Hindawi
Security and Communication Networks
Volume 2022, Article ID 1289175, 16 pages
https://doi.org/10.1155/2022/1289175

Research Article

A Deep Learning Method for Android Application Classification Using Semantic Features

Zhiqiang Wang,[1,2,3] **Gefei Li**,[2] **Zihan Zhuo**,[4] **Xiaorui Ren**,[1] **Yuheng Lin**,[1] **and Jieming Gu**[4]

[1]*Department of Cyberspace Security, Beijing Electronic Science & Technology Institute, Beijing 100070, China*
[2]*State Information Center, Beijing 100045, China*
[3]*Guangdong Provincial Key Laboratory of Information Security Technology, Shenzhen, Guangdong 510006, China*
[4]*National Internet Emergency Center, Beijing, 100029, China*

Correspondence should be addressed to Zihan Zhuo; zzh@cert.org.cn

Received 8 December 2021; Accepted 1 February 2022; Published 24 February 2022

Academic Editor: Robertas Damaševičius

Android has become the most popular mobile intelligent operating system with its open platform, diverse applications, and excellent user experience. However, at the same time, more and more attackers take Android as the primary target. The application store, which is the main download source for users, still does not have a complete security authentication mechanism. Given the above problems, we designed an Android application classification model based on multiple semantic features. Firstly, we use analysis tools to automatically extract the application's dynamic and static features into the text document and use variance and chi-square tests to optimize the features. Combined with natural language processing (NLP), we transform the feature file into a two-dimensional matrix and use the convolution neural network (CNN) to learn features efficiently. Also, to make the model satisfy more application scenarios, we design a dynamic adjustment method according to user requirements, the number of features, and other indicators. The experimental results demonstrate that the detection accuracy of malware is 99.3921%. We also measure this model's performance in detecting a malware family and benign application, with the classification accuracy of 99.5614% and 99.9046%, respectively.

1. Introduction

Android has many devices and users and rich applications as the most popular mobile intelligent operating system, bringing great convenience to people's lives. The open-source Android platform has made more and more mobile terminal manufacturers and developers join the Android alliance. According to the International Data Corporation's global smartphone market data report [1], with the popularization of 5G and the accelerated research and development of 5G smart terminals by notable brands, global smartphone shipments are expected to increase slightly by 1.6% next year. However, at the same time, the security problem of the Android system is also increasingly prominent, which contains more and more sensitive information such as user identity information, location information, and privacy data. At present, the security authentication mechanism for Android application stores is still not complete, and more and more attackers take the Android system as the primary attack target.

The Android platform provides some security mechanisms to limit malware functions, especially Android's permission control mechanism. Android system defines various permissions for developers to protect system resources and provides the corresponding APIs for accessing the above system resources. If an application wants to use these APIs to access user data, system configuration, and other resources, it must apply for the corresponding permissions and obtain the user's consent. However, most users usually blindly grant all permissions, thus destroying the

permission mechanism's effectiveness and failing to limit malware functions. As the primary source for users to download applications, various third-party application stores need to keep malware out and quickly and accurately classify benign applications automatically. Currently, application stores generally classify applications according to categories specified by developers or by analyzing descriptions provided by developers. However, malware developers can easily manipulate this process to evade detection, such as adding unqualified financial applications to the information application interface that is more easily approved. Additionally, as the number of applications explodes, it is becoming critical to classify applications quickly and accurately to improve management efficiency.

At present, most malicious application detection schemes use static or dynamic analysis methods to extract features and then combine machine learning algorithms to identify malicious applications. Some schemes directly visualize the application source code as a gray image or RGB image and use deep learning technology to analyze image features. Although the feature extraction step is omitted, it still belongs to static analysis. MalNet [2] uses CNN and LSTM networks to learn from the grayscale image and opcode sequence and takes a stacking ensemble for malware classification. Ganesh et al. [3] extracted 138 permission features and converted them into 12×12 PNG images and then used CNN to detect malicious applications. Xu et al. [4] used the control flow graph, data flow graph, and their possible combination as the Android application features, then encoded the graph into a matrix, and used them to train the classification model through CNN. Zegzhda et al. [5] proposed an approach for representing an Android application for a CNN, which consists of constructing an RGB image, the pixels of which are formed from a sequence of pairs of API calls and protection levels.

In the above detection method, CNN has two key benefits: local invariance and compositionality. Local invariance allows us to classify an image as a process containing a specific target, no matter where the target appears in the image. Compositionality means that each filter combines features to form a high-level representation, enabling the network to learn more precious features at a deeper level. However, the above methods are static analysis, which cannot wholly characterize the behavior of malicious applications, and need to extract dynamic features. Yuan et al. [6] proposed an online Android malware detection engine that extracted 192 features using static and dynamic analysis techniques and combined with Deep Belief Networks to detect malware, achieving high accuracy. However, there are still some problems, such as low accuracy and long training time for high dimensional feature detection. In NLP, Kim [7] proposed applying CNNs to sentence-level classification problems and achieved excellent results with an uncomplicated model.

This paper combines NLP and CNN to extract static and dynamic features and adds their frequency to describe features more accurately. Then, all the features contained in each application are transformed into a two-dimensional matrix. Finally, we use CNN to learn features efficiently to classify applications quickly. The main contributions of this paper are as follows:

(1) Automated feature extraction: We use four dynamic and static analysis tools to extract features and express the features in the form of "feature name + frequency" to describe the features more accurately and comprehensively. Each type of feature set is independent, and users can flexibly choose analysis tools according to their needs. This paper conveniently adds new feature set types to ensure the long-term validity of the model.

(2) Feature vector generation: We transform the feature file into a two-dimensional matrix using NLP and combine it with CNN, which is excellent in image recognition, to learn features efficiently.

(3) Dynamic adjustment of the model: We design a dynamic adjustment method of parameters according to the user's requirements (binary classification/multiclassification), the number of applications, and the average number of features contained in the feature files, to ensure that the model can always maintain the best detection effect as the applications and features change.

(4) Aiming at the problem of multiclassification of malware families and benign applications, we design a multiclassification method for Android applications based on CNN, which has higher detection accuracy than other methods.

2. Related Work

There are numerous Android malware detection schemes, mainly divided into static analysis methods and dynamic analysis methods. The static analysis method analyzes source code files or executable files without running applications.

EveDroid [8] is an event-aware Android malware detection system that exploits the behavioral patterns in different events to detect new malware based on the insight that events can reflect applications' possible running activities. Kumar et al. [9] proposed an Android malware detection framework based on machine learning and blockchain. Machine learning automatically extracts the malware information using clustering and classification techniques and storing it into the blockchain. Hasegawa and Iyatomi [10] proposed a light-weight Android malware detection method. It treats a minimal part of the target's raw APK file as a short string and analyzes it with one-dimensional CNN. Zhang et al. [11] proposed an Android malware detection method based on the method-level correlation relationship of the application's abstracted API calls. It calculates the confidence of association rules between the abstract API calls to form the behavioral semantics that describes applications and then build the detection system in combination with machine learning. Fang et al. [12] proposed an Android malware familial classification method based on DEX file section features. It first converts the DEX file into RGB image and plain text, respectively, and then extracts the image's and text's

color and texture as features. Finally, a feature fusion algorithm based on multiple kernel learning is used for classification. Apposcopy [13] is a new semantics-based approach for detecting Android malware. It incorporates a high-level language for specifying malware signatures and a static analysis for deciding if a given application matches a given signature. TaeGuen et al. [14] proposed using the method based on presence and similarity to extract features and using a multimodal deep learning method to detect malware. MADAM [15] is a host-based malware detection system for Android devices. It simultaneously analyzes and correlates features at four levels, kernel, application, user, and package, to detect and stop malicious behaviors. Narayanan et al. [16] proposed a method that uses control flow graphs as features and uses online support vector machine algorithms to detect malicious applications. Azad et al. [17] used particle swarm optimization to perform feature selection, a set of features to characterize the behavior of android applications and classify them as legitimate and malicious. Nisa et al. [18] proposed a feature fusion method that combines features extracted from pretrained AlexNet and Inception-v3 deep neural networks with features obtained from images representing malware code using segmentation-based fractal texture analysis (SFTA) and built a multimodal representation of malicious code for classifying grayscale images. Hemalatha et al. [19] describe malware binaries as 2D images and classify them with a deep learning model. Feng et al. [20] analyze and extract two types of features (i.e., manifest attributes and API calls) directly from the Dalvik binary and further update the feature input with matching results between text-based behavioral descriptions and code-level features.

