Hindawi
Security and Communication Networks
Volume 2021, Article ID 6695858, 10 pages
https://doi.org/10.1155/2021/6695858

Research Article

Calibrating Network Traffic with One-Dimensional Convolutional Neural Network with Autoencoder and Independent Recurrent Neural Network for Mobile Malware Detection

Songjie Wei, Zedong Zhang, Shasha Li, and Pengfei Jiang

School of Computer Science & Engineering, Nanjing University of Science and Technology, Nanjing 210094, China

Correspondence should be addressed to Songjie Wei; swei@njust.edu.cn

Received 30 December 2020; Revised 3 February 2021; Accepted 20 February 2021; Published 27 February 2021

Academic Editor: Liguo Zhang

In response to the surging challenge in the number and types of mobile malware targeting smart devices and their sophistication in malicious behavior camouflage, we propose to compose a traffic behavior modeling method based on one-dimensional convolutional neural network with autoencoder and independent recurrent neural network (1DCAE-IndRNN) for mobile malware detection. The design solves the problem that most existing approaches for mobile malware traffic detection struggle with capturing the network traffic dynamics and the sequential characteristics of anomalies in the traffic. We reconstruct and apply the one-dimensional convolutional neural network to extract local features from multiple network flows. The autoencoder is applied to digest the principal traffic features from the neural network and is integrated into the independent recurrent neural network construction to highlight the sequential relationship between the highly significant features. In addition, the *Softmax* function with the *LReLU* activation function is adjusted and embedded to the neurons of the independent recurrent neural network to effectively alleviate the problem of unstable training. We conduct a series of experiments to evaluate the effectiveness of the proposed method and its performance for the 1DCAE-IndRNN-integrated detection procedure. The detection results of the public Android malware dataset CICAndMal2017 show that the proposed method achieves up to 98% detection accuracy and recall rates with clear advantages over other benchmark methods.

1. Introduction

With the rapid development of mobile cellular networks and the prevailing of smart devices, mobile applications have become more indispensable to people's routine life. Meanwhile, application users are confronted with increasingly serious security and privacy threats from mobile malware [1]. The coexistence of modern mobile devices with e-commerce, personal payment, social communication, and other related applications is so critical that we will never overestimate the importance of mobile application security insurance and malware detection.

Many mobile applications operate by interacting with other devices and remote servers on Internet connections. Particularly, when malware carries out malicious actions, it exposes obvious characteristics in network interaction traffic. Previous researches revealed that more than 90% of malware-infected mobile terminals are controlled by botnets controlling malware behavior through network or SMS instructions [2]. The discovery of such fact provides the possibility to explore malware detection methods based on network behavior characteristics as traffic dynamics. Malicious application detection based on network traffic dynamics is to recognize and model malware behaviors from the perspective of network traffic data, which has been recently thoroughly studied by both academia and industry. Malware detection systems based on traffic analysis can be deployed flexibly on network access points or clouds.

Malware detection method based on the statistical characteristics of network flows has the advantages of low

computation cost and avoidance in deep packet inspection and is applicable for both plain and encrypted traffic with privacy protection. However, mobile Internet is a very dynamic and complex network system, and its traffic statistical patterns demonstrate nonstationary, nonlinear, and super chaotic characteristics. When the network application environment or network scale varies, it is difficult to accurately and faithfully reflect the behavior of network traffic based on the characteristics of a single network flow. This is mainly because the accumulated information in network traffic may also change accordingly. Malware network behavior is usually a sequence of continuous behaviors. That is, when a malicious behavior appears in the flow from a source IP, the probability that the following behaviors continue the maliciousness is high, meaning that a network communication behavior has sequential forward and backward relevance. Therefore, most malicious traffic events follow a time-series pattern, and current network traffic can be classified according to the observed past network traffic records [3]. Previous experimental results also show that the sequential characteristics of network flow as one of the important feature dimensions in the traffic detection problem can effectively improve the performance of malicious traffic detection [4, 5].

This paper combines the strength of deep learning models and mobile malware traffic analysis with contributions in the following aspects:

(1) We first propose a mobile malware traffic detection method based on one-dimensional convolutional neural network with autoencoder and independent recurrent neural network (1DCAE-IndRNN) to capture the traffic dynamics and key features of mobile malware's Internet interaction. 1DCAE (one-dimensional convolutional neural network with autoencoder) is adapted to extract the local features of network flow data to ensure its temporal variability and sequence correlation, and the autoencoder helps recognize and concentrate on the efficient features in the traffic.

(2) Then IndRNN (independently recurrent neural network) is designed with fine-tuning to obtain the sequential correlations between high-level sequential-temporal features in traffic flows. It can digest valuable information for malware traffic detection. IndRNN is a revised type of recurrent neural network, which learns the long-term dependence of time pattern in large-scale sequence data. We add the *Softmax* function to IndRNN and use the *LReLU* function as an activation function, which alleviates the instability of IndRNN training to some extent.

(3) We conduct evaluation experiments on publicly available malware dataset and benchmark the proposed method with other typical detection approaches in the literature. The experimental results on the public Android malware dataset CICAndMal2017 show that the method proposed in this paper has higher detection accuracy and recall rate, which are superior to the traditional machine learning methods and the recent deep neural network models by others.

The rest of the paper is organized as follows. Section 2 surveys the recent work in related areas. Section 3 briefly introduces IndRNN. Section 4 explains the structure of the 1DCAE-IndRNN model proposed in this paper and the improvements to IndRNN. Section 5 presents the experimental results and analysis. At last, Section 6 concludes with further insight and discussion.

