Hindawi
Applied Computational Intelligence and Soft Computing
Volume 2022, Article ID 4119500, 22 pages
https://doi.org/10.1155/2022/4119500

Research Article

A Low Computational Cost Method for Mobile Malware Detection Using Transfer Learning and Familial Classification Using Topic Modelling

Saket Acharya, Umashankar Rawat, and Roheet Bhatnagar

Department of Computer Science and Engineering, Manipal University Jaipur, Jaipur, Rajasthan, India

Correspondence should be addressed to Umashankar Rawat; umashankar.rawat@jaipur.manipal.edu

Received 15 December 2021; Revised 28 April 2022; Accepted 16 May 2022; Published 13 June 2022

Academic Editor: Shyi-Ming Chen

With the extensive use of Android applications, malware growth has been increasing drastically. The high popularity of Android devices has motivated malware developers to attack these devices. In recent times, most researchers and scholars have used deep learning approaches to detect Android malware. Although deep learning techniques provide good accuracy and efficiency, they require high computational cost to train huge and complex data sets. Hence, there is a need for an approach that can efficiently detect novel malware variants with a minimum computational cost. This paper proposes a novel framework for detecting and clustering Android malware using the transfer learning and the topic modelling approach. The transfer learning approach minimizes new training data by transferring well-known features from a qualified source model to a destination model, and hence, a high amount of computational power is not required. In addition, the proposed framework clusters the detected malware variants into their corresponding families with the help of Latent Dirichlet Allocation and hierarchical clustering techniques. For performance assessment, we performed several experiments with more than 50K Android application samples. In addition, we compared the performance of our framework with that of similar existing traditional machine learning and deep learning models. The proposed framework provides better accuracy of 98.3% during the classification stage by using the transfer learning approach as compared to other state-of-the-art Android malware detection techniques. The high precision value of 98.7% is obtained during the clustering stage while grouping the obtained malicious applications into their corresponding malware families.

1. Introduction

Among several mobile phone devices, the Android operating system has shown an exponential growth in the last few years due to its numerous potential benefits such as open-source nature, user convenience, and extensibility. Android's openness allows users to download and install billions of free applications. Although Android applications are scanned frequently with Google Play Protect (https://developers.google.com/android/play-protect), still, the users can disable the scanning mechanism and download applications from third-party stores. This motivates malware developers to publish their repackaged malicious Android applications in suspicious third-party stores or websites. When a user downloads and installs repackaged malicious apps, the device security is compromised. According to "*Kaspersky's*" security malware threat report (https://securelist.com/it-threat-evolution-q1-2021-mobile-statistics/102547/), in the third quarter of 2020, approximately 350,000 novel malware applications were detected, of which most of the malicious applications were COVID-19-themed applications. Figure 1 shows the popular Android malware detected in the last quarter of 2020. In the first quarter of 2021, an increase of more than 10,000 malicious samples was seen as compared to the previous quarter.

Conventional malware detection methods have several limitations. With the extensive utilization of machine learning and deep learning techniques in recent years, Android malware detection using these techniques has appeared, and the accuracy of malware detection mechanisms has been significantly improved.

In response to the intense growth of Android malware, extensive research has been conducted on techniques for

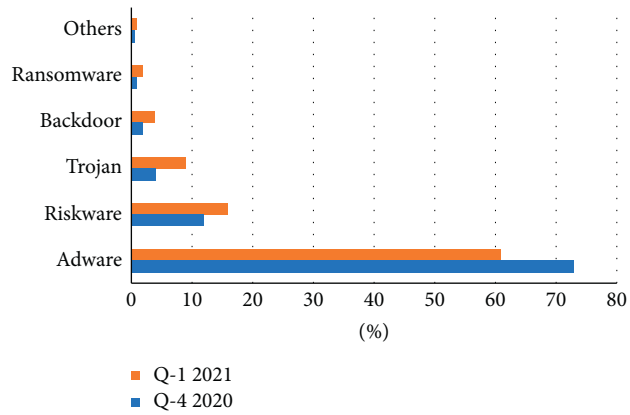

Figure 1: Android malware statistics.

detecting Android malware using deep learning. Researchers have proposed several techniques using various deep learning models and have obtained many research outcomes. However, malware detection techniques based on deep learning require a large amount of labeled data points to identify malicious threats with optimum accuracy. In most cases, the size of the data set for detecting new malware threats is not always large, and to collect a new data set, the search time also increases. Moreover, to identify a new malware threat, the deep learning models need to be trained again for a new data set from scratch, which is not only resource-consuming but also time-consuming. One efficient solution to overcome the issue of high computational complexity and model retraining is to use deep transfer learning technique. The major aim of utilizing the transfer learning approach in our study is to reduce the computational complexity by transferring well-known feature sets from a trained base model to a destination model with very less training data. The detailed description of the transfer learning approach is given in Section 3.

1.1. Our Contribution. This paper proposes a two-stage framework to detect Android malware and group them into their respective families. In the first stage, we performed visualization-based detection using a traditional CNN (Convolutional Neural Network) model and a transfer learning model. Visualization helps to depict malicious behavior because the graphical features of an obfuscated malware do not change during the detection process [1]. CNNs are used to extract visual features because CNNs can efficiently extract deep features from images and videos. Further, we transfer the CNN methodology to the classification phase using transfer learning to speed up the convergence. This provides an advantage of training large networks on both large and small novel data sets with less computational cost. The key goals and advantages of using transfer learning in the initial stage are as follows:

(i) Knowledge utilization: the source model's knowledge can be used to train new target task. Hence, the new target model does not require training from scratch.

(ii) Model development time: the overall time required to develop and train a new model reduces drastically because only the last few data set-specific layers need to be trained.

(iii) Overfitting issues: the problem of overfitting occurs when traditional machine learning and deep learning models are trained on a small data set. Transfer learning overcomes this issue by fine-tuning the model layers.

(iv) Computational cost: deep learning models need a high computational power to train complex and hybrid data sets. This expensive computational cost can be reduced by applying the transfer learning approach.

In the second stage, we utilized two well-known topic modelling approaches *LDA* (*Latent Dirichlet Allocation*) and *hierarchical clustering* to categorize the families of the malicious applications obtained from the classification phase. To cluster the applications, the opcode vector sequences of malicious applications are extracted and the applications with similar functionality are grouped into a common malware family. In our analysis, we considered more than 50K APKs (Android Application Packages). The samples were collected from the official play store (https://play.google.com/store), Drebin data set [2], Malgenome data set [3], and AAGM (Android Adware and General Malware) (https://www.unb.ca/cic/datasets/android-adware.html) data sets that are publicly available for research. These data sets are described in detail in Section 3.

Our experimental results show that the APK parameters learned from the traditional model can be transferred to a target model to classify real-life data sets. By using the conventional CNN technique, we achieved an accuracy of 95.4%, whereas the transfer learning technique improves the accuracy to 98.3%. For familial classification, we utilized the LDA clustering approach. To reduce the irrelevant topics, we used a hierarchical clustering technique which provides the highest precision of 98.7%.

