Hindawi
Security and Communication Networks
Volume 2021, Article ID 9964224, 11 pages
https://doi.org/10.1155/2021/9964224

Research Article

Learning-Based Detection for Malicious Android Application Using Code Vectorization

Lin Liu,[1,2] Wang Ren,[1,2] Feng Xie,[1] Shengwei Yi,[1] Junkai Yi,[3] and Peng Jia[4]

[1]*China Information Technology Security Evaluation Center, Beijing, China*
[2]*College of Electronics, Sichuan University, Chengdu, China*
[3]*Beijing Information Science and Technology University, Beijing, China*
[4]*College of Cybersecurity, Sichuan University, Chengdu, China*

Correspondence should be addressed to Peng Jia; pengjia@scu.edu.cn

Received 8 March 2021; Revised 1 July 2021; Accepted 10 August 2021; Published 19 August 2021

Academic Editor: Leandros Maglaras

The malicious APK (Android Application Package) makers use some techniques such as code obfuscation and code encryption to avoid existing detection methods, which poses new challenges for accurate virus detection and makes it more and more difficult to detect the malicious code. A report indicates that a new malicious app for Android is created every 10 seconds. To combat this serious malware activity, a scalable malware detection approach is needed, which can effectively and efficiently identify the malware apps. Common static detection methods often rely on Hash matching and analysis of viruses, which cannot quickly detect new malicious Android applications and their variants. In this paper, a malicious Android application detection method is proposed, which is implemented by the deep network fusion model. The hybrid model only needs to use the sample training model to achieve high accuracy in the identification of the malicious applications, which is more suitable for the detection of the new malicious Android applications than the existing methods. This method extracts the static features in the core code of the Android application by decompiling APK files, then performs code vectorization processing, and uses the deep learning network for classification and discrimination. Our experiments with a data set containing 10,170 apps show that the decisions from the hybrid model can increase the malware detection rate significantly on a real device, which verifies the superiority of this method in the detection of malicious codes.

1. Introduction

Among all smartphone operating systems, Android has occupied over 85% of market share. Besides, Android powered devices such as cars, fridges, televisions, point of sale (POS) terminals, and ATM booths are expected to flood the user markets within a few years. Due to the popularity of the Android ecosystem, the malware writers are targeting the Android devices exclusively, and the number of Android malicious apps surged exponentially. Android implements a number of security mechanisms like the permission mechanism to ensure the safety of device resources. The permission mechanism of Android is coarse-grained, and users are usually ignorant of the sought permissions. Hackers also proposed the attacks that can bypass the permission mechanism [1–3]. As a result, the effective detection of malware is very important to mitigate security threats in the Android ecosystem. The new malicious Android applications are also emerging. The innovation of Android source code security detection technology needs to be greatly valued [4, 5].

The static analysis and dynamic system-level behavior analysis are common methods used to detect the malicious apps. The static analysis utilizes the reverse-engineering techniques to analyze the source code of the Android application, which relies on the semantic signatures and focuses on analyzing code snippets without executing them [6, 7]. It extracts the static features from the malicious apps, including all string constants [8] and URL addresses in the source code [9, 10], function names of all components in

the source code [11], and any other static information to determine whether an app exhibits malicious behavior. These methods are more resilient to changes in malware codes. The dynamic analysis uses the operating behavior of the software during operation [12, 13]. The dynamic analysis is more effective because it can extract functions that represent unique execution modes, such as system function call sequence [14], function call frequency [15], and function call combination [16, 17]. Interestingly, according to this study, over 98% of new malware samples are in fact variants of an existing malware family. Google uses a dynamic analysis system called Google Bouncer that analyzes APKs submitted to Google Store. Unfortunately, the dynamic analysis techniques that execute the Android apps inside an emulator also suffer from the fact that the malware writers can detect emulators and thus evade detection. The APIs and permissions are usually the focus of APK analysis, because they include a lot of information related to user security, such as user passwords, geographic location, and browsing information. Many applications will apply for more permissions than they really need, which makes users face more permission warnings and increases the risk of permission being exploited [18–20]. In most existing studies, specific API calls [21–23] and specific API call sequences [24] are commonly used for the dynamic characteristics. In order to balance the shortcomings of static and dynamic analysis, a hybrid analysis method of static and dynamic methods has emerged [25–27]. In recent years, as a highly complex network with superior performance and easy implementation, deep neural networks have emerged in various fields, including the field of virus software detection. However, according to the above related research, it can be found that the current network structure in the field of Android malicious application detection has not been studied in depth. Most scholars simply apply the existing network structure, and such a simple application does not give full play to the selected network structure. Therefore, this paper proposes an improved algorithm based on category discrimination to make up for the above problems and improve the accuracy and recall rate of malicious APK classification in response to the shortcomings of the existing virus APK detection algorithms.

In summary, this paper makes the following contributions:

(1) This article designs a feature data set based on the characteristics of the decompiled code of Android software and also designs a neural network to extract code features.

(2) This paper proposes a deep convolution model for Android malicious app detection using multiple receptive fields—DTCNN (Deep Text Convolutional Neural Networks). This model uses a variety of sizes of convolution kernels to take into account local features and short-distance cooccurrence features. The depth and scope of information acquisition are greatly improved, and the different granularities of information can be used to make decisions.

(3) It develops the DTCNN-LSTM hybrid model, an approach that extracts the deep features of the program through the convolution kernels of multiple sizes, and emphasizes the security semantics and the logical order of the code. This method greatly reduces the huge workload that is brought by the feature engineering required in the traditional method.

The rest of this paper is organized as follows: In Section 2, some existing studies using deep learning models and analyzing their shortcomings are briefly reviewed. In Section 3, the static and timing features in the core code of the Android application are extracted and vectorized by decompiling APK files. In Section 4, an improved detection model is proposed. In Section 5, the proposed model is evaluated through experiments and result analysis. Finally, this paper concludes the detection approach in Section 6.

2. Related Work

In recent years, deep learning and machine learning have become a popular branch of data science. Deep learning is currently recognized in the field of machine learning as a very effective algorithm for classification and detection problems. It is widely used in the detection, recognition, and classification tasks of text, images, and other objects. It can also be applied to the direction of malicious Android application detection.

