An Android Malware Detection Method Based on Optimized Feature Extraction Using Graph Convolutional Network

Zhiqiang Wang[1,2(✉)], Zhuoyue Wang[1], and Ying Zhang[1]

[1] Beijing Electronic Science and Technology Institute, Beijing 100070, China
wangzq@besti.edu.cn
[2] State Information Center, Beijing 100045, China

Abstract. With the development of the mobile Internet, mobile devices have been extensively promoted and popularized. Android, as the current popular mobile intelligent operating system, has encountered problems such as the explosive growth of Android malware while bringing convenience to users. The traditional Android malware detection methods have some problems, such as low detection accuracy and difficulty in detecting unknown malware. This paper proposes an Android malware detection method named Android malware detection method based on graph convolutional neural network (AGCN) based on the graph convolutional network (GCN) to solve the above problems. Firstly, we divide the Android software datasets according to family and software features and construct a directed network topology graph. At the same time, the permission features of APK files are extracted and vectorized. Then, we use GCN to learn the features of Android APK files… Finally, we compare AGCN with a multilayer perceptron (MLP), long and short-term memory (LSTM) neural network, bi-directional long and short-term memory (bi-LSTM) neural network, and deep confidence neural network (DCNN) for experiments. Experimental results show that the model has an accuracy of 98.55% for malware detection, demonstrating the detection method's effectiveness.

Keywords: Android Malware · Graph Convolutional Networks · Static Analysis · Graph Features

1 Introduction

As the development of the mobile Internet and the Internet of Things (IoT), smartphones represented by Android phones have rapidly applicated to every aspect of people's lives. Since smartphones contain many sensitive information, more and more attackers are using mobile smart terminals as the primary attack targets. As a result, mobile malwares, such as privacy theft, spam advertisements, and malicious chargebacks, are growing explosively. In the first half of 2021, a total of about 3.85 million new malicious samples were intercepted by 360 Mobile Guard on the mobile terminal, an increase of 267.2%

S. Goel and P. R. Nunes de Souza (Eds.): ICDF2C 2023, LNICST 571, pp. 283–299, 2024.
https://doi.org/10.1007/978-3-031-56583-0_19

compared with the first half of 2020 (1.048 million), and about 2.1 new mobile malicious samples per day on average are intercepted. In the first half of 2022, about 4.23 billion malicious attacks are blocked by 360 Mobile Guard for smartphone users in China, with an average of about 23.386 million malicious attacks per day [1].

The detection of Android malware based on the logical graph is an important research method and hotspot. However, there are problems in the current related work. For example, the feature extraction of logical graph structure is not comprehensive enough, resulting in lower detection accuracy. Moreover, the traditional machine learning algorithm is restricted to the processing of complex functions, and its accuracy is low, so it is difficult to adapt to new samples. Hence, it is a highly extensive research area to explore effective techniques for feature extraction from Android software and enhancing detection efficiency.

This paper conducts research on the detection of Android malware and proposes AGCN based on a graph convolutional neural network. The main innovations and contributions of this paper are as follows:

Design an Android application feature extraction and conversion logic graph algorithm and batch extract the permission features of each application into the corresponding text file (apkpermission.txt). In this technical process, the permission feature document generated by the feature extraction method is used to extract all features. The extracted features are then traversed based on the permission features to generate a permission feature library (Permission Set). Use one-hot encoding to generate feature vectors for determining topology node features. Then, the polar association relationship is constructed, and the directed network topology is formed. And use the topology map structure as an adjacency matrix to transform Android Apply features.

(1) An Android malware detection method based on a graph convolutional network is proposed. Based on the conversion of Android software into logic graphs, an Android malware detection method is designed to improve the traditional deep learning method and detection performance. Through comparison experiments of different neural network architectures and different parameter comparison experiments, the optimal parameter combination of neural network architectures of deep learning model is determined.
(2) Based on a large number of normal and malicious Android software samples, comparative experiments were carried out with four neural network models to verify and evaluate AGCN. The performance of the system is tested according to the performance metrics. Experimental comparisons show that this method has improved accuracy, precision, recall, and F1-score compared with LSTM, MLP, Bi-LSTM, and DBN.

The other sections of this paper are as follows: Sect. 2 introduces related work; Sect. 3 introduces the key technologies and algorithms of the detection method; Sect. 4 introduces the experimental design and result analysis; Sect. 5 summarizes this paper and proposes the following work.

