency, and p_i is the expected frequency at level i. The expected frequency E_i of level i is equal to np_i, k is the number of cells, i=1, 2, 3, ⋯, k.

When the chi-square test is applied to feature selection, we do not need to know the degrees of freedom, do not know the chi-square distribution, we just need to sort according to the calculated χ^2. When the degree of freedom is 1, the larger the chi-square value and the smaller the probability, the more meaningful the corresponding feature, so the chi-square test is used for feature extraction. Generally, in a four-grid table, if the value in the first row and the first column is expressed as a, the value in the second row and the second column is expressed as b, the value in the third row and the third column is expressed as c, and the value in the fourth row and the fourth column is expressed as d. The chi-square test formula in the four-grid table can be transformed into

$$\chi^2 = \sum \frac{(A-E)^2}{E} = \frac{n(ad-bc)^2}{(a+b)(a+c)(b+d)(c+d)}. \quad (2)$$

In the classification algorithm for malware detection, features are usually preceded by a lot of redundancy and high correlation. These features will not only slow down

the classification process and increase the computational overhead, but also prevent the classifier from making accurate decisions. This method fully considers the correlation between features and classes in Android malware data, as well as each feature. It can quickly remove a large number of irrelevant features, and the selected features will greatly reduce the dimensions of the data without affecting the classification accuracy.

At the same time, we use the features that are most relevant to the classification of malware to classify software security threat levels. According to the ranking results, the top 20% of the attributes are classified as high risk A, 20% to 40% are classified as risk B, 40% to 60% are classified as risk C, and 60% to 80% are classified as risk D, the remaining features are classified as risk E, as shown in Table 2.

Table 2 Malware threat level table

Malware threat level	Threat level	Related attribute
High risk	A	Transact,onServiceConnected,bindService,SEND_SMS,READ_PHONE_STATE,⋯
Danger	B	WRITE_HISTORY_BOOKMARKS,TelephonyManager.getSubscriberId,WRITE_SYNC_SETTINGS,⋯
Slight danger	C	READ_CALL_LOG,Android.intent.action.PACKAGE_ADDED,ACCESS_NETWORK_STATE,⋯
Ordinary	D	TelephonyManager.getNetworkOperator,Android.intent.action.SENDTO,SET_ALARM,⋯
Safety	E	ACCESS_SURFACE_FLINGER,Android.intent.action.ACTION_POWER_CONNECTED,⋯

3.2.5 Feature map structure

The convolutional neural network has unique advantages in speech recognition and image processing with its special structure of sharing local weights. It can directly use image data as input, not only without manual image preprocessing and additional feature extraction and other complex operations, and with its unique fine-grained feature extraction method, the processing of the image has reached almost the level of manpower. The feature that images of multi-dimensional input vectors can also be directly input to the network avoids the complexity of data reconstruction in the process of feature extraction and classification. Therefore, we convert the one-dimensional static attribute data into a binary gray image, that is, a black and white image: each pixel has only two possible 0 and 1, where 0 represents black, 1 represents white, and the data type is usually 1 binary bit.

We use the feature selection based on chi-square test to obtain 196 static attributes from the defined 215 static attributes, and then use tensor deformation operation to convert them into binary grayscale images with a width of 14 and a height of 14. The tensor here is the generalization of the matrix to any dimension, and tensor deformation refers to changing the rows and columns of the tensor to get the desired shape. The total number of elements of the deformed tensor is the same as the initial tensor. That is, the 1×196 matrix is converted into a 14×14 binary image.

3.3 Classification model

In recent years, CNN and DNN structures have been widely used in various fields. On the basis of obtaining complete software information to be detected and a large number of features through static analysis, using the deep structural features of the nonlinear mapping between DNN and CNN and the corresponding learning algorithm can automatically mine high-relevance deep features, avoiding large the dilemma of manually screening the behavioral characteristics of big data in the data environment, while improving the accuracy of application software detection. Then, the integrated learning fusion technology is used to effectively integrate multiple models, so that the model can simultaneously use multiple different types of static features to describe the various attributes of the Android application software in an all-round way. Besides, the special network structure can make the features more effective. Here we use ensemble learning to integrate several models. Fig. 2 is a diagram of the DCEL classification architecture.

CNN is composed of neurons with learnable weights and biases, including three parts: convolutional layer, pooling layer, and fully connected layer. CNN-based malware detection is shown in Fig. 3. Usually, there are multiple convolutional layers in a neural network, and multiple convolutional layers sequentially perform convolu-

tion operations on the image to extract information such as the edge and shape of the image. The pooling layer is responsible for down-sampling the data generated by the convolutional layer (that is, reducing the spatial resolution of the input layer) to reduce processing time and enable computing resources to handle larger data scales. The fully connected layer classifies the output generated by the convolutional layer and the pooling layer. Each neuron in this layer is connected to each neuron in the previous layer. The fully connected layer can integrate the local information that is distinguished by categories in the convolutional layer or the pooling layer.