Zhou et al. [28] used convolutional neural networks (CNN) to extract feature information in a sequence of word vectors with a convolution window of a specific size for the word vectors in the text. Convolutional neural network can play a good effect in extracting local feature information, but it always ignores text context information. Pascanu et al. [29] studied the use of RNN (recurrent neural networks) to process the input API sequence and added the maximum pooling layer to achieve the purpose of extracting fixed-length feature sequences from the variable-length input sequence. The structure of RNN makes it commonly used to process short text objects with a logical order, while the malicious Android applications have a long API sequence. Kolosnjaji et al. [30] also used convolutional networks and long- and short-term memory networks to analyze the calling sequence. However, they only model and classify malicious call sequences, and the feature information is not comprehensive enough. Caviglione and others [31] successfully detected information data communication of suspicious software through deep learning algorithms, but the actual detection only through communication information will have a higher probability of misjudgment.

The above research on malicious Android applications through various deep learning and machine learning methods shows that the detection model is developing in the direction of multitechnology integration. Neural networks such as CNN, RNN, and DBN not only play an important role in natural language processing [32], audio analysis [33], and image detection [34], but also have great potential in Android application detection. The program itself is a very

long sequence with only one-dimensional semantic dependence. The extracted operation instruction sequence of Android can be regarded as a text with a fixed dictionary language set, and this text has a very strong local relevance, which is formed by the stack of operation instructions. This paper optimizes the model construction, design of the hidden layer, input data structure, and other aspects to significantly improve the overall performance of the model.

According to the shortcomings of the above research, this paper proposes a novel convolutional network structure, DTCNN (Deep Text Convolutional Neural Network), for code vector classification and establishes a hybrid model combining DTCNN and LSTM (Long Short-Term Memory). It uses longitudinal experiments to verify the superiority of the DTCNN model in detecting the malicious Android applications.

3. Code Vectorization for the Deep Learning Model

The first step is to extract the vectorization description of the massive sample data as the input of the model. In this paper, the significant permissions from the apps are extracted, and the extracted information is applied to detect malware, using deep learning algorithms effectively. The design objective of selecting significant permissions is to detect the malware efficiently and accurately. As stated earlier, the number of the newly introduced malware samples is growing at an alarming rate. Thus, being able to detect the malware efficiently would allow the analysts to be much productive in identifying and analyzing them.

The second step is to design the network model. Through studying the Android malware detection based on the deep neural networks, this paper designs an end-to-end information processing model. The disadvantage of the shallow network model is that it is difficult for it to learn the logic information of the Android application code language completely, so the DTCNN with excellent feature learning ability is used to extract the high-level abstract features of the code vector features. Although the CNN [28] used by Zhou et al. has a powerful local feature mining function, Android malicious applications solve the complex and special structure. The feature information contained in the core code is jumpy and contains a lot of redundant information. It is difficult for CNN to extract the timing information in the code. Therefore, in order to improve the expression of the features on the semantic information of the code, increase the model's ability to learn, and understand the semantic sequence of the security semantic information and the code, the deep learning network structure in this paper is added to LSTM. LSTM is an improved recurrent neural network, which can overcome the problem that the recurrent neural network cannot remember the long-term information under long texts. Therefore, in order to obtain the global and temporal features of the samples, two feature sets will be constructed as the input of the two structures in DTCNN-LSTM model, and the feature dimension will be further strengthened.

In the third step, the designed network is compared with the shallow network to verify the superiority of the proposed method. The detection process is shown in Figure 1.

3.1. Data Set. When training the model, the positive and negative samples should be collected as the learning data to be provided to the DTCNN-LSTM model to prevent the overfitting problems caused by the small data set. The model proposed in this paper uses the static code of the Android application as the source of sample features. The static code is a richer and more comprehensive source of secure semantic information for deep learning than the dynamic files generated by the application running in the sandbox.

The Android application is developed in the Java language, and its code is executed under the interpretation of the Dalvik virtual machine. The Smali is the core code executed inside the Dalvik virtual machine. It has its own set of syntax. Sensitive information is gotten such as the logical structure, the function usage, and the permission calls of the application after analyzing the program's operating principles and process. Therefore, using the source code of the Smali file as the representation form of the semantic features of the Android application, the code logic in the APK is further interpretable and can provide the required features for deep learning. Malicious applications of the same type usually have similar malicious behaviors. For example, a malicious application of privacy stealing type will steal user privacy data beyond the normal requirements, such as retrieval history, GPS location, mail, photo album, account password, and send it to the malicious terminal through WiFi or SMS without the user's authorization. When this malicious behavior occurs, it will trigger the sensitive API function calls, underlying virtual machine instruction calls, etc. Therefore, it is necessary to use deep learning to mine the associations between the features to detect the unknown malicious applications.

This experiment uses the APK decompiling tool, APKTool, to process the sample Android application to obtain the Smali file. In the experiment, the Python script was used to execute specific Shell commands to decompile the APK file to obtain the Smali file automatically. The malicious Android applications may obtain the permissions by calling the sensitive API functions to implement the remote code execution and steal the user's privacy data. For example, the function "getCellLocation( )" can obtain the user's geographic location, and the function "getRunningAppProcesses( )" can obtain the running software information in the user's device. Therefore, the calling sequence of the sensitive API functions has practical significance as a model input feature. The exhaustive search program source code filters out the call flow of some sensitive API functions, focusing on finding the code statements that use invoke-direct, invoke-virtual, invoke-static, invoke-super, and invoke-interface. After filtering the redundant code, the length of the code sequence is integrated to 500 as the first part of the static feature. There are a lot of disassembled codes in the Smali file. In addition to the above API call instructions, the representative Dalvik ones are collected such as jump ones,