2 Related Work

2.1 Research Status at Home and Abroad

Early analysis methods cannot effectively detect new malicious applications. Using traditional machine learning techniques (such as support vector machines, decision trees, random forests, and Bayesian networks) for Android malicious application detection is more effective than earlier methods. At the same time, the detection effect achieved by using deep learning technology is better than traditional machine learning methods. Abdelmonim Naway et al. [2] identified the deep learning method to detect malicious Android applications. In most of these methods, deep learning is combined with static analysis, while only a few are combined with dynamic analysis and hybrid analysis. Zhiqiang Wang et al. [3] combed through the methods of applying deep learning to Android malicious applications detection. The deep learning models are mainly Convolutional Neural Networks (CNN) and Deep Belief Networks (DBN). The extracted features mainly consist of API call features, permission features, and system call features, while component features, intent features, data flow features, and opcode sequence features are also involved to some extent. TaeGuen Kim et al. [4] proposed a multi-modal deep learning method as a malware detection model. To achieve effective feature representation in malware detection, multiple feature types are extracted to reflect the attributes of an Android application from various aspects. The feature extraction method based on existence or similarity is used to improve these features. A total of 7 static features are extracted, including permissions, components, environments, strings, Dalvik opcode sequences, API call sequences, and shared library function opcode features. Each feature type is used to train its corresponding deep neural network's initial network. The training results of the initial network are then used to train the final network.

In addition to the above methods of combining multiple features and using deep learning models for detecting malicious Android applications, a few researchers have used deep learning models to extract and abstract features from images automatically. Peter Zegzhda et al. [5] built a control flow graph through small files to obtain the features of API call sequence, mapped corresponding protection levels according to permissions required for API call, and then converted API call sequence and protection level into pixels of RGB images. The detection model based on a convolutional neural network was used for detection. A classification accuracy of 93.64% is achieved on a dataset consisting of 7192 normal samples and 24461 malicious samples. Luo Shiqi et al. [6] proposed a texture fingerprinting-based method to extract features of malware content. This method uses information from texture images extracted from the sample application code, which is mapped to uncompressed grayscale values according to the texture image-based method. Combined with API call features, a deep belief network is used to classify Android malware. Zhiqiang Wang et al. [7] proposed an Android malware detection framework based on Convolutional Neural Network (CNN). They used static analysis tools and python scripts to automatically extract 1003 static features, and transformed the features of each sample into a two-dimensional matrix as input to the CNN model. Ganesh et al. [8] used static analysis to extract 138 permission features of four categories, and

converted the permission features into 12 × 12 PNG images. They used convolutional neural networks for model training and detection, including 2000 malicious samples and 500 benign samples. The model achieved a detection accuracy of 93% in a total of 2,500 Android applications. Huang et al. [9] converted.dex-like bytecodes to RGB color codes and converted them to fixed-size color images. These color images are input into a convolutional neural network for automatic feature extraction and training. The experimental data set was collected from January 2017 to August 2017. About 2 million benign and malicious Android applications were collected. The experimental results show that the detection accuracy is 98.4225%.

This paper also analyzes the detection of Android malware based on permission characteristics. Wu et al. [10] used static analysis to detect Android malware and considered some static information, including permissions, component deployment, android intent, and application programming interface (API) calls, to describe the behavior of Android applications. After extracting the application features of malware, the K-means algorithm is used to enhance the modeling ability of malware. In the low-rank approximation, singular value decomposition (SVD) is used to determine the number of clusters. Finally, this system uses the KNN algorithm to classify the application as benign or malicious.

Wang et al. [11] explored the risks systematically caused by permissions in Android applications from three perspectives. They analyzed the risks of individual permissions and a set of collaborative permissions and used three feature ranking methods (mutual information, correlation coefficient, and T-test) to rank Android individual permissions. Risky subsets of permissions were identified by using sequential forward selection and principal component analysis. Second, they evaluate the effectiveness of risk permissions for malicious app detection using support vector machines, decision trees, and random forests to deeply analyze the detection results. Finally, they discussed the feasibility and limitations of malicious application detection based on permission requests. KA Talha et al. [12] proposed a permission-based Android malware detection system named APK Auditor, including a signature database, an Android client, and a central server, which uses static analysis to characterize and classify Android applications. Finally, the performance of the system is tested, yielding an accuracy rate of 88%. Qiao et al. [13] proposed a novel machine-learning approach for malware detection by analyzing Android app usage permissions and API function call patterns. They extract binary and numerical features from the source code and resource files of the apps through static analysis, which are then used for qualitative and quantitative evaluation. Finally, various machine learning methods, such as support vector machines, random forest, and neural networks, are employed for classification. Experimental results show that the proposed method can accurately detect Android malware, and the proposed method can help improve users' awareness of potential risks and mitigate malware threats on Android devices.