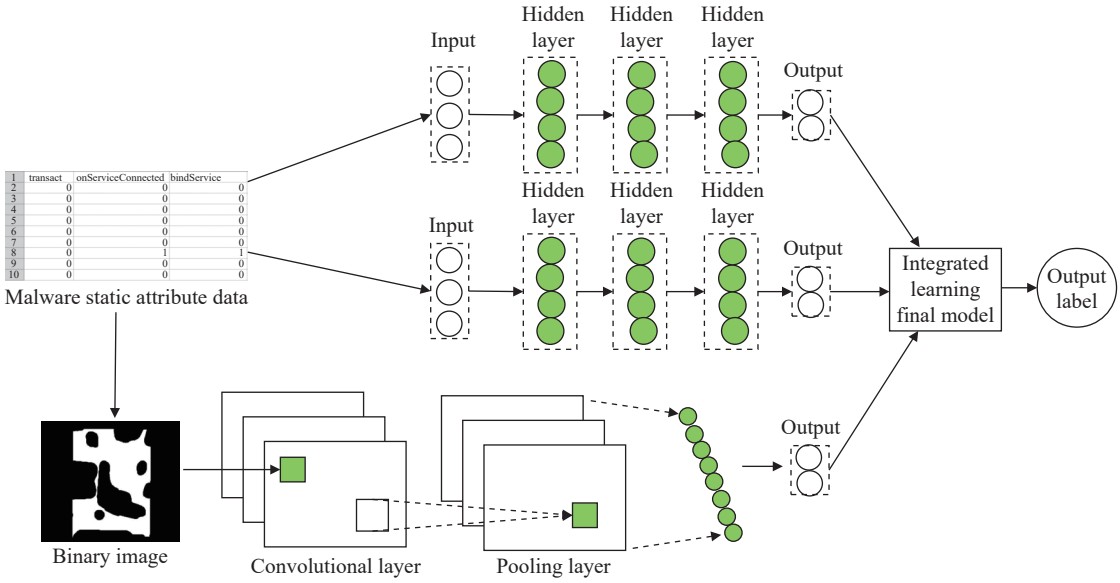

Fig. 2 DCEL classification module

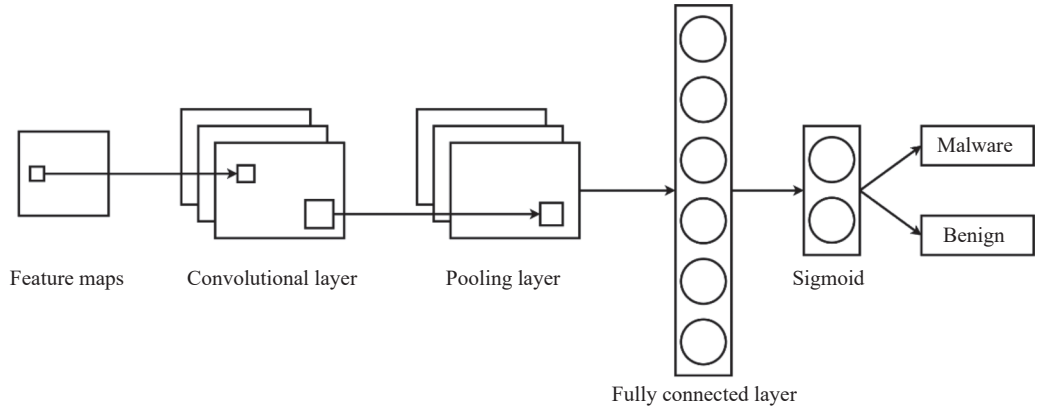

Fig. 3 Malware detection model based on CNN

In CNN, the weight update is based on the backpropagation algorithm. CNN is essentially an input-to-output mapping. It can learn a large number of mapping relationships between input and output without requiring any precise mathematical expressions between input and output. As long as the convolutional network is trained with the known pattern, the network has the ability to map between input and output pairs. The convolutional network performs supervised training, so its sample set is composed of vector pairs of the form: input vector and ideal output vector. All these vector pairs should be derived from the actual "running" structure of the network to simulate the system, and they can be collected from the actual running system. Before starting training, all weights should be initialized with some different random numbers. "Small random number" is used to ensure that the network will not enter a saturated state due to excessive weights, resulting in training failure; "different" is used to ensure that the network can learn normally.

DNN can be understood as a neural network with many hidden layers. Divided by the location of different layers from DNN, the neural network layer inside DNN can be divided into three categories, input layer, hidden layer, and output layer [31], as shown in Fig. 4. Generally speaking, the first layer is the input layer, and the last layer is the output. Layers and the number of layers in the middle are all hidden layers. The layers are fully connected, that is, any neuron in the ith layer must be connected to any neuron in the $(i+1)$th layer. A linear relationship is learned between the output and the input, and the intermediate output result $z=\sum_{i=1}^{m} w_i x_i + b$ is obtained where w_i is the linear relationship coefficient, b is the offset .Then we need to find the appropriate linear coefficient matrix $\boldsymbol{w}$ and offset vector $\boldsymbol{b}$ corresponding to all hidden layers and output layers, so that the output results calculated by all input training samples are as close to or as close as possible to the sample output. Use a suitable loss function to measure the output loss of the training sample, and then optimize the loss function to find the minimized extreme value, a series of linear coefficient matrix $\boldsymbol{W}$, offset vector $\boldsymbol{b}$, which is our result. In DNN, the most common process of optimizing the extremum of the loss function is generally completed step by step through the gradient descent method, and it can also be other iterative methods such as the Newton method and the quasi-Newton method. Since the gradient descent method has three variants, batch, mini-Batch, and random, the difference is only in the selection of training samples during iteration.