The proposed framework is more accurate and less time-consuming and does not require huge amount of computational power and resources. Moreover, the familial classification provides a deep understanding of Android malware. Combining transfer learning with topic modelling and hierarchical clustering provides benefits such as reduced computational power, fast and accurate familial classification, removal of overfitting issues for smaller data sets, and flexibility to add, remove, and transfer features to train new models.

The major contributions of our work are summarized below:

(i) The CNN model is used to identify and locate the visual prototype of Android malware. Root-level features are extracted from malicious images.

(ii) Transfer learning is adopted to accelerate the training phase of the CNN model. Using transfer learning, the model was transferred to the classification phase. The experimental outcomes provided higher accuracy (98.7%) and fewer false positives.

(iii) Topic modelling approaches are used to group malicious apps into their respective families. This grouping provides in-depth knowledge of malware behaviors.

The rest of the article is arranged as follows:

Section 2 presents the related work. Section 3 demonstrates the proposed framework methodology, workflow, detection strategy, experimental setup, evaluation outcomes, and comparative analysis. Section 4 describes familial classification, assessment methods, and evaluation results. Finally, Section 5 concludes the study and discusses further studies.

2. Related Work

2.1. Android Malware Detection Based on Static and Dynamic Approaches. The scholarly world has proposed several techniques and solutions for detecting Android malware. The *static analysis* approach is one of the most famous approaches for analyzing static source code features to detect Android malware. [4, 5] are some of the proposed detection methods based on the static analysis approach. Although this approach is fast, it fails to identify obfuscated malware [6, 7]. To overcome this issue, various solutions have been proposed based on the *dynamic analysis* approach. Rather than checking the source code, the dynamic analysis approach actually executes applications in virtual emulators or real devices to monitor real-time suspicious activities. This requires a large amount of resources and computational hardware.

Wong and Lie proposed IntelliDroid [8], an input generator to analyze Android applications. The proposed generator provides input relative to a dynamic analysis tool and can be paired with dynamic analysis tools. The authors experimented on various inputs and successfully identified malicious behavior and extracted the malicious paths.

Bhatia and Kaushal [9] presented an approach to detect Android malware by performing runtime behavior analysis of the applications. The authors extracted and collected system call traces and identified the frequency of feature sets. They utilized the J48 Decision tree and Random forest algorithms to calculate the accuracy. The results were good; however, their model was trained on a small-sized data set of 50 applications.

Lorenzo et al. proposed the VizMal tool [10] to analyze the runtime traces of Android applications. The authors traced the execution path of the applications and identified vulnerable checkpoints. Though the proposed tool works well for identifying malicious traces, the implementation cost and complexity are high.

2.2. Android Malware Detection Based on Learning Algorithms. The difficulty of manually identifying malicious applications has motivated researchers to utilize machine learning techniques to accelerate and automate the detection mechanism. Popular research studies utilize machine learning algorithms to detect and classify zero-day Android threats.

Unlike machine learning, deep learning techniques automatically choose relevant features to train the detection model efficiently. Therefore, high-level domain information and manual feature selection are not required. This has motivated the researchers to propose various deep learning-based Android malware detection tools and techniques. Karbab et al. [11] proposed the MalDozer framework which can automatically detect Android malware and provides familial classification. The authors utilized deep learning techniques to find malicious applications. They extracted several features from API code sequences of the data set. The framework was implemented on a data set containing more than 30k malicious samples out of 70k total samples. The authors achieved a low false-positive rate, but the framework requires high-end computations to achieve higher efficiency. Yuan et al. [12] implemented DroidDetector, a malicious APK detection engine to detect Android malware. The authors tested thousands of applications and performed a deep investigation using deep learning to detect malware. The proposed engine provided 96.7% accuracy. However, a greater number of critical features can be included in the training stage to improve the accuracy further.

Wang et al. [13] provided a comprehensive survey of deep learning techniques to detect Android malware. The authors investigated several studies based on deep learning architectures and provided a comparative summary. Mu et al. [14] utilized a text classification technique to detect Android malware. The authors extracted API instructions based on the Cuckoo sandbox and applied text processing to detect Android malware. Furthermore, the authors compared the accuracy of their technique with BGRU and LSTM techniques [15]. Their results showed better accuracy; however, the authors did not cluster the detected malware into their families.

Kim et al. [16] proposed a multilayered deep learning framework for Android malware detection. The authors extracted several useful features and refined them using the feature set vector generation technique. Moreover, the authors compared the performance of the proposed framework with other similar deep learning techniques. The proposed framework provided an overall better efficiency; however, it requires high computational power to execute multiple layers. Masum and Shahriar [17] proposed a framework named *Droid-NNet* to classify Android malware using deep learning. The authors utilized publicly available data sets Malegnome-215 [3] and Drebin-215 [2] to compare the performance of their framework with machine learning-based Android malware detection techniques. Their proposed framework provides a good F-beta score and low false positives. However, the evaluation was restricted to two public data sets.

Azmoodeh et al. [18] presented a deep learning framework to identify *Internet of Battlefield Things (IoBT)* malware using Android opcode sequences. The authors converted opcodes into vector space and applied deep Eigenspace learning. Finally, the authors showed the robustness of the proposed technique against code injection attacks. Feng et al. [19] presented a two-layer approach to detect malicious Android applications. The authors utilized

the first layer to extract static features such as intents, components, and permissions. The outcomes of the first layer are fed into the second layer, in which CNN and AutoEncoder are used for malware detection. The experimental outcomes provided a good detection rate, but the proposed framework requires high computational speed for training complex data sets.

Alotaibi [20] presented a new framework called Mal-ResLSTM to detect and classify Android malware. The proposed framework is based on deep residual LSTM. The author captured static features from applications and converted them into vector space. The obtained vector space is then fed into a deep LSTM network to classify malicious and normal applications. The author utilized *Drebin* data set performance evaluation. Booz et al. [21] utilized deep neural networks to classify Android malware by analyzing critical system permissions and third-party permissions. The authors applied the grid search technique to identify several combinations of hybrid attributes for deep learning methods. The obtained results had good accuracy, but the searching process required high number of processing cores to handle complex datasets.

Singh et al. [22] proposed a machine learning-based framework to classify Android applications. The authors extracted gray-scale images from the Drebin data set file and obtained manual features by exploiting the APK files. To extract image files, the algorithms based on image processing are utilized. The authors obtained an accuracy of 93%. The framework is capable to classify Android applications, but overfitting problems can occur if the model needs to be trained on a large data set.

Angelo et al. [23] proposed an approach based on the exploitation of API transitions in the call sequence. The extraction of a subsequence of API calls resulted in a malware classification resistant to evasion techniques. The authors compared the detectors using Markov chain and call sequence algorithms. The study outcomes outperformed various malware detection techniques.

Ficco [24] proposed an approach based on ensemble detection. The author utilized a blend of generic and specialized detectors during the analysis process to enhance the detection randomness and to improve the overall detection rate. Moreover, an alpha-count mechanism is also presented to differentiate the speed of various detectors. This mechanism provides the observation time window length which can affect the detection accuracy.