The above static analysis methods have the advantages of fast detection speed and high efficiency and can detect malware in large quantities. The disadvantage is that they cannot fight against code transformation technology and dynamic malicious payload technology. The dynamic analysis method can overcome the above weakness [21]. It can capture sensitive behaviors in real time dynamically. Feng et al. [22] proposed EnDroid, which uses DroidBox to extract behavioral features through a runtime monitor and uses chi-square feature selection algorithms and ensemble learning to detect malware. Enck et al. [23] designed a Taint Droid detection tool, which marks a variety of sensitive data with taints. It determines whether the application has a privacy data leakage behavior by monitoring the flow path of these contaminated sensitive data in real time in a sandbox environment. Tam et al. [24] proposed a dynamic system based on a virtual machine called CopperDroid, directly detecting system calls to determine the operating system's actions, generating detailed and semantic behavior information to identify malicious applications. However, it can only identify the interaction between the system and the application, not the interaction between the applications. The above methods are not affected by the code transformation technology and can analyze the application's behavior in-depth, but the time is expensive. To analyze Android applications more comprehensively, we use dynamic and static analysis methods to extract features.

3. Architecture

The Android application classification model's overall architecture is shown in Figure 1. It is mainly divided into five modules: feature extraction module, feature preprocessing module, feature vector generation module, deep learning module, and detection module. First, we rename the collected applications with the file hash, remove the duplicate applications, and then store the applications' file hashes with the label "benign" or "malicious" in the database.

In the feature extraction module, we use static and dynamic analysis to batch extract features of applications into text documents. Each line represents a feature, and each application corresponds to a feature file.

In the feature preprocessing module, we use feature selection algorithms to optimize features further. Next, in the feature vector generation module, we convert each feature file into a two-dimensional matrix.

In the deep learning module, the model can flexibly adjust the parameters of the CNN according to the user's need and detection conditions (high precision/high efficiency), detection types (binary classification/multi-classification), the average number of features, and other indicators and select the most appropriate model as the final detection model through training.

Finally, the user submits the application to be tested through the client. The malware detection module firstly checks whether the application already exists in the database through file hash and, if so, directly returns the detection result. If it does not exist, the optimal detection model obtained by the deep learning module is used. Details of each module are shown in Figure 1.

3.1. Feature Extraction Module. In the feature extraction module, for each application to be tested, we use APKTool [25], androguard [26] Drozer, and DroidBox [21] to analyze the applications, obtain the corresponding files, and then extract features from them. The feature consists of two parts separated by spaces, the feature's name, and the frequency of the feature occurrence. It will be omitted if the frequency is 1. If the frequency is 0, this feature is not selected. For each tool, we separately write Python scripts to extract features into text documents automatically and then merge them into the final feature files for each application. The following introduces the four tools and the extracted content.

3.1.1. APKTool. We use APKTool to decompile the application to get AndroidManifest.xml. by parsing XML tree nodes <uses-permission>, <intent-filter>, <uses-feature> to extract permission features, component features, and environmental features.

(1) Permission Features. When an application performs specific operations or accesses certain data, it must apply for corresponding permissions, which means that the permissions defined in the manifest file can indicate the application's behavior. In the feature extraction process, we only extract the permission name. We collect the system

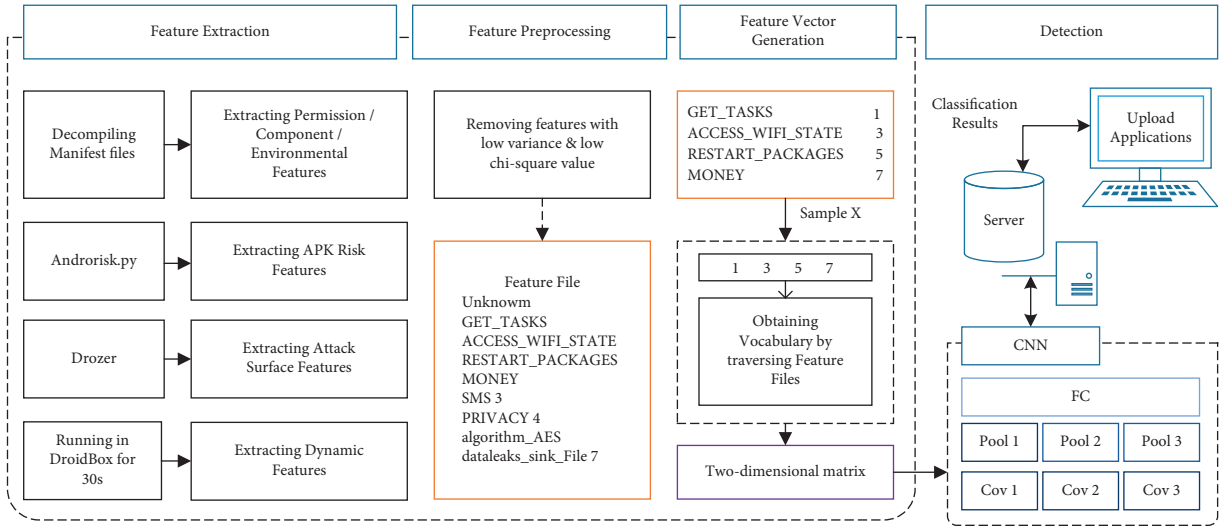


Figure 1: The overall architecture of the model.

permissions of the application by parsing the <uses-permission> tags.

(2) Component Features. Application components are the basic building blocks of Android applications, including Activity, Service, Broadcast Receiver, and Content Provider. Components are called Intents, which can register and receive messages. We can use them to start components or pass some important data to components. We collect component features by parsing the <action> and <category> tags in the <intent-filter> tags.

(3) Environmental Features. It includes hardware or software functions that applications depend on, such as GPS and NFC. Devices lacking specific hardware or software functions will not execute applications that require such special functions. For example, Android devices that do not support wireless charging cannot charge wirelessly. We collect environmental features by parsing <uses-feature> tags.

3.1.2. Androguard. We use the androrisk.py file in androguard to analyze the applications' risk level, and the analysis results and extracted contents are shown in Figure 2.

The analysis results are mainly composed of three parts: DEX, APK, and PERM. The analysis contents of DEX and APK are given in Table 1. PERM is the number of different functional permissions.

3.1.3. Drozer. Drozer is an Android security testing framework. We use "app. package. attacksurface" command to test the attackable points of the applications and extract the attack surface features. The results are shown in Figure 3.

3.1.4. DroidBox. TaintDroid is a dynamic stain detection technology, whose core idea is to mark sensitive data and turn them into pollution sources. In the process of program operation, when these pollution sources spread through interprocess communication, file transfer, etc., TaintDroid will conduct tracking reviews and record in the log to realize the tracking of sensitive data. DroidBox builds on this by performing dynamic stain analysis at the application framework level and redefines the types of stain tags, adding functions such as file manipulation monitoring, network sending and receiving data monitoring, encryption and decryption logging, and log analysis. DroidBox provides two scripts, startemu.sh for launching an emulator dedicated to dynamic analysis of Android apps and droidbox.sh for performing specific dynamic analysis tasks. This paper extracts dynamic behavior features from the operation logs of each application by running the DroidBox for 30 seconds. The specific features are as follows:

(1) Cryptographic Operation. Malicious applications usually use encryption to encrypt root vulnerabilities, malicious payloads, key method identifiers, value-added service SMS, and URLs to remote malicious servers to avoid static detection. So, we count the frequency of encryption, decryption, and key generation and record all encryption algorithms used by the applications.

(2) Network Operation. Malicious applications may receive messages from malicious command and control (C & C) servers through the network and obtain malicious payloads from malicious websites, so that attackers can manipulate the applications to obtain users' private information. We count the frequency of sending and receiving network communication data.

(3) Information Leaks. Information leaks are mainly through the network and files, so we count the number of times dataleaks_operation_write, dataleaks_sink_File, dataleaks_operation_read, dataleaks_-sink_Network occurred. Simultaneously, the leakage of LOCATION, IMSI, ICCID, IMEI, PHONE_NUMBER, LOCATION_GPS is also calculated.

The Analysis Results of Androrisk.py

/mnt/hgfs/share/apk/1.apk
RedFlags
DEX { 'NATIVE' :0, 'DYNAMIC' :1, 'CRYPTO' :1, 'REFLECTION' :1}
APK { 'DEX' :0, EXECTABLE' :0, 'ZIP' :0, 'SHELL_SCRIPT' :0, 'APK' :0, 'SHARED LIBRARIES' :3}
PERM { 'PRIVACY' :0, 'NORMAL' :1, 'MONEY' :0, 'INTERNET' :1, 'SMS' :0, 'DANGEROUS' :1, 'SIGNATUREORSYSTEM' :0, 'CALL' :0, 'SIGNATURE' :0, 'GPS' :0}
FuzzyRisk
VALUE 92.0

Feature File

DYNAMIC
CRYPTO
REFLECTION
SHARED LIBRARIES 3
NORMAL
INTERNET
DANGEROUS

Figure 2: The analysis result and extracted contents of androrisk.py.

