IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received September 19, 2020, accepted September 27, 2020, date of publication October 2, 2020, date of current version October 14, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3028370

Review of Android Malware Detection Based on Deep Learning

ZHIQIANG WANG[1,2], **QIAN LIU**[1], **AND YAPING CHI**[1]
[1]Department of Cyberspace Security, Beijing Electronic Science and Technology Institute, Beijing 100071, China
[2]State Information Center, Beijing 100000, China

Corresponding author: Zhiqiang Wang (wangzq@besti.edu.cn)

This work was supported in part by the Key Lab of Information Network Security, Ministry of Public Security, under Grant C19614, in part by the Fundamental Research Funds for the Central Universities under Grant 328201910, and in part by the China Postdoctoral Science Foundation under Grant 2019M650606.

ABSTRACT At present, smartphones running the Android operating system have occupied the leading market share. However, due to the Android operating system's open-source nature, Android malware has increased dramatically. Malware can steal user privacy and even maliciously charge fees and steal funds. It has posed a severe threat to cyberspace security because traditional detection methods have many limitations. With the widespread application of deep learning in recent years, the method of detecting Android malware using deep learning has gradually attracted widespread attention from scholars at home and abroad. Although scholars have researched Android malware detection using deep learning, there is currently a lack of a detailed and comprehensive introduction to malware detection's latest research results based on deep learning. In order to solve this problem, this study analyzes and summarizes the latest research results by investigating a large number of the latest domestic and international academic papers, summarizing malware detection architecture and detection schemes, and analyzing existing problems and challenges. This review will help researchers better understand the research status and future research directions in this field.

INDEX TERMS Android, malware, deep learning, review.

I. INTRODUCTION

With the rapid development of the mobile Internet, the Android operating system has become the most widely used intelligent terminal operating system in the world due to its many advantages such as open-source, scalability, and convenience. However, because of Android's openness, many benign applications in the Android market hide much malware, and mobile malware has constituted a severe threat to cyberspace security. Specifically, while users enjoy the convenience brought by various Android applications, personal privacy information and essential data (such as payment account information and passwords) are also threatened continuously.

A research report released by the 360 Internet Security Center shows that in the third quarter of 2019, the 360 Internet Security Center intercepted approximately 365,000 new malicious program samples on the Android platform, an increase of 11,000 from the second quarter of 2019 (354,000). On average, about 44,000 new mobile phone malicious program samples are detected every day. Figure 1 shows the statistics of new and intercepted mobile malware in the third quarter of 2019.

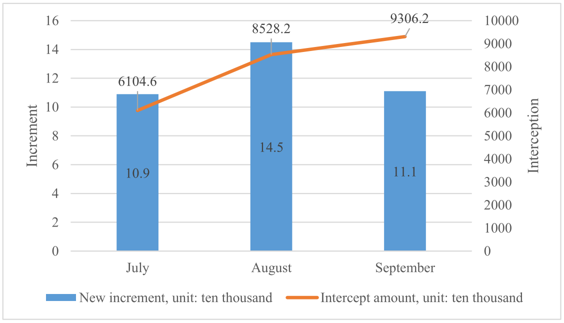

FIGURE 1. Increment and interception of mobile malware in the third quarter of 2019.

Besides, in the third quarter of 2019, new types of malicious programs on the Android platform were mainly expense consumption, accounting for up to 66.5%, followed by

The associate editor coordinating the review of this manuscript and approving it for publication was Yudong Zhang.

privacy theft (17.9%), remote control (10.6%), rogue behavior (3.9%), malicious charge (1.1%), as shown in Figure 2.

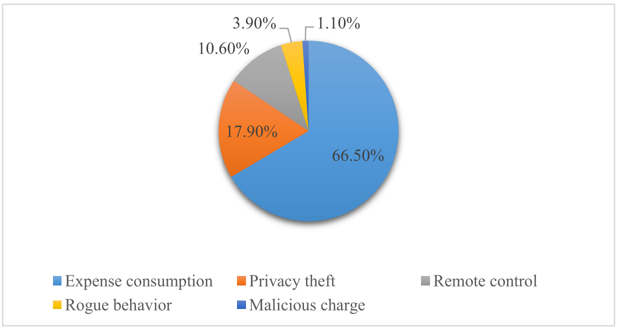

FIGURE 2. Distribution of new types of malware on mobile devices in the third quarter of 2019.

Traditional detection methods have many limitations. With the widespread application of deep learning in recent years, detecting Android malware using deep learning technologies has emerged, and the effectiveness of malware detection has been dramatically improved. In order to cope with the explosive growth of Android malware, much research has been conducted on the methods of Android malware detection using deep learning at home and abroad. Scholars have proposed various detection algorithms and solutions using various deep neural network models and have achieved many research results. However, there is currently no comprehensive review paper detailing these latest research findings. In order to deeply understand the principles, detection architecture, security and challenges, and future research development trends of Android malware detection using deep learning, and to grasp the new trends of domestic and foreign research, it is significant to describe and summarize Android malware detection based on deep learning technology.

The other chapters of this article are arranged as follows: Section II introduces the basic knowledge, Section III describes and analyzes the research progress of Android malware detection based on deep learning in detail, Section IV analyzes and discusses the research results, and Section V introduces the Android malware detection architecture and detection scheme. Section VI introduces the current problems and challenges, Section VII introduces future research directions, and finally the conclusions of this study are presented in Section VIII.

II. ANDROID MALWARE DETECTION

With the continuous upgrading of Android malware avoidance detection technology, the Android malware detection technology has also been continuously developed. The early detection methods based on signature and feature-matching cannot meet needs. Static analysis, dynamic analysis, and hybrid analysis using feature engineering alone could not effectively detect new malware. With artificial intelligence development, the detection method using traditional machine learning can not meet the needs, and the current detection method using deep learning has become a research hotspot.

TABLE 1. APK file structure.

File	Description
assets/	Static files in APK
lib/	Program-dependent Native library
res/	Application resources
META-INF/	Application's signing and certificates
AndroidManifest.xml	Application's global configuration file
classes.dex	DEX executable (Dalvik bytecode)
resources.arsc	The Binary resource configuration file

A. ANDROID APPLICATION

The Android application is programmed in Java, compiled with the Android SDK, and all data and resource files are packaged into an APK (Android Package) file. It is a compressed file with the extension.apk. The APK file containing all content of an Android application is a file used by the Android platform to install the application. APK is a zip compression package. Unpacking this APK package, there is the following structure:

assets: It is used to store static files that need to be packaged into the APK. The difference from res is that the assets directory supports subdirectories of any depth. Users can arbitrarily deploy the folder structure according to their needs, and the files in the res directory can generate the corresponding resource ID in the.R file. Assets cannot automatically generate the corresponding ID. The AssetManager class is required when accessing the assets directory.

lib: It stores the native library files on which the application depends. It is generally programmed in C/C++. The lib library may contain four different types. Depending on the CPU model, it can be roughly divided into an ARM, ARM-v7a, MIPS, X86 corresponding to the ARM architecture, ARM-V7 architecture, MIPS architecture, and X86 architecture, respectively.

res: res is the abbreviation of the resource. This directory stores resource files. All files stored in this folder will be mapped to the Android project's.R file, and the corresponding ID will be generated. When accessing, the resource ID is directly used as R.id.filename. The res folder can contain multiple folders, where anim directory storing animation files, drawable directory storing image resources, layout directory storing layout files, values directory storing some characteristic values, colors.xml storing color values, and dimens.xml defining the size value, string.xml defining the value of the string, and styles.xml defining the style object, the XML folder storing any XML file which can be read by Resources.getXML () at runtime. The raw can be copied directly to arbitrary files in the device, and they do not need to be compiled.

META-INF: It saves the signature information of the application. The signature information can verify the integrity of the APK file. Android SDK calculates all files' integrity in the APK package when packaging the APK and saves this integrity to the META-INF folder. When the application is installed, it first checks the integrity of the APK according to the META-INF folder, so that every file in the APK cannot be falsified. In this way, to ensure that APK applications are not maliciously modified or infected by viruses, it is beneficial to ensure the integrity of Android applications and the system's security. The files included in the META-INF directory are CERT.RSA, CERT.DSA, CERT.SF and MANIFEST.MF. Among them, CERT.RSA is a signature file that the developer uses to sign the APK with a private key. CERT.SF and MANIFEST.MF record the SHA-1 hash value of the files.

AndroidManifest.xml: It is a configuration file for Android applications. It is a configuration file used to describe the ''overall information'' of Android applications. In short, it is equivalent to a configuration file for the ''self-introduction'' of Android applications to Android systems. According to this ''self-introduction,'' the Android system can fully understand the APK application's information. Each Android application must include an AndroidManifest.xml file, and its name is fixed and cannot be modified. When developing Android applications, it is general to register each Activity, Service, Provider, and Receiver in the code into AndroidManifest.xml. Only then can the system start the corresponding components. Besides, this file also contains some permission declarations and SDK version information.

classes.dex: Traditional Java programs. First, the Java file is compiled into a class file. The byte code is stored in the class file, which Java virtual machine can execute through interpretation. The Dalvik virtual machine is optimized in the Java virtual machine and executes Dalvik bytecode, converted from Java bytecodes. Generally, it uses the dx tool in the Android SDK to convert Java bytecode to Dalvik bytecode when packaging Android applications. The dx tool can merge, reorganize, and optimize multiple class files, reducing the volume and shortening the running time.

resources.arsc: it records the mapping between resource files and resource IDs and finds resources based on resource IDs. The development of Android is divided into modules. The res directory is used to store resource files. When resource files need to be called in the code, it only needs to call findviewbyId () to get the resource files. Whenever putting a file into the res folder, the corresponding ID will be automatically generated by the Android Asset Packaging Tool and saved in the.R file. The.R file only guarantees that the compiler does not report an error. When the program is running, the system will look for the corresponding resource path based on the ID and the resources.arsc file is a file used to record the correspondence between these IDs and the resource file location.

B. ANDROID MALWARE DETECTION

1) STATIC ANALYSIS

Static analysis [1] focuses on analyzing the static information obtained without running the application, such as the executable file and source code. It is based on the decompilation technology to perform reverse analysis of the code. The advantage of static analysis is that the calculation overhead is relatively low, and the detection speed is fast. The disadvantage is that malware using obfuscation technology cannot be adequately analyzed by static analysis.

Android applications are released on the applications market in the form of APKs. In terms of static analysis, APKtool is used to decompile the APK files and parses the AndroidManifest.xml file to extract the permissions, package names, components, environment, and intent features. Figure 3 shows the permissions, four major components, and the environment features of the Android application. IDA Pro decompiles the APK file and parses the shared library.so file to obtain the shared library function opcode and ARM opcode features.

Android applications are developed using Java. The development environment (such as Eclipse) converts Java source code into Dalvik executable files (dex files). These files can run on Android's Dalvik virtual machine. Dex is a file format containing compiled code programmed for Android and can be interpreted by the Dalvik virtual machine but cannot be read. In order to convert the dex file to a readable format, smali provides readable code in the smali language. Smali code is the intermediate code for interpretation between Java and Dalvik virtual machines. It can use the baksmali tool to decompile the classes.dex file to obtain the smali file. Parsing the smali code can obtain the Dalvik opcode, API call, string, control flow graph, and data flow graph features.

In addition to parsing smali code for feature extraction, it can also access Java source code files to extract features such as API calls. It is decompressing the APK file to obtain the classes.dex file, it can use the dex2jar tool to convert the dex file into a jar file, and then use the JD-GUI tool to convert the jar file into a java source file.

2) DYNAMIC ANALYSIS

Dynamic analysis [2] is to run an application in a sandbox environment or on a real device. Monitoring the application's status when it is running collects information about the application's behavior and analyzes a series of data information (log, network traffic). Application behavior is typically monitored by accessing private data or using restricted API calls. Dynamic analysis can identify malicious behaviors that are not detected by static analysis methods. The advantage of dynamic analysis is that it can deal with malware using obfuscation techniques, but the disadvantages are that the analysis and detection time is long, and the calculation cost is high.

The dynamic analysis method collects behavior information when the Android application is running and transforms it into features. The dynamic analysis method uses system calls, file access information, network traffic information [3], encryption operation, service opening, telephone call [4], user interaction, system components [5], and other dynamic features resistant to obfuscation. Besides, most researchers collect malicious behavior information within the limited

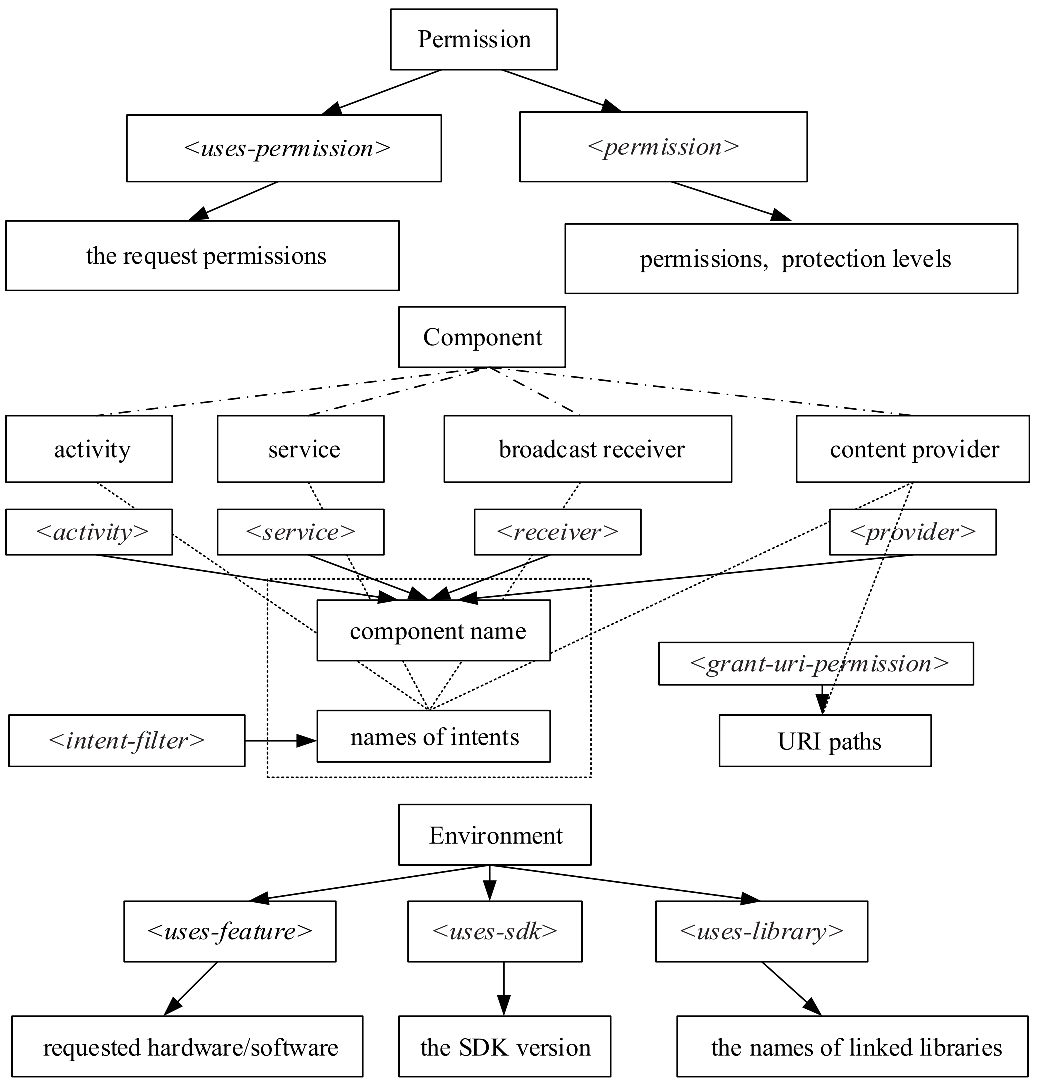

FIGURE 3. **Android application static features.**

running time of Android applications. Malicious software may not carry out malicious activities during this time, so dynamic analysis needs a long time to analyze software [6].