2. Related Work

Mobile malware detection is a critical part for building a mobile network defense system [6]. Various theoretical and practical methods for mobile malware detection are emerging recently. Dhaya and Poongodi [7] proposed a malware detection system based on N-gram algorithm. In their system, the behavioral characteristics of software are first obtained through static analysis. Then the N-gram algorithm is used for classification and identification, so that the category of the software is either benign or malignant. Khatri and Abendroth [8] developed a mobile network-based malware detection system to identify malicious activities in network and protect end users from mobile malware attacks. Nguyen and Yoo [9] built a malicious software detection system for SDN mobile devices based on network behaviors. The system consists of three algorithms: IP blacklist, connection success rate, and connection rate. It develops malicious software detection methods in a more effective and flexible way. However, the fundamental problem of these related works is that they cannot get rid of their dependence on feature engineering; that is, the quality of malware detection methods proposed depends largely on the quality of their feature extraction.

In order to alleviate the dependency problem on feature engineering, mobile malware detection methods based on deep learning have gradually attracted the attention of academia. Amalina et al. [10] proposed a solution for malware detection using machine learning to evaluate various network traffic characteristics. Shabtai et al. [11] proposed a new framework for Android malware detection, first monitoring features from Android devices and then using machine learning classifiers to determine whether the software is malware. Wang et al. [12] proposed a malicious software traffic classification method. The first 784 bytes of single data packets is transformed into a two-dimensional gray-scale image, and then the image is recognized by a convolutional neural network. This method does not need the manually extracted features but directly uses the original flow as input data for the classifier. They conducted experiments based on the self-created traffic dataset USTC-TFC2016, using three classifiers in two cases, and the final average accuracy can reach 99.41%. However, they use both packet header information and payload as evidence for classification, which are limitedly available in a dataset collected in the local

network. In a real network beyond LAN, packet header may change when traversing the network, which challenges the reuse of a previously trained model everywhere [13].

Obviously, an ideal traffic analysis and detection model for malware traffic behaviors is desirable without dependency on packet-content inspection or network environment restriction. While the monitored raw traffic from network is temporal and sequential with features available for extraction on different protocol layers within different time windows and while such features are all mixed up with possible obfuscation and anonymization, the underlying behavior dynamics originated from the application's designed functional and network interaction casualty can never be concealed or anonymized. Instead of recognizing malware by traffic feature appearance, this paper explores the possibility to reconstruct and reorganize the available features to build their interconnection spanning on the time scale, that is, the temporal-sequential correlations and their underlying application network dynamics.

3. IndRNN Basics

IndRNN evolves as a new recurrent neural network model by Li et al. [14]. IndRNN can effectively solve the problem of gradient disappearance and explosion by adjusting the time-based gradient backpropagation and can retain long-term memory and process long sequences. The definition of an IndRNN is

$$h_t = \sigma\left(W_{x_t} + \mu \odot h_{t-1} + b\right), \tag{1}$$

where $x_t \in \mathbb{R}^M$ and $h_t \in \mathbb{R}^N$ represent the input and hidden layer states of IndRNN at time t. $W_t \in \mathbb{R}^{M\times N}$ and $\mu \in \mathbb{R}^N$ represent the weights in the corresponding state, and $b \in \mathbb{R}^N$ is the neuron's bias terms. N and M are the number of neurons in the current layer and the length of the input sequence. σ represents the activation function of the neurons, and $\odot$ is the Hadamard product. For the n-th neuron at time t, the hidden state $h_{n,t}$ can be obtained by the following formula:

$$h_{n,t} = \sigma\left(W_n X_t + \mu_n \odot h_{n,t-1} + b_n\right), \tag{2}$$

where μ_n is the n-th neuron weight of the layer and $h_{n,t-1}$ stands for the previous hidden state. A neuron only receives information from the input and its own hidden state in the previous time step. Each neuron in layers is independent of others and independently processes a type of space-time pattern. The internal structure of IndRNN is shown in Figure 1.

Here, "Weight" represents the input weight processing, and "Recurrent" represents the probability of calculating the weight of each recurrent. "Recurrent + *ReLU*" means a recurrent process with *ReLU* (Rectified Linear Unit) as the activation function in each step. The *ReLU* activation function can reduce the disappearance of gradients between layers, so that multiple layers of IndRNN neural networks can be efficiently stacked. "Batchnorm" stands for batch normalization and is used before or after the activation function.

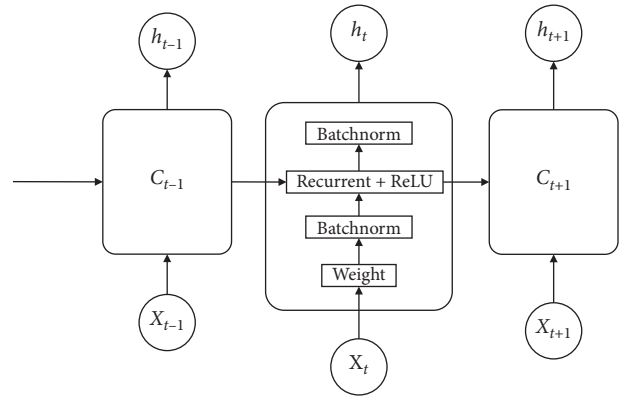

Figure 1: Structure diagram of independent recurrent neural network.

Since IndRNN can solve the problem of gradient explosion and disappearance over time, gradients are efficiently propagated between different steps. Therefore, IndRNN is usually deeper and longer than traditional RNN. IndRNN uses the *ReLU* function as the activation function, and the length of the memory obtained by learning different features is not the same. This greatly enhances the interpretability of the model, and the robustness of the model after training is also improved.