Table 1: The analysis content of androrisk.py.

DEX	Description
NATIVE	Number of calls to non-java code
DYNAMIC	Times of dynamic loading of dex from sd
CRYPTO	Number of hidden dexes
REFLECTION	Number of reflections
APK	Description
DEX	Times of dex use
EXECUTABLE	Number of executions
ZIP	Compressed package
SHELL_SCRIPT	Number of times the script is used
APK	Number of other apks
SHARED LIBRARIES	Number of shared databases

```
dz> run app. package. attacksurface com. glu. android. dinercn
Attack Surface:
  1 activities exported
  1 broadcast receivers exported
  0 content providers exported
  0 services exported
dz>
```

Figure 3: The results of attack surface test.

(4) Sent SMS. Malicious applications usually cause financial charges to infected users. They can secretly subscribe to value-added services by sending several SMS messages without the user's consent. So, we count the frequency of sending text messages.

(5) Service Start. Malicious applications usually perform malicious behavior in background processing contained in in-service components. So, we count the number of times the service has started.

(6) Receiver Action. Malicious applications usually leverage system events to trigger malicious behaviors. Registered Broadcast Receivers can be a fair reflection of the monitored system events. For example, registering the reception of BOOT_COMPLETED intent in malware indicates triggering malicious activity directly after the mobile device's startup.

3.2. Feature Preprocessing Module. To further reduce overfitting and improve the model's training speed and generalization ability, we design a feature preprocessing module. The first step is to remove low-variance features. Through experimental tests, setting the removal rate to 99.95% and above can achieve better detection results. The second step is the chi-square test, which can express the correlation between feature items and categories. The higher the CHI value, the more significant the correlation. So, we remove the features with lower chi-square values in the experiment.

3.3. Feature Vector Generation Module. According to reference [27], after the feature preprocessing module, we traverse all the feature files and obtain all the features that have appeared as the vocabulary. Each feature in the vocabulary is labeled with consecutive numbers to obtain a mapping from feature to label ID. Also, we add an "Unknown" feature to match unknown features that are not in the vocabulary during the detection phase.

Since CNN's input is a vector in continuous space, while NLP uses discrete characters, we need to use word embedding technology to convert each feature file into a two-dimensional matrix when classifying Android applications using CNN. First, we represent each feature in the vocabulary with a vector and randomly initialize the vectors. Then, we update the word vector continuously with training. The length of the word vector depends on the specific situation of the feature set. 50–300 is a common choice, and we set it to 200. We convert the features contained in each feature file into corresponding ID sequences according to the vocabulary, respectively. Then, the feature file is transformed into a two-dimensional matrix according to the ID sequences and the vocabulary. The specific process is shown in Figure 4.

3.4. Deep Learning Module. Deep learning algorithms include CNN, RNN, and LSTM. LSTM and RNN are suitable for learning long time series, and CNN has a better learning ability for local features. In this paper, the features contained in each application are composed of four parts, which have no time sequence and short average length. According to reference [6], the parts associated with permission features can more effectively characterize applications' malice. That is, capturing the relationship between local features can train the model more effectively. So, we finally adopted CNN.

The structure of CNN in the deep learning module is shown in Figure 5.

The first layer is the embedding layer, which is mainly responsible for embedding features into low-dimensional vectors. Then, we perform multiple parallel convolution

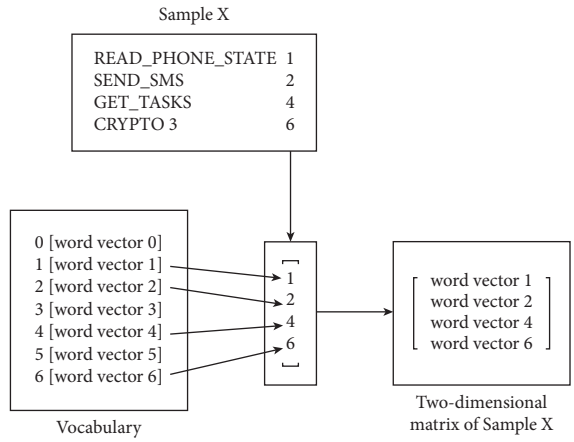


Figure 4: Feature vector generation.

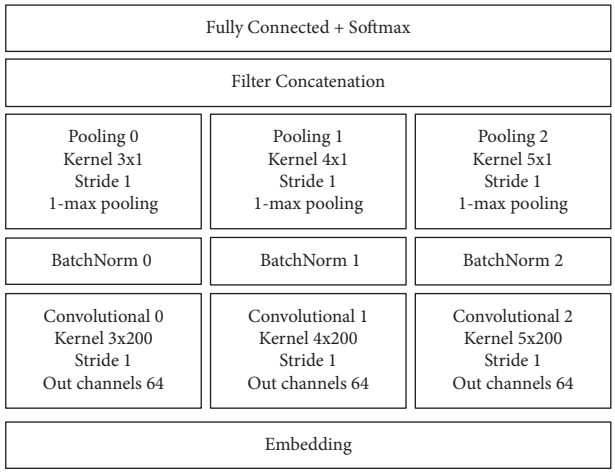


Figure 5: The structure of CNN.

operations, batch normalization, and 1-max pooling on the input matrix and concatenate all the outputs into a fixed-length feature vector. Finally, we classify the results using the full connection layer. The specific description of each layer is as follows:

3.4.1. Convolutional Layer. The convolutional layer is the core of the network. We use multiple filters of different sizes to learn the same area's complementary features, and we can obtain different feature maps. There are three parallel convolutional layers in Figure 5, where the width of the filter is the same as the width of the two-dimensional matrix (i.e., the length of the word vector), and we set it to 200. In this way, after a convolution operation, the two-dimensional matrix becomes a column vector. (3,4,5) is the height of the filter, that is, the relationship between 3, 4, and 5 features. There are 64 filters of each type.

3.4.2. Pooling Layer. The dimension of the feature map generated by each filter varies depending on the number of features and the size of the filter region. Therefore, 1-max pooling is applied to each feature map to induce a fixed-length vector.

3.4.3. Fully Connected Layer. We send the concatenate vectors to the softmax classifier through the full connection layer for classification and use the regularization technology Dropout to prevent overfitting.

3.5. Detection Module. The user submits the application to be tested through the client. The malware detection module firstly checks whether the application already exists in the database through file hash and, if so, directly returns the detection result. If it does not exist, the optimal detection model obtained by the deep learning module is used. The detection result is an excel file, including the file hash and classification results of the application (0 for malicious application and 1 for benign application).

4. Experiments and Evaluation

4.1. Dataset. The dataset includes 18549 malware and 18453 benign applications, of which the malware comes from VirusShare [28] and Drebin [29], and the benign applications come from Google Play Store [30]. They are scanned and detected by VirusTotal [31]. When all virus scanners in VirusTotal treat the application as benign, the application will be included in the benign application dataset.

In practice, we analyzed all applications, but tools do not correctly analyze some applications. So, the features are contained in 100% of the manifest files, 99.6757% of the APK risk files, 99.5081% of the attack surface files, and 72.6420% of the dynamic behavior files. Detailed statistical results are shown in Table 2.

4.2. Experimental Settings. The experimental environment is as follows.

(i) Hardware Dependencies: on the hardware, NVIDIA GPU GeForce RTX 2070, and 8 GB memory are used.

(ii) Software Dependencies: in terms of software, Ubuntu 16.04 LTS, Python 3.6, TensorFlow 1.13.1, Scikitlearn 0.20.3, Numpy 1.16.2, Pandas 0.24.2, and Matplotlib 3.0.3 are used.

(iii) GPU Components: GPU components include NVIDIA GPU driver, CUDA 10.1 and cuDNN v7.5.1.

GPU is used to accelerate the CNN. Tensorflow is used to implement CNN, and Scikitlearn is used to implement various machine learning algorithms.

Android malware detection is a binary classification problem. There are four possible prediction results. The confusion matrix is shown in Table 3. Among them, True Negative (TN) indicates that benign samples are predicted as benign, False Negative (FN) indicates that malicious samples are predicted as benign, False Position (FP) indicates that benign samples are predicted as malicious, and True Position (TP)

Table 2: Detailed statistics of dataset.