According to the technology used to track Android applications' behavior, the dynamic analysis method is divided into the hook-based and log-based methods [7]. The log-based dynamic analysis method is that Android applications are executed in real devices or Android simulators to monitor their behavior using well-known logging tools. The dynamic analysis method based on the hook tool is that the monitor point (hook) is embedded in code to record application activity during execution. These hooks can monitor Android applications' execution, collect information about behavior, track instructions executed, retrieve event sequences, or monitor stored data flows.

For the code coverage problem of dynamic analysis, it is not easy to run all application branches during dynamic analysis, so the hidden code that would be triggered in some specific time or scenario will be ignored. It can lure Android malicious applications to show malicious behavior through some stimulation mechanisms and inducements. It can simulate the regular use of the application by using the user interface event generation tool, simulate the regular interaction between the user and the application, and avoid being detected by a malicious application running in an Android emulator.

3) HYBRID ANALYSIS

The hybrid analysis combines static analysis and dynamic analysis and analyzes the application to extract static and dynamic features, thereby improving recognition accuracy. Its advantage is that it is more comprehensive in application analysis, combining the advantages of static analysis and dynamic analysis. However, the disadvantages are that the analysis and detection time is long, it takes up many system

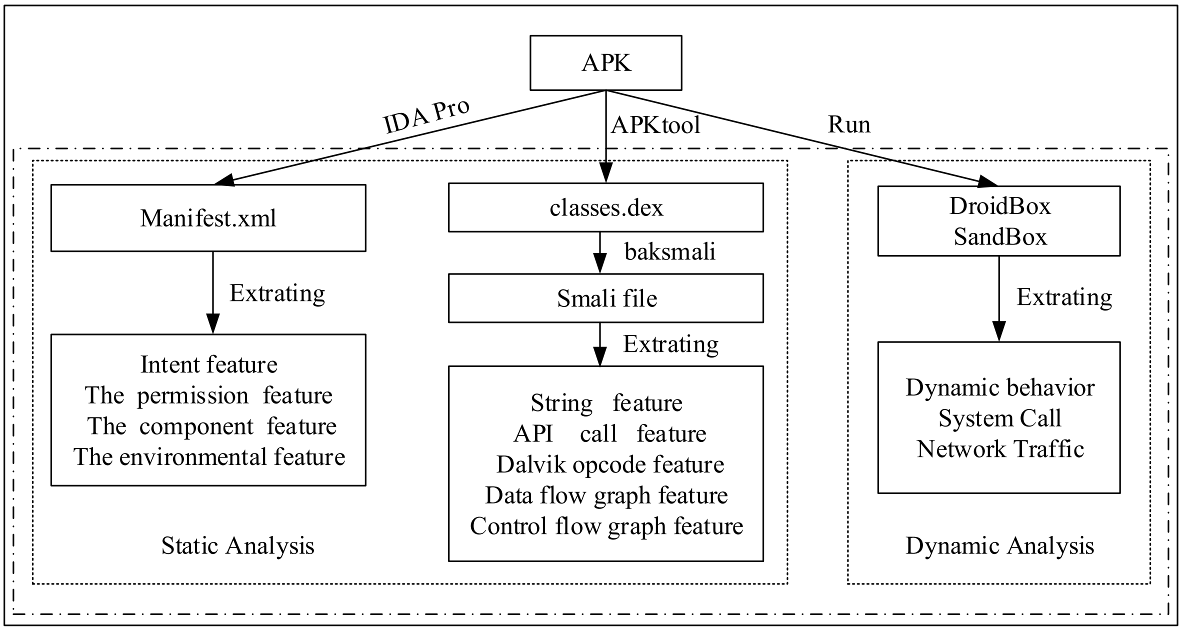

FIGURE 4. **Static and dynamic analysis of android applications.**

resources, and the calculation overhead is enormous. Android application static analysis and dynamic analysis are shown in Figure 4.

We can decompile the application to extract static features from the source code, install the software in an isolated environment, and Android simulator or a real device, to extract dynamic features [8], [9]. Therefore, the hybrid analysis method can not only improve the code coverage but also combat code obfuscation or encryption. Although its high cost and high implementation complexity limit its deployment, the hybrid analysis method is still considered the most profound and comprehensive method.

C. DEEP LEARNING

Deep learning [10] belongs to the sub-fields of Artificial Neural Networks and machine learning [11]. Deep learning is at the pinnacle of development and is widely used in computer vision, speech recognition, and natural language processing. The application of deep learning to Android malware detection has become a significant trend.

A typical deep learning model is a very deep neural network that uses multiple hidden layers of many interconnected neurons to process data. Each layer comprises many different neurons in deep neural networks, each with different weights and possibly different activation functions.

When the data is applied to a neural network, the loss function calculates the prediction error. The optimizer is used to update the weights to reduce the loss function error and improve the accuracy gradually. It trains the data and evaluates the accuracy on the test set.

The most significant value of deep learning lies in the automatic extraction and abstraction of features, eliminating the tediousness of manually extracting features and automatically finding sophisticated and useful high-order features. Each time the neural network deepens, the features that can be extracted become more abstract.

By summarizing the deep learning models used in Android malware detection, it is found that the current detection models are mainly deep belief network (DBN), convolutional neural network (CNN), recurrent neural network (RNN), and deep autoencoder (DAE). The following introduces various deep learning models and analyzes their advantages and disadvantages.

Geoffrey Hinton proposed the deep belief network in 2006. DBN can be used in unsupervised learning, using layer by layer training to solve multi-layer neural networks' optimization problem and provide good initial weight. It is divided into two stages: one is the unsupervised pre-training stage, the other is the supervised backpropagation stage. In the pre-training stage, multiple Restricted Boltzmann Machines (RBM) are superposed, and the multi-layer neural network is used as the hidden variable model to complete the high-level representation of DBN. In the backpropagation phase, the pre-trained DBN is fine-tuned in a supervised manner with labeled samples. RBM is an undirected probability graph model with one layer of observable variables and one layer of hidden variables. The training time is long when using the DBN method, which cannot be effectively used in large-scale learning.

A convolution neural network is a kind of neural network which is used to process data with a similar grid structure, such as image data. The network uses convolution. Convolution is a unique linear operation [12]. The convolution neural network's general structure includes the convolution layer, pooling layer, full connection layer, and output layer. Compared with other deep learning models, the convolutional neural network can give better results in image and speech recognition. Compared with other deep and feedforward neural networks, convolution neural networks use fewer parameters but can obtain higher performance. Convolution neural network uses a convolution kernel to share parameters, which significantly reduces the number of parameters and uses location information. CNN can map low-dimensional features to high-dimensional features and deal with the local related input data well.

A recurrent neural network is a kind of neural network used to process sequence data. Recurrent neural networks have been successfully applied to many temporal problems, such as natural language processing, speech recognition, and machine translation. The RNN performs the same operation on each element in the sequence and the previously calculated output. Therefore, RNN has two inputs: the current state and the previous state, combined to determine the network's final output. RNN has the problem of gradient dispersion and calculation cost. Therefore, some variants of RNN are proposed, such as Long Short-Term Memory (LSTM), Gated Recurrent Unit (GRU). LSTM can effectively solve the problem of long-term dependence of information. Compared with the traditional RNN, the unique feature of LSTM in structure is that it designs the loop body structure skillfully. However, LSTM also has some shortcomings, such as complex structure and high computational complexity. The GRU reduces the calculation cost and simplifies LSTM, so it has higher calculation efficiency and less memory.

The deep autoencoder is an unsupervised neural network. It can use its high-order features to encode itself. Its input and output are consistent. It uses the idea of sparse encoding to reconstruct itself by recombining some sparse high-order features. Many kinds of research using API call graph and control flow graph as features have applied deep autoencoder, which can reduce dimension. DAE's function is not limited to the initialization of some neural network weights, which are difficult to train. It is also possible to use it directly for feature extraction and analysis. The main difference between DAE and other unsupervised learning models is that it does not cluster, but extracts the most useful and frequent high-order features.

III. RESEARCH PROGRESS

As the Android operating system occupies a significant market share and its open-source nature, its security is increasingly threatened and challenged. Researchers have conducted much research on Android malware detection based on deep learning. The following introduces the relevant research progress.

A. MALWARE DETECTION USING DEEP LEARNING BASED ON STATIC ANALYSIS

Abdurrahman Pektaş *et al.* [13] proposed a deep network for Android malware detection. The deep network combines opcode sequence convolution and recursive network of training graph features and optimizes network parameters by applying grid search methods. It extracts the instruction call graph from the application to derive the instruction call sequence and applies pseudo-dynamic analysis, which analyses all execution paths through code analysis. Detection accuracy of 91.42% was achieved on a data set consisting of 24,650 malicious samples and 25,000 benign samples.

Abdurrahman Pektaş *et al.*compared the proposed method with a method based on support vector machine, random forest, logistic regression, and K-nearest neighbor algorithm. The experimental results show that the proposed method's performance metric is better than the method based on traditional machine learning technology.

Yuan *et al.* [14] used a deep belief network model for malware detection through hybrid analysis and compared the detection results under different network model architectures and machine learning models. The experimental results show that the detection method using deep learning is better than the machine learning technology using support vector machine, C4.5, Naïve Bayes, Logistic Regression, multi-layer perceptron, and the detection accuracy is at least 7% higher.

Su *et al.* [15] proposed a deep learning model for Android malware detection, which extracts the requested permissions, used permissions, sensitive API calls, opcodes, and application component features through static analysis. A total of 32,247 features were extracted. The deep learning model based on a deep belief network was used to learn the most typical and essential features, and the support vector machine algorithm was used to classify Android benign and malware. On the data set of 3986 benign applications and 3986 malware, the model's detection accuracy reaches 97.5%.

Kim *et al.* [16] proposed a multi-modal deep learning malware detection model, which extracts multiple feature types to reflect the attributes of the android application from various aspects and uses feature extraction methods based on presence or similarity to refine these features and to achieve effective feature representation in malware detection. It extracts seven types of static features: permissions, components, environment, strings, Dalvik opcode sequences, API call sequences, and shared library function opcode features. Each type of feature is respectively used to train the initial network of the corresponding deep neural network. Training results of the initial network are then used to train the final network. On the data set composed of 13075 malicious samples and 19747 benign samples, the model achieves 98% detection accuracy.

Wang *et al.* [17] proposed a hybrid model based on a deep autoencoder and a convolutional neural network. It uses a deep autoencoder as a pre-training method for convolutional neural networks to reduce training time. Experiments were performed on 10,000 benign applications and 13,000 malware, extracting seven static features(requested permissions, filtering intent, restricted API calls, hardware functions, code-related patterns, and suspicious API calls), and processing the feature set to reduce 34570 features to 413. They use two different CNN architecture called CNN-S and CNN-P model. The detection accuracy of CNN-S is higher than that of CNN-P, reaching 99.82%. DAE-CNN-S model achieves 98.6% detection accuracy, and its training time is 83% less than that of the CNN-S model.

Zhang *et al.* [18] proposed a prototype system DeepClassifyDroid based on a convolutional neural network. This system first performs a static analysis of the application through the feature extraction module and extracts features from the AndroidManifest.xml file and the disassembled Dex file. It generates four different feature sets (permissions, intent filters, API calls, a string constant). It uses feature embedding models to map feature sets of different dimensions into the joint vector space, and finally performs malware detection based on the convolutional neural network. On the data set of 5546 malicious samples and 5224 benign samples, the model's detection accuracy is 97.4%.

Karbab *et al.* [19] proposed an automatic Android malware detection and malware family classification system called MalDozer, which uses deep learning to identify Android malware by extracting the original sequence features of the application's API method calls. On a data set consisting of 33K malware and 38K benign applications, the F1 score is 96% -99%, and the false-positive rate is 0.06% -2%. Besides, the system can be deployed on servers and mobile devices and even IoT devices.

Luo *et al.* [20] proposed an Android malware analysis and detection method based on Attention-Convolutional Neural Network-Long Short Term Memory. The texture fingerprint features of the Android malware are extracted to reflect the similarity of the malware binary file blocks. AndroidMainfest.xml is regarded as a text document, and its contextual text features are extracted through natural language processing. The above features are filtered using a deep belief network. An end-to-end local correlation feature is extracted based on a one-dimensional time-domain convolutional network, and an LSTM model with higher time series modeling capabilities is used to analyze and detect Android malicious code. It achieves 96.4% detection accuracy on Drebin's open-source malware dataset.

Booz *et al.* [21] applied deep neural networks to implement Android malware classification by extracting core Android permissions and user-defined permissions and distinguishing between optional and required permissions. Grid search technology was used to test many combinations of tunable parameters for deep learning models. The best model was determined by adjusting six different hyperparameters to achieve 95% detection accuracy.

Android malware detection and malicious code localization methods based on deep learning are proposed, which obtains behavior sequence features from Android applications and uses bi-directional long short term memory networks to analyze the semantics of behavior sequence segments automatically. By processing the opcode, the bytecode in the APK file, and traversing all the calling procedures in an orderly manner, the method achieves 97.22% accuracy and 98.21% F1 score on the data set composed of 9616 malicious applications and 11982 benign applications [22].

Two end-to-end detection methods of Android malware without human intervention based on deep learning, DexCNN, and DexCRNN, are proposed. The classes.dex file of an Android application was preprocessed into a fixed-size sequence using two resampler methods as input to the deep learning model. In the data set containing 8K benign application and 8K malicious application, the DexCNN method can achieve 93.4% detection accuracy, and the DexCRNN method can achieve 95.8% detection accuracy. Simultaneously, both methods are not limited by the input file's size, do not need artificial feature engineering, and low resource consumption [23].

An Android malware detection method using deep learning is proposed. Firstly, 240 features, including permission, intention, sensitive API calls, and strings, are extracted, and they are sorted by Mean Decrease Impurity of the random forest to reduce the dimension of features and select features with higher importance. Secondly, the feature is transformed into a compact vector based on Word2Vec word embeddings to represent Android malware better. Finally, the classifier is constructed based on the DBN deep learning model, and the detection accuracy of it is 99.25% on the data set composed of 3000 benign samples and 12000 malicious samples [24].

This article proposes a deep learning method for Android malware detection. Through static analysis, it extracts 11 different behavior features, including component, intention, requested permission, hardware, API call, protected API, used permission, code module, string, authentication information, and payload information. A total of 32,856 features were extracted, unique features were learned using a deep learning model based on a deep belief network, and benign and malicious Android software were classified using a support vector machine. On a data set consisting of 3,986 benign applications and 3,986 malicious applications, the model's detection accuracy reached 97.4% when the ratio of benign to malicious applications was 1:1, and the average cost is 6 seconds to analyze and detect each Android application [25].

An Android malware detection scheme called ByteDroid is proposed. ByteDroid directly processes the original Dalvik bytecode, fills and cuts the bytecode, and represents it in One-hot coding. It automatically extracts features and classifies them through a convolutional neural network. For 10479 kinds of malware that apply seven typical obfuscation technologies (general obfuscation, class encryption, string encryption, reflection, and their combination), Bytedroid successfully detected 92.17% of them. ByteDroid maintained its adaptability to obfuscation technology. Because ByteDroid is a malware detection scheme based on static analysis, it cannot detect the malware that dynamically starts the malicious Dalvik bytecode or native code in the library [26].