2.2 Graph Convolutional Neural Networks

Graph Convolutional Network (GCN) was first proposed by Kipf et al., in 2017 [14]. Based on convolutional operations, Kipf et al. have proposed A neural network model f (A, X) for semi-supervised learning of vertices, in which A is the adjacency matrix of the graph, and X is the input data. Adjusting f(·) on the adjacency matrix of the

graph will allow the model to allocate gradient information from the supervised loss L0, and enable it to learn the representation of all nodes. It is a neural network model for processing graph data. The graph convolutional network is a model evolved from the spectral convolutional neural network and the Chebyshev network [15]. The basic structure of the graph convolutional network is shown in Fig. 1.

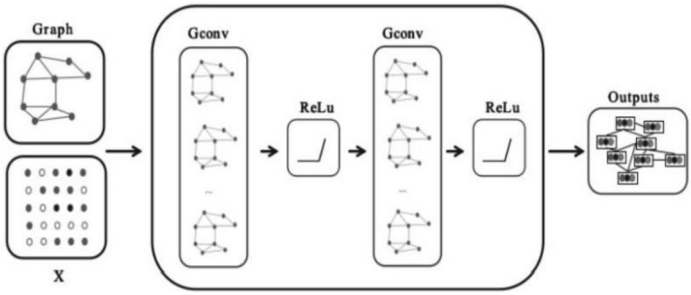

Fig. 1. The basic structure of a convolutional neural network.

The graph convolutional network is similar to the convolutional neural network. The convolution operator is defined on the graph structure, which can continuously expand the node information contained in the node embedding vector during the iterative process [16]. As shown in Fig. 2, a1–a4, the four nodes represent four APK files, and each APK file corresponds to its feature vector p1–p4. The process of graph embedding in graph convolutional neural networks is the process of continuous information propagation, aggregation, and re-propagation among nodes. Each node will get neighbor node information and update itself. Graph data often has a wide range of neighborhood structures near a single node, and the relationship between data is also complex. While CNN is limited to regular data, graph convolutional network has a wider range of applications. GCN can process more structured data, such as social networks and knowledge maps, and extract the spatial characteristics of structured data. GCN directly constructs the learning process on the graph data and provides an end-to-end framework for learning. It has obvious advantages in learning graph data and has better adaptability to learning tasks. GCN puts node representation learning and downstream task learning in the model for end-to-end learning. In order to achieve better adaptability between node feature representation and downstream tasks, GCN supervisory signals guide the parameter updating of task layer and GCN layer. In addition, the structure information and attribute information of the graph are complementary. For the graph with a sparse structure, the addition of attribute information can improve the quality of node representation learning. GCN puts these two kinds of information into the same network layer for simultaneous learning so that the two can synergistically affect the representation of the final node.

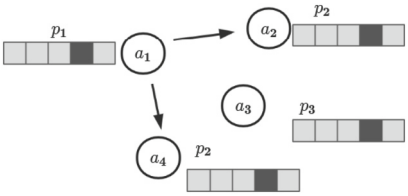

Fig. 2. Node Feature Propagation Process.

3 Detection Method

3.1 Overall Architecture

This paper proposes a GCN-based Android malware detection method. The architecture is shown in Fig. 3. Android software datasets are categorized based on their family and software functions, followed by the construction of a directed network topology graph. Decompile the APK file, unpack it and obtain the AndroidManifest.xml and class.dex files. Then extract the permission features by parsing the XML tree node <uses-permission> of the manifest file and traverse the feature documents of all applications in the feature extraction module. Organize all the permission features as the permission library, and finally compare the features of each APK file with the permission library to determine the node features of the topology map, to train and test the detection model.

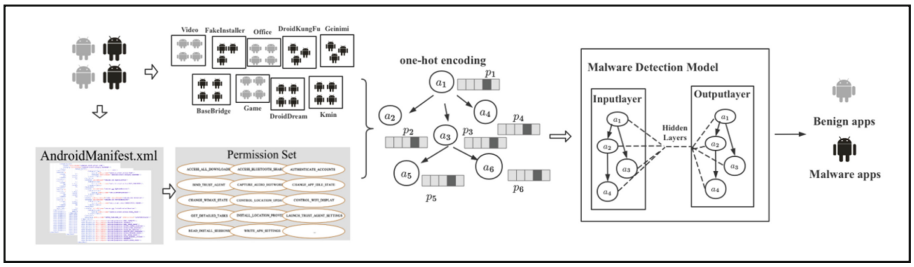

Fig. 3. Detection method architecture.

3.2 Feature Extraction and Processing

This article divides Android APK installation packages according to family and function classification. For example, hackers can remotely access infected mobile devices through DroidKungFu trojans while using the vulnerability root system to disguise themselves. In this paper, we utilize the original information extracted from APK files of each family member to establish a cascade relationship and subsequently construct a directed network topology. Then the topology structure is processed in the form of an adjacency matrix.