4. 1DCAE-IndRNN Model Design and Improvement

Network traffic is essentially sequential data with temporal features, being observed as a single one-dimensional byte flow organized in a hierarchical structure. Among the flow, bytes, packets, sessions, and their combinations are analogous to characters, words, sentences, and the whole article in natural language processing. A typical task of natural language classification usually extracts two levels of features. One is the features between words in a single sentence, which can identify the semantics expressed in the sentence by learning such features. The other is the relationship features between different sentences in a paragraph and then the overall semantics of the whole paragraph in the context. Inspired by this idea, we propose to construct a 1DCAE-IndRNN model to learn local and sequential features of network flows. 1DCAE can extract short-time and frequency-domain features from time-domain signals. It has achieved great success in the field of audio generation and machine translation, being very effective for simple tasks such as sequence classification and time-series prediction and with a faster training speed. IndRNN can learn the traffic sequence information of several previous time points and feed it back to the traffic state of the current time point to realize the learning of traffic sequence characteristics.

The 1DCAE-IndRNN malware traffic detection model, which can learn the correlation and local characteristics between network flow sequences, has a multilayer system structure, including separate input layer, hidden layers, and output layer. The overall 1DCAE-IndRNN network structure designed is shown in Figure 2. After inputting

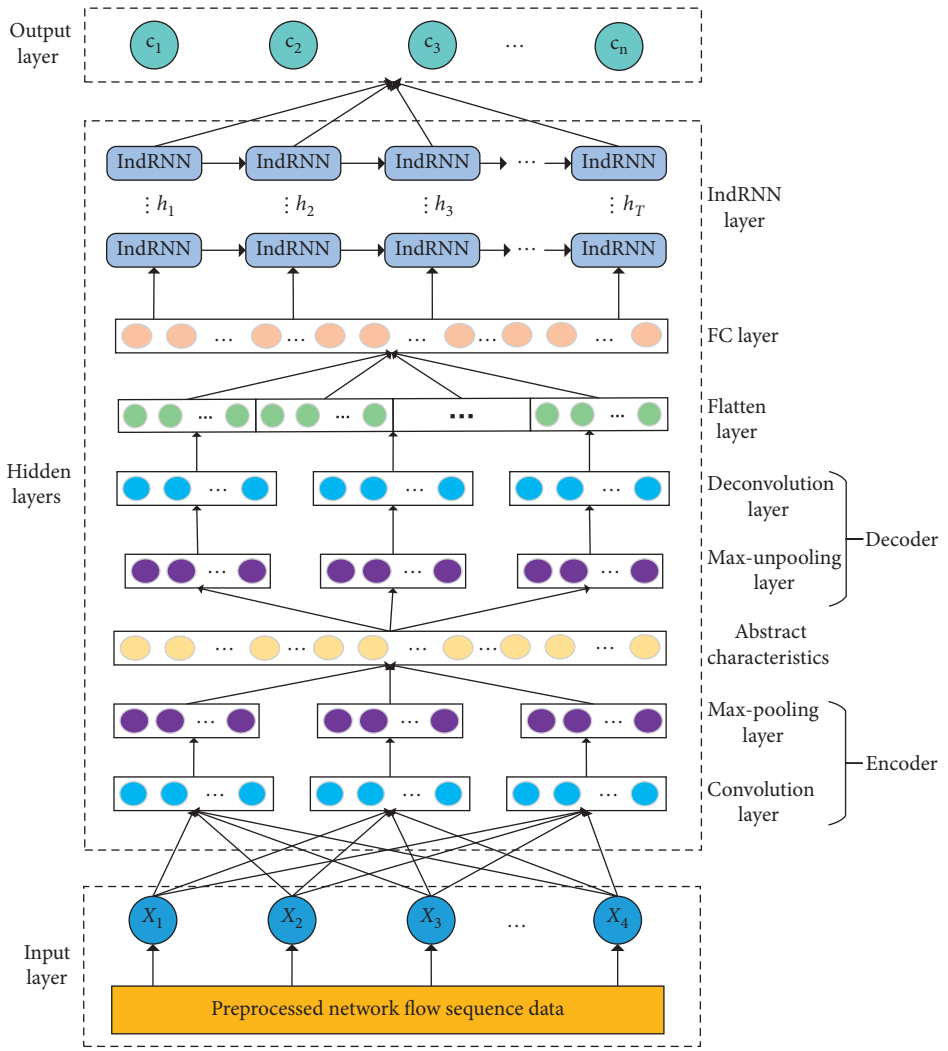

Figure 2: 1DCAE-IndRNN model structure diagram for malware traffic detection.

the traffic data with a sequential relationship into the model, the output layer finally gets the prediction result using the first T time points of data after the hidden layer calculation.

4.1. Input Layer. The input layer is responsible for importing data, which is the network flow features after traffic preprocessing. The 1DCAE-IndRNN input layer requires that the input data is three-dimensional, containing batch size, input dimension, and time steps. Batch size refers to the number of samples fed into the neural network at one time. The time step represents the time point length of the model memory. The input dimension represents the length inherent in a single time dimension of a single sequence of input data.

4.2. Hidden Layers. The hidden layers are the core part of learning and correlating the local and sequential features in network flow data. It consists of 1DCAE layer with the decoder (including convolution layer; and maxpooling layer), the encoder (including max-unpooling layer and deconvolution layer), flatten layer, IndRNN layer, and fully connected layer. Each IndRNN layer contains multiple IndRNN neuron nodes.

1DCNN mainly implements the function of local feature extraction by stacking convolution layer and pooling layer. The convolutional layer can be trained to obtain a set of optimal convolution kernels that meet the minimum loss function and use the convolution kernel to implement automatic feature extraction. The pooling layer can extract the most important features from the convolution layer and reduce the dimension in time dimension. The one-dimensional convolution kernel is equivalent to the observation window on the time axis of the network flow sequence. The convolution operation can extract short-term features on the time axis. The pooling layer further aggregates these short-term features and retains some of them. After the operations in the one-dimensional convolution layer and the pooling layer, the local characteristics of the network flow sequence are well analyzed and retained. The entire 1DCNN workflow is shown in Figure 3. The red and blue rectangles in the figure represent one-dimensional convolution kernels.