	Manifest file	APK risk file	Attack surface file	Dynamic behavior file
Malicious	18549 (100%)	18506 (99.7682%)	18498 (99.7251%)	16769 (90.4038%)
Benign	18453 (100%)	18316 (99.2576%)	18322 (99.2901%)	10110 (54.7878%)
Total	37002 (100%)	36882 (99.6757%)	36820 (99.5081%)	26879 (72.6420%)

Table 3: Confusion matrix.

Type of prediction	Benign	Malicious
Benign	TN	FP
Malicious	FN	TP

indicates that malicious samples are predicted as malicious. The evaluation indicators are as follows: Accuracy (ACC), Precision (PRE), Recall (REC), and F1 score (F1).

Accuracy is defined as the percentage of the total sample that is predicted correctly.

$$\mathrm{ACC} = \frac{(\mathrm{TP} + \mathrm{TN})}{(\mathrm{TP} + \mathrm{EN} + \mathrm{FP} + \mathrm{TN})}. \quad (1)$$

Precision means the probability of a positive sample among all the samples that are predicted to be positive.

$$\mathrm{PRE} = \frac{\mathrm{TP}}{(\mathrm{FP} + \mathrm{TP})}. \quad (2)$$

Recall is the probability of being predicted as a positive sample in a sample that is positive.

$$\mathrm{REC} = \frac{\mathrm{TP}}{(\mathrm{EN} + \mathrm{TP})}. \quad (3)$$

F1 score is the harmonized average of precision and recall.

$$F1 = 2\mathrm{PRE} \cdot \mathrm{REC}\,(\mathrm{PRE} + \mathrm{REC}). \quad (4)$$

4.3. Feature Selection and Analysis. After the feature extraction module, the number of features included in different feature sets is shown in Table 4 (the statistical results in this table only include the feature names and do not include the following parameters, such that "NATIVE 2" and "NATIVE 3" are counted only once in this table).

To have a more detailed understanding of the features and lay the foundation for subsequent experiments, after the feature preprocessing module, we count the types of features contained in each feature set (Type), and the maximum (Max), minimum (Min), and average (Ave) number of features included in all feature files of each feature set. The statistical information is shown in Table 5.

Taking feature set I as an example, we compare the statistical information before and after feature preprocessing and find that the number of feature types and the maximum number of features are significantly reduced. Personality features are greatly reduced, but the reduction of the average number of features is minimal.

4.4. Contrast Experiment with Machine Learning Algorithms. To analyze the effect of the Android malware detection model based on CNN, we select seven machine learning algorithms, namely, k-NearestNeighbor (kNN), Decision Tree (DT), Support Vector Machine (SVM), Logistics Regression (LR), XGBoost, Random Forest (RF), and Multi-Layer Perceptron (MLP) to test the detection effect on three feature sets.

The hyperparameters of seven machine learning algorithms are set by default. The hyperparameters of CNN mainly refer to the experimental results and parameter adjustment suggestions in reference [32]. Through a series of experimental verification, we have obtained the baseline configuration parameters.

The filter region size is (3,4,5), the feature map is 64, and the activation function is RELU. The embedding size and batch size are 200 and 128, respectively. The epoch is set to 10, 1-max pooling is used, and the dropout rate and train set radio are 0.5 and 50%, respectively.

We randomly selected 50% in all applications as the training set, of which 10% as the verification set and the remaining 50% as the test set. The experimental results are shown in Figure 6. The detection effect of CNN is better than that of machine learning algorithms. With the increase of features, the detection effect is getting better and better, proving the effectiveness of the feature set.

4.5. Contrast Experiment between Different Hyperparameters of CNN. This section further analyzes the influence of different hyperparameters on the experimental results. For this reason, keep all other settings unchanged and only change the parameters to be analyzed.

4.5.1. Effect of Filter Region Size. We first perform a coarse linesearch over a single filter region size to find the "best" size for the feature set under consideration. The experimental results are shown in Figure 7. When the filter region size is 1, the performance of the model is weak. When the filter region sizes are 3 and 5, the classification accuracy is the highest. According to the statistical information, the average number of features in each feature file ranges from 13 to 24. For feature files containing more features, the optimal filter region size can be more extensive. When the training set accounts for 50%, the size of the vocabulary corresponding to feature set I is 1233, feature set I + II is 1438, and feature set I + II + III is 1817.

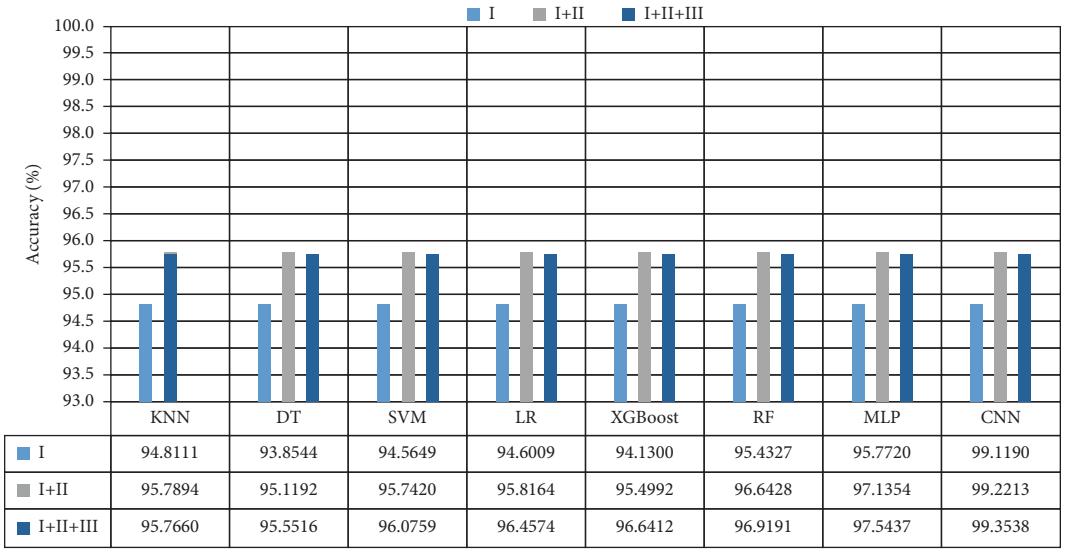
Besides, we combine different region sizes and copies to obtain the best effect. The experimental results can be seen in Table 6. Using different feature sets, the combination of several region sizes near the optimal size can improve the classification performance. However, when we use other

Table 4: The information of feature sets.

		Benign	Malicious
Feature set I	Permission feature	571	276
	Component feature	848	420
	Environmental feature	151	38
	Total	**1570**	**734**
Feature set II	APK risk feature	20	20
Feature set III	Dynamic feature	45	45
	Attack surface feature	4	4

Table 5: The statistical information of feature files.

		Type	Ave	Max	Min
I (before)	Malicious	734	16.4317	126	1
	Benign	1570	13.1863	193	1
I (after)	Malicious	286	16.3737	111	1
	Benign	352	13.0542	149	1
I + II (after)	Malicious	306	23.4436	121	1
	Benign	372	20.0836	164	1
I + II + III (after)	Malicious	355	24.3196	121	1
	Benign	421	20.4258	164	1



	KNN	DT	SVM	LR	XGBoost	RF	MLP	CNN
I	94.8111	93.8544	94.5649	94.6009	94.1300	95.4327	95.7720	99.1190
I+II	95.7894	95.1192	95.7420	95.8164	95.4992	96.6428	97.1354	99.2213
I+II+III	95.7660	95.5516	96.0759	96.4574	96.6412	96.9191	97.5437	99.3538

Figure 6: Detection accuracy of different algorithms under different feature sets.

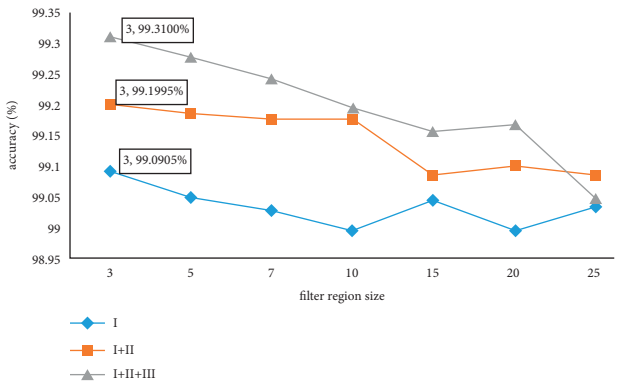


Figure 7: Accuracy of models with different filter region size.

Table 6: Results obtained using different feature sets.