In this study, the permission information is selected as the feature attribute of the topology map node. The permission feature can reflect the behavior of Android software in some aspects. The <uses-permission> tag describes the hardware and software features that Android depends on. Table 1 lists several typical Android malware permission characteristics and attack behavior descriptions [5]. The original APK file in binary format cannot be used as the input of the graph convolutional network model for training directly. To obtain the corresponding feature data, they need to be preprocessed and feature extracted.

Table 1. Permission characteristics and attack behavior description of Android malware.

Malware family	Important Sensitive Permissions	Describe
ADRD	INTERNET, ACCESS_NETWORK_STATE, RECEIVE_BOOT_COMPLETED	worm virus
BaseBridge	NATIVE_CODE	root escalation using a Trojan horse
Bgserv	INTERNET, RECEIVE_SMS, SEND_SMS	worm virus
DroidDreamLight	INTERNET, READ_PHONE_STATE	Information Stealing Trojans
Genimi	INTERNET,SEND_SMS	worm virus
jSMSHider	INSTALL_PACKAGES	Destroy system firmware Trojans
Pjapps	INTERNET,RECEIVE_SMS	worm virus
Zsone	INTERNET, RECEIVE_SMS, SEND_SMS	Malicious sending SMS Trojan horse
zHash	CHANGE_WIFI_STATE	root escalation using a Trojan horse

This article uses APKTool and other decompilation tools to reverse-process Android APK files. After processing, a folder containing AndroidManifest.xml is generated. By parsing the XML tree node <uses-permission> of the manifest file, the permission characteristics can be extracted. The specific steps are designed as follows.

Step 1. Decompile the Android APK file. Use the APKTool to decompile APK files in batches to obtain the folder containing AndroidManifest.xml (Table 2);

Table 2. Partial pseudo-code for permission feature extraction.

Permission feature extraction
Input : APK t apklist R* AndroidManifest.xml c
Output : apkpermission.txt Q*
1.for t in R* do /*Iterate through all apk files*/
2. if t =.apk then
3. change ".apk" to ".zip"
4. (name of ZIP) =(name of APK)
5. for ZIP in R* do /*Iterate through all zip files*/
6. if ZIP is .apk then
7. xmlfile = read(c)
8. create new txtfile with Q*

Step 2. Extracts permission features to form an attribute set. By parsing the XML tree node <uses-permission> of AndroidManifest.xml decompiled in step 1, permission features can be extracted, and the obtained permission features can be store in permission.txt (Table 3);

Table 3. Partial pseudo-code for the feature permission library generation algorithm.

Feature permission library generation algorithm
Input : UpdatePermList.txt t DefaultPermList.txt s
Output: permissionset.txt
1.updatedata ← t
2.for t in list do
3. updateList ← updatedata.split('\n')
4. State defaultdata ← s
5. for s in list do
6. defaultList ← defaultdata.split('\n')
7. newList ← defaultList+list(set(updateList) - set(defaultList))

Step 3. Generate a permission library. Traverse the permission feature file permission.txt generated by all APK files in step 2, sort out all the permission features that have appeared as the permission library permission set;
Step 4. Generate feature vectors. Compare the permission.txt with the Permission Set generated in step 3, and use one-hot encoding to generate feature vectors. 1 means that the feature exists 0 means that the feature does not exist, and finally, generate node features (Table 4);
Step 5. Build connection relationships. For each APK file, edges are established between all positions of 1 in its permission feature vector and the corresponding positions of 1 in other permission feature vectors to construct the outgoing edge set for this application.;

Table 4. Partial pseudo-code for feature vector generation algorithm.

Feature vector generation algorithm

```
Input : apkpermission s permissionset t
Output: csvasterdict a csvrowdata b
/*Function to create a .csv file to store the data in the supplied license list*/
1.function CSVFormatter():
2.      testfile ← open(UpdatedPermList.txt)
3.      permlist ← data.split('\n')
4.      b.append(NAME)
5.      /*ADD NAME COLUMN*/
6.      b ← permlist + b
7.     b.append(CLASS)
8.     /*ADD PERMISSION COLUMN*/
9. end funtion
10./*Ability to create datasets from existing .csv files and provided APK file folders*/
11.function Bagger(datastoredir):
12.     if datastoredir is "./MalwareAPK then
13.        TYPE ← 1
14.     else
15.        TYPE ← 0
16.     end If
17.     if len(ApkNameList)≠0 then
18.        for ApkName in ApkNameList do
19.           permissions ← root.findall("uses-permission")
20.           a ← dict.fromkeys(fieldnames,0)
21.           a[NAME] ← ApkName
22.           a[CLASS] ← TYPE
23.           for perm in permissions do
24.              for att in perm.attrib do
25.                 permelement ← perm.attrib[att]
26.                 a[permelement] ← 1
27.        return result
```

Step 6. Build network topology maps. Summarize the edge set constructed in step 5 to form the entire network topology;

Step 7. Generate adjacency matrixes. The network topology structure is represented by an adjacency matrix in step 6, with each node's eigenvector stored as a node attribute and its outgoing edge set recorded in the matrix. In this way, the entire topology graph can be efficiently represented and computed through the adjacency matrix.