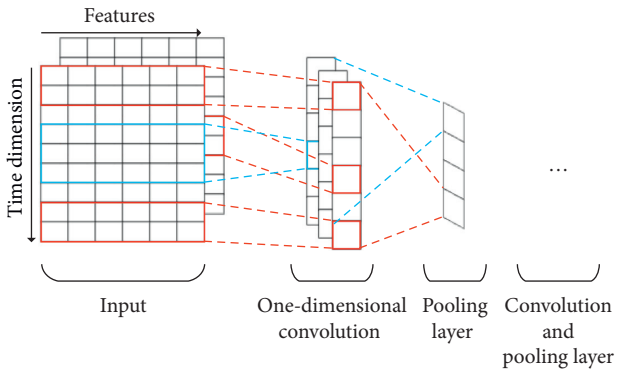

Figure 3: Workflow of one-dimensional convolutional layer.

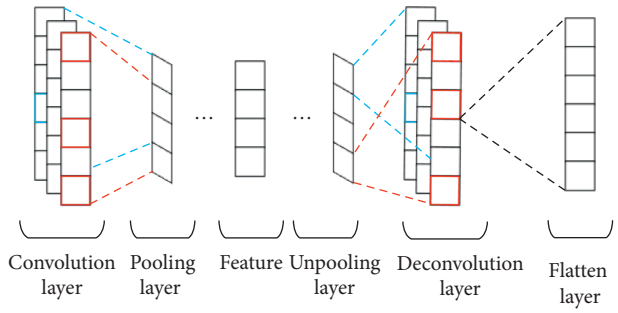

Figure 4: Workflow of encoder and decoder layer.

The encoder performs a convolution operation on the input layer data to extract local features based on 1DCNN. The decoder is to perform a deconvolution operation on the hidden layer data, so that the output data are restored to the dimension of the input data. Then the neural network minimizes the error between input and output by iterating repeatedly. The entire encoder workflow is shown in Figure 4.

Flatten layer is used for the transition from the 1DCAE layer to the IndRNN layer; that is, multidimensional input is one-dimensionalized. After the 1DCAE layer, a fully connected layer is needed to map the high-dimensional features to the low-dimensional ones and retain the useful information.

In IndRNN, we use the *ReLU* activation function to easily implement efficient stacking of multilayer IndRNN networks. However, since the *ReLU* function can output 0, the whole history information of neurons may be discarded and has to be relearned from the current time. Therefore, the stability of IndRNN in the training process is insufficient, and the prediction results are very unstable. Due to the temporal and dynamic characteristics of network traffic, it is difficult for the model to learn the rules and capture useful mutation information.

To alleviate the above problem, we put the recurrent weight information of IndRNN into the *Softmax* function and replaced the *ReLU* function with the *LReLU* function. The general meaning of the *Softmax* function is to normalize the vector, highlighting the maximum value and suppressing other components far below the maximum value. For the n-th neuron at time t, the hidden state $h_{n,t}$ becomes

$$h_{n,t} = \sigma\left(W_n X_t + \frac{e^{\mu_n \odot h_{n,t-1}}}{\sum_{i=1}^{T} e^{\mu_n \odot h_{n,t-1}}} + b_n\right). \tag{3}$$

Softmax function calculates the probability of each repeated weight, which affects the size of its loss function and ensures the effective propagation of gradient information. The *LReLU* function has a small slope for the negative value input, and the calculation method is shown in equation (4).

$$y_i = \begin{cases} x_i, & x_i \geq 0, \\ a_i x_i, & x_i < 0, \end{cases} \tag{4}$$

where a_i is a fixed parameter in the (0, 1). Since the derivative of the *LReLU* function is always not zero, it can reduce the occurrence of "dead" neurons, which effectively solves the problem that neurons do not learn when the *ReLU* function enters the negative range. In this way, it can be determined which weight in the recursive weight matrix has a larger probability value. If the weighted probability is large, the weighted probability is retained, otherwise suppressed. The improved IndRNN not only evaluates the impact of the previous time step on the current time step with more accuracy but also reduces the training complexity of the model. The internal structure of the improved IndRNN is shown in Figure 5.

4.3. Output Layer. The output layer consists of a fully connected layer, and the input is the output of the last hidden layer. The number of nodes in the output layer is the number of categories in the training set. After the multiclassification function *Softmax* is processed, the final classification result is obtained.

5. Experiments and Evaluation

5.1. Experimental Environment. All the following experiments are conducted on a desktop with 16 GB of RAM, Intel Core i7-7700 3.6 GHz CPU, and Nvidia GeForce GTX 1660Ti graphics card. Python 3.5 is used for programming. 1DCAE-IndRNN model is built on Keras (2.1.0), an open-source deep learning framework with TensorFlow (1.12.0) as the back end, with GPU acceleration. Benchmark algorithms are implemented based on the machine learning framework scikit-learn (0.18.0) and the deep learning ones on Keras.

5.2. Dataset Introduction and Preprocessing. CICAndMal2017 [15] is a new Android malware dataset proposed by the Canadian Institute for Cybersecurity. The collectors ran both the malware and benign applications on real smartphones to avoid runtime behavior modification of advanced malware samples that can detect the emulator environment. They installed 5,000 of the collected samples (426 malware and 5,065 benign) on real devices. The dataset includes benign and malicious application samples, which are divided into four malware categories, including adware, ransomware, scareware, and SMS malware. Malicious samples come from 42 unique malware families.

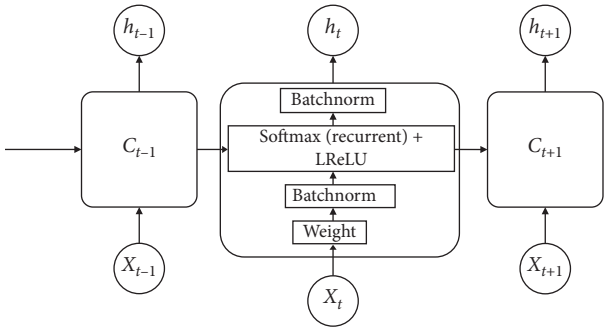

Figure 5: Internal structure of the improved IndRNN.