Region size	Train time (s)	ACC (%)	PRE (%)	REC (%)	F1 (%)
Feature set I					
(3)	37	99.0905	98.9735	99.1794	99.0763
(5)	39	99.0508	98.3917	99.6995	99.0413
(3,4,5)	63	99.1190	99.0872	99.1216	99.1044
(2,3,4,5)	80	99.0962	98.7040	99.4683	99.0847
(3,3,3)	62	99.0280	99.2337	98.7864	99.0096
(3,3,3,3)	74	99.1303	98.7719	99.4683	99.1189
(7,7,7)	81	99.0621	98.9616	99.1331	99.0473
Feature set I + II					
(3)	36	99.1995	99.0590	99.3307	99.1947
(5)	38	99.1886	98.7705	99.6050	99.1860
(3,4,5)	63	99.2213	99.2966	99.1332	99.2148
(2,3,4,5)	76	99.2430	99.1781	99.2978	99.2379
(3,3,3)	58	99.1886	99.2096	99.1551	99.1824
(3,3,3,3)	71	99.2267	98.9524	99.4953	99.2231
Feature set I + II + III					
(3)	43	99.3100	98.8877	99.7360	99.3100
(5)	44	99.2771	99.0153	99.5381	99.2760
(3,4,5)	70	99.3538	99.0061	99.7030	99.3534
(2,3,4,5)	85	99.2552	99.0794	99.4281	99.2534
(3,3,3)	69	99.2662	99.0044	99.5271	99.2650
(3,3,3,3)	79	99.3045	98.9089	99.7030	99.3044

filter combinations far from the optimal region size, we cannot achieve a better detection effect than a single filter. For example, when using Feature Set I, the detection effect of a single filter (3) is better than that of filter combinations (7,7,7).

When all the features are applied, the model performs best when the filter region size is (3,4,5). The classification accuracy is 99.3538%, and the training time is 70 seconds. The change of accuracy and loss on the training set and verification set is shown in Figures 8 and 9. The orange line results from the training set, and the blue line is the result of the verification set. The model tends to be stable after 1,250 batch iterations, with the verification set's accuracy floating around 99.4% and the loss value floating around 0.02.

4.5.2. Effect of Number of Feature Maps. We again hold other configurations constant and test the influence of the number of feature maps. The experimental results are shown in Figure 10. The "best" number of feature maps depends on the feature sets. As the number of feature maps increases, the classification effect is getting better and better. However, when it exceeds the critical value marked in Figure 10, the model may reduce the classification accuracy due to overfitting. On the other hand, as the number of feature maps increases, the model's training time is longer.

4.5.3. Effect of Regularization. Dropout is a common regularization strategy. During the learning process of a neural network, it temporarily discards some units with a certain probability and discards the weights of all nodes connected to them. Keeping the other settings unchanged, we use the feature set I + II + III to test the effect of Dropout. The experimental result is shown in Figure 11. When the number of feature maps is 64, the model performs best when the dropout rate is 0.5. The classification accuracy is 99.3538%, and the training time is 70 seconds. At this time, the number of randomly generated network structures is also the largest. When the number of feature maps is 500, the model performs best when the dropout rate is 0.6. The classification accuracy is 99.3921%, and the training time is 200 s. By comparing the two sets of data, we can find that Dropout's effect is more noticeable when the amount of data is large. When the network has the risk of overfitting, we can try the following methods to prevent overfitting: 1. Applying batch standardization between convolution layers can regularize and avoid the gradient disappearance and reduce training time. 2. When Dropout is applied to the full connection layer, the dropout rate can be set to about 0.5.3, combining learning rate decay and Adam optimization algorithm to improve the model's detection effect further.

4.6. Contrast Experiment with Deep Learning Algorithms. To show the performance of our model, we investigated similar approaches that have been previously proposed. Table 7 shows the results of the investigations. Many existing methods utilize the malware samples from the VirusShare. Therefore, we include the performance in the table when using the samples from the VirusShare in the detection test.

As shown in Table 7, our model's detection accuracy and the F1-score values are higher than the other methods. [14, 33] adopt the method of generating feature vectors based on existence. If there is a corresponding feature in the application, it is expressed as 1; if it does not exist, it is expressed as 0. The vectors generated by this method are too sparse, and the dimension of feature vectors is high. Our method can overcome these shortcomings and achieve better detection results.

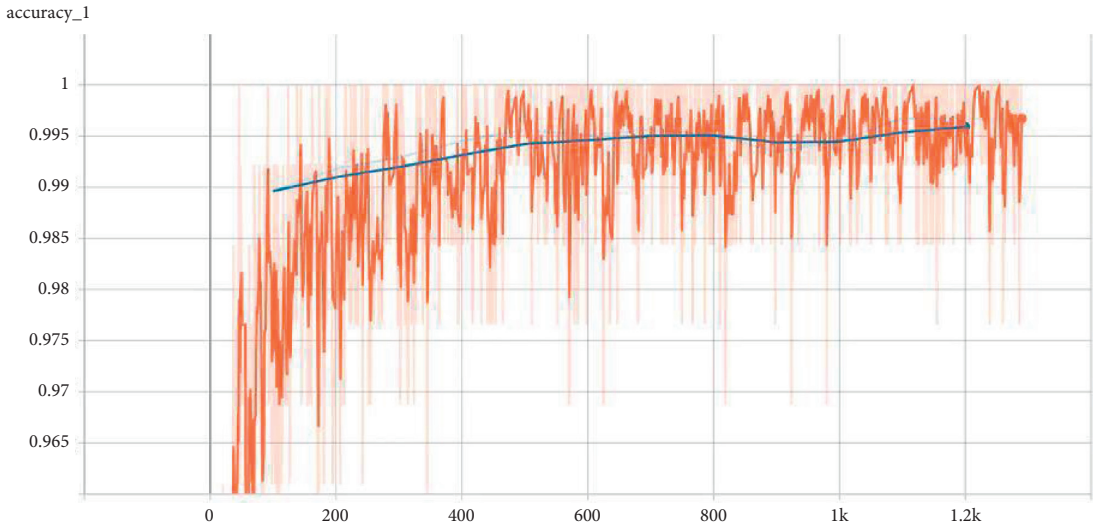


Figure 8: The change of accuracy.

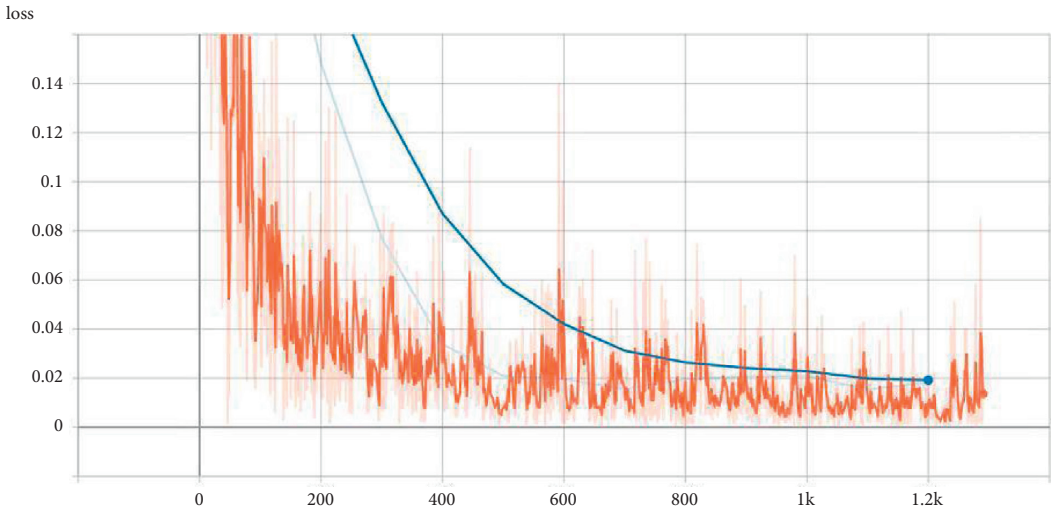


Figure 9: The change of loss.

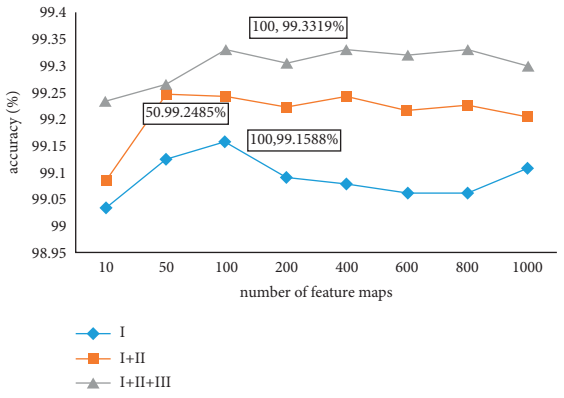


Figure 10: Effect of number of feature maps for different feature sets.