Figure 4 shows the process of converting Android software into a logic diagram.

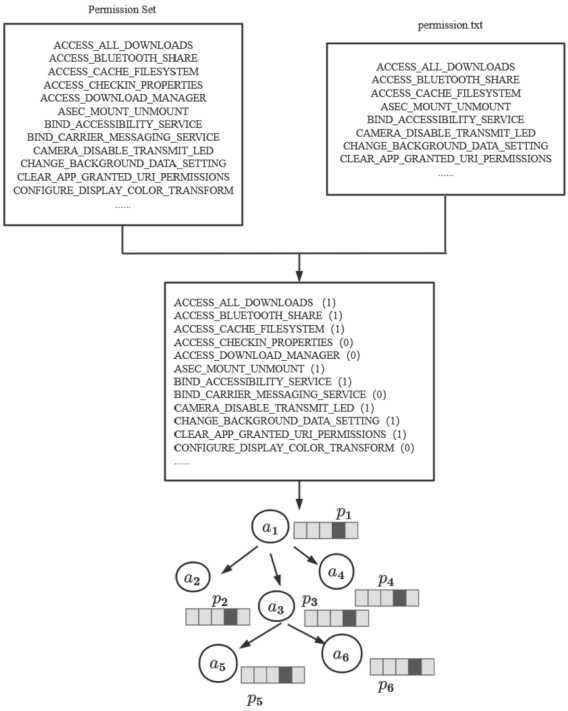

Fig. 4. Schematic diagram of converting Android software into a logic diagram

3.3 Detection Classification

This paper proposes a malware detection method based on a graph convolutional network. In the PyTorch-based Android malware detection system, Fig. 5 illustrates the graph convolutional network structure. The method uses convolutional graph layers, pooling layers, and fully connected layers to implement the detection function. The first layer of the method is the embedding layer, whose role is to convert the discrete input data into a continuous vector representation. Then, multiple parallel operations are performed on the input matrix, such as convolution operations, batch normalization, and max pooling. The parameter of the graph convolution layer is a convolution kernel, which performs convolution operations on the graph structure. During training, these convolution kernels are continuously adjusted to minimize the loss function of the meshes. The convolution kernels mainly obtain the characteristics of the current and adjacent nodes. They obtain local features through aggregation functions. Then shallow learning training is performed to obtain high-dimensional features for classification tasks. Next, the pooling layer concatenates the outputs generated by the convolutional layers into a fixed-length feature vector. Finally, the fully connected layer sends the concatenated vectors to the Softmax classifier for classification and uses regularization technology Dropout to prevent overfitting. The advantage of this method is that it can handle graph-structured data and has certain flexibility and scalability. Therefore, the method has better applicability and scalability in Android malware detection.

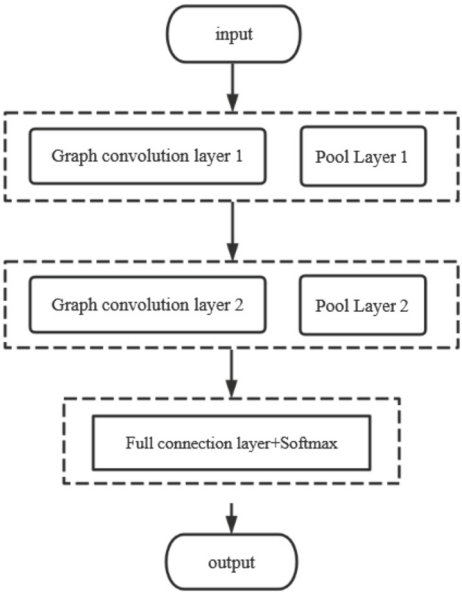

Fig. 5. Neural Network Architecture

The detection and classification part of this paper includes two modules of training and testing. The specific process of the training module is to input the topological graph and node feature vector generated by the feature extraction and processing module into the graph convolutional network. The trained and learned model is used as the final detection model. The test module inputs the test set into the detection model, obtains the final classification result, and calculates evaluation indices to assess the model's classification performance. For the defined model evaluation index, see Sect. 4.