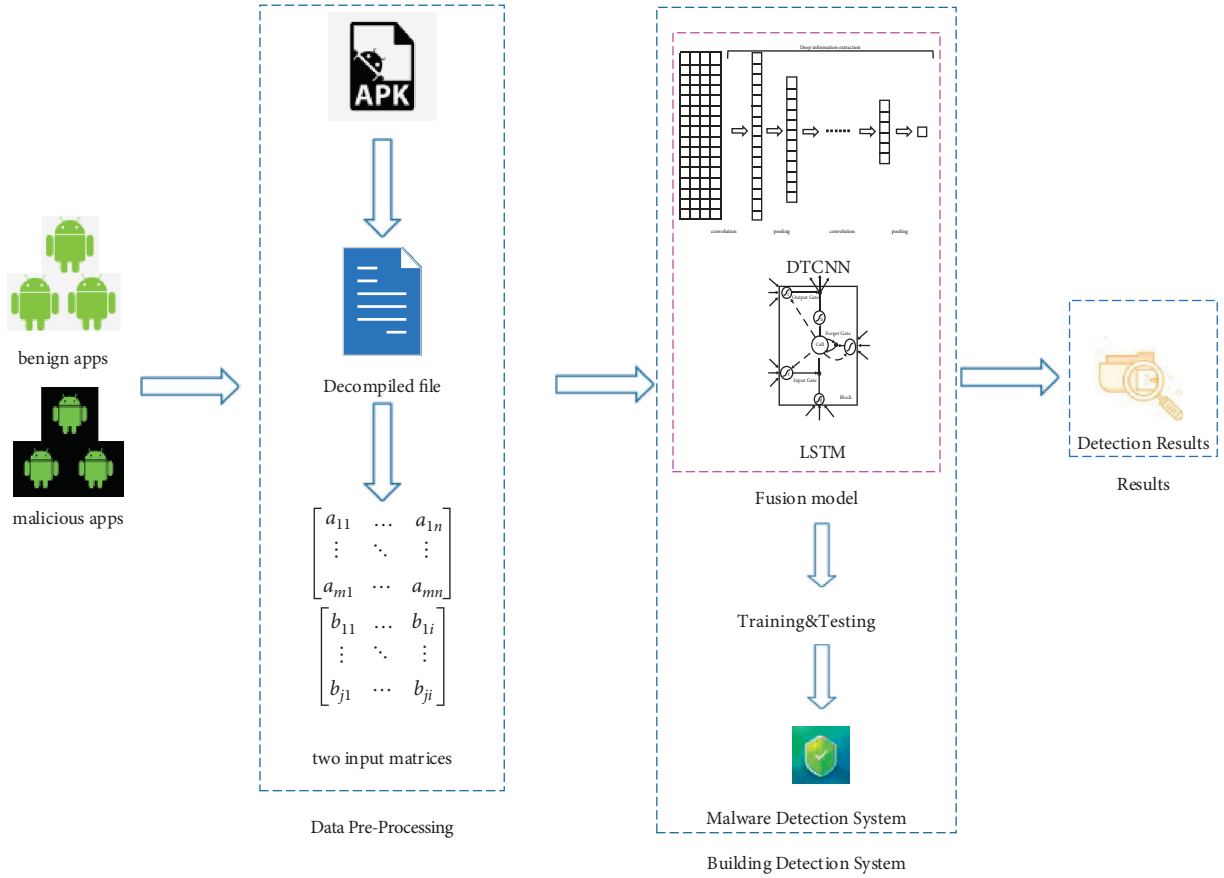

Figure 1: Detection process.

data operation ones, return ones, and other ten functions of instructions. They are taken as the static features of the first two parts. In order to reduce information redundancy, the data is filtered while retaining features, and the Dalvik instruction is described using a custom mapping method. The partial Dalvik directives simplified comparative table is shown in Table 1.

In the experiment, the length of the input sequence of the feature extraction network is 10000. In order to ensure that the input matrix has the same format, the samples are subjected to zero padding. In summary, the input sequence of this experiment is divided into two parts, Group A is a sensitive API function call sequence of length 500, and Group B is a Dalvik instruction sequence of length 10000. Group A and Group B are used as the input of the LSTM module and the DTCNN module, respectively.

3.2. Code Vectorization. In the deep neural network, a large amount of calculation processing and data mapping operations are performed, and only the digital vector matrix can be used as a reasonable network input. The frequency of API calls in the Smali file and the number of other Dalvik instructions are relatively large. If all code information is encoded directly and converted into the vectors, it will cause a "dimensional disaster." It is difficult to express the safety semantic information of the sample and the logical order of the code. In order to solve the problem of data dimension and enable the LSTM network to make better use of the security semantics and the logical order of the code, this paper converts the source text to readable low-dimensional vectors. The word2vec is a word vector training tool released by the Mikolov and others in 2013. Once released, it becomes an important text vectorization tool in the field of natural language processing. Compared with the traditional text representation, the word2vec can improve the approximation of synonyms in a high-dimensional space. The word2vec has two calculation modes, CBOW and skip-gram, which describe the relationship between the words from different angles. This paper chooses to use the skip-gram method with short training time and high accuracy. The model structure is shown in Figure 2. This method infers the context of the target words within a sliding window. The corpus of all data sets will be used in the experiment to train the word2vec model.

Given a set of word sequence $w_1, w_2, \ldots, w_T$, the skip-gram is known to maximize the following formula:

$$L = \frac{1}{T} \sum_{t=1}^{T} \sum_{-c \le i \le c, i \ne o} \log p(w_{t+i} | w_t), \tag{1}$$

where c is the number of words in the context. As the value of c increases, the semantic relationship of the code in the high-dimensional space and the training time will increase. $p(w_{t+i}|w_t)$ is defined using the Softmax function:

Table 1: Partial Dalvik directives simplified comparative table.