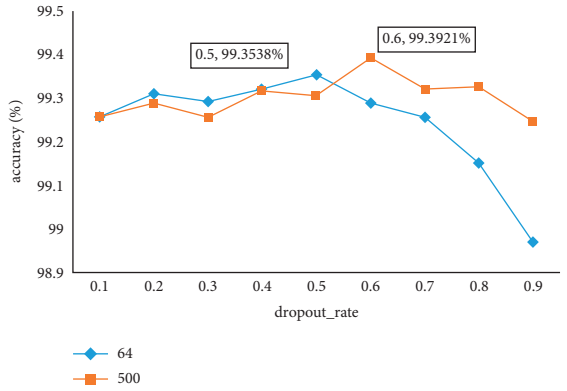


Figure 11: Effect of regularization.

Table 7: The results of the investigations.

System	Dataset		Algorithm	ACC/F1
	Benign applications	Malware		
Ours	Google play store: 18453	VirusShare + drebin: 18549	CNN	99.3538%/99.3534%
[33]	Google play store: 3000	Android malware genome project + Virusshare: 8000	DBN	NA/95.05%
[14]	Google play store: 19747	VirusShare: 13075 malgenome project: 1209	DNN	98%/NA
[34]	360 security company and wandoujia app store:10948	Drebin + VirusShare: 8652	CNN	96.54%/95.89%
[35]	AndroZoo: 5215	VirusShare + android malware genome project: 5442	DBN	98.71%/NA
[36]	Google play store: 8000	VirusShare: 8000	CNN	95.8%/NA

We measured the method proposed in [36], and the result is shown in Table 8. The feature vectorization method of [5] first reads the classes.dex file as an unsigned vector and then converts the vector into a fixed size by resampling. Resampling algorithms commonly used in image processing include Nearest Neighbor Interpolation, Bilinear Interpolation, and Bicubic Interpolation. Unlike two-dimensional images, [5] uses a similar method to resample one-dimensional sequences and convert the original bytecode of classes.dex into a fixed-size sequence. Besides, our model uses various kinds of features to reflect the various aspects of applications.

4.7. Dynamic Adjustment Method. To make the model meet more application scenarios, we design a dynamic adjustment method of the model according to the user requirements, the number of applications, and the average number of features. The detailed description is shown in Figure 12.

4.7.1. Mode Selection. The model includes three detection modes, namely, Android malware detection, Android malware familial classification, and benign application classification. Users can select the corresponding mode according to their own needs.

4.7.2. Training Set. We will regularly update and add the training set to ensure the model's adaptability and long-term effectiveness for Android malware detection mode. For Android malware familial classification mode, users need to upload the applications according to the familial classification or add/delete the familial applications based on Drebin. For benign application classification mode, users need to upload applications according to the application store's classification.

4.7.3. Tool Selection. Users can choose the detection tool according to their own needs (high efficiency/high accuracy), but to ensure the detection effect, APKTool is a required tool. Throughout the previous experiment, we observed that even if only feature set I is used, the model can still maintain a high accuracy of 99.1303% while detecting rapidly.

4.7.4. Obtaining Indicators. According to the tools that are selected by the user, feature extraction and preprocessing operations are performed to obtain indicators: the total number of feature files and the average number of features.

4.7.5. Dynamic Generation of Optimal Model. For multiclassification mode, users need to select detection tools and upload training sets. According to the indicators and conversion table shown in Table 9, CNN, with different parameters, is automatically generated to train the data. The model with the highest detection accuracy is stored as the final detection model.

Table 8: Comparison of experimental results.

	PRE (%)	REC (%)	ACC (%)	F1 (%)
Ours	99.0061	99.7030	99.3538	99.3534
[36]	95.4	96.2	95.8	95.8

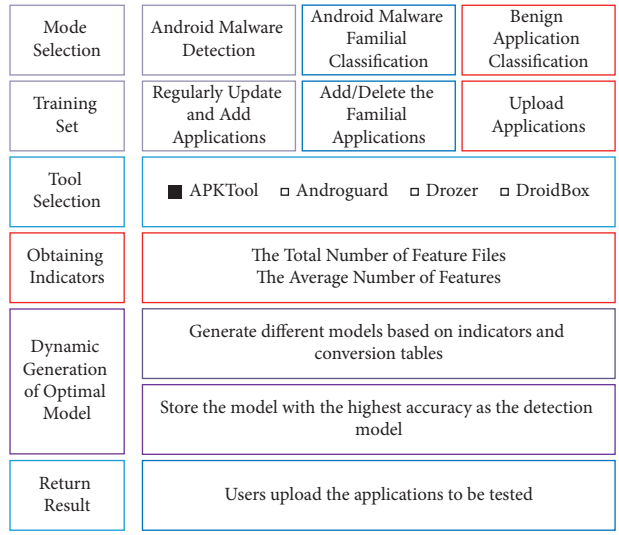


Figure 12: Dynamic adjustment method.

4.7.6. Return Result. The user uploads the applications to the server, and the server uses the detection tools selected by the user to perform feature extraction and preprocessing operations. Then, use the optimal mode generated in the previous step for detection, get the detection result, and return it to the user.

5. Detection of Malware Families

This section uses the Drebin dataset to evaluate the model's performance on the classification of malware families. The structure of CNN is shown in Figure 13, and the configuration parameters are as follows. The filter area size is set to (2, 3, 4, 5), the feature map is 16, and the activation function is RELU. The embedding size and batch size are 200 and 32, respectively, the epoch is set to 10, and 1-max pooling is selected for pooling. The dropout rate is 0.5, and the train set radio is 90%.

All malware belongs to known malware families. The basic information is shown in Table 10, including the number of applications of each malware family, the number of feature types included in feature set I, and feature set I + II.

In all applications, 90% are randomly selected as the training set, of which 10% are used as the verification set, and the remaining 10% are used as the test set. When the filter region size is (2,3,4,5), the model performs optimally, and the classification accuracy is 99.5614%. Two applications are classified incorrectly, which is higher than the 94.5% classification accuracy of EnDroid [22]. With the addition of feature set II, the classification accuracy and stability of the model have improved. The change of accuracy and loss on the training set and verification set are shown in Figures 14 and 15. The orange line is the result of the training set, and the blue line is the result of the verification set. The model tends to be stable after 1040 batch iterations, with the verification set's accuracy floating around 99% and the loss value floating around 0.04.

6. Detection of Benign Applications

This section evaluates the performance of the model on multiple classifications of benign applications. We collect nine types of applications from the Xiaomi App Store. The basic information is shown in Table 11, including the number of applications of each type (Apps), the types of features they contain (Features), and the maximum (Max), minimum (Min), and average (Ave) number of features contained in the feature files. From Table 11 [27], we can infer that the game applications do not have a specific function because the game applications contain only 191 types of features, and the average number of game applications' features is the smallest. While the sports applications features' minimum number is 11, which means they have specific functions. For example, accurately tracking sports routes, distances, speeds, and altitudes through GPS and measuring heartbeat frequencies and pulses related to medical sports are the typical functions in most sports applications. These applications need to get relevant permissions, namely, positioning permissions (ACCESS_FINE_LOCATION and ACCESS_COARSE_ LOCATION) and sensor permissions (BODY_SENSORS). It is feasible to implement classification according to the applications' features. Besides, to ensure the efficiency of classification management, we only use feature set I.

Table 9: Conversion table.

Number of features files	Number of filters	The average number of features	Filter region size
0–5000	16	10–50	(3,4,5) (3,3,3,3) (2,3,4,5)
5001–10000	32	51–100	(5,6,7) (5,5,5,5) (4,5,6,7)
10001–20000	64	100+	Step1: Search for a single filter region
20001–30000	128		Step2: Search for the combination of filters

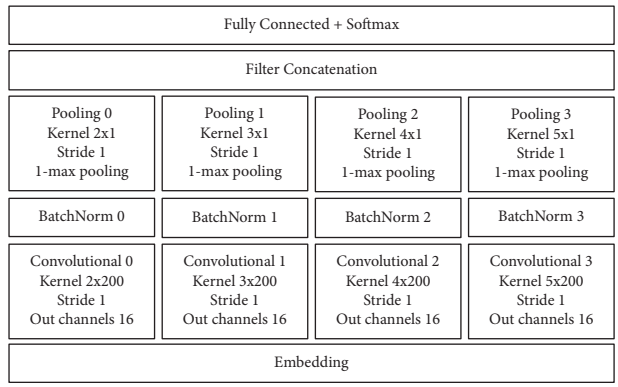


Figure 13: The structure of the CNN.