4 Experimental Test and Result Analysis

4.1 Dataset and Experimental Environment

The experimental data set in this paper comes from Drebin [17] and CICDataset, which contains 4202 benign samples, 4679 malicious samples, and 16 malicious application families. Some malicious families and the number of samples are shown in Table 5. In this paper, it is randomly divided into training set, verification set, and test set according to the ratio of 6:2:2. The running and testing of the model are based on the Ubuntu 16.04 operating system, the CPU used is AMD-5800X, and the memory is 32G.

Table 5. Partial malware family datasets.

Serial number	Malware family	Number of samples	Training set	Validation set	Test set
1	FakeInstaller	1127	676	200	251
2	DroidKungFu	968	580	180	208
3	Plankton	882	529	170	183
4	BaseBridge	727	436	130	161

4.2 Determination of Model Parameters

1) Different learning rates

In neural network models, the learning rate is a difficult hyperparameter to set [18]. This is because its size can greatly affect the detection ability of the model. The learning rate controls the speed or step size of error allocation when the weight is updated each time. A higher learning rate can speed up model learning but may result in sub-optimal weights. A lower learning rate can improve the model's weight optimization, including global optima, but may increase training time. Therefore, in order to achieve better performance, the learning rate needs to be gradually reduced during training. By optimizing the learning rate, the model's performance can be improved, and better results can be achieved in experiments. In the comparison experiment, a learning rate value of 0.1 achieved the best detection result with 99.49% accuracy. The results of the comparative experiment are shown in Table 6, and the changing trend of the accuracy rate is shown in Fig. 6.

Table 6. Comparison of experimental results with different learning rates.

Iterations	Learning rate	Accuracy
Epoch = 200	0.1	99.49%
	0.05	98.55%
	0.01	97.50%
	0.005	75.58%
	0.001	68.50%
	0.0001	48.36%

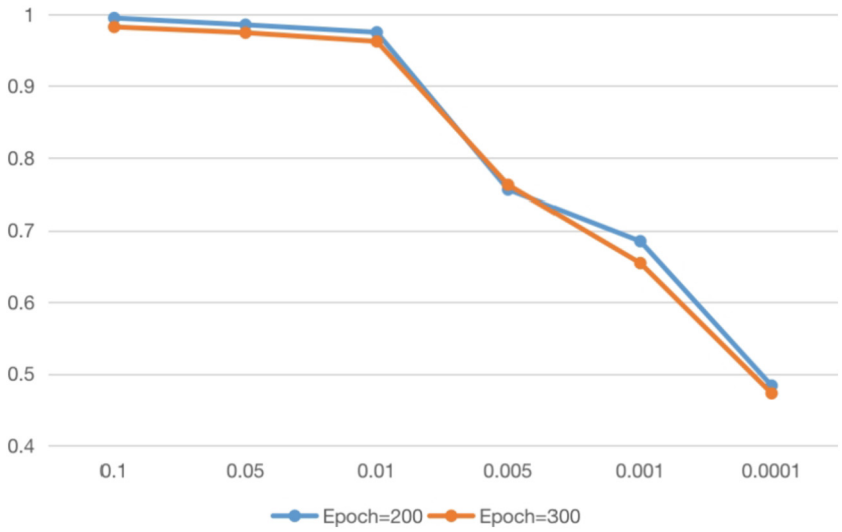

Fig. 6. Changes in accuracy at different learning rates

2) Different iterations

Epoch means to conduct a complete training of the neural network model by using all the sample data in the training set. The number of iterations is the value of Epoch, namely the number of complete training [19]. After multiple rounds of training, the model updates its weights and parameters on the same set to gradually approach the minimum loss function space, improving accuracy and reducing loss. At the same time, methods such as regularization techniques and pruning can also be used to avoid over-fitting problems. The comparison experiment shows that when the number of iterations is 200, the model achieves 99.49%, which is the best detection result of the comparison experiment. The results of the comparative experiment are shown in Table 7, and the changing trend of the accuracy rate is shown in Fig. 7.

Table 7. Comparison of experimental results with different iterations.

Learning rate	Iterations	Accuracy
lr = 0.1	100	92.2%
	150	96.82%
	200	99.49%
	250	98.57%
	300	98.25%
	350	87.24%
	400	75.47%