This article proposes a detection method of Android malware based on deep autoencoder. It designs a specific autoencoder structure that reduces the dimension of API feature vectors extracted and transformed from APK, and uses a logistic regression model for binary classification. The experimental results show that the detection method has the best effect when the weight ratio between benign training samples and malicious training samples is 1:4 on the data set composed of 5000 benign samples and 1200 malicious samples, and the recall rate and F1 value can reach 93% and 64.3% respectively [27].

They propose an anti-obfuscation classification method for Android malicious applications integrating Recurrent Neural Network and Convolutional Neural Network, with anti-obfuscation ability and lightness. This method extracts application package name, authentication data, permission, and intention features from multiple short strings. The sample data set consists of 1152750 benign samples and 1279389 malicious samples. This method reduces the training time of the RNN model and achieves a 97.7% true positive rate when the false positive rate is 0.01 [28].

This article proposes a multi-mode malware detection method based on multiple convolutional neural networks, which uses permissions, API, and URL features to train sub-networks. It uses a backtracking method to solve the limitations of malware detection's poor interpretability based on neural networks. The backtracking method selects the most important features that make vital contributions to classification decision-making. This method reduces the detection time and achieves 96.54% accuracy on the data set composed of 10948 benign samples and 8652 malicious samples [29].

This article proposes a malware detector called MSNdroid based on the native API call, permissions, system API call features, and Deep Belief Network (DBN). It applies deep learning to native code features to detect Android malware. It extracts 5154 features from a dataset composed of 5442 malicious applications and 5215 benign applications. The performance of MSNdroid is better than Random Forest (RF), Decision Tree (DT), Support Vector Machine (SVM), and K-Nearest Neighbor (KNN) machine learning model. The accuracy of MSNdroid is 98.71%, and the false-negative rate is 0.7% [30].

A detection system for Android mobile malware using optimized convolutional neural network learning opcode sequence features is proposed. In this system, an improved static feature extraction method is adopted, and the k-max pooling strategy is adopted in the pooling stage of the convolution neural network, which makes the model retain important feature information and relative position information of features in the pooling stage. On the data set composed of 12500 benign samples and 10000 malicious samples, the system's detection accuracy reaches 99% [31].

B. MALWARE DETECTION USING DEEP LEARNING BASED ON DYNAMIC ANALYSIS

Hou *et al.* [32] proposed a new dynamic analysis method called Component Traversal, which can automatically execute code routines for a given Android application. By extracting the Linux kernel system calls from the samples, constructing a weighted directed graph, it applies a deep learning framework with a stack-based autoencoder model based on the graph's features to detect unknown Android malware. The detection accuracy of 93.68% was achieved on a data set consisting of 1500 malicious samples and 1500 benign samples.

Martinelli *et al.* [33] designed a method based on convolutional neural networks to detect system calls using dynamic analysis. Many interactions and system events are generated during application execution by running the application on a real physical device. The original system call features are extracted, and feature data is cleaned. It achieves 90% detection accuracy on a dataset consisting of 3536 benign samples and 3564 malicious samples.

Yeh *et al.* [34] inserts monitoring functions into the target application to record API call events during the preprocessing stage. It uses DroidBox to analyze Android applications and marks activities dynamically and extracts dynamic features. It converts a series of event logs into flat data with two-dimensional features and uses a model based on convolutional neural networks for malware detection. Using 16,000 benign and 16,000 malware as training data, and using 1,000 benign and 1,000 malware as test data, its prediction accuracy is 93.012%.

This article proposes a dynamic analysis system DL-Droid based on deep learning for Android malware detection, which uses a stateful input generation method to enhance code coverage to achieve high detection performance. In the real Android device, 31125 sample data are used for experiment instead of Android simulator, and the first 420 API calls, intents, and permission features are selected by information gain algorithm to achieve a 97.8% detection rate, which is better than the seven popular traditional machine learning classifiers [35].

A fully connected deep neural network with dropout and ReLu activation functions is proposed to classify mobile malware using low-level monitoring features such as CPU utilization, memory utilization, network data, sensor data. The accuracy of 99.79% is achieved on a dataset composed of 24343 benign samples and 8779 malicious samples [36].

This article proposes a method based on multiple dynamic behaviors features to detect malware using integrated learning combined with multiple basic machine learning models. These features include system-level behavior tracking and common application-level malicious behaviors, such as personal information stealing, advanced service subscription, and malicious service communication. Besides, the Chi-square feature selection algorithm is used to remove noise features, irrelevant features, and redundant features to extract key behavior features. Based on 8806 benign samples and 5213 malicious samples, the best detection accuracy of 96.49% is achieved using the stacking integration method. The proposed method only takes IP and port as the features of network operation, which will result in the loss of network-based malware. This method can only detect malicious behavior during the analysis process and can be considered to improve dynamic analysis coverage [37].

C. MALWARE DETECTION USING DEEP LEARNING BASED ON HYBRID ANALYSIS

Yuan *et al.* [38] combined static analysis with dynamic analysis to extract 192 features such as requested permissions and sensitive API functions. By designing an online Android malware detection engine (DroidDetector) based on a deep belief network, it applies deep learning to distinguish between malicious and benign applications. By comparing and analyzing the features extracted from malicious samples and benign samples, analyzing the features with a high frequency of occurrence and the features with significant differences, it links the features in static analysis with the features in dynamic analysis to better characterize Android malware to build advanced representations. Experiments were performed on a data set consisting of 20,000 benign samples and 1760 malicious samples, and a detection accuracy rate of 96.76% was achieved.

Yuan *et al.* [14] used static analysis to extract the requested permissions and sensitive API static features and used dynamic analysis to extract dynamic behavior features. A total of 200 features were extracted for malware detection. A deep belief network model was used to achieve 96% detection accuracy on a dataset consisting of 250 benign samples and 250 malicious samples, and the detection results under different network model architectures and different machine learning models were compared.

This article proposes a hybrid analysis based detection method for Android malware, HADM, which extracts requested permissions, permission request APIs, used permissions, advertising networks, intent filters, sensitive calls, network APIs, providers, and low-level instruction sequences static features, and system call sequences dynamic features. HADM applies a deep neural network to learn the features of the feature vector set, connects the new features with the original features, and uses hierarchical multi-core learning to build a hybrid classifier. It achieves 94.7% of the best classification accuracy on the data set composed of 4002 benign samples and 1886 malicious samples [39].

D. MALWARE DETECTION USING DEEP LEARNING BASED ON IMAGE PROCESSING

Zegzhda *et al.* [40] constructed a control flow graph through a smali file to obtain the features of the API call sequence, and mapped the corresponding protection level according to the permissions required for the API call, and then converted the API call sequence and protection level into pixels of the RGB image. It used a convolutional neural network model to detect malware. Classification accuracy of 93.64% was achieved on a data set consisting of 7192 benign samples and 24461 malicious samples.

Ganesh *et al.* [41] used static analysis to extract 138 permission features in four categories, converted the permission features into 12 × 12 PNG images, and used convolutional neural networks for model training and detection. In 2500 Android applications, it achieved 93% detection accuracy, including 2000 malicious samples and 500 benign samples.

Huang *et al.* [42] converted the bytecodes of classes.dex into RGB color codes and converted them into color images with a fixed size. The color images were input to a convolutional neural network for automatic feature extraction and training. The data set was collected from January 2017 to August 2017. About 2 million benign and malicious Android applications were collected, achieving a detection accuracy of 98.4225%.

Shiqi *et al.* [43] proposed a method based on texture fingerprints to extract malware content features. This method uses the information of the texture image extracted from the sample application code. According to the texture image method, these codes are mapped to the uncompressed gray value and then combined with API call features to detect the Android malware using the deep belief network. On the data set of 6956 samples, the model achieves 95.6% detection accuracy.

Yen and Sun *et al.* [44] proposed a method that can detect the malware by visualizing the importance values of the words in the APK code as images and then using a convolutional neural network to detect. They decompiled the DEX file to get the Java source code. They used the term frequency-inverse document frequency method to get each word's importance value and then used the simhash algorithm and gjb2 algorithm to generate images. On the data set composed of 720 benign samples and 720 malicious samples, the model's detection accuracy rate is 92.67%.

Xu *et al.* [45] used semantic graph representations, that is, control flow graphs, data flow graphs, and their possible combinations as features to characterize Android applications. The graphs are then encoded into matrices and used to train a classification model via a convolutional neural network. Samples were collected from four data sets: Marvin, Drebin, VirusShare, and ContagioDump, achieving 99% detection accuracy.

Pektaş *et al.* [46] used the API call graph as a graphical representation of all possible execution paths that malware can track during its runtime. The graph embedding method was used to transform the API call graph's embedding into a low-dimensional numerical vector feature set, which was introduced into the deep neural network. Evaluation experiments were performed on different graph embedding algorithms and network configuration parameters, and classification accuracy of 98.86% was achieved on a dataset consisting of 33,139 malicious samples and 25,000 benign samples.

E. MALWARE FAMILY MULTI-CLASSIFICATION USING DEEP LEARNING

Li *et al.* [47] detect malware through a fine-grained malware detection engine based on deep neural networks, which displays detailed family categories and other malware information, not just whether the application is malicious or not. By extracting static features such as permission features and API call features, the detection model achieves a detection accuracy of 97.16% and a false positive rate of 0.1% on the DREBIN dataset consisting of 5560 malicious samples and 123453 benign samples.

This article proposes an Android malware family classification method based on the Dalvik executable file (DEX file). This method transforms the DEX file into RGB image and plain text, then extracts the Generalized Search Trees(GIST) texture features, color features, and plain text features of the image as to features, and uses the feature fusion algorithm for classification based on multi-core machine learning. 96% classification accuracy is achieved on the Android malware dataset (AMD) with 24553 malicious samples [48].

A method of static malicious sample detection based on deep neural networks is proposed. The method obtains gray images directly from executable samples and uses a gray histogram to collect a group of features from each image to build multiple classifiers. Experiments were carried out on the real samples of 50,000 (24,553 malware in 71 families and 25,447 benign samples) to detect whether the analyzed samples were malware, detect their families and related variants, and achieve multi-classification of malware families with an accuracy of 92.9% [49].

In this article, an image-based fine-tuning convolutional neural network architecture for malware family's multi-classification is proposed. The original malware binary files are transformed into color images. The accuracy in the Malimg malware data set (9435 samples) is 98.82%, and the accuracy of Android mobile data sets (14733 malware and 2486 benign samples) is over 97.35%. This method uses data enhancement to improve the algorithm's performance and can detect malware processed by obfuscation technology [50].

Using the code's gray image converted from the binary bytecode of the malware DEX file, an Android malware family classification method based on deep learning is proposed. The deep learning classifier is constructed by reusing the feature extraction layer of the convolutional neural network Google perception V3, which has been successfully trained on large datasets for traditional image classification tasks. It can automatically learn and distinguish features from malware images and achieve 97.7% accuracy on the data sets of 4892 malware samples and 30 malware families [51].

IV. RESEARCH STATUS ANALYSIS

Through the summary of the above the latest research work, the current research situation is analyzed in detail.

Studies [15]–[31] combined static analysis and deep learning. It used APKTool, Dex2Jar, BackSmali, and other decompilation tools to decompile and statically analyze the Android APK file, extract static features, and use static feature vectorization as the input of the deep neural network model. The training set is used to train the model iteratively, and the test set is used to test the model to realize the detection of Android malware. The extracted features are mainly permission features and API call features. Component features, intent features, data flow features, and opcode sequence features are all involved. However, malware behavior cannot be fully characterized by static features alone, and dynamic feature extraction is also required.

[13], [16], [17], [20], [23], [27]–[29] adopt multi-modal deep learning, that is, use multiple different deep neural network models for detection, each type of feature is specially used to train the corresponding sub deep neural network, and the training results of the sub neural network are used to train the final neural network. In [13], [21], grid search technology is used to test many combinations of adjustable grid search technology is used to test many combinations of adjustable parameters for deep learning neural network models, and different super parameters are adjusted to determine the best classification model. In contrast, the adjustment of artificial super parameters takes much time. In [17], [27], the deep automatic encoder is used as the deep neural network's pre-training method to reduce the original feature vector's dimension and shorten the training time. In [16], [17], [25], it has processed the features, in [16], it uses the existing or similarity-based feature extraction method to improve the static features, to achieve the effective feature representation in malware detection. In [17], it codes all features and uses the feature code to represent each application. In [25], it uses the deep learning model based on the deep belief network to process the feature, using a behavior feature learning algorithm to select behavior features. The detection system proposed by [19], [25], [28] is lightweight and can be deployed flexibly. The system proposed by [19] can be deployed not only on servers but also on mobile devices and even the Internet of things devices. The proposed system spends an average of 6s to analyze and detect each Android application [25], and the scheme is light enough to be deployed on Android devices with limited resources [28]. In [13], [18], [30], it compared and analyzed the proposed method with the methods based on support vector machine, random forest, logical regression, and k-nearest-neighbor algorithm. The experimental results show that the proposed method's detection performance is better than that of the method based on traditional machine learning, showing the advantages of deep learning. In [22], [24]–[26], [31], it adopts the method of one-hot coding to vectorize the extracted static features. The feature vectors obtained by one-hot coding are sparse and high-dimensional, so further processing is needed to reduce the feature vectors' dimension and sparsity. In [22], [26], [28], it can analyze APK files processed by obfuscation technology and maintain the adaptability to obfuscation technology.

Studies [32]–[37] combined dynamic analysis and deep learning, extracted dynamic features such as system calls as data input, and used deep neural network models for Android malicious application detection. In [32]–[37], it only performed dynamic analysis without static analysis and lacked static features extraction, which could not fully characterize malicious applications' behavior. The system scheme proposed in [32] has been integrated into commercial Android anti-malware. Linux kernel system call is resistant to malware evasion technology. The system uses a component traversal method to automatically execute the whole application code to perform all application program components for malware analysis. Future work includes introducing the generation of random events on each component to improve the component traversal method further, further explore how to impose sparse constraints on autoencoder to produce better detection performance, and study other deep learning models for Android malware detection. The scheme proposed in [33] uses real Android physical devices to conduct dynamic analysis on APK, and the other schemes proposed in the literature use Android simulators to analyze APK. Only from the detection accuracy aspect, the detection effect based on Android physical devices needs to be improved, and its detection accuracy is lower than that based on Android simulators. The data set used in the detection scheme proposed in [32], [33] is small, and the reliability of the detection results needs to be further tested. In [34], a planarization data input method is proposed. By combining attributes and the proposed planarization input format, the convolutional neural network can reduce the dimension of the k-skip-n-gram. The convolutional neural network is applied to Android malware detection, which opens the door to deep learning in other fields. In [36], it uses a fully connected neural network to detect applications by identifying low-level monitoring features that can distinguish benign applications from malicious applications. Its researchers plan to develop improved anti-malware software based on deep learning in the future, without requiring an Android device root or kernel level modification.

In [14], [38], [39], it combines deep learning with hybrid analysis to detect Android malware using static and dynamic features. The number of malicious samples in the data set less than 2000. Because the data set is small, the reliability of the test results needs further study. In [38], it uses association rule mining techniques to process the static and dynamic feature. A binary value represents each feature. When a feature appears in the application, its value is 1. Otherwise, it is 0. Its researchers pointed out that more fine-grained features should be extracted to characterize Android applications. Such feature sets can cover more aspects of Android malware to characterize better and detect malware. In [39], hierarchical multi-core learning is applied to combine different kernel learning results of different features, and static features are transformed into vector-based representations, dynamic features are transformed into n-gram vector and n-gram graph. A deep neural network is trained for each feature vector set, and advanced features learned by the neural network are combined with original features to form new features vector set.