CICAndMal2017 dataset is fully labeled and includes network traffic, logs, API/SYS calls, phone statistics, and memory dumps of 42 malware families, with 84 features extracted during all the three execution states (installation, before restart, and after restart). The dataset contains more than 1.8 million pieces of data, covering most of the dynamic features of Android software. This dataset is suitable for validating the effectiveness of dynamics-based malware detection. In order to accurately evaluate the proposed 1DCAE-IndRNN's ability to recognize mobile malware traffic, this paper conducts experiments as binary classification tasks of benign and malicious applications.

In order to facilitate the calculation and analysis of data by the model, we perform the following preprocessing steps on the CICAndMal2017 dataset. Firstly, we exclude nonnumeric features, including Flow ID, Source IP, Source Port, Destination IP, Destination Port, and Timestamp. These nonnumeric features cannot be quantitatively modeled by the neural network and are irrelevant to maliciousness. Secondly, we select all the statistical characteristics of network traffic, totaling 68, some of which are shown in Table 1. We use 0 to fill in missing value fields. Finally, we use the min–max normalization method to reshape the data. The linear transformation of the original data by min–max normalization makes the result fall in 0~1. One-third of the total data is used as the test set and the rest for training.

5.3. Malware Detection Experiment and Analysis. In order to train a 1DCAE-IndRNN model structure with better performance, this experiment builds a one-dimensional convolutional layer containing 1 to 3 layers and an IndRNN layer containing 1 to 4 layers based on the model designed in Section 4. There are 12 different 1DCAE-IndRNN models. The filling mode used on the boundary of the one-dimensional convolution layer is "valid," and the sliding steps are all 1. The number of neuron nodes in the fully connected layer between the 1DCAE layer and the IndRNN layer is set to 64. The slope of the *LReLU* function in the negative interval is set to 0.05. The specific structural parameters of the 1DCAE layer and the IndRNN layer are shown in Tables 2 and 3.

Here, Conv1D (x, y) and deConv1D (x, y) indicate that the convolution kernel size is x and the number of convolution kernels is y. Maxpooling (2) and unMaxpooling (2) represent the max-pooling layer window size of 2. IndRNN (n) indicates that there are n IndRNN neuron nodes.

Although the robustness of IndRNN after training is stably good, due to the strong learning ability of the neural network model itself, overfitting may still occur. Therefore, a dropout layer is added after each IndRNN layer to prevent overfitting, and the neuron dropout rate of the dropout layer is set to 0.5.

In order to reduce the computational cost and randomness of model training, our experiment uses the minibatch training method, and the batch size of this experiment is set to 128. The training loss function of the model is the cross-entropy loss function. The optimizer uses the Adam method. The initial learning rate is 0.0005 and the number of training rounds is 150. Due to the small scale of the datasets, this experiment configures the time step of 1DCAE-IndRNN to 10, where 1DCAE-IndRNN can remember the traffic sequence information of the previous 10 time periods. The model extracts the local features of network flow data through 1DCAE layer, uses IndRNN layer to learn the sequential association information between high-level features, and finally outputs the prediction category through a fully connected layer.

The designed 1DCAE-IndRNN models with 12 different structures are each trained 3 times, and the average accuracy of each structure is obtained based on the results of the 3 experiments. The output results with different structures are shown in Table 4, where the structure indexes correspond to those in Tables 2 and 3.

From the experimental results in Table 4, we can notice that the classification accuracy of 1DCAE-IndRNN models with different structures has some differences. The accuracy of most models is more than 90%. When the number of 1DCAE layers is 2 and the number of IndRNN layers is 3, the classification accuracy of the models is the highest, up to 98.32%. With a small 1DCAE layer, the classification performance of the model can be improved by increasing the number of IndRNN layers, although such increase may not continue infinitely. For example, when the number of 1DCAE layers is 2, with the number of IndRNN layers being 4, the classification accuracy of the model declines. But there is no significant decline, which is inseparable from the advantages and robustness of the gradient in the training of the IndRNN model. When the number of 1DCAE layers is 1, adding another 1DCAE layer, the classification accuracy of the model has been greatly improved, indicating that the 1DCAE layer is very effective in extracting local features of network flow sequence data, and two 1DCAE layers are suitable for the model. The experimental data in this paper have fewer feature dimensions. Complicating the model may not improve the detection performance of the model but will definitely increase the training and test time overhead for the model. Therefore, we select the optimal hidden layer structure (2 1DCAE layers; 3 IndRNN layers) and conducts 3 experiments again. Accuracy, recall, F1-score, and training time are calculated as evaluation metrics. The experimental results are shown in Table 5. It can be seen that the performance of the 1DCAE-IndRNN model is still very stable, and the training time is significantly elongated.

Table 1: Partially selected features.

Feature	Description
Fwd_Packet_Length_Std	Standard deviation of the size of packet in forward direction
Init_Win_Bytes_Forward	Total number of bytes sent in the initial window in the forward direction
Init_Win_Bytes_Backward	Total number of bytes sent in the initial window in the backward direction
Bwd_Packet_Length_Min	Minimum of the size of packet in backward direction
Total_Length_BwdPackets	Total size of the packet in backward direction
Flow_IAT_Max	Maximum interarrival time of the packet

Table 2: Structural parameters of different designs for 1DCAE layer.