Table 10: The basic information of malware families.

Malware families	Samples	Feature set I	Feature set I + II
FakeInstaller	925	57	76
DroidKungFu	666	116	135
Plankon	625	133	152
Opfake	613	45	64
GinMaster	339	114	133
BaseBridge	328	66	85
Iconosys	152	33	52
$K_{\min}$	147	30	49
FakeDoc	132	49	68
Geinimi	91	43	62
Adrd	91	105	124
DroidDream	81	77	96
ExploitLinuxLotoor	69	67	86
MobileTx	69	10	29
Glodream	69	41	60
FakeRun	61	24	43
SendPay	59	18	37
Gappusin	58	46	65
Imlog	43	11	30
SMSreg	40	34	53
TOTAL	4658		

In the most relevant work, Shabtai et al. [37] use machine learning algorithms to classify tool and game applications by extracting static features from dex files and manifest files. Wang et al. [38] achieve an accuracy of 82.93% in benign application classification by extracting 11 static features and using a collection of multiple machine learning classifiers. Based on API relationships analysis and CNN, an automatic classification method for Android applications is proposed by Fan et al. [39]. It classifies applications into 24 categories with an average accuracy of 88.9%.

The configuration parameters of CNN are as follows, and its structure is the same as the malicious Android application familial classification model. The pooling and activation function select 1-max pooling and ReLU, respectively, the filter area size is (2,3,4,5), and the feature map is 32. The embedding size and batch size are 200 and 32, respectively, the epoch is set to 10, the dropout rate is set to 0.5, the train set radio is 80%, and the vocabulary is 1288. In all applications, 80% are randomly selected as the training set, of which 10% are used as the validation set, and the remaining 20% are used as the test set.

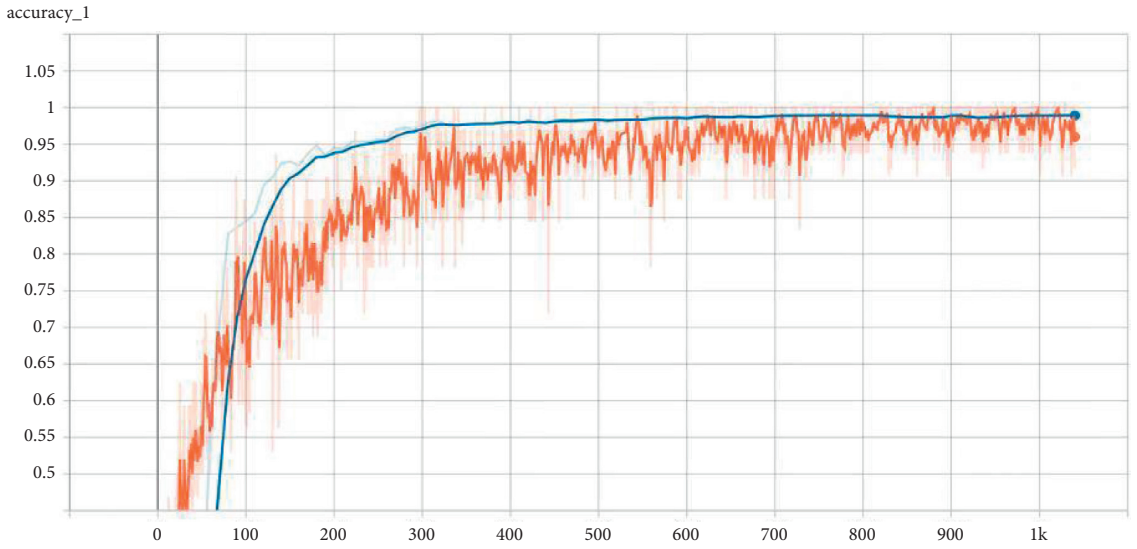


Figure 14: The change of accuracy.

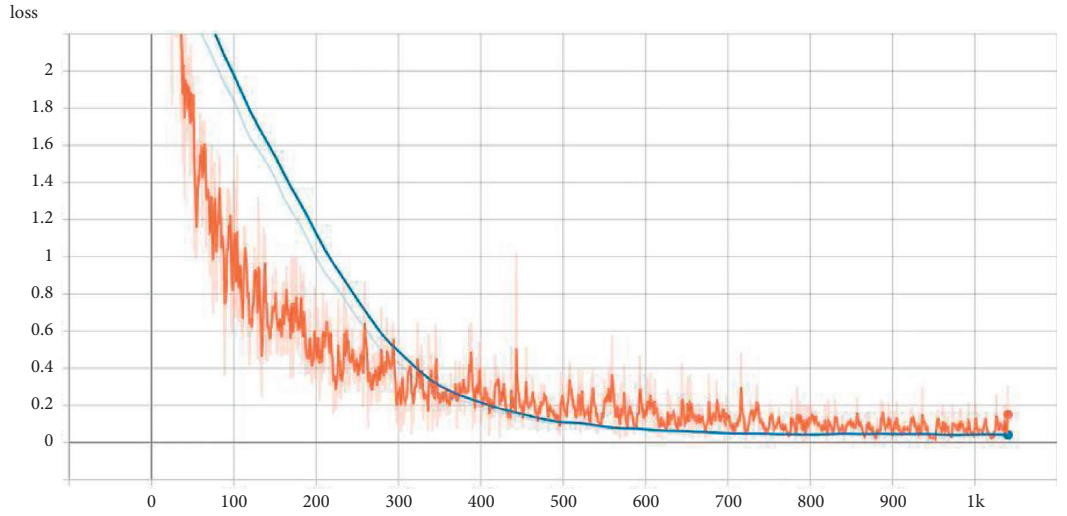


Figure 15: The change of loss.

Table 11: The basic information of feature files.

Type	Apps	Features	Max	Min	Ave
Game	697	191	59	1	13.4562
Book reading	597	450	121	4	33.5343
Audiovisual	659	520	98	3	34.8574
Chat social	741	484	161	1	39.4507
Sports	212	299	111	11	37.9057
News	552	381	122	2	37.5163
Shopping	803	567	132	1	39.7298
Financial	763	405	89	2	40.1782
Camera	323	289	107	1	23.2601

The experimental results are shown in Table 12. After 193s, the accuracy rate reaches 99.9046%. Only one application is misclassified. Suda, which is originally a chat social application, is misclassified as a camera application. After analysis, we found that Suda is a comprehensive application. It includes functions of chat social, taking pictures, and editing pictures simultaneously.

Table 12: Classification accuracy of benign application.

Region size	Train time (s)	ACC (%)
(3)	101	99.4275
(5)	103	99.5229
(3,4,5)	164	99.6183
(2,3,4,5)	193	99.9046
(3,3,3)	163	99.4275
(3,3,3,3)	188	99.6183

7. Conclusion and Future Work

In this paper, we design an Android application classification model based on multiple semantic features. It can extract multiple types of static and dynamic features automatically. We use feature selection algorithms to remove irrelevant or noisy features and extract critical features. These key features help identify dangerous behaviors in unknown applications more effectively. Then, we use CNN to implement classification. We verify the model's effectiveness, the usefulness of the feature sets, and the feature vector generation method's effect through a series of experiments. The model also performs well on malware familial classification and benign application classification and has a short training time.

Despite the effectiveness of our model, several issues remain to be resolved. Our future work will focus on addressing the following problems. During the dynamic analysis process, only 72.6420% of the applications can be correctly analyzed by DroidBox, and 45 useful features are extracted. We would investigate to combine input generator tools IntelliDroid [40] to improve dynamic analysis coverage and extract dynamic features in more detail. Finally, we will further refine our classification model to enable more accurate malware detection.

Data Availability

The data used to support the findings of this study have been deposited at https://github.com/blackwall0321/malicious_applications_detection.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This research was financially supported by China Postdoctoral Science Foundation funded project (2019M650606), the Opening Project of Guangdong Provincial Key Laboratory of Information Security Technology (202B1212060078-12), First-class Discipline Construction Project of Beijing Electronic Science and Technology Institute (3201012), and the National Key Research and Development Program of China under Grant (2018YFB0803401).

References

[1] International Data Corporation, "Global smartphone market data report [EB/OL]," 2020, https://www.idc.com/getdoc.jsp?containerId=prCHC45975020.