In [40]–[46], it is based on the image method combined with deep learning to detect Android malicious applications. In [40], [41], by converting the extracted features into images, using an imaging method to vectorize the features, the Android malware detection problem is converted into an image classification problem, and the neural network is trained by using image data. There are mature schemes for image classification using a convolutional neural network. The critical problem in the field of Android malware detection is vectorization representation. The specific method of transforming features into images needs further study. In [42]–[44], it processes code, including the bytecode of classes.dex, application code, and words of APK code. It does not involve the extraction of static or dynamic features without decompilation technology and reverse engineering. It directly processes the files in APK, converts the files into images, and extracts image texture features from images using a neural network. The specific method of converting code into image needs further analysis, discussion, and research. In [45], [46], the method of graphical representation is used. API call graph, control flow graph, and data flow graph are taken as Android applications' features. These graphs are vectorized and encoded into a numerical vector feature set.

In [47]–[51], it uses deep learning to classify Android malicious application families. The detection results show detailed family categories and other information of malicious applications, not just whether the application is malicious. In [48]–[51], it converts file codes into images, and use a neural network to classify Android malware families. See Table 2 for a detailed analysis and comparison. Table 2 is a comparative study of using deep learning models to detect Android malware, and it analyzes and discusses some latest research results. The size and types of the publicly available Android malware family multi-classification datasets are minimal, which needs further research.

Most of the surveyed papers used CNN and DBN to detect Android malware. The highest accuracy was 99.82% with a 0.21% false positive rate obtained by CNN [17]. In [17], the proposed model combines DAE with CNN and uses DAE as a pre-training CNN method to reduce the training time. In order to reduce the dimension of the dataset, the proposed model encodes all seven kinds of static features and uses the feature code to indicate each app. The limitation of this model is that it cannot be resistant to obfuscating techniques. In [27], the proposed method uses DAE to reduce the API feature vector's dimension and uses logistic regression binary classification model to detect malware. Using other deep learning models instead of logistic regression classification model may improve the detection accuracy. In [24], the proposed method achieves a 99.25% precision rate by using DBN. It extracts static features and ranks them by mean decrease impurity in the random forest to select higher importance features. Moreover, it transforms the features into high-level representation using Word2Vec. In most of the surveyed papers, one-hot encoding is used for vectorization of features. When the number of extracted features is large, the feature vector's dimension using one-hot encoding is high, and the feature vector is sparse. Word2Vec is used to generate dense vectors to overcome the shortcomings of one-hot encoding. Replacing one-hot encoding with Word2Vec may improve the accuracy of the model. In [24], the number of benign

TABLE 2. A comparison of deep learning models to detect Android malware.

Reference	Year	Feature	Feature Processing	Feature Vectorization	Deep Learning model	Contribution	Limitations	Future outlook
[22]	2020	API sequence	First call traversal based on bytecode instructions	One-hot encoding	Bi-directional-LSTM	Weight allocation strategy	Time consuming	Multi classification of malware family.Improve the localization method of malicious code.
[23]	2020	Original bytecode of classes.dex	none	Take the DEX file as an unsigned vector, and then convert the vector to a fixed size by resampling	CNN,CNN+LSTM	This method is not limited by the size of input file, does not need artificial feature engineering, and has low resource consumption, so it can realize end-to-end detection	The pattern learned by the proposed detector is still difficult to understand	Cross platform malware detection research. Add other files (for example, Androidmanifest.xml) as model input.
[35]	2020	Application properties, actions/events, permissions	Information gain, feature ranking	One-hot encoding	DNN	Using state based input generation to enhance code coverage	Time consuming	Introduce adaptive technology
[48]	2020	Texture feature, color feature and text feature of image	Feature fusion algorithm	Simhash algorithm, GIST algorithm	multiple kernel learning	Handling multiple DEX files for Android Applications	Process DEX files only	Study .so files and extract more features from them
[49]	2020	Gray image texture	Gray histogram	Get gray image from executable sample	DNN	Detection of families and family variants	RGB image may be better	Explore whether formal verification technology can help achieve better performance
[50]	2020	color image	none	Binary file to color image	CNN	Detect hidden code, confused malware and malware family variants	Small data set	The transformation from malicious code to color image needs further study
[25]	2020	Eleven types of static features	Feature learning	One-hot encoding	DBN	A model can select behavior feature through DBN.	Small data set, too many features	Multi classification of malicious application family
[13]	2019	Instruction call graph,instruction call sequence	none	Three additional features of flow graph are calculated: the depth of graph, the number of leaves and the maximum number of leaves. These features are combined into a feature vector	LSTM+CNN	Using a grid search method to find the best parameters of network and find the combination of super parameters	Limitations of pseudo dynamic analysis	Adopt dynamic analysis
[29]	2019	Dangerous permission, sensitive API, malicious URL	none	Identifier Mappings	CNN	Multimodal malware detection method, provide interpretability for detection results	Inherent limitations of static analysis	Using dynamic analysis to improve the anti confusion ability of the system
[24]	2019	Permission, intent, API call, string	Mean Decrease Impurity of Random Forest	One-hot encoding+Word2Vec	DBN	Using word2vec to transform features into advanced representations	Inherent limitations of static analysis	Using dynamic analysis to improve the anti confusion ability of the system
[26]	2019	Dalvik bytecode sequence	none	One-hot encoding	CNN	Maintain the adaptability to confusion technology, Processing the original Dalvik bytecode directly	It can't detect the malware that dynamically starts the malicious Dalvik bytecode or native code in the library.	Explore more complex network architecture, dynamic analysis to detect dynamically loaded code, the interpretability of neural network.
[51]	2019	Gray code image	none	2048-dimensional feature vector of image obtained by feature extraction module	CNN	The feature extraction layer of convolutional neural network, which has been successfully trained on large data sets for traditional image classification tasks, is reused to construct deep learning classifier	Process DEX files only	Using localization and global features of Android code image to solve malware countermeasures such as relocation of binary segments, building large-scale Android malware data set with family tags.
[31]	2019	Opcode sequence	none	One-hot encoding,Two-dimensional matrix	CNN	In the pooling stage of convolution neural network, K-max pooling strategy is adopted	Inherent limitations of static analysis	Add multiple features and use dynamic analysis to improve the anti confusion ability of the system
[30]	2019	Permission, API call	none	none	DBN	Integrating native layer API features into feature set	Inherent limitations of static analysis	Analyze data flow and interaction between native API and system API
[17]	2019	seven types of static features	Code all features and represent each application with feature code	Feature matrix	DAE,CNN	Two different CNN architectures are developed, DAE is used as pre-training method	Inherent limitations of static analysis	Using dynamic analysis to improve the anti-confusion ability of the system
[28]	2019	Application package name,authentication data, permissions, intent	none	Package (128), Certificate(128), Permissions+intent actions(256) to 512-dimensional vectors	RNN+CNN	Resistance to confuse and lightweight enough to deploy on Android devices	Inherent limitations of static analysis	Dimension reduction of feature vector

samples is small compared with the number of malware collected. To get an accurate estimation, more benign samples need to be collected. In [19], the original sequence of API method calls extracted from the DEX assembly is used as input to the model. The proposed model replaces each API method call with an identifier, resulting in a sequence of numbers. The proposed model can serve as a malware detection system that is deployed on servers and mobile and IoT devices. The proposed model may be the most practical in the papers surveyed. It pushes toward portable detection solutions, which enhances small devices' security. The limitation of this model is that it is not resilient against dynamic code loading and reflection obfuscation. Most of the proposed methods do not consider the actual deployment. These methods can be applied to anti-virus software to enhance the ability of anti-virus software to detect malware.

As shown in Table 2, the research results are compared and analyzed in detail from the extracted features, feature processing methods, feature vectorization methods, deep learning models, contributions, limitations, and prospects. From the limitations and prospects of each study, we can find ways and directions for improvement.

In these papers, many detection models use a combination of various deep learning models for detection, such as CNN + LSTM [23], DAE + CNN [17], RNN + CNN [28]. Compared with the model using a deep learning model, the hybrid detection model can also achieve satisfactory results. The combination of various deep learning models makes use of the advantages of each model, which can also be used as the research direction in the future. The detection model based on deep learning adopts static analysis, but it can detect obfuscated malware [22], [26], [28]. In [28], the package and certificate information of obfuscated malware variants is available, and the proposed model identifies obfuscated malware according to these features. In [22], the proposed method deals with the opcode and the bytecode in the APK files and analyzes various API sequences features. Whether an application is obfuscated or not makes no difference in

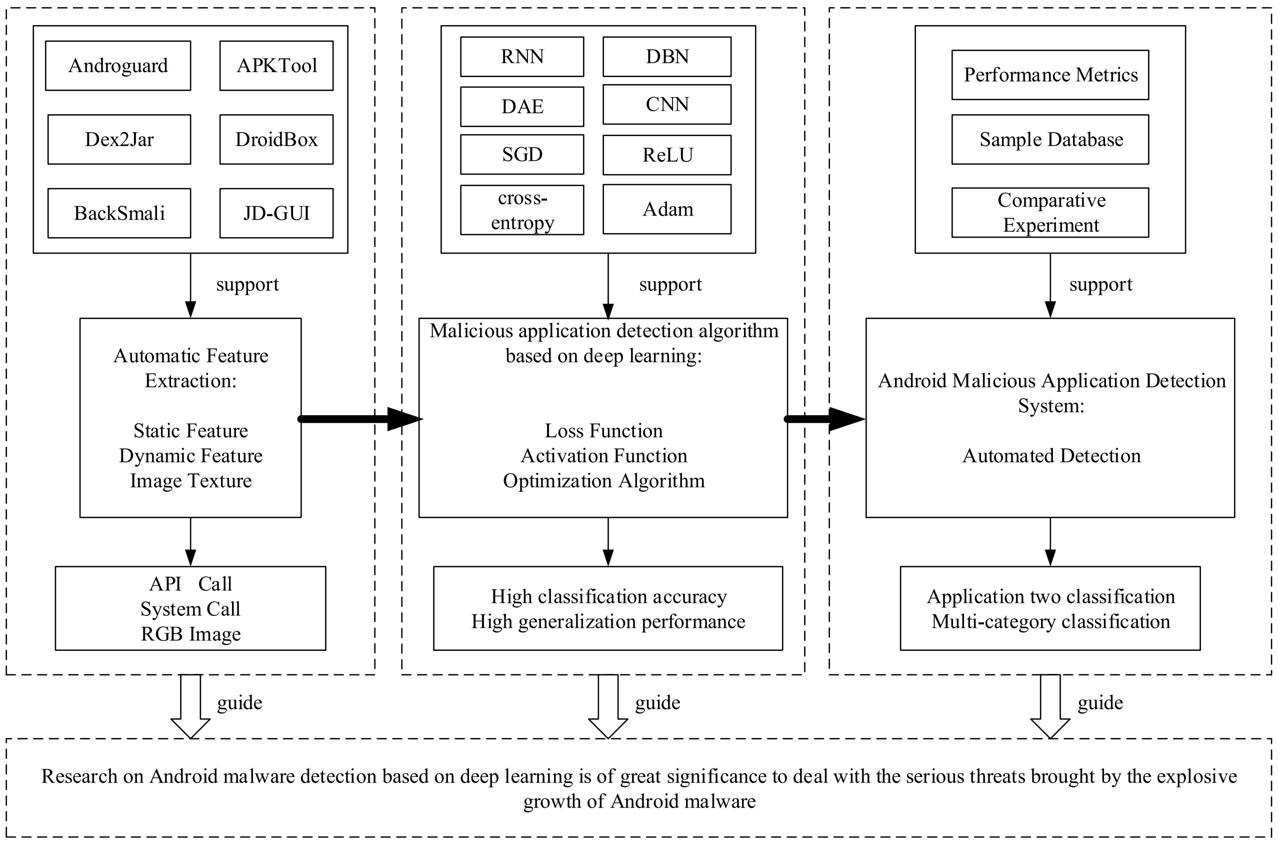
Androguard
APKTool
Dex2Jar
DroidBox
BackSmali
JD-GUI
support
Automatic Feature Extraction:
Static Feature
Dynamic Feature
Image Texture
API Call
System Call
RGB Image
RNN
DBN
DAE
CNN
SGD
ReLU
cross-entropy
Adam
support
Malicious application detection algorithm based on deep learning:
Loss Function
Activation Function
Optimization Algorithm
High classification accuracy
High generalization performance
Performance Metrics
Sample Database
Comparative Experiment
support
Android Malicious Application Detection System:
Automated Detection
Application two classification
Multi-category classification
guide
guide
guide
Research on Android malware detection based on deep learning is of great significance to deal with the serious threats brought by the explosive growth of Android malware

FIGURE 5. **Android malware detection architecture based on deep learning.**

their case. In [22], [28], they theoretically explain that the detection model could detect obfuscated malware, but do not use obfuscated malware to verify. In [26], the proposed model uses Dalvik bytecode as a static feature. They train the CNN model using the dataset provided by FalDroid [52], and use the dataset from Android PRAGuard [53] for the evaluation, which contains 10479 malware obfuscated by seven different obfuscation techniques. The CNN model effectively detects the malware applied with trivial obfuscation, which only affects the string and does not change the bytecode's instructions. As for the obfuscation techniques which affect the bytecode sequence, the model has a poor performance. Since these proposed methods are based on static analysis using deep learning, they inherently fail to detect the malware that dynamically loaded from native shared libraries such as.so (shared object) files. The dynamic analysis technology can effectively overcome this deficiency. However, most research papers use static analysis technology, mainly due to the high computational cost and long detection time of the dynamic analysis model. Improving the ability of a static analysis model based on deep learning to detect obfuscated malware needs further research. In [33], the dynamic analysis model based on deep learning uses real Android devices to conduct dynamic analysis on APK. Most of the dynamic analysis models based on deep learning in the research papers use an Android simulator for analysis and detection. When malware detects that it runs in the Android simulator, it will stop its malicious behavior. Therefore, researchers should consider using real Android devices in future research.

V. ANDROID MALWARE DETECTION BASED ON DEEP LEARNING

Based on the analysis and summary of many domestic and foreign academic papers, the system architecture and detection scheme of Android malware detection using deep learning technology are summarized.

Figure 5 shows the Android malware detection architecture using deep learning. The decompilation and static analysis of the Android APK file is performed using Androguard, APKTool, Dex2Jar, BackSmali, and other decompilation tools to extract the static features. It uses DroidBox to dynamically analyze the Android application and extract the dynamic features from the Android application running status and operation log. The image rule uses the RGB image generation algorithm to convert the files in the APK into RGB images, and it uses a deep learning-based malware detection algorithm to extract image texture features from the RGB images. It designs neural network models in aspects of the loss function (cross-entropy), activation function (ReLU [54]) and optimization algorithm (Stochastic Gradient Descent algorithm [55], Adam algorithm) while referring to existing deep learning models, such as

Recurrent Neural Network, Deep Belief Network [24], Deep-AutoEncoder [56], Convolutional Neural Network [57]. The Android malware detection system is implemented based on this through the automatic extraction of features and the Android malware detection algorithm's realization. The system is trained according to the sample database, its performance is tested according to performance metrics, and various comparative experiments are performed to realize the automation of the system detection.

Figure 6 shows Android malware detection scheme based on deep learning. The process of detecting malicious Android applications using deep learning is schematically illustrated in detail. The detection scheme is divided into five stages: data set construction, feature extraction, feature selection, feature vectorization, training, and detection. The detection scheme is obtained by summarizing the relevant research. The actual research does not necessarily include all five stages. For example, some researchers have no feature selection in the detection scheme.