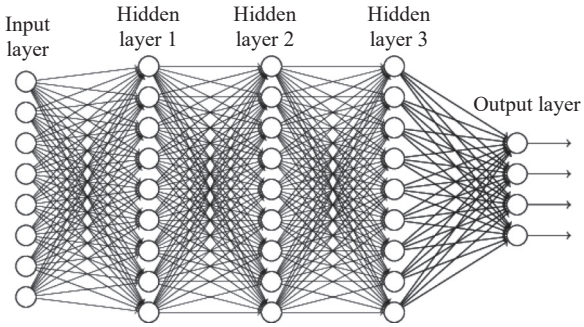

Fig. 4 Detection model based on DNN

The pseudo-code of the DCEL algorithm is like Algorithm 1, where N_{cnn} is the number of convolutional blocks, and N_{dnn} is the number of hidden layers of the neural network. Table 3 shows the detailed parameters of the CNN model used.

Algorithm 1 DCEL

Input: x rows of malware samples, each row has k columns of attributes

Output: Category of each sample

```
b=x;
a=x.reshape()//Convert to binary image
for i = 1; i<=N_cnn; i++ do
    a=Conv2D(a)
    a=Relu(a)
    a=maxpooling(a)
    result1=fullyconnected(a)
end for
for i = 1; i<=N_dnn; i++ do
    b1=Dense(b)
    b1=Relu(b1)
    result2=fullyconnected(b1)
end for
for i = 1; i<=N_dnn; i++ do
    c1=Dense(b)
    c1=Relu(b1)
    result3=fullyconnected(c1)
end for
result=MajorityVoteClassifier(result1,result2, result3)
Return result
```

Table 3 Network parameters of CNN

Layer(type)	Output shape	Parameter
Conv2d_3(Conv2D)	(None, 12, 12, 32)	320
Max_pooling2d_3(MaxPooling2D)	(None, 6, 6, 32)	0
Conv2d_4(Conv2D)	(None, 4, 4, 64)	18 496
Max_pooling2d_4(MaxPooling2D)	(None, 2, 2, 64)	0
Flatten_2(Flatten)	(None, 256)	0
Dense_3(Dense)	(None, 512)	131 584
Dense_4(Dense)	(None, 1)	513

The first convolution layer C1 performs a convolution operation and has 32 kernels of size 3×3. The result of the C1 layer is 32 feature maps with a size of 12×12. After the C1 layer, there is a 2×2 max pool operation, and the result is 32 feature maps with a size of 6×6. The kernel size of the second convolutional layer C2 is also 3×3, but there are 64 channels. The result is 64 feature maps of size 4×4. After the second 2×2 max pool layer P2, 64 feature maps with a size of 2×2 are generated. The last two layers are fully connected layers, the result sizes are 512 and 1, respectively, and the overall number of trainable parameters is 150 913. The activation layer uses the Relu function and the sigmoid function to output the category. In the DNN in DCEL, DNN model 1 consists of two

intermediate fully connected layers, each with 16 hidden units, using Relu as the activation function, and Dropout to alleviate the occurrence of overfitting. DNN model 2 consists of four intermediate fully connected layers, each with 32 hidden units, and also uses Relu as the activation function and Dropout to alleviate the occurrence of overfitting.

4. Experiment

4.1 Experimental environment

The experimental environment and configuration of this article are shown in Table 4.

Table 4 Experimental software and hardware environment

Hardware environment	Software environment
CPU 2.11GHz Intel Core i5 GPU NVIDIA GeForce MX250 RAM 16GB 2667Mhz DDR4	OS Windows 10 Python 3.6.4 Pytorch 1.4.0 Sklearn 0.19.1 Pandas 0.22.0 Numpy 1.14.2

4.2 Dataset

Malware detection is inseparable from the support of effective malicious sample sets. The malicious sample sharing website provides a large number of malicious samples for analysis and use by researchers, which has greatly promoted the development of malware detection technology. The malware dataset can be directly used for malware detection and family identification. We use the dataset provided by Yerima et al. [13], which is based on the Drebin [14] and Malgenome [16] datasets.

The Drebin and Malgenome datasets consist of four types of features: Manifest Permission, API call signature, Intent, and Commands signature. Table 5 shows the relevant information for each dataset.