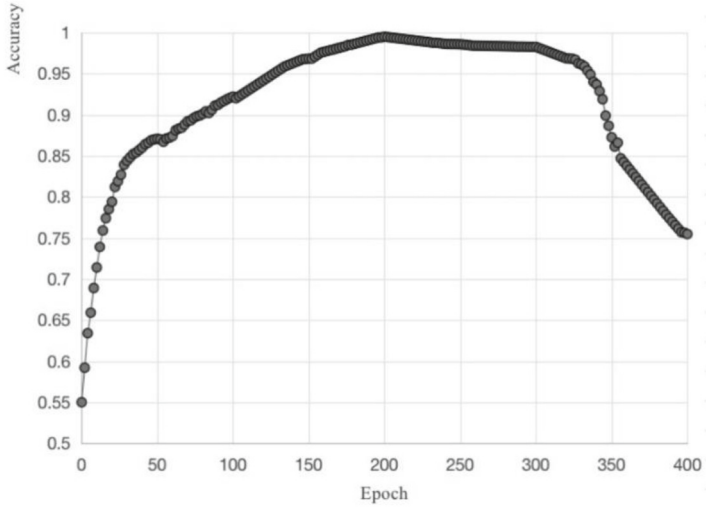

Fig. 7. Changes in accuracy rate for different iterations

4.3 Model Evaluation Metrics

Android malicious application detection is a binary classification problem. There are four possible prediction results. TN indicates that benign samples are predicted to be benign, FN indicates that malicious samples are predicted to be benign, FP indicates that benign samples are predicted to be malicious, and TP indicates that malicious samples are predicted to be malicious. We use the four parameters of accuracy, precision, recall, and F1-score to evaluate the detection model. The parameters and calculation methods are as follows.

The accuracy rate indicates the ratio of the number of correct judgments to the number of all judgments. The higher the accuracy rate, the better the classifier.

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

Accuracy rate is a measure of accuracy that represents the ratio of the number of correctly judged positive examples to the number of all judged positive examples.

$$Precision = \frac{TP}{TP + FP}$$

The recall rate is a measure of coverage, which measures how many positive cases are classified as positive cases.

$$Recall = \frac{TP}{TP + FN}$$

F1-score represents the harmonic average evaluation index of precision and recall.

$$F1 - score = \frac{2Precision \times Recall}{Precision + Recall}$$

4.4 Comparative Experiment

In order to verify the effectiveness of the GCN-based Android malware detection model, this paper selects four machine learning algorithms, namely, MLP, LSTM, Bi-LSTM, and DBN, to compare model detection performance. This paper defines a two-layer GCN. The permission database generated by the feature extraction module has 3669 permission features. Therefore, the dimension of the model input is 3669, the dimension of the hidden layer is set to 16, and the output dimension of the last layer of GCN is set to 2 categories. Using ReLU as the activation function, multiple sets of features are extracted and finally classified by the softmax classifier. We randomly use 60% of the dataset as a training set, 20% as a validation set, and 20% as a test set.

4.5 Experimental Results and Analysis

We conduct an evaluation experiment on the Android malware detection method. The core experimental process in this experiment is divided into two stages: training and detection. The test set in this paper consists of 20% malicious and 20% benign samples. The test set contains 641 malicious samples and 1056 benign samples.

Through experiments, the detection method in this paper detected nine malicious applications and 1047 benign applications among the benign test applications, 625 malicious applications, and 16 benign applications among the malicious test applications. The evaluation criteria in this paper include accuracy, precision, recall, and F1-score. Compared with the multilayer perceptron model, long-term, short-term neural memory network model, bidirectional long-term, short-term memory neural network model, and deep confidence neural network model, the experimental results of different deep learning models are shown in Table 8. The experimental results prove the designed effectiveness of Android malware detection methods (Fig. 8).

Table 8. Comparison of experimental results of different deep learning models.

deep learning model	Accuracy	Precision	Recall	F1-score
AGCN	**99.490%**	**99.148%**	**98.495%**	**98.820%**
MLP	98.370%	98.374%	98.371%	98.371%
LSTM	98.460%	98.465%	98.466%	98.466%
Bi-LSTM	97.320%	97.343%	97.328%	97.325%
DBN	93.380%	93.378%	93.377%	93.377%

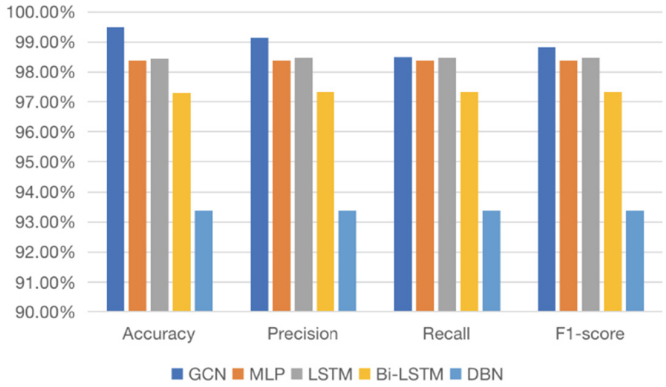

Fig. 8. Comparison of experimental results of different deep learning models

5 Conclusion

This paper conducts thorough research and analysis on Android malware detection technology. Based on the function and family classification of Android software, the construction method of the topology graph is designed. The feature extraction method is designed based on the permission characteristics of Android software. The processed Android softwares are trained and tested, and the deep learning method used is compared with the multilayer perceptron, long-term, short-term memory neural network, bidirectional long-term, short-term memory neural network, and deep confidence neural network. By comparing multiple evaluation indicators, It is verified that the proposed model has a certain degree of advancement compared with the traditional model.