Layers	Structural parameters
1	Conv1D (20, 8), maxpooling (2) unMaxpooling (2), deConv1D (20, 8)
2	Conv1D (20, 8), maxpooling (2), Conv1D (30, 4) deConv1D (30, 4), unMaxpooling (2), deConv1D (20, 8)
3	Conv1D (20, 8), maxpooling (2), Conv1D (30, 4), maxpooling (2) Conv1D (40, 2), maxpooling (2) unMaxpooling (2), deConv1D (40, 2), unMaxpooling (2), deConv1D (30, 4), unMaxpooling (2), deConv1D (20, 8)

Table 3: Structural parameters of different designs for IndRNN layer.

IndRNN layers	Structural parameters
1	IndRNN (20)
2	IndRNN (30), IndRNN (30)
3	IndRNN (30), IndRNN (40), IndRNN (30)
4	IndRNN (30), IndRNN (40), IndRNN (40), IndRNN (30)

Table 4: Test set accuracy of 1DCAE-IndRNN models with different hidden layer structures.

IndRNN1DCAE	1 (%)	2	3	4 (%)
1	86.21	87.16%	91.79%	91.23
2	91.32	95.74%	98.32%	97.32
3	93.29	96.48%	97.11%	95.23

Table 5: Experimental results of the optimal 1DCAE-IndRNN model.

Number	Accuracy (%)	Recall (%)	F1-score	Time (min)
1	98.32	98.32	98.31	43.6
2	98.27	98.26	98.26	41.5
3	98.29	98.27	98.27	42.4

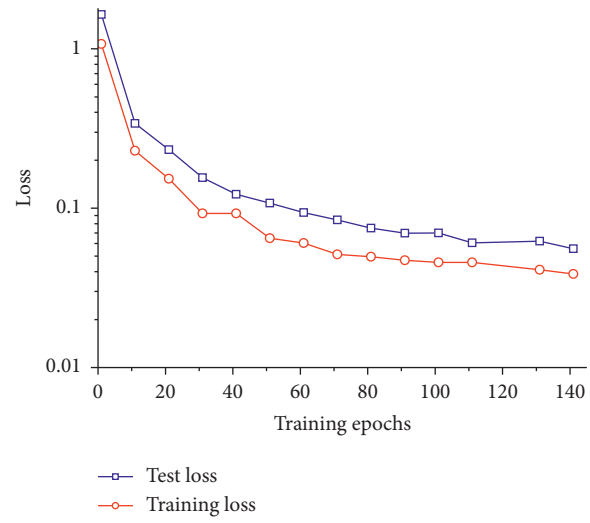

Figure 6: Optimal 1DCAE-IndRNN model loss value curves.

Figure 6 shows the loss curve of the optimal 1DCAE-IndRNN model in training and test. It can be seen that, in the first 40 rounds of training, the loss value of the model decreases rapidly, which shows that the convergence speed of the model is fast. After 80 rounds of training, the training set loss of the model has become stable. During the whole training process, the training loss value and test loss value of the model do not fluctuate greatly, indicating that the training stability of the model is ensured. The improvement of IndRNN effectively alleviates the problem of model training instability. Therefore, the 1DCAE-IndRNN model proposed in this paper is superior in terms of convergence speed, training stability, and classification performance.

In addition, we investigate the influence of the number of training data on the classification performance of the 1DCAE-IndRNN model. We use different numbers of training sets to train the model, and each trained model is evaluated on the complete test set. It is worth mentioning

that the proportion of various types of traffic in the training set is unchanged. The value range of the training data ratio is [0.1, 0.2, 0.4, 0.6, 0.8, 1.0]. Figure 7 shows the test set accuracy curve when training with different scale training sets.

According to the experimental results in Figure 6, when the proportion of training data is only 0.1, the accuracy of the test set is low due to insufficient features extracted by the model, and the fluctuation is obvious during training, and the convergence speed is also slow. When the amount of training data is continuously increased, more and more features are extracted by the model, where the final classification accuracy of the model is rising, and the convergence speed is continuously accelerated with a more stable training process. Therefore, although smaller training dataset can achieve 92.89% accuracy with less training time, the training process is prone to instability, and the model convergence speed is slow. More training data tends to make the detection performance of the model better with a slight increase in training time. Under the premise of ensuring the accuracy of model detection, the amount of training data needs to be controlled within a certain range to obtain the most suitable training time and classification performance.

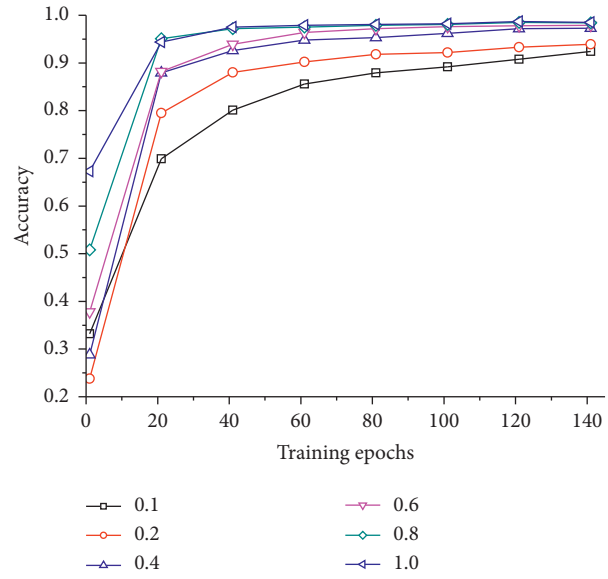

Figure 7: Test set accuracy curves with different training set sizes.

5.4. Comparison with Other Methods. In order to verify the advantages of the 1DCAE-IndRNN method proposed in this paper for malware traffic detection, two classical machine learning methods and three deep learning methods are selected from the literature for experimental comparison. Classical machine learning methods include random forest (RF) [16] and XGBoost [17]. Deep learning methods include deep neural networks (DNN) [18], recurrent neural networks (RNN) [19], and long short-term memory (LSTM) [20]. The application of these benchmark methods is briefly surveyed in the related work section. The time steps for LSTM and RNN are set to 10. Figure 8 shows the classification accuracy, recall rate, and F1-score of the five comparison classification models and 1DCAE-IndRNN on the test set.