Table 5 Malware dataset

Parameter	Malgenome-215	Drebin-215
Number of malware	1 260	5 560
Number of benign	2 539	9 476
Total number	3 799	15 036
Number of features	215	215

Malgenome was built by the Zhou and Jiang team for over a year. It contains 49 families, of which 2 539 are benign and 1 260 are malware samples from the Android Malware Genome Project. The team has spent a lot of manpower and material resources on it. The samples in the dataset are analyzed in detail, and the results of the analysis show: (i) 86% of malware is produced by repackaging, that is, the producer selects popular applications for decompilation, implants malicious loading code, and then repacks it. Compile and upload to the market to attract users to download for free; (ii) 36.7% of malware uses known platform vulnerabilities to increase privileges; (iii) 93% of malware uses HTTP-based protocols to receive control commands from C&C servers. This data integration is a collection of reference malware samples widely used by the malware research community. Drebin is developed by Arp et al. based on Malgenome. The Drebin dataset analyzes the detection results of 10 anti-virus engines in VirusTotal. If more than two engine results indicate that they are malicious, they are added to the dataset and the tags returned by the engines are used as their family information. VirusTotal is a system website that integrates 53 anti-virus engines. The dataset contains 179 families, a total of 5 560 malware. The Drebin sample is also public and widely used in the research community. Table 6 shows the relevant characteristics of the Drebin and Malgenome datasets.

Table 6 Drebin and Malgenome dataset description

Type and number of features	Feature
Manifest Permission(113)	SEND_SMS, READ_SMS, RECEIVE_SMS, READ_PHONE_STATE, WRITE_SMS, ⋯
API call signature(73)	Transact, onServiceConnected, bindService, attachInterface, Ljava.lang.Class.getField, ⋯
Intent(23)	Android.intent.action.BOOT_COMPLETED, Android.intent.action.SEND_MULTIPLE, ⋯
Commands signature(6)	Remount, chown, /system/bin, /system/app, ⋯

4.3 Evaluation index

In this section, we will introduce and discuss experiments for evaluating DCEL performance. The same configuration is used in the Drebin-215 and Malgenome-215 experiments to maintain consistency. The training set is used to construct the DCEL model through the hierarchical tenfold cross-validation method. In order to facilitate

subsequent performance comparison, the following indicators are predefined. The performance of the proposed DCEL classification mechanism is evaluated by calculating performance metrics such as accuracy, error rate, recall rate, precision, *F*-measure, and AUC value. In (3) to (7), TP represents the number of true positives, TN represents the number of true negatives, FP represents the number of false positives, and FN represents the number of false negatives.

(i) Accuracy: the proportion of correctly classified instances in the total sample

$$\text{Accuracy} = \frac{\text{TP}+\text{TN}}{\text{TP}+\text{TN}+\text{FP}+\text{FN}} \tag{3}$$

(ii) Error rate: the percentage of misclassified instances in the total sample

$$\text{Error rate} = \frac{\text{FP}+\text{FN}}{\text{TP}+\text{TN}+\text{FP}+\text{FN}} \tag{4}$$

(iii) Recall rate/detection rate: the proportion of malicious instances that are correctly detected as malicious

$$\text{Recall} = \frac{\text{TP}}{\text{TP}+\text{FN}} \tag{5}$$

(iv) Precision: the proportion of malicious instances that are correctly detected as malicious

$$\text{Precision} = \frac{\text{TP}}{\text{TP}+\text{FP}} \tag{6}$$

(v) *F*-measure: the harmonic average of precision rate and recall rate

$$F\text{-measure} = \frac{2\times\text{Precision}\times\text{Recall}}{\text{Precision}+\text{Recall}} \tag{7}$$

(vi) AUC value: the size of the area under the ROC curve.

(vii) Detection time: the time it takes to test the model. This is the time (in seconds) to test the built model from the test set.

4.4 Experimental results and analysis

In this section, we evaluate the performance of the proposed model. The experiment in this article is mainly carried out on the Drebin dataset. In order to verify the versatility of DCEL, we verify it on the Malgenome dataset. Fig. 5 and Fig. 6 show the loss and accuracy of DCEL on the Drebin training dataset and validation dataset. It shows that the loss value of the model gradually decreases, the accuracy rate increases and the model does not overfit.

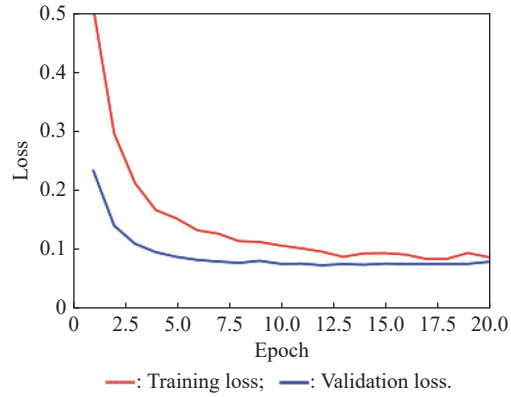

Fig. 5 Loss value of DCEL in training and validation datasets

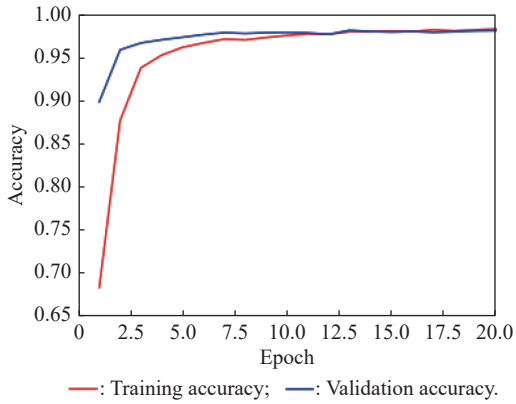

Fig. 6 Accuracy of DCEL in training and validation datasets

We design four experiments to test DCEL:

Experiment 1: On the Drebin dataset, a variety of machine learning classification algorithms are used for experimental comparison with the proposed DCEL model. Table 7 compares classification algorithms such as *K*-means, naive Bayes (NB), support vector machine (SVM), logistic regression (LR), decision tree (DT), random forest (RF), DroidFusion [13] and DCEL. From the performance of the model on indicators, we can see that the accuracy rate, precision rate, recall rate, and *F* value of the DECL classification algorithm are better than other algorithms, but the time spent testing the model is relatively high. Compared with other ensemble learning methods, the computational cost is lower. For the malware dataset Drebin, the DCEL algorithm is superior to other traditional machine learning algorithms in indicators such as accuracy and recall. Because this method will first use the original data DNN to generate a model, then convert the original data into a gray image using a CNN to generate the model, and finally use the ensemble learning method to get the final classifier. The feature engineering of deep learning is fully automated, learning all features at one time, without manual design, avoiding the inaccuracy of the feature selection algorithm in selecting

sample attributes, and improving the accuracy of classification. The ensemble learning can combine the advantages of these classifiers to make up for their shortcomings and provide better solutions. However, compared with other algorithms, ensemble learning takes a longer time because of the integration of multiple models. Experimental results prove that using ensemble learning combined with deep learning algorithms can effectively improve the classification performance of malware detection models.

Table 7 Comparison between DCEL and traditional machine learning algorithms in terms of accuracy, AUC value and other indicators

Algorithm	Accuracy	Error rate	Recall rate	Precision	*F*-measure	AUC	Time/s
K-means	0.716 994	0.283 006	0.915 674	0.568 259	0.701 299	0.759 768	0.06
DT	0.979 381	0.020 619	0.983 820	0.983 820	0.983 820	0.977 703	0.03
LR	0.979 049	0.020 951	0.985 908	0.981 299	0.983 598	0.976 455	0.02
SVM	0.977 719	0.022 281	0.983 299	0.981 761	0.982 529	0.975 609	0.03
Gaussian NB	0.708 015	0.291 985	0.552 192	0.981 447	0.706 747	0.766 930	0.1
AdaBoost	0.963 086	0.036 914	0.977 557	0.964 967	0.971 221	0.957 615	0.23
RF	0.983 705	0.016 295	0.971 586	0.983 302	0.977 409	0.981 096	0.31
DroidFusion[13]	0.984	0.016	0.984	0.992	0.988	None	None
DCEL	0.990 772	0.009 228	0.991 127	0.989 578	0.990 352	0.995 831	0.15

Experiment 2: Experiment 2 is conducted based on the Drebin dataset to compare the classification performance of individual CNN, DNN1, DNN2, and DCEL. DNN1 contains two intermediate fully connected layers, each of which has 16 hidden units, uses Relu as the activation function, and Dropout alleviates the occurrence of overfitting; DNN2 contains four intermediate fully connected layers, each with 32 hidden units, using Relu as the activation function and Dropout to mitigate the occurrence of overfitting. Table 8 shows the performance comparison of DCEL with CNN and DNN. The performance of DCEL is better than other algorithms. As can be seen from Fig. 7 to Fig. 11, DCEL is better than single CNN and DNN in terms of Accuracy, *F* value, and AUC. The reason for this result is that each classifier of DCEL is trained on different subsets with different errors, and combining the three classifiers gives the best classification boundaries. This helps us to find a global solution that can reduce the false positive rate and improve detection accuracy.

Table 8 Comparison of DCEL and a single model in terms of accuracy and other indicators

Algorithm	Accuracy	Error rate	Recall rate	Precision	*F*-measure	AUC
DNN1	0.987 695	0.012 305	0.989 562	0.988 530	0.989 045	0.984 698
DNN2	0.982 042	0.017 958	0.984 821	0.967 544	0.976 106	0.986 398
CNN	0.984 037	0.015 963	0.973 660	0.982 585	0.978 102	0.992 325
DECL	0.990 772	0.009 228	0.991 127	0.989 578	0.990 352	0.995 831