The Android malware detection method based on the graph convolutional network proposed in this paper still has some areas to be improved: 1) It only extracts and analyzes static features, and the detection effect of dynamic features needs to be verified and analyzed experimentally. 2) Currently, The deep learning model for Android malware detection is vulnerable to adversarial attacks. In the future, the adversarial samples automatically generated by the generative adversarial network can be used to retrain deep learning classifiers for adversarial training. The abstraction of the deep learning model can be modified through adversarial training.

References:

1. Data source. https://pop.shouji.360.cn/safe_report/Mobile-Security-Report-202106.pdf
2. Naway, A., Li, Y.: A review on the use of deep learning in Android malware detection (2018)
3. Wang, Z., Liu, Q., Chi, Y.: Review of Android malware detection based on deep learning. IEEE Access **8**, 181102–181126 (2020)
4. Guen, K.T., Kang, B.J., Mina, R., et al.: A multi-modal deep learning method for Android malware detection using various features. IEEE Trans. Inf. Forensics Secur.Secur. **14**(3), 773–788 (2018)
5. Zegzhda, P., Zegzhda, D., Pavlenko, E., et al.: Applying deep learning techniques for Android malware detection, pp. 1–8 (2018)

6. Shiqi, L., Shengwei, T., Long, Y., et al.: Android malicious code classification using deep belief network. KSII Trans. Internet Inf. Syst. **12**(1), 454–475 (2018)
7. Wang, Z., Li, G., Chi, Y.: Multi-classification of Android applications based on convolutional neural networks. In: 2020 International Conference on Computer Science and Application Engineering, Sanya, Hainan, P.R.China (2020)
8. Ganesh, M., Pednekar, P., Prabhuswamy, P., Nair, D.S., Park, Y., Jeon, H.: CNN-based Android malware detection. In: Proceedings of the International Conference on Software Security and Assurance (ICSSA), pp. 60–65 (2017)
9. Huang, T.H.-D., Kao, H.-Y.: R2-D2.: color-inspired convolutional neural network (CNN)-based Android malware detections. In: Proceedings of the IEEE International Conference on Big Data (Big Data), Dec. 2018, pp. 2633–2642
10. Wu, D.-J., Mao, C.-H., Wei, T.-E., Lee, H.-M., Wu, K.-P.: DroidMat: Android malware detection through manifest and API calls tracing. In: 2012 Seventh Asia Joint Conference on Information Security, Tokyo, Japan, pp. 62–69 (2012). https://doi.org/10.1109/AsiaJCIS.2012.18
11. Wang, W., Wang, X., Feng, D., Liu, J., Han, Z., Zhang, X.: Exploring permission-induced risk in Android applications for malicious application detection. IEEE Trans. Inf. Forensics Secur.Secur. **9**(11), 1869–1882 (2014). https://doi.org/10.1109/TIFS.2014.2353996
12. Talha, K.A., Alper, D.I., Aydin, C.: APK Auditor: permission-based Android malware detection system. Digit. Invest. **13**, 1–14 (2015)
13. Qiao, M., Sung, A.H., Liu, Q.: Merging permission and API features for Android malware detection. In: 2016 5th IIAI International Congress on Advanced Applied Informatics (IIAI-AAI), Kumamoto, Japan, pp. 566–571 (2016). https://doi.org/10.1109/IIAI-AAI.2016.237
14. Tan, Y.: A survey of text classification methods based on graph convolutional neural network
15. Gao, H., Cheng, S., Zhang, W.: GDroid: Android malware detection and classification with graph convolutional network. Comput. Secur.. Secur. **106**, 102264 (2021)
16. Feng, P., Ma, J., Li, T., et al.: Android malware detection via graph representation learning. Mob. Inf. Syst. **2021**, 5538841 (2021)
17. Arp, D., Spreitzenbarth, M., Hubner, M., et al.: Drebin: effective and explainable detection of Android malware in your pocket. In: NDSS 2014, vol. 14, pp. 23–26 (2014)
18. Pan, Y., Ge, X., Fang, C., et al.: A systematic literature review of android malware detection using static analysis. IEEE Access **8**, 116363–116379 (2020)
19. Chiang, W.L., Liu, X., Si, S., et al.: Cluster-GCN: an efficient algorithm for training deep and large graph convolutional networks. In: Proceedings of the 25th ACM SIGKDD International Conference On Knowledge Discovery & Data Mining, pp. 257–266 (2019)