2.3. Android Malware Detection Based on Robustness. Cai et al. [25] presented a novel dynamic and robust Android app classification technique called DroidCat based on dynamic method calls and ICC intents. The proposed technique efficiently handles reflection without depending on system calls and permissions. Moreover, the technique provides better robustness when compared to other similar state-of-the-art static and dynamic malware detection techniques. The authors extracted features from a behavioral categorization study. In addition, 34,343 apps from distinct sources during nine years were used in the evaluation, and performance measure was calculated. They achieved a better F1 score accuracy of 97% when compared with two state-of-the-art techniques.

Suarez-Tangil et al. [26] presented the DroidSieve approach based on static features classification. The proposed method exploits obfuscating feature sets and processes the static features in parallel. The authors achieved a detection accuracy of 99.82% with no false positives and familial classification accuracy of 99.26%.

Suarez-Tangil and Stringhini [27] conducted a deep analysis of Android malicious app behavior consisting of more than 1.2 million malware samples with 1.28 K families during a period from 2010 to 2017. The authors aimed to understand the evolution of repackaged malware, separated the components that were unrelated to malware, and analyzed the behavior of malicious riders using differential analysis. The samples were collected from distinct antivirus vendors.

Cai [28] conducted a study to build a systematic environment that continuously mines the mobile software ecosystem. The author focused on the behavioral evolution and performed a big dimension characterization study of the ecosystem. Further, an ecological interaction among three attributes, namely, user apps platform, mobile platform, and app users, was considered. The results provide sustainable app development and security.

Cai et al. [29] presented a study on the execution strategy of Android applications. The authors analyzed the behavior of Android applications about malware concerning execution paths, structures, methodology scopes, and callbacks. They observed the app execution structure concerning the security platform. Further, the authors traced ICCs and methods of more than 30,000 applications including 15,451 benign apps and 15,183 malicious apps created from 2010 to 2017. Among these apps, the behavioral structure and similarities were identified to trace the difference in the apps.

Cai and Ryder [30] presented a study on Android apps based on longitudinal characterization to identify and observe the build and execution nature of the applications. The authors utilized a lightweight static approach to analyze the execution code of 17,664 applications developed in eight years. Further, they analyzed that applications' functionalities strongly depend on Android system architecture, and this dependency tends to increase with time. Also, the Activity components invocate life-cycle callbacks, and the event callbacks rely on the user interface rather than the system interface. In addition, the ICCs do not contain data payloads.

2.4. Android Malware Detection Based on Sustainability. Fu and Cai [31] investigated the deterioration issues in Android malware detectors. The authors analyzed four state-of-the-art detectors and concluded that the performance of the existing solutions degrades with time. Moreover, the authors proposed a novel approach based on a longitudinal characterization study of applications focusing on runtime behaviors. A comparison was also performed between the proposed approach and the four state-of-the-art techniques to analyze the deterioration problem.

Xu et al. [32] proposed DroidEvolver to detect Android malware which can update automatically without human intervention. The model does not require retraining, and it gets updated through online learning techniques, and hence, the high computational cost is not needed. The authors experimented on 33,294 benign apps and 34,722 malware apps that were created over six years. Further, the authors compared the model with the state-of-the-art MamaDroid model [33] and outperformed it in terms of average F1 measure.

Cai [34] proposed a practical malware detector that can sustain over time without the need for retraining. The author deeply analyzed sustainability issues related to learning-based classifiers. Initially, the author defined sustainability metrics and created the DroidSpan classification system which focuses on sensitive access distribution capturing. Moreover, the author evaluated and compared the sustainability of DroidSpan with five state-of-the-art detectors having 13,627 benign apps and 12,755 malware apps. The proposed system outperformed the baseline detectors in sustainability. Cai and Jenkins [35] proposed a sustainable Android malware detector that can detect novel malware without retraining. The authors investigated the runtime behaviors of applications and discriminated benign apps from malware by analyzing the behavior. The proposed model can sustain over five years without the requirement of frequent retraining.

3. Proposed Framework

In this article, a two-stage framework is proposed to classify and combine Android applications. During the first phase, the Android applications were collected from various sources such as Google Play Store (containing benign APKs only), Malgenome, Drebin, and AAGM data sets, respectively. The feature sets are extracted from the collected APK files, and these feature sets are preprocessed and converted into binary vectors. To categorize the applications into malicious and benign apps, static and dynamic parameters are extracted from a set of Android applications. These applications are obtained from the official play store, third-party app stores, and publicly available datasets. Static parameters [36] include system services, intent, version, manifest permissions, strings, and components. These parameters provide metadata information stored in the application. Unlike static parameters, the dynamic parameters [37] such as system calls, files, logs, and network activities provide information about the behavior and control flow of an application. Further, the binary vectors are converted into gray-scale images, and these binary images are used as inputs to the source model for training and classification purpose. The generation process of images from Android APK files is demonstrated in Figures 2 and 3 shows the proposed framework. The static and dynamic feature sets are discussed below:

(i) Manifest.Permissions: every Android application contains the *Manifest* file, which is used to provide information about packages, strings, resources, services, etc. One of the package files inside the manifest file is the *manifest.permissions* file which is used to provide permission to applications. This file is used to validate permissions during the installation process of the application. Critical permissions include access to SMS, location, and storage.

(ii) API calls: Application Programming Interface (API) calls are runtime function calls. They are initiated when an application wants to interact with system resources.

(iii) String values: these values are used to provide text information of resources. These files are normally stored in *Strings.xml* file. Information such as app history, version, list of permissions, and app size is contained in this file.

(iv) Intent: intents are used to trigger activities in an application. When an application wants to perform a task, intents are triggered to provide runtime APK binding.

(v) Dalvik code: these are executable codes that are obtained from Java bytecodes. Dalvik codes are not used in novel Android versions, and they are replaced by *ART* (*Android.runtime)* library. The ART library is used to provide debugging options to identify bugs in the application.

(vi) Services: Android services are the components responsible for background activities while an application is running. Services do not require user intervention.

(vii) Version: this specifies the information about the current version of an APK file. Versions usually change when an application is updated.

(viii) Component: Android APKs are categorized into components for better storage. The components store activity, intents, permissions, and resource files.

(ix) System calls: the system calls are used by the applications to access Android operating system resources. They work as system-level APIs to interact with system files.

(x) Runtime libraries: Android runtime (ART) is used to provide diagnostic and debugging options.

The feature sets are converted into feature vectors and ranked according to their importance. For example, the importance of parameter "version" is less than the importance of parameter "Permissions." Similarly, all the feature parameters are ranked according to their priority. By ranking the parameters, the insignificant features can be removed and filtering can be done. After filtering the feature sets, the significant feature sets are converted into binary gray-scale images. This conversion is described in the next section.