Custom description instructions	Meaning	Partial representative prefix
C	Compare	cmp-long/cmpg-float/cmpl-float
G	Jump	goto
R	Return	return/return-void/return-wide
M	Operating	move/move-object
D	Definition	const/const-wide/const-string
T	Read	aget/aget-wide
V	Function call	invoke-virtual/invoke-super
P	Write	aput/aput-wide
I	Judgment	if-le/if-lt/if-eq

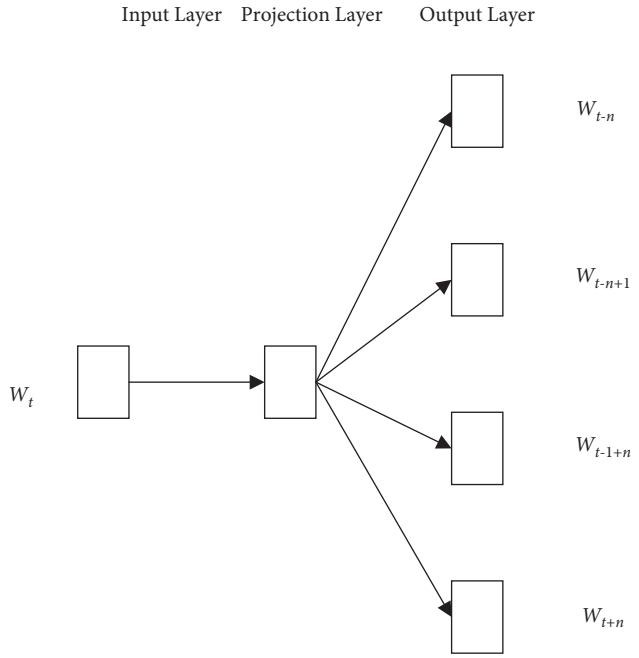

Figure 2: Skip-gram model.

$$p(w_O|w_I) = \frac{\exp\left(v'^{T}_{w_O} v_{W_I}\right)}{\sum_{w=1}^{W} \exp\left(v'^{T}_{w} v_{W_I}\right)}, \quad (2)$$

where v'_w and v_w represent the output and input vector description of w, respectively, W is the dictionary size, the Gensim library training word vector is used in the specific experiment, and the output code vector dimension is 100.

4. Deep Learning Model

Based on the structural characteristics of the code vector, this paper proposes a novel deep convolutional network model called DTCNN. It fuses the improved recurrent neural network to obtain the DTCNN-LSTM model, which completes the extraction of all the deep information of the input code vector.

This paper uses the cleaned Android application decompiled code as the training data, which is highly similar to the text data in the field of natural language processing. Both have the logical and local characteristics. The research results in the field of natural language processing indicate that the fusion model of the convolutional neural network and the recurrent neural network has obtained good experimental results. At the same time, the code data of the different modules in this experimental data set has great differences. If the traditional single model is adopted, the characteristics of different dimensions will be ignored. For this kind of data, a fusion model that takes into account the timing and capturing local features will obtain better inference results. The distance of some related codes in the code sequence data set may be far, and it is necessary to use the characteristics of the convolution network to extract it. However, the current convolutional neural networks used for text classification are mostly the shallow networks. In order to overcome the weakness of the shallow network to learn the security semantic information of Android malware, the DTCNN deep convolutional network is used to extract the sample information. The extracted features have a more essential characterization of the local correlation of the sample data.

Although the LSTM model can overcome the long-term memory problem of RNN and can process the sequence data with a long timeline, the processing effect of the LSTM will decrease when the text length exceeds a certain threshold. Because the API call flow and other Dalvik instruction data in the Smali file used in this study are large, it is difficult to extract the long-distance associations only by using the LSTM. Therefore, this study extracts the deep local features through the deep convolutional networks. At the same time, in order to make up for the shortcomings of the convolutional networks in understanding the logical relationship of the code security semantics and extract the logical information and the time-order relationship of the application, a long-term or short-term memory layer is added to the model. The two-part feature vectors are fused to classify them.

4.1. DTCNN. Inspired by the TextCNN model for the text classification, a deep convolutional network—DTCNN—that extracts the local code information is proposed. Figure 3 is a schematic diagram of the DTCNN structure. The B-Group feature vector set obtained by data processing is used as the input of the DTCNN structure to perform the local depth feature extraction.

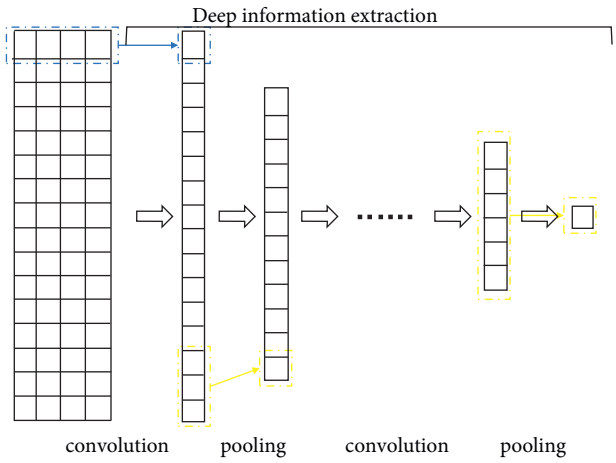

Figure 3: Structure of DTCNN.

In a convolutional network, let $x_i \in R^k$ be a k-dimensional character vector, and in a code sentence of length n, the dimension of the word vector of the i-th word is k. The convolution operation applies a convolution kernel to a window of h characters to generate a new feature. For example, a new feature is generated by

$$c_i = f\left(w \cdot X_{i:i+h-1} + b\right). \tag{3}$$

$b \in R$ is the bias term, and f is a nonlinear function. The convolution kernel is applied to every possible substring in the string to form a feature set:

$$c = [c_1], [c_2], \ldots [c_{1n-h+1}]. \tag{4}$$

Here, $c \in R^{n-h+1}$.

The network structure of the DTCNN includes an input layer, the multiple parallel convolutional layers, and the pooling layers. The convolutional layer and the next adjacent pooling layer are called a convolution module. The DTCNN consists of the multiple convolutional modules. This study uses a DTCNN composed of 9 convolution modules to extract the features. The convolutional layer of the network uses four different heights of the convolution kernels, and their heights, h, are 2, 3, 4, and 5, respectively. In order to reduce the training time and the memory requirements, the number of the above convolution kernels is 50, 100, 150, and 200 and the corresponding convolutional layers are 9, 7, 6, and 5, respectively.

Based on the experimental classification effect, the pooling layer uses the maximum pooling method. The nonlinear activation function uses the ReLU. In order to extract much more local features, the convolution step is always set to 1, the height of the convolution window used in the first convolution layer is h, the width is determined by the word vector dimension, and the width of the remaining convolution kernel is 1. The size and step size of the pooling window are set to h. The final pooling step size of the pooling layer is determined by the input dimensions of the adjacent convolutional layer. The output of a convolution kernel is a 1×1 feature vector. When the height, h, of the convolution kernel is 2, the size of the convolution kernel is $2 \times k$, the output after the first layer of convolution is one-dimensional data, the size of the filter in the connected pooling layer is 2×1, and the step size is 2. The size and step size of the subsequent convolution kernel remain unchanged. In the last pooling layer, the pooling window dimension is equal to the previous convolutional layer data input dimension, so a 1×1 feature vector can be obtained.