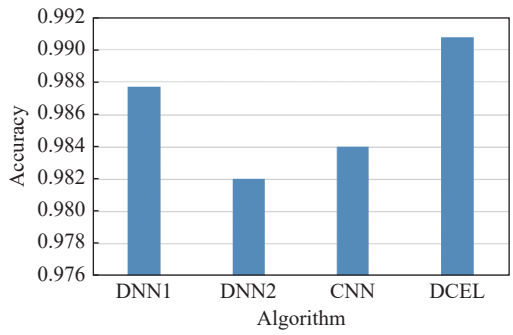

Fig. 7 Comparison of accuracy between DCEL and other neural network algorithms

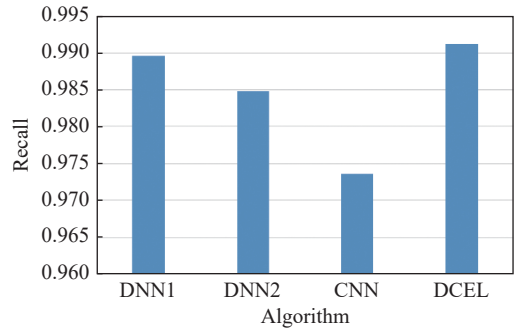

Fig. 8 Comparison of the recall rate of DCEL and other neural network algorithms

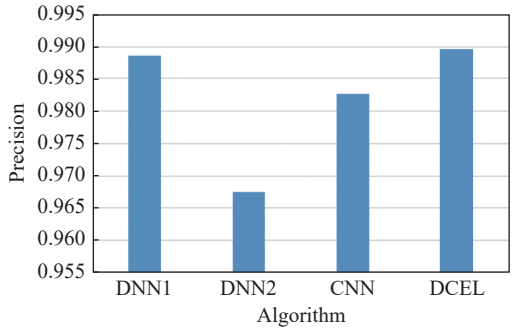

Fig. 9 Comparison of precision between DCEL and other neural network algorithms

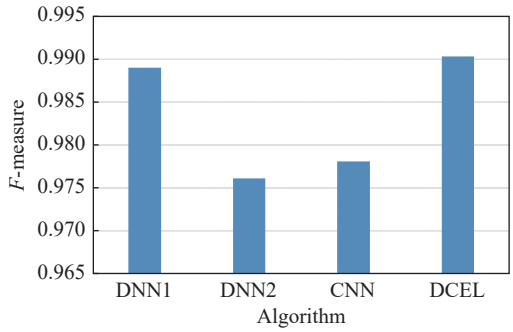

Fig. 10 Comparison of *F*-measure between DCEL and other neural network algorithms

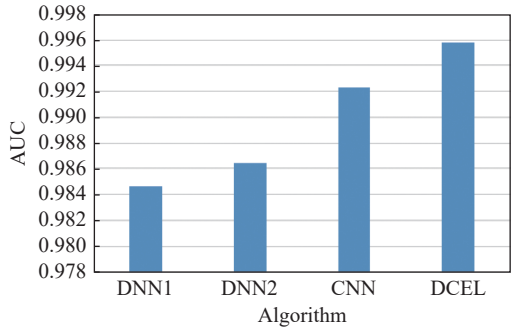

Fig. 11 Comparison of AUC between DCEL and other neural network algorithms

Experiment 3: The classification performance of the DCEL model is verified by using the Malgenome dataset to prove that the method has a certain universality and can be used in actual malware detection systems. The results of Table 9 and Fig. 12 show that the DCEL algorithm has a high accuracy rate and high detection accuracy. It can be seen that the detection mechanism for Android malware proposed in this article has high practicability and effectiveness for the detection of massive Android malware.

Table 9 Performance comparison between DCEL and other models on the Malgenome dataset

Algorithm	Accuracy	Error rate	Recall rate	Precision	*F*-measure	AUC	Time/s
DNN1	0.990 789	0.009 211	0.991 379	0.978 723	0.985 011	0.999 004	0.005
DNN2	0.989 474	0.010 526	0.984 127	0.984 127	0.984 127	0.988 126	0.011
CNN	0.988 158	0.011 842	0.978 448	0.982 684	0.980 562	0.999 339	0.028
DCEL	0.994 737	0.005 263	0.991 379	0.991 379	0.991 379	0.999 371	0.051

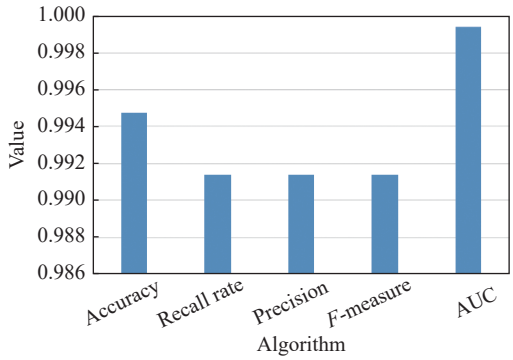

Fig. 12 Classification performance of DCEL on the Malgenome dataset

Experiment 4: Compare the categories of characteristics of different malware threat levels and find the categories of common characteristics of malware. We use correlation feature technology to find the most relevant features, and classify the threat level of malware through the correlation of features. As can be seen from Table 10, among the characteristics of Type A threats, API call signature accounts for 42.5%, and Manifest Permission accounts for 9.7%. Among the characteristics of Type B threats, API call signature accounts for 23.3%, and Manifest Permission accounts for 15%. Among the characteristics of E-type threats, API call signature accounts for 4%, and Manifest Permission accounts for 29.2%. Here we can conclude that high-risk software uses more API call signature and Intent, and less Manifest Permission. The security software uses less API call signature and Intent, and more Manifest Permission. We believe that for malware detection, the more API call signature a software uses, the greater the threat of the software, and the greater the probability that the software is malware.