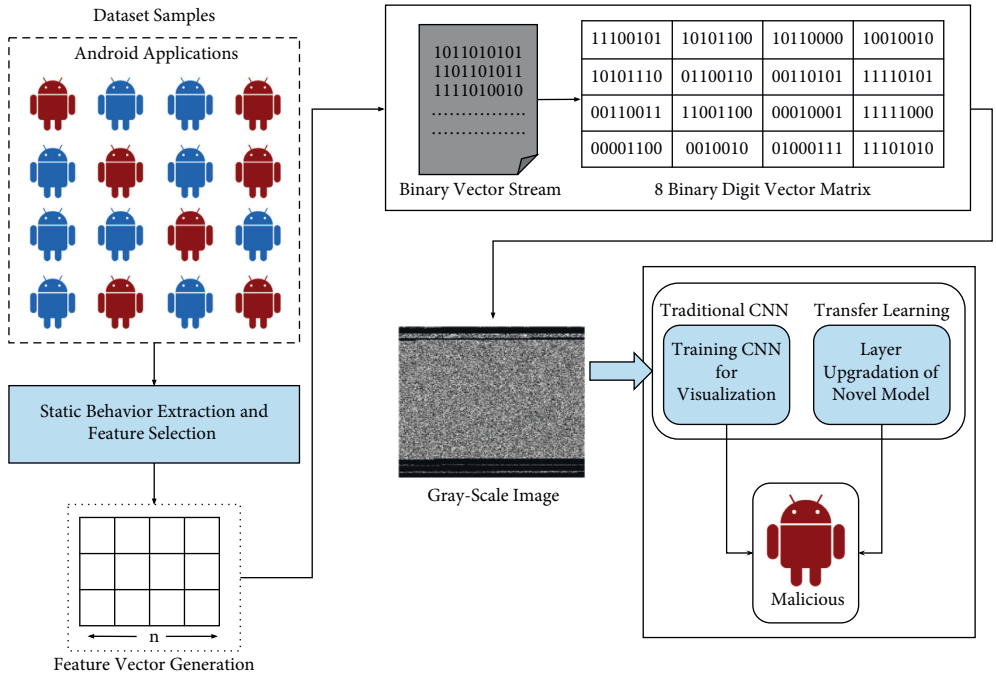

Figure 2: APK to gray-scale image conversion process.

3.1. Workflow. In our article, we collected testing samples from Android's official play store using a crawler tool. This tool crawls over the entire application database and extracts APK files from it. The applications stored in the play store database are usually benign because Google uses the Google Play Protect tool to periodically scan applications in real time.

If an application is removed from the play store, it indicates that the application is either vulnerable or suspicious. After obtaining the benign APKs using a crawler tool, the benign app data set and hybrid data set (containing benign and malicious applications) are filtered, and irrelevant features are removed as shown in Figure 4. The obtained feature set is converted to gray-scale images by selecting the binary values from APk files. Furthermore, the gray-scale images are used as an input to the CNN model for training purpose. To reduce the complexity, the initial layers of the CNN model are transferred to a novel model using transfer learning. The final few layers were fine-tuned and updated. Finally, the malicious applications obtained after the training and classification phase were used for the familial classification process. In the overall workflow, the filtering and scanning processes are performed rigorously to obtain real-time updates.

3.2. Generating Images from Android Applications. According to the research criterion, APK visualization can be performed efficiently by using static features of APK file such as *AndroidManifest.xml*, Dalvik files, string xml files, and resource files [38]. In this article, the malicious images were extracted using these files from malicious APKs. Gray-scale images are obtained by converting the files into binary vector pixels. The data set APK archives are extracted to generate the components required for creating image data sets. The APK data have been interpreted as a binary stream and kept in a binary array vector matrix. The APK files are disassembled to generate the 8-bit binary files, and then, they are mapped to gray-scale range of image which is generally 0–255. The binary streams are transformed into vector array matrix to construct a gray-scale image as shown in Figure 2. The major drawback with AndroidManifest.xml and resources.arsc files is that these files generated images of size 64 pixels. And this cannot be expanded to 256 pixels because data loss occurs. As every byte in the binary vector matrix can have a value in the range 0–255, every byte has been transformed into a pixel.

The image generation steps are given as follows:

(i) Step-1: the data sets having APK archives are extracted to obtain the files namely *AndroidManifest.XML*, *Resources.arsc*, *Classes.dex*, and *jar files*.

(ii) Step-2: the generated files are disassembled to produce 8-bit binary files. The data in the files are interpreted as binary data, and the binary vector streams are generated.

(iii) Step-3: the binary vector streams are transformed into an 8-bit array vector matrix.

(iv) Step-4: the gray-scale images are constructed using the array vector matrix and are stored in an image data set.

The generated image is used as an input for the traditional CNN model and transfer learning model. In the transfer learning model, the last few layers are updated and initial layers are frozen as these layers specify generalized feature sets. The reason for choosing the CNN model is that it is capable of visualizing the overall geographical areas of an