The DTCNN module of this experiment finally outputs 500 1×1 feature vectors.

4.2. LSTM Network. The RNN (recurrent neural networks) is a neural network that can process and predict the time series data. The expanded RNN is equivalent to a multilayer feed-forward neural network, which can transfer the information layers. However, in the case of relatively complex training text, it is difficult for the RNN to solve the problem of long-term dependence. The LSTM can perform better in longer sequences than the ordinary RNN. The LSTM uses the memory unit to replace the traditional hidden neuron node and transfers the memory information from the initial position of the sequence to the end of the sequence avoiding the problem of long-term dependence. This memory method is generally realized through three gating mechanisms, namely, input gate, forget gate, and output gate. The LSTM cell structure is shown in Figure 4.

The input gate is used to control the input of the current node unit state. The output gate is used to control how much the current unit state is filtered out. The forget gate controls the degree to which the previous unit state is forgotten.

The neuron's state update calculation method can be expressed as

$$i_t = \sigma\left(W_i\left[h_{t-1}, x_t\right] + b_i\right), \tag{5}$$

$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right), \tag{6}$$

$$f_t = \sigma\left(W_f\left[h_{t-1}, x_t\right] + b_f\right), \tag{7}$$

$$c_t = f_t \cdot c_{t-1} + i_t \cdot c_t', \tag{8}$$

$$c_t' = \tanh\left(W_c\left[h_{t-1}, x_t\right] + b_c\right), \tag{9}$$

$$h_t = o_t \cdot \tanh\left(c_t\right). \tag{10}$$

Here, σ represents the Sigmoid function; tanh represents the hyperbolic tangent function; W_i, W_o, W_f represent the weight matrix of the input gate, output gate, and forget gate, respectively; x, h are the input and output of the memory unit; f_t, o_t, and i_t are the forget gate, input gate, and output gate; c_t', c_t are the candidate value and the new memory cell state; h_t is the final output; and b_i, b_i, and b_f, respectively, represent the offset vector corresponding to each gate.

Equations (5)–(7) represent the calculation process of the input gate, output gate, and forget gate, respectively. Equations (8)–(9) are used to update the state of the memory cell. Equation (10) first applies the tanh to get the current state of the memory cell and then determines the final state through the output gate. Finally, the deep extraction and

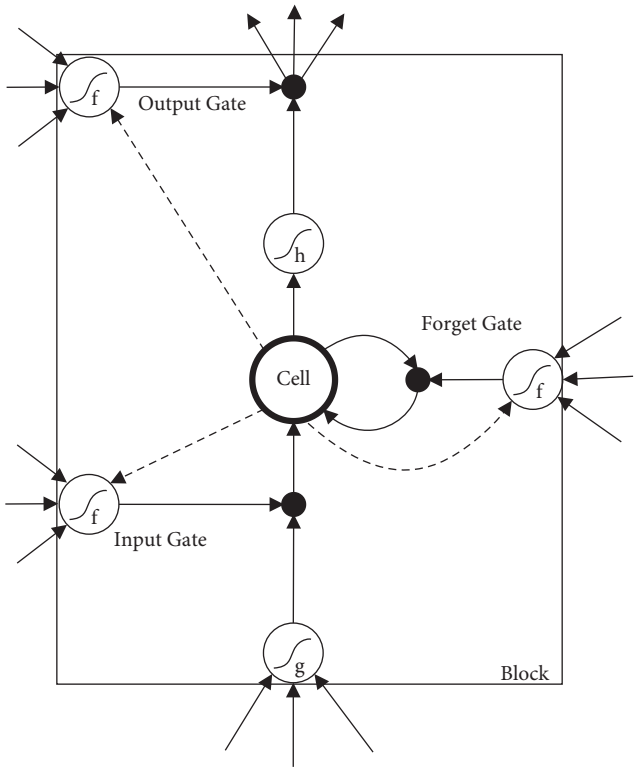

Figure 4: Structure of the LSTM cell.

output of code vector features are completed. The structural characteristics of the LSTM neural network make it have the advantage of processing the data-based language sequences. It has a good classification effect in some semantic processing tasks, which can make up for the shortcomings of the convolutional networks in terms of long-term memory, so it is selected as a deep feature extract being a part of the module.

4.3. Classification with the Fusion Model of DTCNN-LSTM. After extracting the deep information from the DTCNN and LSTM neural networks, the feature vectors are obtained and sent to the fully connected layer. This module needs to complete classification of the feature vectors. Out of consideration of the overall consistency of the model, this study will use a fully connected network and Softmax classifier as the output module of the entire model. Considering that there are many layers of the network structure model, the model sets the activation function ReLU in the DTCNN and the fully connected layer, so as to accelerate the convergence and reduce the learning cycle. In order to prevent all the feature selectors from working together in each iteration and always highlight or weaken some specific features, a dropout mechanism is used between the two fully connected layers. Each training randomly selects 50% of the hidden layer nodes to carry out to make the weight update not dependent on some inherent features and avoid the problem of overfitting and weak generalization ability of the model.

The fusion model is finally shown in Figure 5.

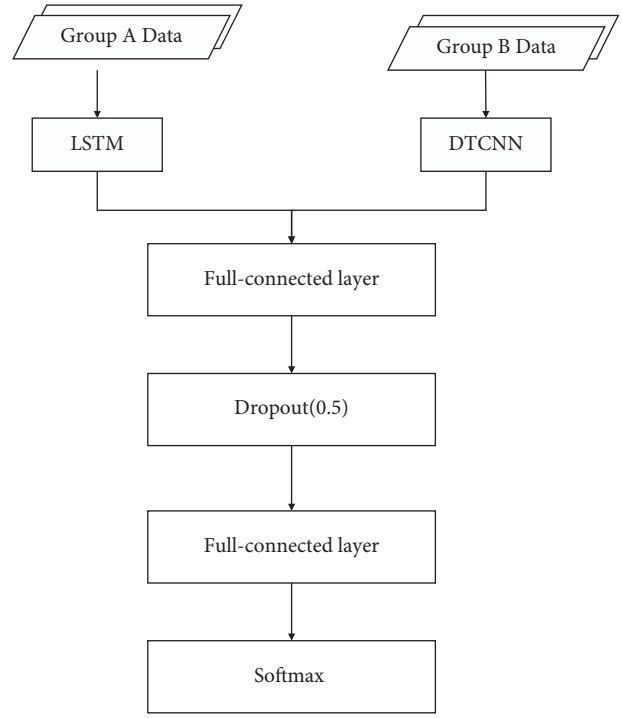

Figure 5: DTCNN -LSTM fusion model.