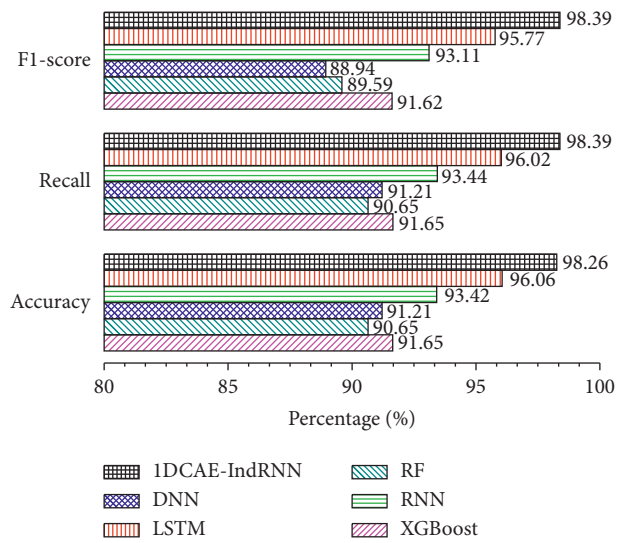

Figure 8: Comparison of classification performance with other methods.

According to the experimental results in Figure 8, the classification performance of the 1DCAE-IndRNN-based malware traffic detection method proposed in this paper is superior to all the comparison algorithms. Because XGBoost, DNN, and random forest models do not have the time memory function, they cannot take advantage of the sequential characteristics between network flows. The classification accuracy of these three models is lower than that of RNN and LSTM, and the F1-score is also poorer. RNN and LSTM are both models utilizing the time sequence characteristics between network flows, and the classification accuracies reach over 93%. However, they do not have 1DCAE to extract the local feature information of network flow sequence. Moreover, when the time step of RNN is a little longer, information loss is likely to occur. Therefore, the classification performance of RNN and LSTM is lower than that of the 1DCAE-IndRNN model.

For a real network environment, the time efficiency of the applied malware traffic detection model is not ignorable. The shorter the online identification execution time is, the better it is to take defensive measures in real time. The shorter the model training time is, the faster it is to update the model parameters. Therefore, this paper compares the training time of a round of deep learning model (DNN, RNN, LSTM, and 1DCAE-IndRNN) on the same dataset and the test time of single data after the model training. The experimental results are concluded in Figures 9 and 10, respectively. It is worth mentioning that these deep learning models use the same GPU acceleration hardware during training.

Since the convolution and pooling operation of 1DCAE greatly reduces the number of characteristic parameters and the number of IndRNN parameters is relatively small, the one-round training time of the 1DCAE-IndRNN model proposed is 31 seconds, and the test time of single data on the trained model is only 74 milliseconds, showing high training and detection efficiency. DNN does need to learn the

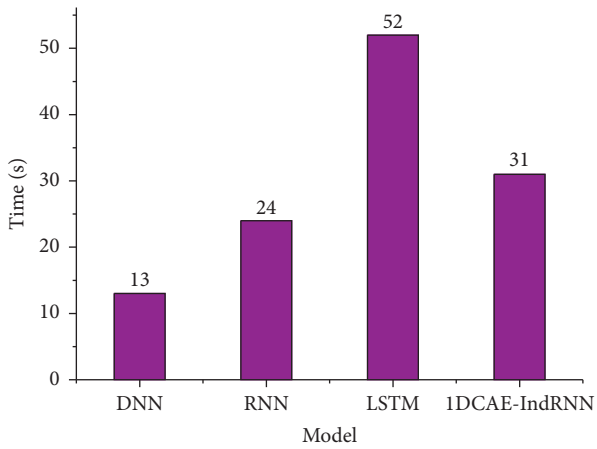

Figure 9: Single round training time of different neural network models.

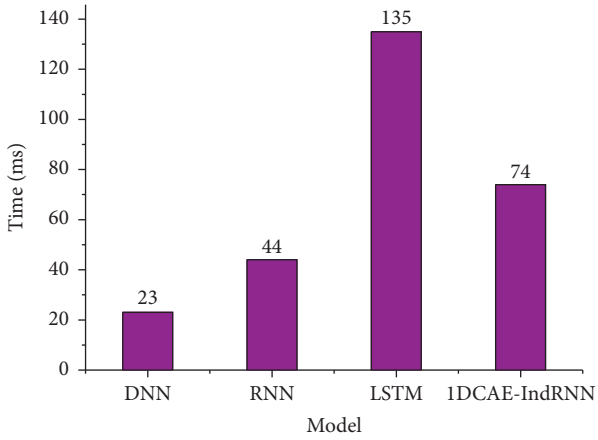

Figure 10: Test time of single data with different neural network models.

sequential information of multiple network flows, so the single round training time and single data test time are the smallest. Since the LSTM needs three gates to control the learning of the model, the parameters of the model itself are very large, which leads to a longer time for the parameter updating in the LSTM model. The number of parameters in RNN is less than that in the 1DCAE-IndRNN model, so the training time and test time of a single round are proportionally lower than the 1DCAE-IndRNN model.