In the data set construction stage, benign applications are downloaded through AndroZoo, Google App store, and other platforms. Malicious applications are collected through the Drebin data set, Android malware genome project, Android malware dataset, and other data sets. APK samples in the data set are divided into a training set, verification set, and test set according to a certain proportion. The training set is a sample for learning, which is used to train the neural network model. The verification set is used to estimate the generalization error in or after training and update the super parameters. The test set is used to evaluate the neural network model's generalization error after the learning process.

In the feature extraction phase, tools such as IDA Pro and APKtool are used to decompile APK files. The baksmali tool is used to get the smali file from the classes.dex. Dex2jar tool is used to get the jar file from the classes.dex, and then use the JD-GUI tool to get the java file from the jar file. The static features of Android applications are extracted from AndroidManifest.xml, smali files, and java files. By using DroidBox, SandBox, Android applications are installed in Android Virtual Device to conduct dynamic analysis on its operation, and dynamic features are extracted from the running log or using hook tools. The image-based method is used to obtain image texture features from the image. The binary byte sequence is obtained from Androidmanifest.xml and classes.dex, binary byte sequence is used to obtain a pixel matrix according to the RGB image generation algorithm, an RGB image is finally obtained according to a pixel matrix.

In the feature selection stage, it processes the original feature data set. It uses the association rule mining algorithm, information gain algorithm, chi-square statistical algorithm, and frequency sorting algorithm to select the appropriate features from the original feature set and remove the noise, irrelevant and redundant features.

In the feature vectorization stage, the features in the feature data set are vectorized by the Word2Vec [58], one-hot encoding [59], Euclid distance measurement [16], and identifier mapping [60]. These vectors will be input into the neural network model for model training. The vectorization process is described in detail in Section V(C)'s feature vectorization representation part.

Training and testing phase. The neural network model consists of many layers of the neural network, which can be generally divided into the input layer, hidden layer, and output layer. By training the training set's sample data, the neural network model is trained to find the best combination of super parameters. The super parameters of the neural network model mainly include the number of layers of the neural network, the number of neurons in each layer, the number of iterations, the selection of optimizer, learning rate, batch size, the selection of activation function, convolution kernel size, convolution kernel number, pooling layer size, step length and dropout rate. The neural network model is tested with a test set to test its classification performance and generalization ability.

We can use performance metrics to test system model performance. The test results can be expressed in the form of a table called a confusion matrix, as shown in Table 3, which has four parameter types. True Positive (TP) refers to samples that get marked positive that are positive or marking a malicious file as malicious. False Positive (FP) refers to samples that get marked positive that are NOT positive or marking a benign file as malicious. True Negative (TN) refers to samples that get marked negative that are negative or marking a benign file as benign. False Negative (FN) refers to samples that get marked negative that are NOT negative or marking a malicious file as benign.

TABLE 3. Confusion Matrix.

	Predicted Positive	Predicted Negative
Actual Positive	TP	FN
Actual Negative	FP	TN

The confusion matrix can show the detection results of the system intuitively. To further evaluate the system's performance, based on the confusion matrix shown in Table 3, the researchers calculated more evaluation indexes.

TPR, True Positive Rate, also known as sensitivity or recall rate, is the proportion of positive samples detected to the actual positive samples. The calculation formula is (1):

$$TPR = \frac{TP}{P} = \frac{TP}{TP+FN} \tag{1}$$

FNR, False Negative Rate, is the proportion of positive samples detected as negative to the actual positive samples. The calculation formula is (2):

$$FNR = \frac{FN}{P} = \frac{FN}{TP+FN} \tag{2}$$

FPR, False Positive Rate, is the proportion of negative samples detected as positive to the actual negative samples.

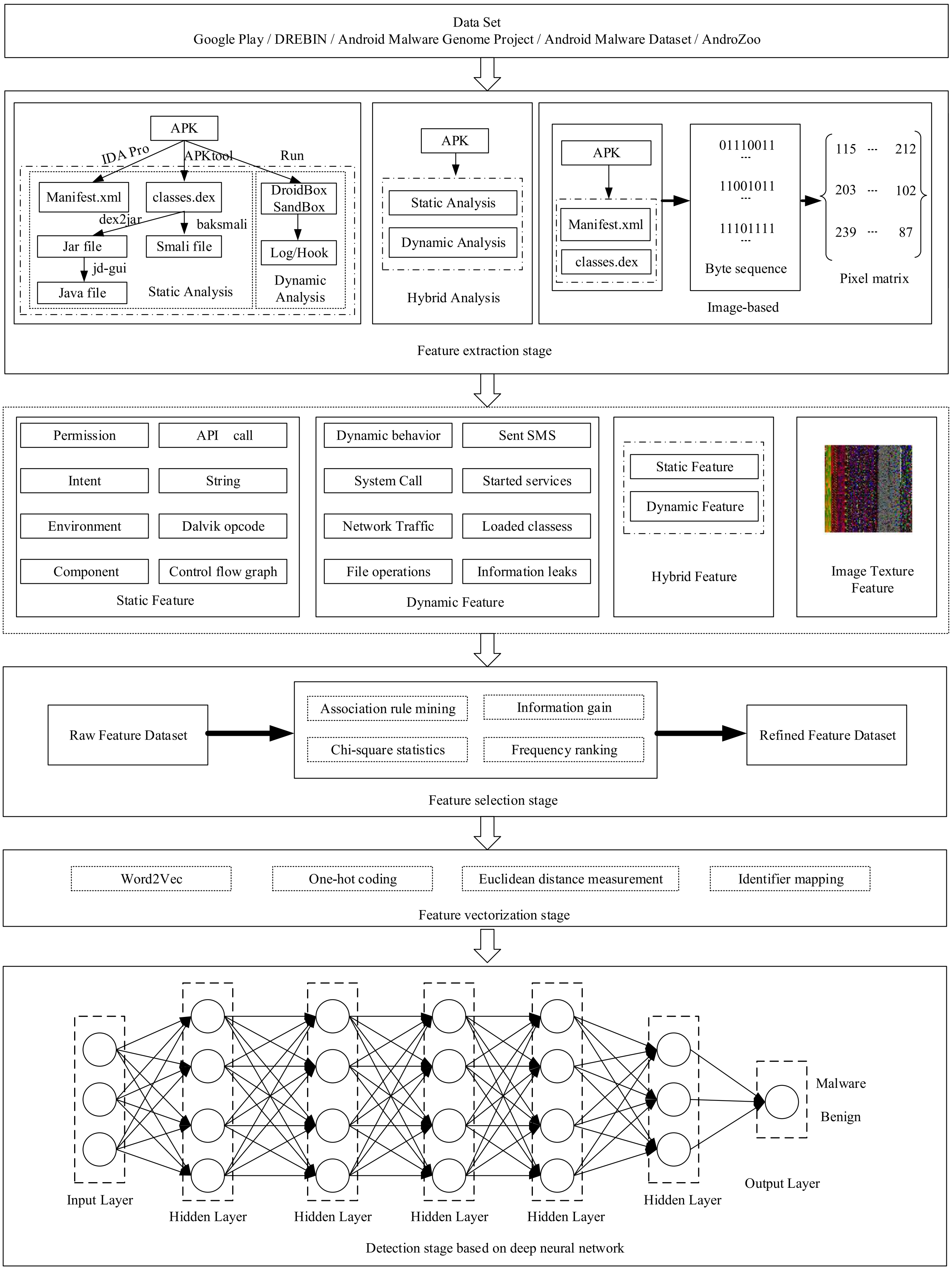

FIGURE 6. **Android malware detection scheme based on deep learning.**

The calculation formula is (3):

$$FPR = \frac{FP}{N} = \frac{FP}{FP + TN} \tag{3}$$

TNR, True Negative Rate, is the proportion of negative samples detected as negative to the actual negative samples, the calculation formula is (4):

$$TNR = \frac{TN}{N} = \frac{TN}{FP + TN} \tag{4}$$

Accuracy, the system's ability to detect the whole data set's samples, is the ability of the system to detect positive samples as positive samples and negative samples as negative samples. The calculation formula is (5):

$$Accuracy = \frac{TP + TN}{P + N} = \frac{TP + TN}{TP + FP + TN + FN} \tag{5}$$

Precision is the proportion of the number of samples that are positive to the number of samples that are detected as positive. The calculation formula is (6):

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

F-measure, the weighted harmonic mean of precision and recall, is closer to the smaller of the two. The calculation formula is (7)(8):

$$F\_measure = \frac{\left(\alpha^2 + 1\right) \times Precision \times Recall\ Rate}{\alpha^2 \times (Precision + Recall\ Rate)} \tag{7}$$

$$F1 = \frac{2 \times Precision \times Recall\ Rate}{Precision + Recall\ Rate} \quad \alpha = 1 \tag{8}$$

To evaluate the generalization performance of the model, we need not only a practical and feasible experimental estimation method but also an evaluation standard to measure the generalization ability of the model, which is called performance measurement. Performance measurement reflects the task requirements. When comparing different models' capabilities, different performance measures often lead to different evaluation results, which means that the model's performance is relative. What kind of model is fine depends not only on the algorithm and data but also on the task requirements. We have so many indicators to meet all tasks' needs because accuracy can not meet all task requirements. In the book "machine learning," written by Zhou Zhihua, it is mentioned that taking the watermelon problem as an example, assuming that the watermelon farmers bring a cart of watermelons, we use the trained model to discriminate these watermelons. Accuracy measures how many proportions of melons are correctly identified. However, if we are concerned about "how many of the selected watermelons are good melons" or "how many of all the good melons are picked out," recall and precision are more suitable performance measures for such requirements. Therefore, it is not comprehensive to consider only the accuracy index. Precision refers to the prediction result, which indicates how many samples that get marked positive are real positive samples. The recall rate refers to the original sample, which indicates how many positive examples in the sample are predicted correctly. In short, precision means "finding rightly," and recall rate is "finding all." Therefore, they are essential for classification. When the probability of error classification between different categories is not taken into account, indicators such as accuracy are often misleading. Which index to use depends on the specific situation. Achieving the highest accuracy does not necessarily mean that the classifier can correctly predict. Therefore, it is necessary to evaluate the reliability of the proposed system in combination with other indicators.

VI. PROBLEMS AND CHALLENGES

By investigating the latest papers of deep learning applied to Android malware detection, the problems and challenges faced by deep learning applied to malware detection are summarized.

A. DATA SET UPDATE

In reality, most Android applications are non-malicious. This basic fact brings significant challenges to the detection of Android malware using deep learning. A large-scale and continuously updated malicious sample data set is required to take full advantage of deep learning. Most researchers' data set is still relatively small and old, and the sample data in the data set cannot be updated in time.

The most commonly used data set is the DREBIN data set [61], which consists of 5,560 malicious samples and 123,453 benign samples, but the collection time of this sample was from August 2010 to October 2012. The Android Malware Genome Project data set [62] consists of 1260 malicious samples and 863 benign samples, of which the malicious samples can be divided into 49 categories. The collection time of the samples is from August 2010 to October 2011. The Contagio dataset consists of 1150 malicious samples collected in 2011. Android Malware Dataset [63], [64] is composed of 24553 malicious samples, among which there are 135 categories of malicious samples, which are collected from 2010 to 2016. The VirusShare dataset is publicly available through the web site, and the 2016 dataset contains 65536 malicious samples. Android PRAGuard Dataset [53] consists of 10479 malicious samples collected in 2015. Marvin data set [65] consists of 10572 malicious samples and 75996 benign samples collected in 2015. The ISCX Android Botnet Dataset [66] consists of 1929 malicious samples, which can be divided into 14 categories. The collection time of the samples is from 2010 to 2014. Details of these datasets are shown in Table 4.

Robin Nix *et al.* [59] used system API call sequences to detect and classify Android applications and designed a convolutional neural network for sequence classification, achieving 99.4% detection accuracy on 216 malicious samples and 1016 benign sample dataset. Zhenlong Yuan *et al.* [14] used static analysis to extract the requested permissions and sensitive API static features and used dynamic analysis to extract dynamic behavior features. A total of 200 features

TABLE 4. Comparison of different data sets.

Data Set	Collection Time	Benignware	Malware	References	Access
Android Malware Genome Project	2010.8-2011.10	863	1260(49 classes)	[62]	http://www.malgenomeproject.org/
DREBIN	2010.8-2012.10	123453	5560(179 classes)	[61]	https://www.sec.cs.tu-bs.de/∼danarp/drebin/index.html
Contagio	2011	none	1150	none	http://contagiodump.blogspot.com/
Android Malware Dataset	2010-2016	none	24553(135 classes)	[63] [64]	http://amd.arguslab.org
VirusShare	2016	none	65536	none	https://virusshare.com/. https://github.com/zautomata/virusshare
Android PRAGuard Dataset	2015	none	10479	[53]	http://pralab.diee.unica.it/zh/AndroidPRAGuardDataset
Marvin	2015	Training set:50501 Testing set:25495	Training set:7406 Testing set:3166	[65]	paper
ISCX Android Botnet Dataset	2010-2014	none	1929(14 classes)	[66]	paper
AndroZoo	2018.1	25000	none	[68]	https://androzoo.uni.lu

were extracted for malware detection. A deep belief network model was used to achieve a detection accuracy of 96% on a dataset consisting of 250 malicious samples and 250 benign samples. Robin Nix *et al.* [67] use a deep neural network to identify and classify malware. It decompiles APK files to obtain smali files and designs a parser to extract core API call sequences from smali files, and then converts these API call sequences into feature vectors used as input to the convolutional neural network for classification. It achieved 99.4% detection accuracy on a dataset of 800 APK files.

Although the above methods achieve more than 95% detection accuracy, they are implemented on a small data sample set, and the reference significance needs to be considered.

Since collecting sample data is a very time-consuming and energy-consuming project, most researchers use their sample database and do not want to make it public.

The more training samples and test samples in the dataset, the more types of malware, and the real-time update of sample data can improve the deep learning model's generalization ability, thereby achieving excellent accuracy in real-world Android malware detection. As shown in Table 4, the currently used data sets' size and collection time are uneven, resulting in a lack of comparability between the research results. It is very one-sided to rely on the performance metrics of the neural network model for evaluation. Therefore, an open and standardized sample database is needed as the standard.

B. FEATURE SELECTION

The Android malware detection used by deep learning is mainly based on permission features, API call features, and system call features. These features are not sufficient to completely summarize all the characteristics of Android applications. Selecting suitable and sufficient feature types has a positive effect on improving deep learning-based malware detection effectiveness.

Feature learning is essential for Android malware detection, and it can reduce feature size and improve detection accuracy.

Yuan *et al.* [38] studied deep learning using association rule mining technology to characterize the features of Android malware and conducted experiments on the correlation between 192 features to find associations often used only by Android malware. The frequency of features in malicious and benign samples only reflects the trend of feature differences between them.

Zhao *et al.* [69] proposed a feature extraction and selection tool, which is a feature-based machine learning method for malware detection on the Android platform. The tool selects features based on how often they appear in Android applications, and it uses the selected features to detect Android malware.

Chakradeo *et al.* [70] extracted 182 features through static analysis. It relies on attribute-based selection: features extracted independently and subset-based extraction, which considers the dependencies between features.

Shabtai and Elovici *et al.* [71] used a feature ranking algorithm to select a subset of features from 88 features and the top 10, 20, and 50 features from the original data set.

Guoyin *et al.* [72] used permissions, API calls, component declarations, and string information as features to be extracted, and used association rules and chi-square statistical methods to perform feature screening on all extracted features, removing unaffected and redundant features.