Table 10 Categories of samples with different threat levels

Malware threat level	Number of Manifest Permission	Number of API call signature	Number of Intent	Number of commands signature
A	11	31	1	0
B	17	17	3	2
C	25	12	4	2
D	27	10	5	2
E	33	3	10	0

5. Conclusion and future work

5.1 Conclusion

In this article, we propose DCEL based on neural networks, which uses model fusion methods to improve the predictive ability of deep learning algorithms. In order to prove the effectiveness of the method, we conduct a large number of experiments on the Drebin and Malgenome datasets processed by the feature selection method. We observe the interesting properties of malware and the advantages of model fusion, and propose correlations based on features to find the category of the attributes most used by malware. This mechanism DCEL can effectively combine deep learning algorithms to improve the accuracy of detection. This method will first use the original data DNN to generate a model, then convert the original data into a gray image using a CNN to generate the model, and use the ensemble learning method to get the final classifier. Through experiments, the effectiveness of the mechanism is proved, and the experimental results of the model compared with it are given. At the same time, we also discuss the types of features in different malware threat levels, and find that the more API call signature a software uses, the greater the threat of the software, and the greater the probability that the software is malware.

5.2 Future work

There are three limitations regarding our work and related future work. First of all, the main purpose of this article is to use representation deep learning methods to prove the effectiveness of malware classification, so we do not study the parameter tuning of algorithms such as CNN and DNN. In practical applications, the size and number of classes of malware data are not fixed, and the generalization ability of our method needs further verification. Secondly, the dynamic malware detection method uses classic machine learning methods and uses many time-series features to prove their high efficiency. Our method only uses static characteristics and completely ignores time characteristics. We will study how to incorporate these temporal features in future work, such as using recurrent neural networks. Finally, the work of this article only classifies known malware. The ability to identify unknown malware is also very important to protect the safety of users. How to further study how to increase the ability of unknown malware in future work.

References

[1] AJMA B, ADDAB C. Research-supported mobile applications and internet-based technologies to mediate the psychological effects of infertility: a review. Reproductive BioMedicine Online, 2021, 42(3): 679–685.
[2] ZHANG T, BAI Y, SONG M Y, et al. Research on test methods for AI industrial application capabilities of smart mobile terminals. Journal of Computer and Communications, 2021, 9(12): 106–115.
[3] IDC. Smartphone market share. https://www.idc.com/promo/smartphone-market-share/os.
[4] WANG H Y, LIU Z, LIANG J Y, et al. Beyond google play: a large-scale comparative study of chinese Android app markets. Proc. of the Internet Measurement Conference, 2018: 293−307.
[5] 360 Lab. 2019 Android malware annual report. http://zt.360.cn/1101061855.php?dtid=1101062360&did=211012248.
[6] CEN L, GATES C S, SI L, et al. A probabilistic discriminative model for Android malware detection with decompiled source code. IEEE Trans. on Dependable and Secure Computing, 2015, 12(4): 400–412.
[7] KOUMARAS H, MAKROPOULOS G, BATISTATOS M, et al. 5G-enabled UAVs with command and control software component at the edge for supporting energy efficient opportunistic networks. Energies, 2021, 14(5): 1480.
[8] ELAYAN O N, MUSTAFA A M. Android malware detection using deep learning. Procedia Computer Science, 2021, 184(2): 847–852.
[9] BAKOUR K, UNVER H M. DeepVisDroid: Android malware detection by hybridizing image-based features with deep learning techniques. Neural Computing and Applications, 2021, 33: 11499–11516.
[10] KHARIWAL K, GUPTA R, SINGH J, et al. R-MFdroid: Android malware detection using ranked manifest file components. International Journal of Innovative Technology and Exploring Engineering, 2021, 10(7): 55–64.
[11] ONWUZURIKE L, MARICONTI E, ANDRIOTIS P, et al. MaMaDroid: detecting Android malware by building Markov chains of behavioral models. ACM Transactions on Privacy and Security, 2019, 22(2): 1–34.
[12] MARIN G, CASAS P, CAPDEHOURAT G. Deep in the dark-deep learning-based malware traffic detection without expert knowledge. Proc. of the IEEE Security and Privacy Workshops, 2019: 36−42.
[13] YERIMA S Y, SEZER S. Droidfusion: a novel multilevel