6. Conclusions

This paper proposes a mobile malware traffic detection method based on 1DCAE-IndRNN, which aims to solve the problem that the current mobile malware traffic detection algorithm is struggling with capturing the dynamic changes and key temporally local information in abnormal traffic flows from mobile applications. In our design of 1DCAE-IndRNN, 1DCAE can extract local features of multiple network flows, and the independent recurrent neural network obtains the sequential relations between high-level features. In addition, activation functions are added to the IndRNN neurons to effectively alleviate the instability in the training procedure. We utilize the CICAndMal2017 dataset to conduct a series of simulation experiments and study the impact of the model structure variation and the training data volume difference on the detection performance. Experimental results show that the method proposed in this paper has higher detection accuracy, recall, and F1-score, which is superior to the common classic machine learning methods and the recently proposed deep neural network models. We demonstrate that, by capturing and characterizing the dynamic features of network traffic from the malicious application software, it is promising to differentiate the network behaviors between benign and malicious applications. And behavior-based classification has more insight into malware maliciousness than code appearance. In future work, we plan to further investigate how to generalize such network traffic modeling on behavior dynamics on malware family level. That means using the proposed method to not only separate benign from malware but also cluster and group those malicious ones for code heritage and attack originality.

Data Availability

CICAndMal2017 is a new Android malware dataset provided by the Canadian Institute for Cybersecurity. Lashkari, Arash Habibi et al. "Toward Developing a Systematic Approach to Generate Benchmark Android Malware Datasets and Classification." 2018 International Carnahan Conference on Security Technology (ICCST). IEEE, 2018.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This work was supported in part by the National Natural Science Foundation of China under Grants nos. 61802186 and 61472189.

References

[1] D. He, S. Chan, and M. Guizani, "Mobile application security: malware threats and defenses," *IEEE Wireless Communications*, vol. 22, no. 1, pp. 138–144, 2015.

[2] Y. Zhou and X. Jiang, "Dissecting android malware: characterization and evolution," in *Proceedings of IEEE Symposium on Security and Privacy Conference*, vol. 95-109, San Francisco, CA, USA, May 2012.

[3] R. Vinayakumar, K. Soman, and P. Poornachandran, "Applying convolutional neural network for network intrusion detection," in *Proceedings of International Conference on Advances in Computing, Communications and Informatics*, vol. 1222-1228, IEEE, Udupi, India, September 2017.

[4] W. Wang, Y. Sheng, J. Wang et al., "HAST-IDS: learning hierarchical spatial-temporal features using deep neural networks to improve intrusion detection," *IEEE Access*, vol. 6, pp. 1792–1806, 2017.

[5] Z. Zou, J. Ge, H. Zheng et al., "Encrypted traffic classification with A convolutional long short-term memory neural network," in *Proceedings of International Conference on High*

Performance Computing and Communications, IEEE, Fiji, Oceania, December 2018.

[6] M. Mimura and H. Tanaka, "Long-term performance of a generic intrusion detection method using Doc2vec," in *Proceedings of International Symposium on Computing & Networking*, IEEE, Silicon Valley, CA, USA, January 2017.

[7] R. Dhaya and M. Poongodi, "Detecting software vulnerabilities in android using static analysis," in *Proceedings of International Conference on Advanced Communications, Control and Computing Technologies*, IEEE, Ramanathapuram, India, May 2014.

[8] V. Khatri and J.. Abendroth, "Mobile guard demo: network based malware detection," in *Proceeding of Trustcom/BigDataSE/ISPA*, IEEE, Washington, DC, USA, May 2015.

[9] T.-H. Nguyen and M. Yoo, "A behavior-based mobile malware detection model in software-defined networking," in *Proceedings of International Conference on Information Science and Communications Technologies*, Moscow, Russia, September 2017.

[10] F. Amalina, A. Feizollah, and N. Badrul Anuar, "Evaluation of machine learning classifiers for mobile malware detection," *Soft Computing*, vol. 20, no. 1, pp. 343–357, 2014.

[11] A. Shabtai, U. Kanonov, Y. Elovici, C. Glezer, and Y. Weiss, "Andromaly": a behavioral malware detection framework for android devices," *Journal of Intelligent Information Systems*, vol. 38, no. 1, pp. 161–190, 2012.

[12] W. Wang, M. Zhu, X. Zeng et al., "Malware traffic classification using convolutional neural network for representation learning," in *Proceedings of International Conference on Information Networking*, IEEE, Shenzhen, China, March 2017.

[13] H.-k. Lim, J.-B. Kim, J.-S. Heo et al., "Packet-based network traffic classification using deep learning," in *Proceedings of International Conference on Artificial Intelligence in Information and Communication (ICAIIC)*, IEEE, Jeju Island, South Korea, April 2019.

[14] S. Li, W. Li, C. Cook et al., "Independently recurrent neural network (IndRNN): building A longer and deeper RNN," in *Proceedings of IEEE/CVF Conference on Computer Vision and Pattern Recognition*, Salt Lake City, UT, USA, June 2018.

[15] A. Lashkari, A. Kadir, L. Taheri et al., "Toward developing A systematic Approach to generate benchmark android malware datasets and classification," in *Proceedings of International Carnahan Conference on Security Technology (ICCST)*, IEEE, Montreal, Canada, October 2018.

[16] C. Wang, T. Xu, and X. Qin, "Network traffic classification with improved random forest," in *Proceedings of International Conference on Computational Intelligence and Security*, IEEE, Shenzhen, China, December 2015.

[17] Q. Chen, C. Guestrin, and X. Xgboost, "A scalable tree boosting system," in *Proceedings of ACM SIGKDD International Conference*, pp. 785–794, Anchorage, AK, USA, August 2016.

[18] W. Elmasry and A. Akhan Akbulut, "Halim zaimet, deep learning approaches for predictive masquerade detection," *Security and Communication Networks*, vol. 2018, Article ID 9327215, 24 pages, 2018.

[19] J. Du, V. Chi-Man, C. Philip Chen et al., "Novel efficient RNN and LSTM-like architectures: recurrent and gated broad learning systems and their applications for text classification," *IEEE Transactions on Cybernetics*, vol. 992 pages, 2020.

[20] C. H. Park and G.-H. Lee, "Jamming prediction for radar signals using machine learning methods," *Security and Communication Networks*, vol. 2020, Article ID 2151570, 9 pages, 2020.