Shabtai *et al.* [73] analyzed the network traffic data of Android applications and used feature selection algorithms to select the most useful features in the network traffic data.

Cong *et al.* [74] used the Apriori algorithm of association rules on the extracted permission features, namely connection and pruning, to select and process the permission feature set.

Yanping *et al.* [75] used the information gain algorithm in their research to select features and select important and suitable features for detection.

Siqi *et al.* [76] used static analysis and dynamic analysis to extract malicious code behavior and required permission features and used the chi-square test to pre-process the extracted features to eliminate features that are not related to the required features.

More detailed features should be extracted to characterize Android applications. A more comprehensive and fine-grained feature set can cover more aspects of Android malware and be used to characterize better and detect malware, such as combining static and dynamic features to improve malware detection accuracy.

C. FEATURE VECTORIZED REPRESENTATION

One method is to design a neural network model for Android malware detection and train it from scratch. Using transfer learning technology and using existing trained neural networks, and applying them to new and different data sets to implement malware detection through feature extraction and parameter tuning is another method. Transfer learning refers to copying knowledge from a trained network to a new network to solve similar problems. Currently, this method is mainly used for malware detection using deep learning.

In order to use the existing trained deep neural network to deal with the problem of Android malware detection, the problem description needs to be converted into a vector, because the use of deep learning in any field first needs to solve the problem of vectorization. The input is converted into a digital vector, and the existing deep neural network model processes the vector.

The Android malware detection problem can be transformed into a problem with proven solutions, such as converting binary program files into images and then using image recognition methods to classify and identify them [77].

Zegzhda *et al.* [40] constructed a control flow graph through a smali file to obtain the features of the API call sequence, then mapped the corresponding protection level according to the permissions required for the API call, and then converted the API call sequence and protection level into pixels of the RGB image. A hash function is used to calculate a 32-bit hash value according to the API call signature value, and the 24 least significant bits are obtained from the hash value, and the protection level corresponding to the permissions required by the API call is normalized. The 3-byte hash value and protection level are compiled into RGB pixel values and Alpha channel values. Finally, the RGBA pixels are converted into RGB images with black background according to the function.

Ganesh *et al.* [41] used static analysis to extract 138 permission features. According to the number, the extracted permission features are numbered from 0 to 137 and then correspond to the PNG image pixel point's position number. Because there are 138 features, the image size is 12×12, and zero paddings are performed for the remaining 6 pixels. When the application has the feature whose number is 2, the value of the second pixel of the image is 1.

Huang and Kao [42] decompressed the APK file to obtain the classes.dex file and displayed it in the form of bytecode. Then hex was mapped from bytecode to the three channels of RGB's pixel values through rules, and then the DEX file is converted into an RGB image according to the RGB pixel values.

The above researchers have published some research results using the image method. These results show that malware's image mode is not fixed, and some conversion methods are simple. The use of image data to represent malware requires further research.

The most common method is to convert the static and dynamic features of Android applications into word vectors and use some mathematical models for vectorized representation in addition to the image method. These mathematical models mainly come from existing results in natural language processing, such as Word2Vec [58], Glove [78].

Yuan *et al.* [38], Zhu *et al.* [79], Karbab *et al.* [19] all use feature vector generation methods based on existence, whose elements only represent the existence of features in a malicious feature database, such as strings, permission, component, and environment feature vectors. The feature values in the malicious feature database correspond to the feature vector elements, and each feature value is searched for in the features extracted from the input application. If there is no specific feature value in the extracted features, the absence of the feature value is represented as 0; otherwise, the feature value's presence is represented as 1 in the vector.

Kim *et al.* [16] used the similarity-based feature vector generation method, and the features were in the form of frequency lists. The frequency values can be very different, so it first uses the min-max scaling method to normalize the input application features so that they are in the range [0, 1]. Then, using the Euclidean distance metric, each malware representative (the centroid of the cluster) in the malware database is compared with the input application features. Among each malware representative's distances, the minimum distance is selected to be converted into similarity, and the calculated similarity is recorded in the corresponding element of the feature vector.

Li *et al.* [80] defined the $|S|$ dimensional vector space to map feature sets into Boolean expressions to capture these features' dependencies. The application x is mapped into this space by constructing a vector V (x) so that for each feature s extracted from x, the value of the corresponding dimension is set to 1, and the values of all other dimensions are set to 0.

Nix *et al.* [59] use one-hot encoding to encode each API call and convert the API call to an integer index and then to the corresponding one-hot vector.

Juwono *et al.* [60] detect and classify the Cuckoo sandbox's dynamic behavior log. For each sample, the original dynamic behavior report is JSON structured data converted to text format. Each line in the text represents a dynamic behavior. It reads all dynamic behavior logs to obtain all dynamic behaviors that have occurred and serve them as a lexicon. Labeling each word in the lexicon with consecutive numbers, it can get the mapping of dynamic behavior to the label id. A fixed-length vector represents each word in the lexicon, and the length of this vector is a selectable parameter. The sample's dynamic behavior is converted into a corresponding id sequence via the lexicon, and then the sample is converted into a two-dimensional matrix according to the id sequence and the vector of each id in the lexicon.

How to efficiently and accurately vectorize the features or programs affects the detection effect of malware. The existing vectorization methods are relatively simple and cannot wholly summarize the features of Android applications. The feature vectors are too sparse. Using discrete numerical features instead of binary numerical features for vectorized representation of features can be considered in the future. For example, if permission is requested three times or a dynamic behavior occurs three times, their corresponding feature values can be set to a discrete value of 3 instead of 1 (a binary value).

D. NEURAL NETWORK TRAINING

The neural network model is the core part of the implementation of malware detection systems. The design and optimization of neural network models are the keys to improving malware detection systems' detection and classification capabilities. Neural network models still have overfitting, difficult to adjust parameters, gradient vanishment, and initialization problems. Although there are currently corresponding solutions to these problems, these methods still need to be improved and supplemented.

1) OVERFITTING

Overfitting is a common problem in deep learning. It means that the accuracy of deep learning model prediction increases on the training set, but decreases on the test set. It also shows that the deep learning model only memorizes the training set's data features and has poor generalization performance.

To train a deep neural network, it must build a vast data set to take advantage of deep learning to avoid overfitting. Overfitting is prone to occur when the data set is small. Transfer learning can be used as a strategy to avoid overfitting, using existing neural networks to detect Android malware on small data sets.

2) PARAMETERS ARE DIFFICULT TO ADJUST

The deep learning model parameters are difficult to adjust, and different parameter settings have a significant impact on the experimental results obtained in the end. Deep learning is usually not a convex optimization problem, and it is full of locally optimal solutions. A deep learning model may have many locally optimal solutions that can achieve the desired effect, but the optimal global solution is easily overfitted.

For the parameter setting of deep learning models, it is necessary to repeatedly adjust to reach the desired result, so adaptive methods such as Adam [81], Adadelta [82] appear, and these methods can reduce the burden of adjusting parameters. Besides, the grid search method can be used to find the best parameters of the deep learning model. Nevertheless, the difficulty of adjusting the parameters is still a severe problem in deep learning.

3) GRADIENT VANISHMENT

Too deep a network will bring Gradient Vanishment problems and affect deep neural network training. At the beginning of neural network training, Sigmoid was used as the activation function. When the number of layers of the neural network is large, the Sigmoid function's gradient value will gradually decrease in backpropagation, and it will decrease exponentially after multi-layer transmission. The gradient value becomes very small when passed to the previous layers. In this case, updating the neural network parameters based on the feedback of the training data will be very slow, and the role of training cannot be achieved basically.

4) INITIALIZATION

The method of neural network initialization is also needed to be considered. The initialization method is more important or even critical for complex convolutional networks, recurrent networks, or deeper fully-connected networks. The method of neural network initialization will directly affect the network training time and effect.

5) ADVERSARY ATTACK

Adversarial sample attacks based on a generative adversarial network (GAN) pose a threat to the Android malware detection system using deep learning [83], [84]. Like many machine learning models, neural networks are be easily manipulated by their inputs [85]. These operations take the form of adversarial samples, that is, by adding selected and usually artificially distinguished perturbations to the input to force the target model to misclassify the samples. These samples take advantage of the training phase's defects and the inherent linearity of the components used by the model, even if the entire model is non-linear. This attack scenario is mainly due to the lack of sufficient data for deep learning models. When using a deep learning model for classification tasks, because there is scarcely enough data to help the model make decisions in the entire feature space, the discriminant model will expand the distance between samples and decision boundaries to obtain better classification results. At the same time, the area of each category in the feature space is expanded. The advantage of this method is that it can make classification easier. However, the disadvantage is that it includes some logical feature spaces that do not belong to the current category, allowing attackers to generate adversarial samples from this feature space [86].

To a certain extent, it can protect the Android malware detection system based on deep learning by deploying an adversary sample detector to filter adversary samples. Heng Li *et al.* [87]. proposed a new adversary sample attack method based on bi-objective GAN. More than 95% of the adversary samples generated by this attack method successfully misled the Android malware detection system equipped with an adversary sample detector. Heng Li *et al.* [87] used bi-objective GAN to generate adversary samples with incomplete information. Bi-objective GAN extended the traditional GAN by using two discriminators with different targets. The generator generated candidate adversary samples. Discriminator 1 tried to distinguish between malicious samples and benign samples. Discriminator 2 tried to distinguish between adversary samples and benign samples. In the proposed GAN model, two discriminator boot generators achieve two goals simultaneously, namely, breaking through the firewall (adversary sample detector) and avoiding malware detection.

6) EVASION ATTACK

In evasion attacks, the attacker manipulates malicious samples during testing so that a trained classifier will misclassify them as benign without affecting the training data. Therefore, whether it is a targeted attack or an indiscriminate attack, it depends on whether the attacker targets a specific computer or conducts an indiscriminate attack. The goal of the attacker is to destroy system integrity [88].

Biggio *et al.* [89] proposed a simple algorithm for avoiding classifiers with discriminative discriminant functions. Studies show that even if the attacker's knowledge of the classification system being attacked is limited, the attacker can only understand the classifier's copy from a small proxy data set and can still avoid the detection of the classification model with a high probability.

7) CONCEPT DRIFT

Because attackers frequently change malware to avoid being detected by deep learning models, when using a deep neural network to classify malware, concept drift often occurs [90]. Deep learning faces the problem of concept drift [91], that is, the detection effect of detection model based on deep learning will decline over time [92]. On the premise that the distribution of training data and test data is stable, the deep learning model has good prediction performance. However, malware usually changes constantly to avoid deep learning model detection, leading to changes in data distribution and degradation of deep learning models [93]. When the detection model based on deep learning uses the old data set, it is prone to concept drift. In [94], they suggest updating models and updating datasets to deal with concept drift. How to effectively solve the problem of concept drift is a problem to be solved.

8) POISONING ATTACK

A poisoning attack is a typical security threat in the process of neural network training. In data poisoning attacks, adversaries create artificial associations between inputs and tags by interfering with inputs or modifying tags. In the training process of the model, malicious data, namely toxic data, is added. The machine learning model trained on toxic data will learn the incorrect association between input and label and become inaccurate for clean input, thus reducing the model's accuracy [95].

Many poisoning attacks have been found for various machine learning models, such as [96], [97]. In [96], they develop a Projected Gradient Ascent (PGA) algorithm to compute Label Contamination Attack (LCA) on a family of empirical risk minimizations and develop a defense algorithm to identify the data points that are most likely to be attacked. Feature selection is not clear whether its use may be beneficial or even counterproductive when intelligent attackers poison training data. In [97], they shed light on this issue by providing a framework to investigate popular feature selection methods' robustness. In [98], [99], they put forward the countermeasures to prevent poisoning attack and realize the purpose of detecting and filtering poisoning data. In [98], they construct approximate upper bounds on the loss across a broad family of attacks, for defenders that first perform outlier removal followed by empirical risk minimization. Their bound comes paired with a candidate attack that often nearly matches the upper bound, giving them a powerful tool for quickly assessing defenses on a given dataset. In [99], they study the susceptibility of collaborative deep learning systems to adversarial poisoning attacks. They demonstrate that the attacks have a 99% success rate for misclassifying specific target data while poisoning only 10% of the entire training dataset for collaborative deep learning systems. They propose a system that detects malicious users and generates an accurate model to defend against poisoning attacks.

VII. FUTURE DIRECTIONS

The application of deep learning in Android malware detection has been a research hotspot in recent years. It has crucial significance for cyberspace security and has attracted widespread attention. Based on the existing problems in the research of deep learning applied to Android malware detection, the following four future research directions are pointed out:

A. DEFEND AGAINST ADVERSARIAL ATTACKS

Existing deep learning models for Android malware detection and classification are susceptible to threats from adversarial attacks. Research on the adversarial attacks faced by deep learning applied to Android malware detection is in its infancy. The solution is mainly concentrated in the field of computer vision. It is using adversarial generated adversarial samples to retrain the deep learning classifier for adversary training. The purpose of adversary training is to modify the abstractness of the deep learning model. Choosing an appropriate number of adversary training samples is a challenge.

YanJinpei *et al.* [86] used several methods to defend against adversary attacks. Firstly, regularization was added to the deep learning model so that the model would not overfit the training set and promote the closure of the benign sample feature space. L2 regularization is used to maintain conservative discrimination results for unknown feature spaces to prevent adversarial samples using these feature spaces from deceiving malware detection classifiers. Secondly, adversarial training is conducted by making adversarial samples in advance, then using these samples to train deep learning models through online learning, thereby enhancing deep learning models' ability to resist adversarial attacks. Thirdly, using ensemble learning with different classifiers can prevent adversarial attacks to a certain extent. Fourthly, selecting features of a specific category, such as *N*-gram. Fifthly, using the method of gradient masking [100], the model outputs hard decisions (predicted target categories), instead of outputting the probabilities of different categories, so that it is difficult for an attacker to obtain useful gradients to construct attack samples.

Filtering out the adversary samples by an adversary sample detector can protect the Android malware detection system based on deep learning technology. An adversary sample detector's construction can be divided into three stages: data collection, model training, and adversary sample detection. Benign samples are obtained by extracting features from benign and malicious applications, then adversary samples are generated by some methods, and classifiers are trained by benign samples, adversary samples, and their labels. The labels of adversary samples are set as 1; the benign sample label is set to 0. Finally, a trained classifier is used as the firewall to prevent the ring breaking detection system [87].

After analyzing the fundamental defects of the deep neural network, Wang *et al.* [101] pointed out that the effectiveness of the methods to defend against the adversary attack by increasing the training data and the complexity of deep learning model is limited, and these methods can not provide the theoretical guarantee of the robustness of the attack. Qinglong Wang *et al.* proposed a new adversary technology to prevent attackers from building useful adversary samples by randomly eliminating the samples' features. In the data set composed of 14679 malware variants and 17399 benign applications, the proposed technology's robustness is verified theoretically. The experimental results show that the technology improves the deep neural network's robustness to the adversary samples while maintaining high classification accuracy.

At present, there is not much research on defense against adversarial attacks, but with the increase of security requirements in the future, adversarial attacks will become an important research direction. A complete and detailed analysis of adversarial attacks will be needed, and relevant experimental evaluations will be conducted.

B. APPLICATION MULTI-CLASSIFICATION

At present, the detection of Android malware using deep learning is concentrated in the field of two classifications, which is merely distinguishing whether an Android application is malicious or not. There is not much research on the multi-classification of Android malware families. The Android malware multi-classification further explains Android malware's maliciousness, indicating the category to which the malware belongs.