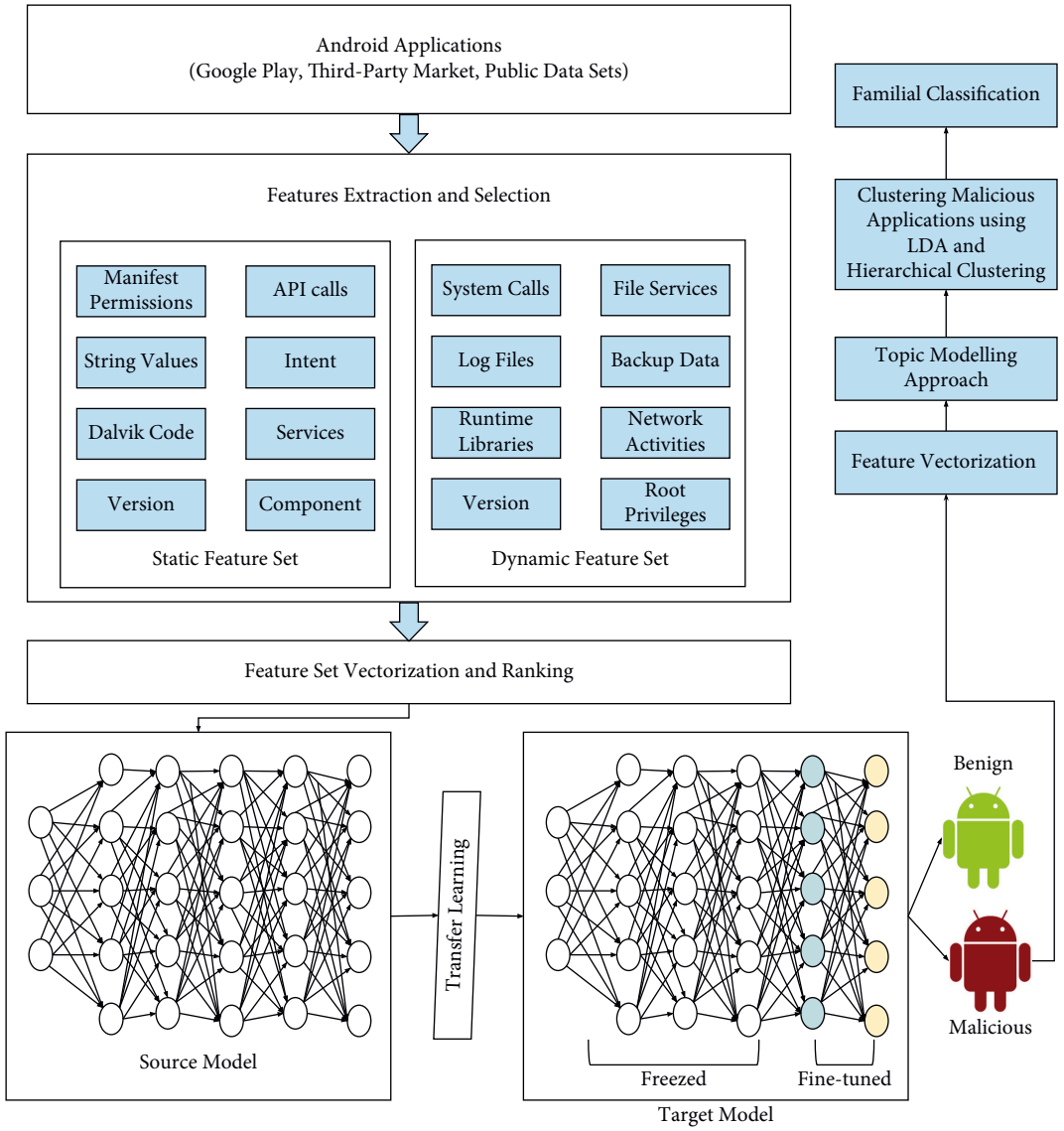

Figure 3: Proposed framework.

image. Transferring the learned attributes of initial layers helps the novel model to train similar data sets more efficiently with less time.

3.3. Convolutional Neural Networks. A CNN is a type of neural network that is feedforward in nature. CNNs are best suited for classifying images and videos as they efficiently extract geographical parameters and provide good accuracy with very few false positives [39, 40]. CNNs can extract data set parameters in narrow image regions as well as full image frames. In the initial stage, CNNs contain convolutional layer and dense layer that are interconnected with each other. The important parameters are separated from the input data set in the training phase with the help of convolutional layer. In this way, the input data set is reduced, and training process is accelerated. To further reduce the feature-set size, CNNs contain max pooling layer which is responsible for merging various neurons to a maximum value. The above-mentioned CNN layers and a completely connected dense layer form the CNN architecture as shown in Figure 5.

A completely connected dense layer takes an input feature set obtained from previous layers and generates output. The most useful advantage of using CNNs is that they can be fine-tuned to extract essential features from malicious data set images without performing feature engineering. However, the CNNs are vulnerable to overfitting problem for smaller data sets. To overcome the

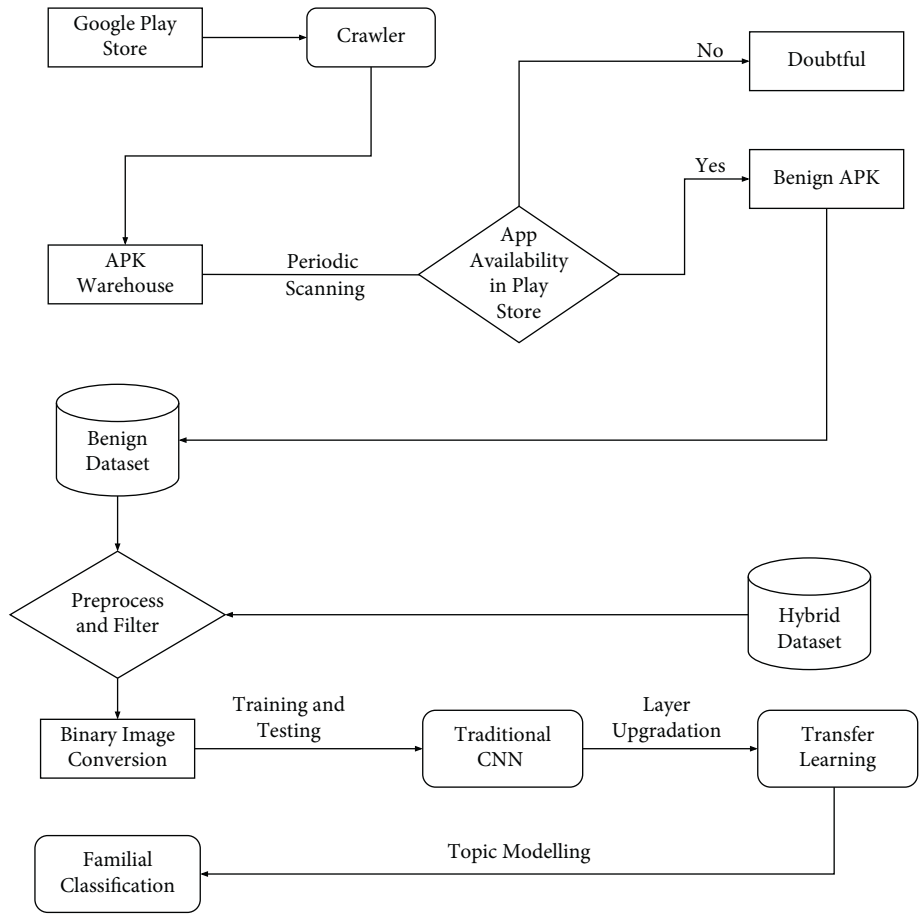

Figure 4: Workflow of proposed framework.

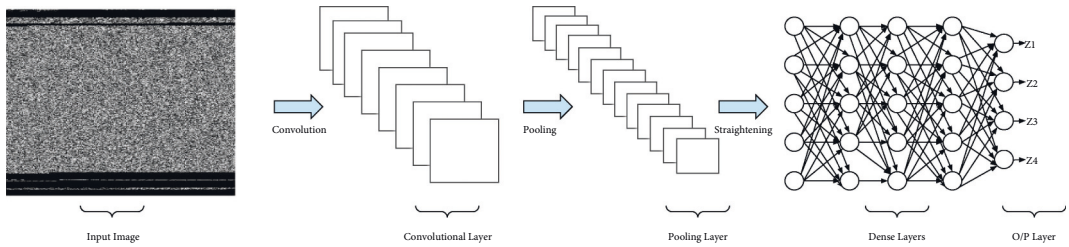

Figure 5: CNN model architecture.

problem of overfitting, the training data can be normalized. A CNN model is able to adopt images of any size. However, to overcome any data loss, we normalized the images to a scale, such as A X B. To input an image of size C XD, there are two possibilities as shown in the following equation:

$$(AXB)\text{is}\begin{cases} >(CXD), & \text{Rescale\_image}, \\ <=(CXD), & \text{Edge\_fill} = 0. \end{cases} \tag{1}$$

Malicious APK image data set is generated and malicious features are extracted to classify malware graphically. The objective function of generated image in the CNN network is defined as

$$L(p_i, v_i) = \frac{1}{P_{\text{cls}}} \sum L_{\text{cls}}(p_i, p_i^*) + \lambda \frac{1}{P_{\text{reg}}} \sum p_i^* L_{\text{reg}}(v_1, v_1^*), \tag{2}$$

where p_i is the target probability of the anchor.

$$p_i^* = \begin{cases} 0, & \text{negative\_flag}, \\ 1, & \text{positive\_flag}. \end{cases} \tag{3}$$

The cross-entropy is defined as

$$L_{\text{cls}}(p_i, p_i^*) = \frac{1}{P_{\text{cls}}} \sum_i -\log[p_i^* p_i + (1 - p_i^*)(1 - p_i)]. \tag{4}$$

3.4. Malicious APK Detection Based on Transfer Learning

3.4.1. Transfer Learning. Deep transfer learning constructs a model by working on the last few layers of already trained deep neural network models. This is achieved by transferring the knowledge or model learned in a particular model to another model. The basic objective is to enhance the performance and accuracy by transferring the core features of the trained model to another model [7, 41]. For example, a model trained on a large image data set can be utilized to train the target model on a smaller data set efficiently. This process is illustrated in Figure 6.