According to the previous sections, the malicious Android detection system algorithm using the DTCNN-LSTM fusion model is as follows:

Input: a two-part code vector set

Output: 0 (malicious)

1 (benign)

Step 1: screen from the massive sample data, extract the required API call instructions and the Dalvik Opcodes, perform the code vectorization, and express in the form of feature vector.

Step 2: use the static feature block as the input of the DTCNN-LSTM fusion model. The vector set extracts the local abstract features through DTCNN and captures the high-level features in the pooling layer. The LSTM neural network further extracts the timing information for the logical sequence of the code language context.

Step 3: use the TensorFlow framework programming to fuse the features of the DTCNN and the LSTM output.

Step 4: connect the two fully connected layers and the dropout layer, use the Softmax classifier to classify the fusion feature set in Step 3, and finally complete the classification detection of the malicious Android application.

5. Experiment Results and Analysis

5.1. Experimental Data. The quality of the deep learning model is affected by the number and quality of the training samples. The model needs sufficient data samples for training, and it

needs to ensure a reasonable distribution of positive and negative samples. We select the number of benign apps to be three times the malicious apps to maintain balance during training, because the imbalanced data set can result in skewed models.

In order to increase the validity of the model results, this experiment collected enough Android application installation packages as the training samples and the test samples from various sources at home and abroad. This paper regards the benign Android applications mainly from the domestic Android application market and the Google Play Store. Although there is no guarantee that there will be absolutely no malware in the application market, this paper captures the Android applications that have the most downloads and the highest praise rates in the Android application market in China and abroad. The malicious Android application samples used in this study come from the foreign virus database VirusShare. The total sample set contains 2584 positive samples and 7584 negative samples.

5.2. Experimental Environment and Parameters. The experimental environment is shown in Table 2.

In the experiment, in order to balance the training speed and the invalid convergence, the batch size was set to 35. During code vectorization, the word embedding vector dimension of the training output is set to 100, the number of hidden layer units is also set to 100, and the training window size is set to 5. The gradient descent optimization algorithm used to optimize the model parameters is Adam. The loss function is the cross-entropy error. The learning rate is set to 0.1. The filling method is the VALID. The unit number of the hidden layer of the LSTM module is set to 100, and the forget bias is set to 2.0.

5.3. Evaluation Index. In terms of evaluating the effectiveness of the proposed model, the experiment considers the combination of the actual category of the experimental sample and the model prediction category and divides it into true positives, false positives, true negatives, and false negatives. The "confusion matrix" of the test results is shown in Figure 6.

The accuracy rate and the recall rate are used to evaluate the model test results. For the test sample data set, the accuracy rate refers to the ratio of the number of samples correctly identified by the model to the total number of samples, and the recall rate refers to the ratio of the number of correct positive samples to the total number of positive samples in the test data set. The definition formulas of the accuracy rate and the recall rate are as follows:

$$\begin{aligned} \text{accuracy rate} &= \frac{TP + TN}{TP + FP + TN + FN}, \\ \text{recall rate} &= \frac{TP}{TP + FN}. \end{aligned} \tag{11}$$

5.4. Results and Analysis. In order to solve the problem of model overfitting, this paper uses K-fold cross-validation for model tuning in the model realization stage. In K-fold cross-validation, the initial sample is divided into K parts: one part is retained as the data for the verification model, and the other K-1 parts are used as the training set. The cross-validation is repeated K times, and K detection models are obtained. The average result of the K models represents the detection performance of this model under the K-fold cross-validation method. In this article, the K value is set to 4. Figure 7 shows the results obtained for each cross-validation fold. The last set of data in the figure is the average value obtained by the fourfold cross-validation. The accuracy rate is 95.0%, and the recall rate is 93.9%. The DTCNN-LSTM model achieves a good prediction effect.

Table 2: Experimental environment.

Name	Configuration
Operating system	Ubuntu 16.04
CPU	Intel core i5-7500
GPU	Tesla M40 24GB
RAM	DDR3 8G
Development framework	Keras

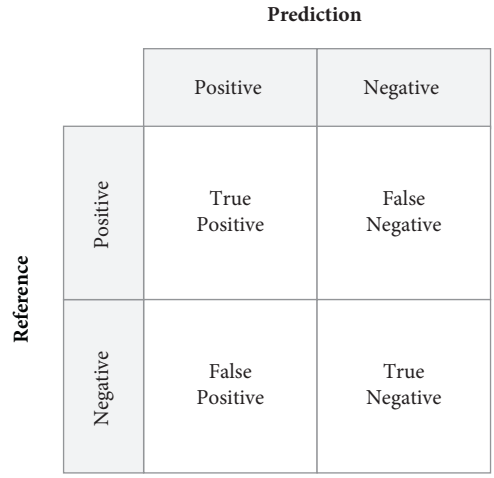

Figure 6: Confusion matrix.

In order to verify that the evolution model proposed in this article has stronger detection capabilities than the other deep learning models that have been used for malware detection, this experiment uses the same batch of samples to compare our model with multiple deep learning models. In the comparative experiment, CNN-LSTM is a hybrid model based on convolutional network and long short-term memory proposed by Wang [35]. Unlike DTCNN-LSTM, the convolutional neural network in CNN-LSTM uses a shallow convolutional network. The DCNN used in the comparison experiment is a two-layer convolutional network for malicious applications proposed by Mclaughlin [36]. The convolution kernel in DCNN has a single size. Except for separate LSTM model and LSTM structure in CNN-LSTM, the models use Group B characteristic vector as input.