As shown in Table 5, it is the family names and corresponding sample numbers of the 20 largest malware families in the DREBIN dataset. Most data sets only include malicious samples and benign samples, and there is no classification of malware families. With the explosive growth of malware, more and more malware types have appeared, such as the emergence of ransomware applications. With the deepening of cyberspace security research, malware detection's security requirements will continue to increase, and research on the multi-classification of malware families will gradually receive more attention.

TABLE 5. 20 malware family types in the DREBIN dataset.

Id	Family	Number	Id	Family	Number
A	FakeInstaller	925	K	Adrd	91
B	DroidKungFu	667	L	DroidDream	81
C	Plankton	625	M	LinuxLotoor	70
D	Opfake	613	N	GoldDream	69
E	GingerMaster	339	O	MobileTx	69
F	BaseBridge	330	P	FakeRun	61
G	Iconosys	152	Q	SendPay	59
H	Kmin	147	R	Gappusin	58
I	FakeDoc	132	S	Imlog	43
J	Geinimi	92	T	SMSreg	41

C. HYBRID ANALYSIS

The current research focuses on the combination of static analysis and deep learning. There is less research on the use of dynamic analysis for detection, and less research on the hybrid detection that combines static analysis and dynamic analysis. It is mainly due to the long detection time of dynamic analysis and the high resource consumption rate. On the contrary, static analysis has high detection efficiency and fast speed. However, static analysis cannot effectively analyze applications that use code obfuscation, shell protection, encryption, and repackaging technologies. With the improvement of attackers' attack methods, malware is capable of evading static detection. Static detection has been unable to meet the increasing security requirements in the future, and dynamic detection can effectively analyze malware using code obfuscation technologies. The combination of static features and dynamic features can adequately characterize malware, and it is useful to use hybrid detection to improve the accuracy of malware detection.

There are some inherent limitations in the method of dynamic analysis only. When a lack of necessary resources, UI operations, or library, the dynamic analysis platform DroidBox may not trigger malware's malicious behavior in

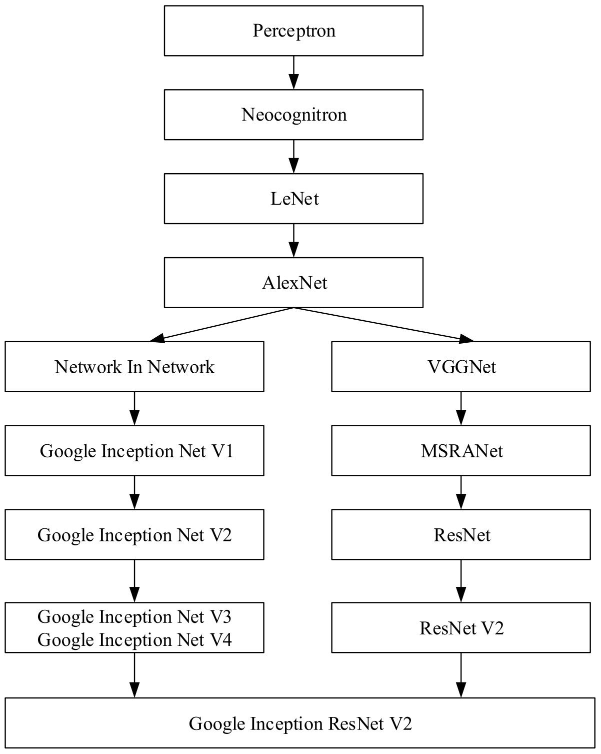

FIGURE 7. Convolutional neural network development diagram.

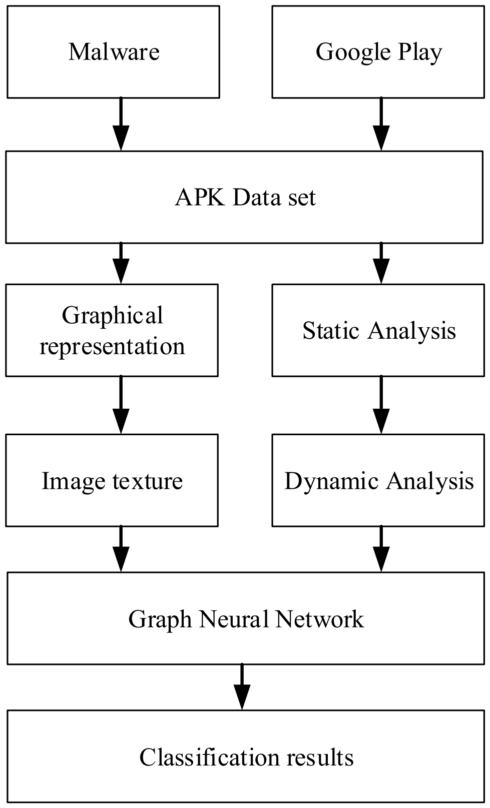

FIGURE 8. Graph neural network detection architecture.

some cases. For example, if the malware attempts to acquire the malicious load, use a third-party server to connect, command, and control the server to obtain instructions when these servers cannot connect, the malware may not continue to perform its malicious operations. Besides, some malware will only perform malicious behavior when some system events (such as receiving SMS) happen or users perform specific operations (such as entering legal user identity and password). If the analysis platform cannot execute these events, the malicious behavior of this malware cannot be triggered. Malware may detect its runtime environment and stop performing malicious operations, in which case the analysis platform will not record valid behavior information. This evasion technology can be realized through the Android emulator's identification or monitoring process. Because of the above problems in the dynamic analysis, static analysis is needed to assist analysis and judgment to identify malware more accurately.

D. DEEP LEARNING MODEL

Deep learning models are a core part of the implementation of malware detection systems based on deep learning. The design and optimization of models are the keys to improving malware detection systems' detection and classification capabilities. The deep learning models currently used for detection are Deep Belief Network, Convolutional Neural Network, Recurrent Neural Network, and DeepAutoEncoder. In order to improve the detection effect of deep learning models, the approach can be divided into two types: improved adjustment and optimization on the structure of the neural network, the left branch of Figure 7, and the other is the increase of the depth of the neural network, as shown on the right branch of Figure 7.

In addition to the current mainstream deep learning detection models, applying new deep learning models to malware detection is also a research direction, such as the application of Graph Neural Network [102] for detection.

Graph Neural Networks (GNN) is a useful framework for graph representation learning [103].GNN follows a neighborhood aggregation scheme in which the representation vectors of a node are calculated by recursively aggregating and transforming its neighboring nodes' representation vectors. Many graph neural network variants have been proposed, and the latest results have been achieved on both node and graph classification tasks. We can try representing the APK file with a graph and use graph neural networks to detect malware through image texture features combined with dynamic behavior features, as shown in Figure 8.

VIII. SUMMARY

As the Android operating system occupies the leading market share and its open-source nature, its security is increasingly threatened and challenged. In order to deal with the explosive growth of Android malware, research on Android malware detection using deep learning has been a research hotspot in recent years, but currently lacking a detailed and comprehensive introduction to the research progress of Android malware detection using deep learning. Firstly, the paper introduces the basic knowledge of Android malware detection technology. Then it sort outs, analyzes, and summarizes Android malware detection's latest research progress based on deep learning and summarizes the Android malware detection architecture and detection scheme based on deep learning. Then it analyzes the problems and challenges of Android malware detection using deep learning. Finally, it looks forward to the future research direction.

ACKNOWLEDGMENT

The authors would like to thank anonymous reviewers for their valuable comments.

REFERENCES

[1] Y. Feng, S. Anand, I. Dillig, and A. Aiken, "Apposcopy: Semantics-based detection of Android malware through static analysis," in *Proc. 22nd ACM SIGSOFT Int. Symp. Found. Softw. Eng.*, 2014, pp. 576–587.

[2] M. Y. Wong and D. Lie, "IntelliDroid: A targeted input generator for the dynamic analysis of Android malware," in *Proc. NDSS*, vol. 16, 2016, pp. 21–24.

[3] S. K. Dash, G. Suarez-Tangil, S. Khan, K. Tam, M. Ahmadi, J. Kinder, and L. Cavallaro, "DroidScribe: Classifying Android malware based on runtime behavior," in *Proc. IEEE Secur. Privacy Workshops (SPW)*, May 2016, pp. 252–261.

[4] S. Rasthofer, S. Arzt, M. Miltenberger, and E. Bodden, "Harvesting runtime values in Android applications that feature anti-analysis techniques," in *Proc. NDSS*, 2016.

[5] J. Qiu, S. Nepal, W. Luo, L. Pan, Y. Tai, J. Zhang, and Y. Xiang, "Data-driven Android malware intelligence: A survey," in *Proc. Int. Conf. Mach. Learn. Cyber Secur.* Springer, 2019, pp. 183–202.

[6] M. Rhode, P. Burnap, and K. Jones, "Early-stage malware prediction using recurrent neural networks," *Comput. Secur.*, vol. 77, pp. 578–594, Aug. 2018.

[7] K. Bakour, H. M. Ünver, and R. Ghanem, "The Android malware detection systems between hope and reality," *Social Netw. Appl. Sci.*, vol. 1, no. 9, p. 1120, Sep. 2019.

[8] R. Mahmood, N. Mirzaei, and S. Malek, "EvoDroid: Segmented evolutionary testing of Android apps," in *Proc. 22nd ACM SIGSOFT Int. Symp. Found. Softw. Eng.*, 2014, pp. 599–609.

[9] T. Vidas, J. Tan, J. Nahata, C. L. Tan, N. Christin, and P. Tague, "A5: Automated analysis of adversarial Android applications," in *Proc. 4th ACM Workshop Secur. Privacy Smartphones Mobile Devices*, 2014, pp. 39–50.

[10] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[11] E. Alpaydin, *Introduction to Machine Learning*. Cambridge, MA, USA: MIT Press, 2020.

[12] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, *Deep Learning*, vol. 1. Cambridge, MA, USA: MIT Press, 2016.

[13] A. Pektaş and T. Acarman, "Learning to detect Android malware via opcode sequences," *Neurocomputing*, vol. 396, pp. 599–608, Jul. 2019.

[14] Z. Yuan, Y. Lu, Z. Wang, and Y. Xue, "Droid-sec: Deep learning in Android malware detection," in *Proc. ACM Conf. SIGCOMM*, 2014, pp. 371–372.

[15] X. Su, D. Zhang, W. Li, and K. Zhao, "A deep learning approach to Android malware feature learning and detection," in *Proc. IEEE Trustcom/BigDataSE/ISPA*, Aug. 2016, pp. 244–251.

[16] T. Kim, B. Kang, M. Rho, S. Sezer, and E. G. Im, "A multimodal deep learning method for Android malware detection using various features," *IEEE Trans. Inf. Forensics Security*, vol. 14, no. 3, pp. 773–788, Mar. 2019.

[17] W. Wang, M. Zhao, and J. Wang, "Effective Android malware detection with a hybrid model based on deep autoencoder and convolutional neural network," *J. Ambient Intell. Humanized Comput.*, vol. 10, no. 8, pp. 3035–3043, Aug. 2019.

[18] Y. Zhang, Y. Yang, and X. Wang, "A novel Android malware detection approach based on convolutional neural network," in *Proc. 2nd Int. Conf. Cryptogr., Secur. Privacy*, 2018, pp. 144–149.

[19] E. B. Karbab, M. Debbabi, A. Derhab, and D. Mouheb, "MalDozer: Automatic framework for Android malware detection using deep learning," *Digit. Invest.*, vol. 24, pp. S48–S59, 2018.

[20] S. Luo, Z. Liu, B. Ni, H. Wang, H. Sun, and Y. Yuan, "Android malware analysis and detection based on attention-cnn-lstm," *J. Comput.*, vol. 14, no. 1, pp. 31–44, 2019.

[21] J. Booz, J. McGiff, W. G. Hatcher, W. Yu, J. Nguyen, and C. Lu, "Tuning deep learning performance for Android malware detection," in *Proc. 19th IEEE/ACIS Int. Conf. Softw. Eng., Artif. Intell., Netw. Parallel/Distrib. Comput. (SNPD)*, Jun. 2018, pp. 140–145.

[22] Z. Ma, H. Ge, Z. Wang, Y. Liu, and X. Liu, "Droidetec: Android malware detection and malicious code localization through deep learning," 2020, *arXiv:2002.03594*. [Online]. Available: http://arxiv.org/abs/2002.03594

[23] Z. Ren, H. Wu, Q. Ning, I. Hussain, and B. Chen, "End-to-end malware detection for Android IoT devices using deep learning," *Ad Hoc Netw.*, vol. 101, Apr. 2020, Art. no. 102098.

[24] T. Chen, Q. Mao, M. Lv, H. Cheng, and Y. Li, "DroidvecDeep: Android malware detection based on Word2Vec and deep belief network," *TIIS*, vol. 13, no. 4, pp. 2180–2197, 2019.

[25] X. Su, W. Shi, X. Qu, Y. Zheng, and X. Liu, "DroidDeep: Using deep belief network to characterize and detect Android malware," *Soft Comput.*, pp. 1–14, 2020.

[26] K. Zou, X. Luo, P. Liu, W. Wang, and H. Wang, "ByteDroid: Android malware detection using deep learning on bytecode sequences," in *Proc. Chin. Conf. Trusted Comput. Inf. Secur.* Springer, 2019, pp. 159–176.

[27] N. He, T. Wang, P. Chen, H. Yan, and Z. Jin, "An Android malware detection method based on deep autoencoder," in *Proc. Artif. Intell. Cloud Comput. Conf.*, 2018, pp. 88–93.

[28] W. Y. Lee, J. Saxe, and R. Harang, "SeqDroid: Obfuscated Android malware detection using stacked convolutional and recurrent neural networks," in *Deep Learning Applications for Cyber Security*. Springer, 2019, pp. 197–210.

[29] D. Zhu, T. Xi, P. Jing, D. Wu, Q. Xia, and Y. Zhang, "A transparent and multimodal malware detection method for Android apps," in *Proc. 22nd Int. ACM Conf. Modeling, Anal. Simulation Wireless Mobile Syst.*, 2019, pp. 51–60.

[30] X. Qin, F. Zeng, and Y. Zhang, "MSNdroid: The Android malware detector based on multi-class features and deep belief network," in *Proc. ACM Turing Celebration Conf.-China*, 2019, pp. 1–5.

[31] D. Li, L. Zhao, Q. Cheng, N. Lu, and W. Shi, "Opcode sequence analysis of Android malware by a convolutional neural network," *Concurrency Comput. Pract. Exper.*, vol. 32, no. 18, p. e5308, 2020.

[32] S. Hou, A. Saas, L. Chen, and Y. Ye, "Deep4MalDroid: A deep learning framework for Android malware detection based on linux kernel system call graphs," in *Proc. IEEE/WIC/ACM Int. Conf. Web Intell. Workshops (WIW)*, Oct. 2016, pp. 104–111.

[33] F. Martinelli, F. Marulli, and F. Mercaldo, "Evaluating convolutional neural network for effective mobile malware detection," *Procedia Comput. Sci.*, vol. 112, pp. 2372–2381, 2017.

[34] C.-W. Yeh, W.-T. Yeh, S.-H. Hung, and C.-T. Lin, "Flattened data in convolutional neural networks: Using malware detection as case study," in *Proc. Int. Conf. Res. Adapt. Convergent Syst.*, 2016, pp. 130–135.

[35] M. K. Alzaylaee, S. Y. Yerima, and S. Sezer, "Dl-Droid: Deep learning based Android malware detection using real devices," *Comput. Secur.*, vol. 89, Feb. 2020, Art. no. 101663.

[36] P. Faruki, B. Buddhadev, B. Shah, A. Zemmari, V. Laxmi, and M. S. Gaur, "DroidDivesDeep: Android malware classification via low level monitorable features with deep neural networks," in *Proc. Int. Conf. Secur. Privacy*. Springer, 2019, pp. 125–139.