Transfer learning reduces the computational cost and time complexity as training the model from scratch is not required. With the increase in network complexity, the time complexity, computational cost, and overfitting problems increase drastically. To resolve these issues, the features are extracted by applying fine-tuning and freezing process [42]. Using these two processes, the network features of the learning target can be modified. For example, fine-tuning stops generalized network features to be trained and updates higher level model features. This is achieved by freezing the initial few layers of the source model. After fine-tuning the layers, the novel updated layers are attached to the source model.

Figure 7 shows the basic architecture of transfer learning approach.

In Figure 7, the starting layers of the model are frozen and the final layers are fine-tuned. This is achieved by changing the features at the last layers without modifying the initial layers. The reason for modifying the last layers is that these layers utilize most of the data set, whereas the initial layers deal with generic features such as *AndroidManifest* file, version, metadata, strings, and other static features. By freezing the initial layers, the computational power requirement is reduced drastically as only the last layers need to be trained.

The necessary feature sets of *AndroidManifest.XML*, *classes.dex*, and *resources.arsc* files present inside the image data set are stored in the final few layers of the model, and they are transferred to the target model to train hybrid and smaller image data sets. Hybrid data sets are prepared by using a blend of publicly available data sets. The extracted APK files are filtered to separate necessary files such as *classes.dex*, *manifest.xml*, and *resources.arsc*. This reduces detection time and computational power requirements. Because only the last few layers need to be changed, the learning rate is usually smaller than that of the traditional models. The problem of overfitting is reduced by using transfer learning, as only essential features are transferred to the novel model and generalized features are frozen.

3.4.2. Detection Mechanism. CNN models are typically trained on a large image data set as an input. However, the data set samples are limited. Training the CNNs on smaller data sets creates the problem of overfitting. Transfer learning overcomes this issue by training the target model on selected layers only. In this article, we utilized transfer learning of the CNN network which is trained on a hybrid data set containing benign and malware applications and then transferred the learning to train our Android APK gray-scale image data set. This process is demonstrated by the flowchart shown in Figure 8.

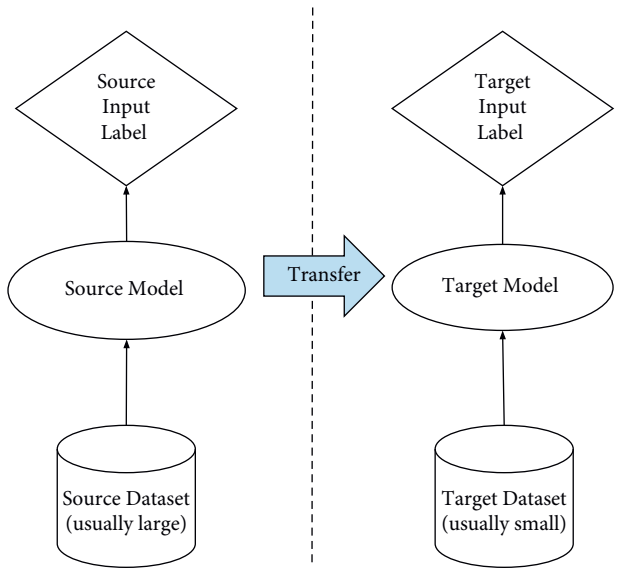

Figure 6: Transfer learning process.

The pretrained CNN network model is transferred to the Android APK visual malicious classification model. This transfer learning reduces the training time complexity of the target classification model. The deep features of the model are able to extract geographical features of images. The target model can be trained on a small data set with good accuracy and less false positives. To reduce errors in the image detection process, we define a hybrid loss function to fine-tune the network features. The schematic figure is depicted in Figure 9.

$$Hl(p_i, v_i) = \frac{1}{P_{\text{cls}}} \sum Hl_{\text{cls}}(p_i, p_i^*) + \lambda \frac{1}{P_{\text{reg}}} \sum p_i^* Hl_{\text{reg}}(v_1, v_1^*) + Hl(w), \tag{5}$$

where

$$Hl(p_i, p_i^*) = \frac{1}{P_{cls}} \sum_i -\log[p_i^* p_i + (1 - p_i^*)(1 - p_i)],$$
$$p_i^* = \begin{cases} 0, & \text{negative\_flag}, \\ 1, & \text{positive\_flag}, \end{cases} \tag{6}$$

p_i is the target probability of the anchor.

$$Hl_{\text{bbox\_score}}(v_i, v_i^*) = \lambda \frac{1}{N_{\text{reg}}} \sum p_i^* S(v_i - v_i^*), \tag{7}$$

v_i^* is the vector representing the coordinates of the bounding rectangle with respect to the positive flag. $v_i = \{v_x, v_y, v_w, v_h\}$ is a parameter coordinate vector of predicting rectangle. S is the smooth function for the softmax layer.

The main implementation consists of the following steps:

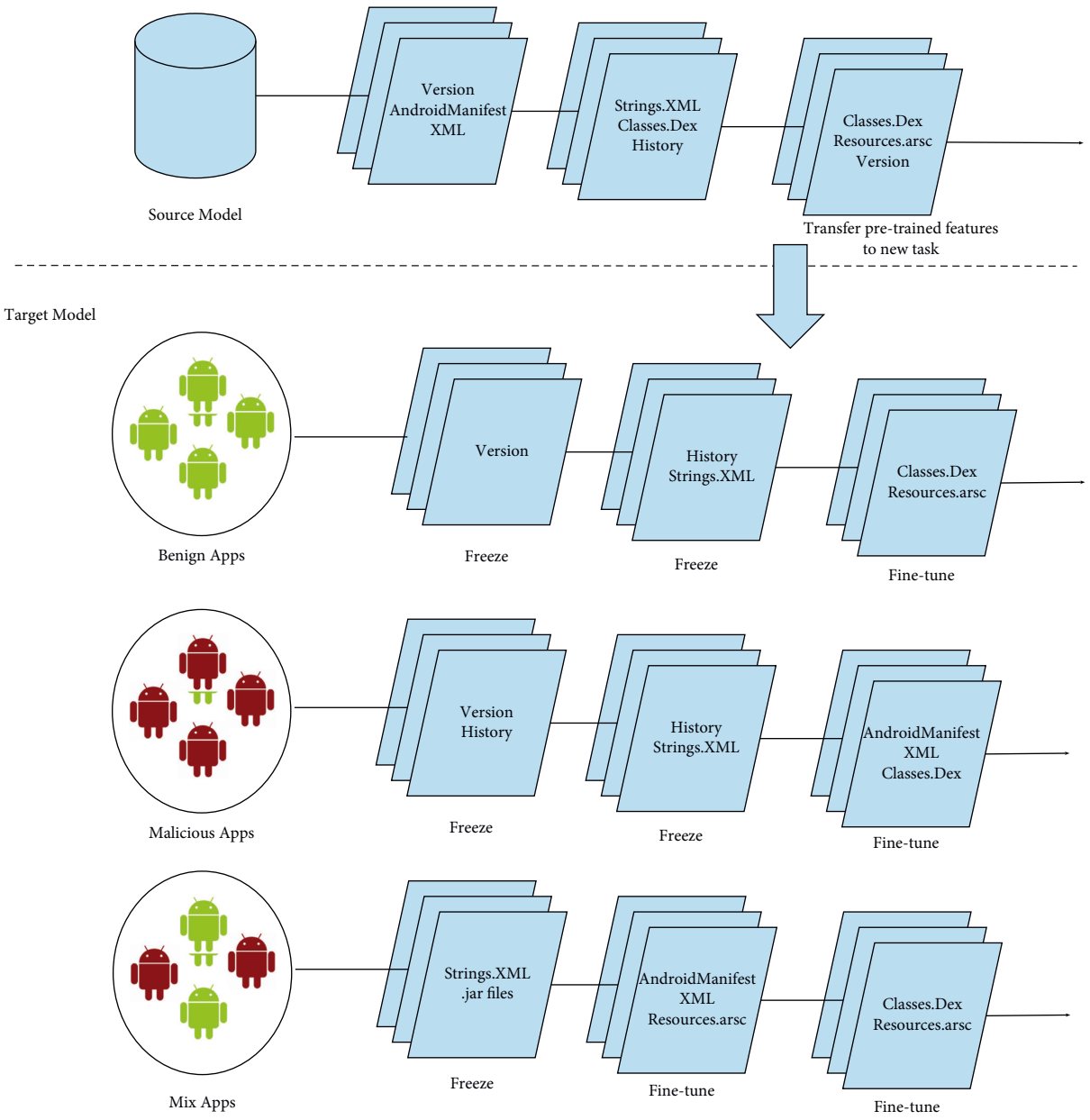

Figure 7: Layers upgradation in transfer learning mechanism.

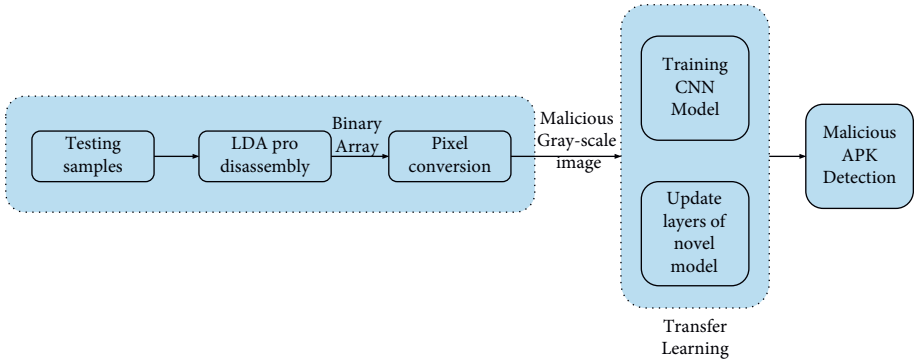

Figure 8: Malicious APK detection mechanism.

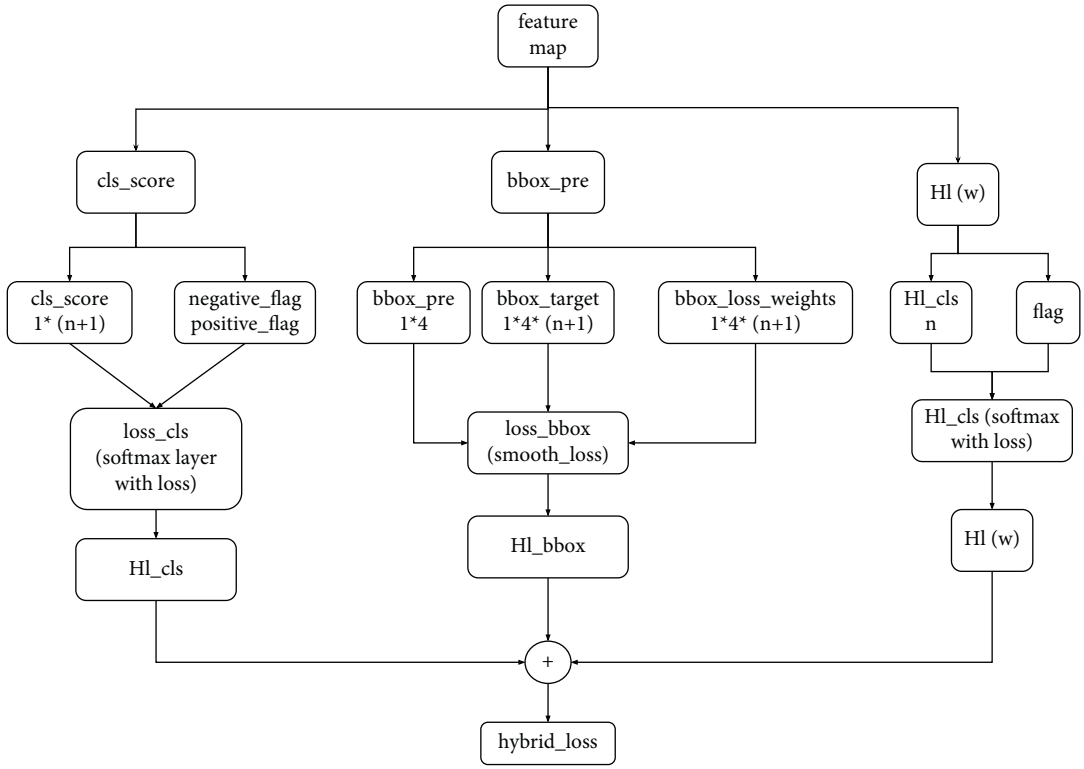

Figure 9: Step-by-step calculation of hybrid loss function.

(i) Data set description: We utilized three distinct Android malware data sets: the Drebin data set containing 5560 APKs, the Malgenome data set containing 1200 APKs, and the AAGM data set containing 1900 APKs, respectively. These data sets are publicly available for research purposed. The reason for selecting Malgenome and AAGM data set is to check the accuracy of the transfer model by training the model on this smaller data set. Moreover, the overfitting problem also needs to be analyzed. To generate pure benign data set, we utilized the crawler tool to extract benign APKs from Google Play Store. Table 1 shows the various data sets used in our experimental study.