[2] J. Yan, Y. Qi, and Q. Rao, "Detecting malware with an ensemble method based on deep neural network," *Security and Communication Networks*, vol. 2018, no. 1, pp. 1–16, 2018.

[3] M. Ganesh, P. Pednekar, P. Prabhuswamy, D. S. Nair, Y. Park, and H. Jeon, "CNN-based Android malware detection," in *Proceedings of the 2017 International Conference on Software Security and Assurance (ICSSA)*, pp. 60–65, IEEE, Altoona, PA, USA, July 2017.

[4] Z. Xu, K. Ren, S. Qin, and F. Craciun, "CDGDroid: android malware detection based on deep learning using CFG and DFG," in *Proceedings of the International Conference on Formal Engineering Methods*, Springer, Cham, pp. 177–193, 2018.

[5] P. Zegzhda, D. Zegzhda, E. Pavlenko, and G. Ignatev, "Applying deep learning techniques for Android malware detection," in *Proceedings of the 11th International Conference on Security of Information and Networks*, vol. 7, September 2018.

[6] Z. Yuan, Y. Lu, and Y. Xue, "Droiddetector: android malware characterization and detection using deep learning," *Tsinghua Science and Technology*, vol. 21, no. 1, pp. 114–123, 2016.

[7] Y. Kim, "Convolutional neural networks for sentence classification," arXiv:1408.5882, 2014.

[8] T. Lei, Z. Qin, Z. Wang, Q. Li, and D. Ye, "EveDroid: event-aware android malware detection against model degrading for IoT devices," *IEEE Internet of Things Journal*, vol. 6, no. 4, pp. 6668–6680, 2019.

[9] R. Kumar, X. Zhang, W. Wang, R. U. Khan, J. Kumar, and A. Sharif, "A multimodal malware detection technique for android IoT devices using various features," *IEEE Access*, vol. 7, pp. 64411–64430, 2019.

[10] C. Hasegawa and H. Iyatomi, "One-dimensional convolutional neural networks for Android malware detection," in *Proceedings of the 2018 IEEE 14th International Colloquium on Signal Processing & Its Applications (CSPA)*, March 2018.

[11] H. Zhang, S. Luo, Y. Zhang, and L. Pan, "An efficient android malware detection system based on method-level behavioral semantic analysis," *IEEE Access*, vol. 7, no. 7, pp. 69246–69256, 2019.

[12] Y. Fang, Y. Gao, F. Jing, and L. Zhang, "Android malware familial classification based on DEX file section features," *IEEE Access*, vol. 8, no. 8, pp. 10614–10627, 2020.

[13] Y. Feng, I. Dillig, S. Anand, and A. Aiken, "Apposcopy: automated detection of Android malware (invited talk)," in *Proceedings of the 2nd International Workshop on Software Development Lifecycle for Mobile*, pp. 13-14, Hong Kong, China, November 2014.

[14] K. TaeGuen, K. BooJoong, R. Mina, S. Sezer, and E. G. Im, "A multimodal deep learning method for android malware detection using various features," *IEEE Transactions on Information Forensics and Security*, vol. 14, pp. 773–788, 2019.

[15] A. Saracino, D. Sgandurra, G. Dini, and F. Martinelli, "MADAM: effective and efficient behavior-based android malware detection and prevention," *IEEE Transactions on Dependable and Secure Computing*, vol. 15, no. 1, pp. 83–97, 2016.

[16] A. Narayanan, L. Yang, L. Chen, and L. Jinliang, "Adaptive and scalable android malware detection through online learning," in *Proceedings of the International Joint Conference*

on Neural Networks, pp. 2484–2491, Vancouver, BC, Canada, July 2016.

[17] M. A. Azad, F. Riaz, A. Aftab, S. K. J. Rizvi, J. Arshad, and H. F. Atlam, "DEEPSEL: a novel feature selection for early identification of malware in mobile applications," *Future Generation Computer Systems*, vol. 129, pp. 54–63, 2022.

[18] M. Nisa, J. H. Shah, S. Kanwal et al., "Hybrid malware classification method using segmentation-based fractal texture analysis and deep convolution neural network features," *Applied Sciences*, vol. 10, no. 14, p. 4966, 2020.

[19] J. Hemalatha, S. A. Roseline, S. Geetha, S. Kadry, and R. Damaševičius, "An efficient DenseNet-based deep learning model for malware detection," *Entropy*, vol. 23, no. 3, p. 344, 2021.

[20] R. Feng, S. Chen, X. Xie, G. Meng, S.-W. Lin, and Y. Liu, "A performance-sensitive malware detection system using deep learning on mobile devices," *IEEE Transactions on Information Forensics and Security*, vol. 16, pp. 1563–1578, 2021.

[21] P. Chaurasia, "Dynamic analysis of Android malware using droidbox," Dissertations & Theses, Gradworks, 2015.

[22] P. Feng, J. Ma, C. Sun, X. Xu, and Y. Ma, "A novel dynamic android malware detection system with ensemble learning," *IEEE Access*, vol. 6, no. 6, pp. 30996–31011, 2018.

[23] W. Enck, P. Gilbert, S. Han et al., "TaintDroid," *ACM Transactions on Computer Systems*, vol. 32, no. 2, pp. 1–29, 2014.

[24] K. Tam, S. J. Khan, A. Fattori, and L. Cavallaro, "CopperDroid: automatic reconstruction of android malware behaviors," in *Proceedings of the Internet Society Network and Distributed System Security Symposium*, pp. 15–26, San Diego, California, February 2015.

[25] "APKtool," 2019, https://ibotpeaches.github.io/Apktool.

[26] A. Desnos, G. Gueguen, and S. Bachmann, "Androguard package [EB/OL]," https://androguard.readthedocs.io/en/latest/api/androguard.html.

[27] Z. Wang, G. Li, and Y. Chi, "Multi-classification of android applications based on convolutional neural networks," in *Proceedings of the CSAE 2020: The 4th International Conference on Computer Science and Application Engineering*, Sanya, China, October 2020.

[28] "VirusShare," 2019, https://virusshare.com.

[29] D. Arp, M. Spreitzenbarth, M. Hübner, H. Gascon, and K. Rieck, "DREBIN: effective and explainable detection of android malware in your pocket," in *Proceedings of the 2014 Network and Distributed System Security Symposium*, San Diego, California, February 2014.

[30] Google Play Store, https://play.google.com/store, 2019.

[31] VirusTotal, https://www.virustotal.com/ko[Online], 2019.

[32] Y. Zhang and B. Wallace, "A sensitivity analysis of (and practitioners' guide to) convolutional neural networks for sentence classification," 2015, https://arxiv.org/abs/1510.03820.

[33] D. Zhu, H. Jin, Y. Yang, D. Wu, and W. Chen, "DeepFlow: deep learning-based malware detection by mining Android application for abnormal usage of sensitive data," in *Proceedings of the 2017 IEEE symposium on computers and communications (ISCC)*, pp. 438–443, IEEE, Heraklion, July 2017.

[34] D. Zhu, T. Xi, P. Jing, D. Wu, Q. Xia, and Y. Zhang, "A transparent and multimodal malware detection method for android apps," in *Proceedings of the 22nd International ACM Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, pp. 51–60, FL, Miami Beach, USA, November 2019.

[35] X. Qin, F. Zeng, and Y. Zhang, "MSNdroid: the Android malware detector based on multi-class features and deep belief network," in *Proceedings of the ACM Turing Celebration Conference-China*, pp. 1–5, Chengdu, China, May 2019.

[36] Z. Ren, H. Wu, Q. Ning, I. Hussain, and B. Chen, "End-to-end malware detection for android IoT devices using deep learning," *Ad Hoc Networks*, vol. 101, p. 102098, 2020.

[37] A. Shabtai, Y. Fledel, and Y. Elovici, "Automated static code analysis for classifying android applications using machine learning," in *Proceedings of the International Conference on Computational Intelligence and Security*, pp. 329–333, Nanning, China, December 2010.

[38] W. Wang, Y. Li, X. Wang, J. Liu, and X. Zhang, "Detecting Android malicious apps and categorizing benign apps with ensemble of classifiers," *Future Generation Computer Systems*, vol. 78, pp. 987–994, 2018.

[39] W. Fan, Y. Chen, Y. A. Liu, and F. Wu, "DroidARA: android application automatic categorization based on API relationship analysis," *IEEE Access*, vol. 7, pp. 157987–157996, 2019.

[40] M. Y. Wong and D. Lie, "IntelliDroid: a targeted input generator for the dynamic analysis of android malware," in *Proceedings of the Network & Distributed System Security Symposium*, Ontario, Canada, January 2016.