[37] P. Feng, J. Ma, C. Sun, X. Xu, and Y. Ma, "A novel dynamic Android malware detection system with ensemble learning," *IEEE Access*, vol. 6, pp. 30996–31011, 2018.

[38] Z. Yuan, Y. Lu, and Y. Xue, "Droiddetector: Android malware characterization and detection using deep learning," *Tsinghua Sci. Technol.*, vol. 21, no. 1, pp. 114–123, Feb. 2016.

[39] L. Xu, D. Zhang, N. Jayasena, and J. Cavazos, "HADM: Hybrid analysis for detection of malware," in *Proc. SAI Intell. Syst. Conf.* Springer, 2016, pp. 702–724.

[40] P. Zegzhda, D. Zegzhda, E. Pavlenko, and G. Ignatev, "Applying deep learning techniques for Android malware detection," in *Proc. 11th Int. Conf. Secur. Inf. Netw.*, 2018, pp. 1–8.

[41] M. Ganesh, P. Pednekar, P. Prabhuswamy, D. S. Nair, Y. Park, and H. Jeon, "CNN-based Android malware detection," in *Proc. Int. Conf. Softw. Secur. Assurance (ICSSA)*, Jul. 2017, pp. 60–65.

[42] T. H.-D. Huang and H.-Y. Kao, "R2-D2: ColoR-inspired convolutional NeuRal network (CNN)-based Android malware detections," in *Proc. IEEE Int. Conf. Big Data (Big Data)*, Dec. 2018, pp. 2633–2642.
[43] L. Shiqi, T. Shengwei, Y. Long, Y. Jiong, and S. Hua, "Android malicious code classification using deep belief network," *KSII Trans. Internet Inf. Syst.*, vol. 12, no. 1, 2018.
[44] Y.-S. Yen and H.-M. Sun, "An Android mutation malware detection based on deep learning using visualization of importance from codes," *Microelectron. Rel.*, vol. 93, pp. 109–114, Feb. 2019.
[45] Z. Xu, K. Ren, S. Qin, and F. Craciun, "CDGDroid: Android malware detection based on deep learning using CFG and DFG," in *Proc. Int. Conf. Formal Eng. Methods*. Springer, 2018, pp. 177–193.
[46] A. Pektaş and T. Acarman, "Deep learning for effective Android malware detection using API call graph embeddings," *Soft Comput.*, vol. 24, no. 2, pp. 1027–1043, 2019.
[47] D. Li, Z. Wang, and Y. Xue, "Fine-grained Android malware detection based on deep learning," in *Proc. IEEE Conf. Commun. Netw. Secur. (CNS)*, May 2018, pp. 1–2.
[48] Y. Fang, Y. Gao, F. Jing, and L. Zhang, "Android malware familial classification based on DEX file section features," *IEEE Access*, vol. 8, pp. 10614–10627, 2020.
[49] F. Mercaldo and A. Santone, "Deep learning for image-based mobile malware detection," *J. Comput. Virol. Hacking Techn.*, vol. 16, pp. 157–171, 2020.
[50] D. Vasan, M. Alazab, S. Wassan, H. Naeem, B. Safaei, and Q. Zheng, "IMCFN: Image-based malware classification using fine-tuned convolutional neural network architecture," *Comput. Netw.*, vol. 171, Apr. 2020, Art. no. 107138.
[51] Y. Sun, Y. Chen, Y. Pan, and L. Wu, "Android malware family classification based on deep learning of code images," *IAENG Int. J. Comput. Sci.*, vol. 46, no. 4, 2019.
[52] M. Fan, J. Liu, X. Luo, K. Chen, T. Chen, Z. Tian, X. Zhang, Q. Zheng, and T. Liu, "Frequent subgraph based familial classification of Android malware," in *Proc. IEEE 27th Int. Symp. Softw. Rel. Eng. (ISSRE)*, Oct. 2016, pp. 24–35.
[53] D. Maiorca, D. Ariu, I. Corona, M. Aresu, and G. Giacinto, "Stealth attacks: An extended insight into the obfuscation effects on Android malware," *Comput. Secur.*, vol. 51, pp. 16–31, Jun. 2015.
[54] R. H. R. Hahnloser, R. Sarpeshkar, M. A. Mahowald, R. J. Douglas, and H. S. Seung, "Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit," *Nature*, vol. 405, no. 6789, pp. 947–951, Jun. 2000.
[55] L. Bottou, "Large-scale machine learning with stochastic gradient descent," in *Proc. COMPSTAT*. Springer, 2010, pp. 177–186.
[56] S. Lange and M. Riedmiller, "Deep auto-encoder neural networks in reinforcement learning," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Jul. 2010, pp. 1–8.
[57] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2012, pp. 1097–1105.
[58] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," 2013, *arXiv:1301.3781*. [Online]. Available: http://arxiv.org/abs/1301.3781
[59] R. Nix and J. Zhang, "Classification of Android apps and malware using deep neural networks," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, May 2017, pp. 1871–1878.
[60] J. T. Juwono, C. Lim, and A. Erwin, "A comparative study of behavior analysis sandboxes in malware detection," in *Proc. Int. Conf. New Media (CONMEDIA)*, 2015, p. 73.
[61] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, K. Rieck, and C. Siemens, "DREBIN: Effective and explainable detection of Android malware in your pocket," in *Proc. NDSS*, vol. 14, 2014, pp. 23–26.
[62] Y. Zhou and X. Jiang, "Dissecting Android malware: Characterization and evolution," in *Proc. IEEE Symp. Secur. Privacy*, May 2012, pp. 95–109.
[63] Y. Li, J. Jang, X. Hu, and X. Ou, "Android malware clustering through malicious payload mining," in *Proc. Int. Symp. Res. Attacks, Intrusions, Defenses*. Springer, 2017, pp. 192–214.
[64] F. Wei, Y. Li, S. Roy, X. Ou, and W. Zhou, "Deep ground truth analysis of current Android malware," in *Proc. Int. Conf. Detection Intrusions Malware, Vulnerability Assessment*. Springer, 2017, pp. 252–276.
[65] M. Lindorfer, M. Neugschwandtner, and C. Platzer, "MARVIN: Efficient and comprehensive mobile app classification through static and dynamic analysis," in *Proc. IEEE 39th Annu. Comput. Softw. Appl. Conf.*, vol. 2, Jul. 2015, pp. 422–433.
[66] A. F. A. Kadir, N. Stakhanova, and A. A. Ghorbani, "Android botnets: What URLs are telling us," in *Proc. Int. Conf. Netw. Syst. Secur.* Springer, 2015, pp. 78–91.
[67] R. A. Nix, "Applying deep learning techniques to the analysis of Android APKs," Tech. Rep., 2016.
[68] L. Li, J. Gao, M. Hurier, P. Kong, T. F. Bissyandé, A. Bartel, J. Klein, and Y. Le Traon, "AndroZoo++: Collecting millions of Android apps and their metadata for the research community," 2017, *arXiv:1709.05281*. [Online]. Available: http://arxiv.org/abs/1709.05281
[69] K. Zhao, D. Zhang, X. Su, and W. Li, "Fest: A feature extraction and selection tool for Android malware detection," in *Proc. IEEE Symp. Comput. Commun. (ISCC)*, Jul. 2015, pp. 714–720.
[70] S. Chakradeo, B. Reaves, P. Traynor, and W. Enck, "MAST: Triage for market-scale mobile malware analysis," in *Proc. 6th ACM Conf. Secur. Privacy Wireless Mobile Netw.*, 2013, pp. 13–24.
[71] A. Shabtai and Y. Elovici, "Applying behavioral detection on android-based devices," in *Proc. Int. Conf. Mobile Wireless Middlew., Operating Syst., Appl.* Springer, 2010, pp. 235–249.
[72] Z. Guoyin, Q. Jiaxing, and L. Xiaoguang, "Android malicious behavior detection method based on Bayesian network," *Comput. Eng. Appl.*, vol. 52, no. 17, pp. 16–23, 2016.
[73] A. Shabtai, L. Tenenboim-Chekina, D. Mimran, L. Rokach, B. Shapira, and Y. Elovici, "Mobile malware detection through analysis of deviations in application network behavior," *Comput. Secur.*, vol. 43, pp. 1–18, Jun. 2014.
[74] W. Cong, Z. Renbin, and L. Gang, "Bayesian Android malware detection technology based on correlation features," *Comput. Appl. Softw.*, vol. 34, no. 1, pp. 286–292, 2017.
[75] X. Yanping, W. Chunhua, H. Meijia, Z. Kangfeng, and Y. Shan, "Android malicious application detection technology based on improved naive Bayes," *J. Beijing Univ. Posts Telecommun.*, vol. 39, no. 2, pp. 43–47, 2016.
[76] Z. Siqi, "Android malware detection based on improved Bayesian classification," *Radio Commun. Technol.*, vol. 40, no. 6, pp. 73–76, 2014.
[77] L. Nataraj, S. Karthikeyan, G. Jacob, and B. Manjunath, "Malware images: Visualization and automatic classification," in *Proc. 8th Int. Symp. Vis. Cyber Secur.*, 2011, pp. 1–7.
[78] J. Pennington, R. Socher, and C. Manning, "GloVe: Global vectors for word representation," in *Proc. Conf. Empirical Methods Natural Lang. Process. (EMNLP)*, 2014, pp. 1532–1543.
[79] D. Zhu, H. Jin, Y. Yang, D. Wu, and W. Chen, "DeepFlow: Deep learning-based malware detection by mining Android application for abnormal usage of sensitive data," in *Proc. IEEE Symp. Comput. Commun. (ISCC)*, Jul. 2017, pp. 438–443.
[80] W. Li, Z. Wang, J. Cai, and S. Cheng, "An Android malware detection approach using weight-adjusted deep learning," in *Proc. Int. Conf. Comput., Netw. Commun. (ICNC)*, Mar. 2018, pp. 437–441.
[81] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014, *arXiv:1412.6980*. [Online]. Available: http://arxiv.org/abs/1412.6980
[82] M. D. Zeiler, "ADADELTA: An adaptive learning rate method," 2012, *arXiv:1212.5701*. [Online]. Available: http://arxiv.org/abs/1212.5701
[83] K. Grosse, N. Papernot, P. Manoharan, M. Backes, and P. McDaniel, "Adversarial perturbations against deep neural networks for malware classification," 2016, *arXiv:1606.04435*. [Online]. Available: http://arxiv.org/abs/1606.04435
[84] K. Grosse, N. Papernot, P. Manoharan, M. Backes, and P. McDaniel, "Adversarial examples for malware detection," in *Proc. Eur. Symp. Res. Comput. Secur.* Springer, 2017, pp. 62–79.
[85] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," 2013, *arXiv:1312.6199*. [Online]. Available: http://arxiv.org/abs/1312.6199
[86] J. Yan, Y. Qi, and Q. Rao, "Detecting malware with an ensemble method based on deep neural network," *Secur. Commun. Netw.*, vol. 2018, pp. 1–16, Mar. 2018.
[87] H. Li, S. Zhou, W. Yuan, J. Li, and H. Leung, "Adversarial-example attacks toward Android malware detection system," *IEEE Syst. J.*, vol. 14, no. 1, pp. 653–656, Mar. 2020.

[88] A. Demontis, M. Melis, B. Biggio, D. Maiorca, D. Arp, K. Rieck, I. Corona, G. Giacinto, and F. Roli, "Yes, machine learning can be more secure! A case study on Android malware detection," *IEEE Trans. Dependable Secure Comput.*, vol. 16, no. 4, pp. 711–724, Jul. 2019.
[89] B. Biggio, I. Corona, D. Maiorca, B. Nelson, N. Šrndić, P. Laskov, G. Giacinto, and F. Roli, "Evasion attacks against machine learning at test time," in *Proc. Joint Eur. Conf. Mach. Learn. Knowl. Discovery Databases*. Springer, 2013, pp. 387–402.
[90] D. Hu, Z. Ma, X. Zhang, P. Li, D. Ye, and B. Ling, "The concept drift problem in Android malware detection and its solution," *Secur. Commun. Netw.*, vol. 2017, pp. 1–13, 2017.
[91] Z. Wang, M. Tian, X. Zhang, J. Wang, Z. Liu, C. Jia, and I. You, "A hybrid learning system to mitigate botnet concept drift attacks," *J. Internet Technol.*, vol. 18, no. 6, pp. 1419–1428, 2017.
[92] X. Wang, Z. Wang, W. Shao, C. Jia, and X. Li, "Explaining concept drift of deep learning models," in *Proc. Int. Symp. Cyberspace Saf. Secur.* Springer, 2019, pp. 524–534.
[93] Z. Wang, M. Qin, M. Chen, C. Jia, and Y. Ma, "A learning evasive email-based P2P-like botnet," *China Commun.*, vol. 15, no. 2, pp. 15–24, Feb. 2018.
[94] K. Xu, Y. Li, R. H. Deng, and K. Chen, "DeepRefiner: Multi-layer Android malware detection system applying deep neural networks," in *Proc. IEEE Eur. Symp. Secur. Privacy (EuroS&P)*, Apr. 2018, pp. 473–487.
[95] J. Shen, X. Zhu, and D. Ma, "TensorClog: An imperceptible poisoning attack on deep neural network applications," *IEEE Access*, vol. 7, pp. 41498–41506, 2019.
[96] M. Zhao, B. An, W. Gao, and T. Zhang, "Efficient label contamination attacks against black-box learning models," in *Proc. IJCAI*, 2017, pp. 3945–3951.
[97] H. Xiao, B. Biggio, G. Brown, G. Fumera, C. Eckert, and F. Roli, "Is feature selection secure against training data poisoning?" in *Proc. Int. Conf. Mach. Learn.*, 2015, pp. 1689–1698.
[98] J. Steinhardt, P. W. W. Koh, and P. S. Liang, "Certified defenses for data poisoning attacks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 3517–3529.
[99] S. Shen, S. Tople, and P. Saxena, "AUROR: Defending against poisoning attacks in collaborative deep learning systems," in *Proc. 32nd Annu. Conf. Comput. Secur. Appl.*, 2016, pp. 508–519.
[100] N. Papernot, P. McDaniel, I. Goodfellow, S. Jha, Z. B. Celik, and A. Swami, "Practical black-box attacks against machine learning," in *Proc. ACM Asia Conf. Comput. Commun. Secur.*, 2017, pp. 506–519.
[101] Q. Wang, W. Guo, K. Zhang, A. G. Ororbia, X. Xing, X. Liu, and C. L. Giles, "Adversary resistant deep neural networks with an application to malware detection," in *Proc. 23rd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2017, pp. 1145–1153.
[102] J. Zhou, G. Cui, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, and M. Sun, "Graph neural networks: A review of methods and applications," 2018, *arXiv:1812.08434*. [Online]. Available: http://arxiv.org/abs/1812.08434
[103] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" 2018, *arXiv:1810.00826*. [Online]. Available: http://arxiv.org/abs/1810.00826

ZHIQIANG WANG was born in China, in 1985. He received the Ph.D. degree in information security from Xidian University. He is currently an Assistant Professor with the Department of Computer Science and Technology. His research interests include system security and network security.

QIAN LIU received the B.Eng. degree in electronic information engineering from North China Electric Power University, Beijing, China, in 2018. He is currently pursuing the master's degree in electronics and communication engineering. His current research interests include deep learning and information security.

YAPING CHI was born in China in 1969. She received the master's degree in computer science and technology from the Beijing University of Technology. She is currently a Professor with the Department of Cyberspace Security, Beijing Electronic Science and Technology Institute. Her research interests include network security and cloud computing security.