The test results are shown in Table 3.

From the experimental data in Table 3, it can be seen that the other models have a shallower layer than the DTCNN-LSTM model. The detection effect is weakened to various

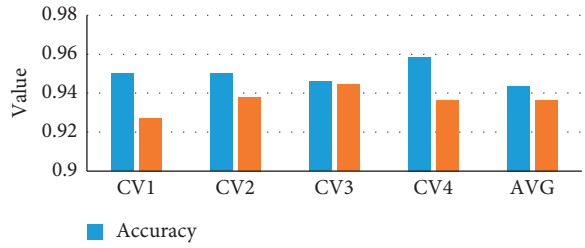

Figure 7: Cross-validation result.

Table 3: Comparison of lateral experiments using different models.

Models	Accuracy rate	Recall rate
LSTM	0.795	0.814
DCNN	0.806	0.803
CNN-LSTM	0.914	0.892
DTCNN-LSTM	0.932	0.930

degrees. These results indicate that the deep convolutional architecture of the DTCNN can characterize the malicious applications more efficiently than the shallow architectures. In the task of detecting the Android malicious applications, single local information may not affect the classification. The maliciousness is determined by the interaction of multiple local information, so the malicious code's ability to express logical features under a single convolutional network is poor.

The prediction accuracy of the DTCNN-LSTM model has improved. The LSTM network captures the logical relationship of the input features effectively and makes up for the deficiencies of a single DTCNN. Although the LSTM network has a long-term memory function, its ability to extract the high-level local information is weak. Only the safe semantic expression of the code context is learned, so the accuracy rate is low. Combining it with other models greatly improves the overall detection effect. This indicates that, in the Android malicious application detection tasks, abstracting the features to a higher level while paying attention to the learning of the dependence of the input feature vector can get a good classification effect.

6. Conclusion

This paper establishes a malicious Android application detection model based on the deep convolutional network DTCNN and LSTM network and implements the Android malware detection algorithm.

The model is horizontally compared with a single convolutional network model, a long and short memory network model, and a CNN-LSTM fusion model to verify the effectiveness of the model. The positive and negative sample sets used in this paper are from the domestic Android application market and VirusShare. In the experiment, the source code acquisition and information filtering of the application were first carried out. The random parts were selected from the positive and negative sample sets as the training data set and the test data set. The code vectorization operation was performed, and the DTCNN-LSTM model parameters were trained using the feature vectors. Finally, the Android malicious applications in the test set are classified and identified. It is shown that the fusion model performs well in understanding the security semantics of the malicious Android applications and extracting the local information. This result shows that the organic combination of the modeling methods using different functions and different optimization principles is a way to improve the effectiveness of the model based on the deep network structure. An end-to-end detection model which can automatically acquire the feature expression capabilities is still the direction of future development.

There are still some deficiencies in this study and areas that can be improved. Firstly, the number of samples in deep learning has a greater impact on the model training results. The data set used in the experiment is relatively small. Doing more experiments with a larger data set is required to improve the accuracy of the model. Secondly, the feature extraction using the static source code of the Android application will generate an input vector with an excessively large number of dimensions, which cannot guarantee the complete validity of the information therein. The selection and filtering of static features need to be improved.

Data Availability

The APK samples data used to support the findings of this study are available from the corresponding author upon request, and the website is https://virusshare.com/.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

References

[1] M. Chua and V. Balachandran, "Effectiveness of android obfuscation on evading anti-malware," in *Proceedings of the 8th ACM Conference on Data and Application Security and Privacy*, pp. 143–145, Tempe, AZ, USA, March 2018.

[2] M. A. Jerlin and K. Marimuthu, "A new malware detection system using machine learning techniques for API call sequences," *Journal of Applied Security Research*, vol. 13, no. 1, pp. 45–62, 2018.

[3] G. Canfora, A. De Lorenzo, E. Medvet, F. Mercaldo, and A. C. Visaggio, "Effectiveness of opcode N-grams for detection of multifamily android malware," in *Proceedings of the Availability, Reliability and Security (ARES), 2015 10th International Conference on. IEEE*, pp. 333–340, 2015.

[4] B. Wang, Y. Yao, S. Shan et al., "Neural cleanse: identifying and mitigating backdoor attacks in neural networks," in *Proceedings of the 2019 IEEE Symposium on Security and Privacy*, pp. 707–723, SP), San Francisco, CA, USA, May 2019.

[5] 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.

[6] W. Martin, F. Sarro, Y. Jia, Y. Zhang, and M. Harman, "A survey of app store analysis for software engineering," *IEEE Transactions on Software Engineering*, vol. 43, no. 9, pp. 817–847, 2017.

[7] F. Ahmad, N. Badrul, A. Karim, and M. Faizal, "Discovering optimal features using static analysis and a genetic search based method for android malware detection," *Frontiers of Information Technology & Electronic Engineering*, vol. 19, no. 6, pp. 712–736, 2018.

[8] T. G. Kim, B. J. Kang, M. Rho, S. Sakir, and I. Eul Gyu, "A multimodal deep learning method for android malware detection using various features," *IEEE Transactions on Information Forensics and Security*, vol. 14, no. 3, pp. 773–788, 2019.

[9] M. Nauman, T. A. Tanveer, S. Khan, and T. Ali Syed, "Deep neural architectures for large scale android malware analysis," *Cluster Computing*, vol. 21, no. 1, pp. 569–588, 2018.

[10] N. V. Duc and P. T. Giang, "NADM: neural network for android detection malware," in *Proceedings of the 9th International Symposium on Information and Communication Technology*, pp. 449–455, Danang City, Vietnam, December 2018.

[11] W. Wang, M. Zhao, and J. Wang, "Effective android malware detection with a hybrid model based on deep autoencoder and convolutional neural network [J]," *Journal of Ambient Intelligence Human Computing*, vol. 10, no. 8, pp. 3035–3043, 2019.

[12] D. Kirat and G. Vigna, "MalGene: automatic extraction of malware analysis evasion signature," in *Proceedings of the ACM Conference on Computer and Communications Security*, pp. 769–780, ACM Press, Denver, CO, USA, October 2015.