classifier fusion approach for Android malware detection. IEEE Trans. on Systems, Man, and Cybernetics, 2019, 49(2): 453–466.
[14] ARP D , SPREITZENBARTH M , HUBNER M, et al. Drebin: effective and explainable detection of Android malware in your pocket. Proc. of the Network and Distributed System Security Symposium, 2014: 23−26.
[15] ZHOU Y J, JIANG X X. Dissecting android malware: characterization and evolution. Proc. of the IEEE Symposium on Security and Privacy, 2012: 95–109.
[16] YAO H, WANG Y, YANG Y X. Range estimation of few-shot underwater sound source in shallow water based on transfer learning and residual CNN. Journal of Systems Engineering and Electronics, 2023, 34(4): 839–850.
[17] KULEVOME D K B, WANG H, WANG X G. Deep neural network based classification of rolling element bearings and health degradation through comprehensive vibration signal analysis. Journal of Systems Engineering and Electronics, 2022, 33(1): 233–246.
[18] LIU X Y, WENG J, ZHANG Y, et al. Android malicious application detection based on APK signature information feedback. Journal of Communications, 2017, 38(5): 190–198. (in Chinese)
[19] ZHENG M, SUN M S, LUI J C S. DroidAnalytics: a signature based analytic system to collect, extract, analyze and associate android malware. Proc. of the 12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications, 2013: 163−171.
[20] YAN Y. Research on Android malware detection based on permission correlation. Science & Technology, Economy, Market, 2017(9): 10–11. (in Chinese)
[21] SATO R, CHIBA D, GOTO S. Detecting Android malware by analyzing manifest files. Proceedings of the Asia Pacific Advanced Network, 2013, 36: 23−31.
[22] DU Y, WANG J F, LI Q. An Android malware detection approach using community structures of weighted function call graphs. IEEE Access, 2017, 5: 17478–17486.
[23] ZHANG J X, QIN Z, ZHANG K H, et al. Dalvik opcode graph based Android malware variants detection using global topology features. IEEE Access, 2018, 6: 51964–51974.
[24] BURGUERA I, ZURUTUZA U, NADJM-TEHRANI S. Crowdroid: behavior-based malware detection system for android. Proc. of the 1st ACM Workshop on Security and Privacy in Smartphones and Mobile devices, 2011: 15−26.
[25] SINGH A K, WADHWA G, AHUJA M, et al. Android malware detection using LSI-based reduced opcode feature vector. Procedia Computer Science, 2020, 173: 291–298.
[26] ROY A, JAS D S, JAGGI G, et al. Android malware detection based on vulnerable feature aggregation. Procedia Computer Science, 2020, 173: 345–353.
[27] UNVER H M, BAKOUR K. Android malware detection based on image-based features and machine learning techniques. SN Applied Sciences, 2020, 2: 1299.
[28] COPTY F, DANOS M, EDELSTEIN O, et al. Accurate malware detection by extreme abstraction. Proc. of the 34th Annual Computer Security Applications Conference, 2018: 101−111.
[29] MACHIRY A, REDINI N, GUSTAFSON E, et al. Using loops for malware classification resilient to feature-unaware perturbations. Proc. of the 34th Annual Computer Security Applications Conference, 2018: 112−123.
[30] KIM T G, KANG B J, RHO M, et al. A multimodal deep learning method for android malware detection using various features. IEEE Trans. on Information Forensics and Security, 2019, 14(3): 773–788.
[31] GUO F H, XU B W, ZHANG W A. Training deep neural network for optimal power allocation in islanded microgrid systems: a distributed learning-based approach. IEEE Trans. on Neural Networks and Learning Systems, 2022, 33(5): 2057–2069.

Biographies

XU Xiaolong was born in 1977. He received his B.S. degree in computer and its applications, M.S. degree in computer software and theories and Ph.D. degree in communications and information systems at Nanjing University of Posts & Telecommunications, Nanjing, China, in 1999, 2002 and 2008, respectively. He worked as a postdoctoral researcher at Station of Electronic Science and Technology, Nanjing University of Posts & Telecommunications from 2011 to 2013. He is currently a professor in College of Computer, Nanjing University of Posts & Telecommunications. His current research interests include cloud computing and big data and information security.
E-mail: xuxl@njupt.edu.cn

JIANG Shuai was born in 1995. He received his B.E. degree in computer science and technology from Nanjing University of Posts and Telecommunications, Nanjing, China, in 2018. He is currently working as a researcher for Jiangsu Key Laboratory of Big Data Security & Intelligent Processing, Nanjing, China. His research interests include big data mining and information security.
E-mail: 1018041226@njupt.edu.cn

ZHAO Jinbo was born in 1999. He received his B.E. degree in Internet of Things engineering from Nanjing University of Posts and Telecommunications, Nanjing, China, in 2021. He is currently working for his Ph.D. degree in information network at Nanjing University of Posts and Telecommunications. His research interests include artificial intelligence and its application.
E-mail: 2021070705@njupt.edu.cn

WANG Xinheng was born in 1968. He received his B.E. and M.S. degrees in electrical engineering from Xi'an Jiaotong University, Xi'an, China, in 1991 and 1994, respectively, and Ph.D. degree in computing and electronics from Brunel University, Uxbridge, U.K., in 2001. He is currently a professor of networks with the School of Computing and Engineering, University of West London, London, U.K. His current research interests include wireless networks, Internet of Things, converged indoor positioning, cloud computing, and applications of wireless and computing technologies for health care.
E-mail: xinheng.wang@uwl.ac.uk