[13] G. Canfora, F. Mercaldo, E. Medvet, and C. A. Visaggio, "Detecting android malware using sequences of system calls," in *Proceedings of the 3rd International Workshop on Software Development Lifecycle for Mobile, DeMobile2015—Proceedings*, pp. 13–20, ACM Press, Bergamo, Italy, May 2015.

[14] X. Xiao, S. Zhang, F. Mercaldo, G. Hu, and A. K. Sangaiah, "Android malware detection based on system call sequences and LSTM," *Multimedia Tools and Applications*, vol. 78, no. 4, pp. 3979–3999, 2019.

[15] S. Hou, A. Saas, L. Chen, and Y. Ye, "Deep4MalDroid:a deep learning framework for android malware detecti detection based on linux kernel system call graphs," in *Proceedings of the 2016 IEEE/WIC/ACM International Conference on Web Intelligence Workshops*, pp. 104–111, Omaha, NE, USA, October 2016.

[16] X. Xiao, Z. Wang, Q. Li, Y. Jiang, and S. Xia, "Back-propagation neural network on Markov chains from system call sequences: a new approach for detecting Android malware with system call sequences," *IET Information Security*, vol. 11, no. 1, pp. 8–15, 2017.

[17] A. Abderrahmane, G. Adnane, C. Yacine, and G. Khireddine, "Android malware detection based on system calls analysis and CNN classification," in *Proceedings of the 2019 IEEE Wireless Communications and Networking Conference Workshop (WCNCW)*, pp. 1–6, IEEE, Marrakech, Morocco, April 2019.

[18] H. Kang, J.-W. Jang, A. Mohaisen, and H. K. Kim, "Detecting and classifying android malware using static analysis along with creator information," *International Journal of Distributed Sensor Networks*, vol. 11, no. 6, Article ID 479174, 2015.

[19] A. Sharma and A. Doegar, "Permission-set based detection and analysis of android malware," *Springer Cyber Security*, vol. 729, no. 1, pp. 231–239, 2018.

[20] X. Li, J. Liu, Y. Huo, R. Zhang, and Y. Yao, "An android malware detection method based on android manifest file," in *Proceedings of the International Conference on Cloud Computing & Intelligence Systems*, pp. 239–243, IEEE, Beijing, China, August 2016.

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

[22] S. Sabhadiya, J. Barad, and J. Gheewala, "Android malware detection using deep learning," in *Proceedings of the 2019 3rd International Conference on Trends in Electronics and Informatics (ICOEI)*, pp. 1254–1260, Tirunelveli, India, April 2019.

[23] S. Hou, A. Saas, Y. Ye, and L. Chen, "DroidDelver: an android malware detection system using deep belief network based on API call blocks," in *Proceedings of the International Conference on Web-Age Information Management*, pp. 54-55, Nanchang, China, June 2016.

[24] N. Huang, M. Xu, N. Zheng, T. Qiao, and K. K. Choo, "Deep android malware classification with API-based feature graph," in *Proceedings of the 18th IEEE International Conference on Trust, Security and Privacy in Computing and Communication s/13th IEEE International Conference on Big Data Science and Engineering*, pp. 296–303, Rotorua, New Zealand, August 2019.

[25] M. Spreitzenbarth, T. Schreck, F. Echtler, A. Daniel, and H. Johannes, "Mobile-Sandbox: combining static and dynamic analysis with machine-learning techniques," *International Journal of Information Security*, vol. 14, no. 2, pp. 141–153, 2015.

[26] H. Cai, N. Meng, B. Ryder, and D. Yao, "DroidCat: effective android malware detection and categorization via app-level profiling," *IEEE Transactions on Information Forensics and Security*, vol. 14, no. 6, pp. 1455–1470, 2019.

[27] W. Han, J. Xue, Y. Wang, L. Huang, Z. Kong, and L. Mao, "MalDAE: detecting and explaining malware based on correlation and fusion of static and dynamic characteristics," *Computers & Security*, vol. 83, pp. 208–233, 2019.

[28] C. Zhou, C. Sun, Z. Liu, and C. M. Lau, "A C-LSTM neural network for text classification," *Computer Science*, vol. 1, no. 4, pp. 39–44, 2015.

[29] R. Pascanu, J. W. Stokes, H. Sanossian, M. Marinescu, and A. Thomas, "Malware classification with recurrent networks," in *Proceedings of the IEEE International Conference on Acoustics*, pp. 1916–1920, IEEE, South Brisbane, Australia, April 2015.

[30] B. Kolosnjaji, A. Zarras, G. Webster, and C. Eckert, "Deep learning for classification of malware system call sequences," in *Proceedings of the Australasian Joint Conference on Artificial Intelligence*, pp. 137–149, Bobart, Australia, December 2016.

[31] L. Caviglione, M. Gaggero, J.-F. Lalande, W. Mazurczyk, and M. Urbanski, "Seeing the unseen: revealing mobile malware hidden communications via energy consumption and artificial intelligence," *IEEE Transactions on Information Forensics and Security*, vol. 11, no. 4, pp. 799–810, 2017.

[32] J. Chen, L. I. Weihua, J. I. Chen et al., "Bi-directional long short-term memory neural networks for Chinese word segmentation," *Journal of Chinese Information Processing*, vol. 32, no. 2, pp. 29–37, 2018.

[33] X. U. Shaoping, Z. Guizhen, L. I. Chongxi, L. Tingyun, and T. Yiling, "A fast random-valued impulse noise detection algorithm based on deep belief network," *Journal of Electronics and Information Technology*, vol. 41, no. 5, pp. 1130–1136, 2019.

[34] Q. Cai, Y. Pan, T. Yao, C. Yan, and T. Mei, "Memory matching networks for one-shot image recognition," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern*

Recognition, pp. 4080–4088, Salt Lake City, UT, USA, April 2018.

[35] N. Mclaughlin, D. Adam, G. J. Ahn et al., "Deep android malware detection," in *Proceedings of the ACM on Conference on Data & Application Security & Privacy*, March 2017.

[36] C. Wang, W.-d. Qiu, P. Tang et al., "Android malware detection based on CNN and LSTM," *Communications Technology*, vol. 51, no. 9, pp. 2209–2214, 2018.