(i) Pretraining the model: to pretrain the model, we utilized a combination of all the data sets including benign and malicious APKs. Most of the general features are common to all the images.

(ii) Fine-tuning the layers: we applied forward and backward propagation methods to balance the weights of the pretrained model. This is required to freeze the general feature sets of the initial model layers. After freezing the initial parameters, we transferred the feature set to a novel model. We chose a learning rate of 0.001 to modify the convolutional layer. The deep feature sets obtained by the initial few layers are general, so we kept these layers as it is while transferring. The middle and last few layers affect the target output of the model, so we fine-tuned these layers and updated the learning rate to 0.0001. Finally, we set the objective function to train the novel model.

(iii) Model Training: after obtaining the new configuration file, we trained the novel model by maintaining the learning rate of 0.0001. The process flowchart is depicted in Figure 10.

In Figure 10, the feature sets of malware APK files are stored in "*dex*" files. These "*dex*" files are disassembled and converted into binary files for binary classification. The pixels are obtained during the conversion process, and each binary file corresponds to a gray-scale image. The malicious images are stored in the "resource" folder inside the main file. The CNN model is trained on hybrid data set containing benign and malware files. The config file which was used to train the CNN model is modified in such a way that the initial layers are kept as it is. The middle and final layers are modified, and this configuration is stored in *config* file. Finally, the novel transfer model is trained using this modified config file.

3.4.3. Training Methodology. To distribute the data set equally, we utilized 7-fold cross-validation. The gray-scale images are fed into the CNN network in a randomized manner, for example, genuine and malicious APK images can enter the network in a single shot. This reduces the time required to serialize the images. We utilized *Tensorflow* [43] and Torch [44] libraries in our experimental evaluation. To identify the hyperparameters, we utilized performance metrics such as "precision," "recall," "efficiency," and "F1-

Table 1: Data set description.

Dataset	Time period	Benign apps		Malicious apps		
		Source	#Samples	Source	#Samples	#Families
Drebin	2010–2012	Drebin public data set	9476	Drebin public data set	5560	179
Malgenome	2014–2015	Malgenome	700	Malgenome	1260	10
AAGM	2017–2018	CIC-AAGM2017	1500	CIC-AAGM2017	500	42
Hybrid	2020–2021	Drebin public data set, Google Play, CCCS-CIC-AndMal-2020	250000	CCCS-CIC-AndMal-2020	200000	191

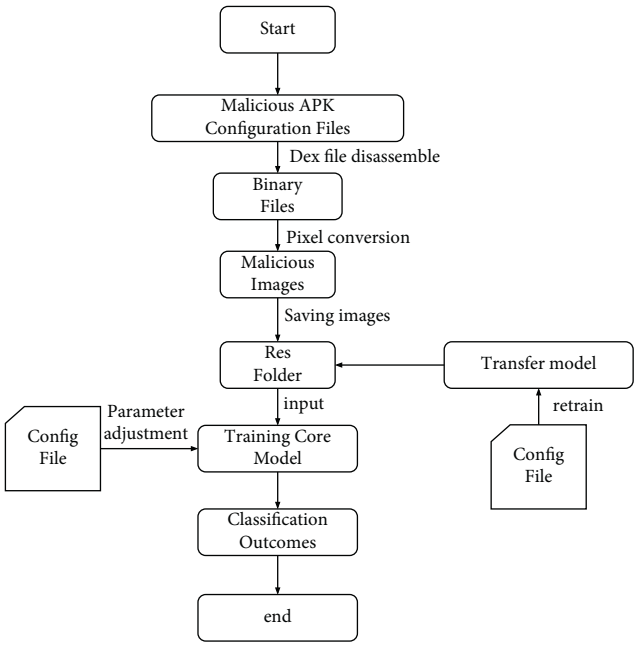

Figure 10: Experimental flowchart.

score." Total epochs of [15, 25, 35, 55, 75, 90, 100] are executed in our evaluation considering the hardware and software computational power. To reduce the false positives, we utilized "recall" score. The optimal blend of hyperparameters we got by using 120 epochs for Drebin data set and 15 epochs for hybrid data set, respectively. The classification statistics for the hybrid data set are presented in Table 2. The outcomes are generated by training the CNN model on a larger hybrid data set. The classification statistics for Malgenome and AAGM data sets are shown in Tables 3 and 4, respectively. The precision values are better as compared to the precision values of CNN classification.

This indicates that the true false positives are less, and training predictions are almost accurate. In addition, the binary classification for benign and malicious outcomes is obtained by setting the learning threshold to a very low value.

3.4.4. Detection Strategy. To detect the malicious samples, we executed the classification test among hybrid data set comprising play store benign APKs, Drebin data set APKs, Malgenome APKs, and AAGM APKs, each containing 5000, 5560, 1200, and 1900 samples, respectively. The evaluation

Table 2: Classification statistics for hybrid data set (Play Store/Drebin/Malgenome/AAGM).

Category	Precision	F-score	Recall	Support
Genuine APKs	0.952	0.935	0.92	1123
Malware APKs	0.952	0.935	0.92	1400

Table 3: Classification statistics using transfer learning (Malgenome data set).

Category	Precision	F-score	Recall	Support
Genuine APKs	0.960	0.944	0.93	1228
Malware APKs	0.970	0.954	0.94	1401

Table 4: Classification statistics using transfer learning (AAGM data set).

Category	Precision	F-score	Recall	Support
Genuine APKs	0.980	0.969	0.96	1352
Malware APKs	0.970	0.970	0.97	1563

result of the data set produces a score of 95.2% with a false positive rate of 5.4%. This cross-validated score is executed per fold, and it is distributed evenly among the data set. Furthermore, we applied the transfer learning approach and achieved a cross-validated score of 98.3%. While applying transfer learning, we fine-tuned the hyperparameters of the CNN layer and dense layer and updated the *configuration file*. For this purpose, we utilized the objective hybrid loss function defined in Equation 2.

Transfer learning approach achieves better performance and less false positives as compared to the traditional CNN model. The performance outcomes are presented in Table 5. According to Table 5, the transfer learning approach performs better in terms of improved accuracy, less FPR and computational cost, and no overfitting issues. The convergence rate is also fast with the transfer model as the complete model does not need to be trained from scratch.

We also compared these results with those of similar studies. To compare the models, we chose accuracy rate or efficiency rate, number of false positives, and percentage of malware detected. A comparative analysis of distinct techniques and models is shown in Table 6.

Table 7 shows the comparison between training time and computational power overhead of the CNN model and transfer learning model for different data sets. Some of the important metrics used to evaluate the performance of

Table 5: Comparative analysis of performance of two models.

Method	Overall accuracy (%)	False positive rate (%)	Computational cost	Complexity
Traditional CNN	95.2	5.4	High	High
Transfer learning	98.3	3.2	Low	Low

Table 6: Comparative analysis of experimental outcomes of different techniques.

Study	Accuracy (in %)	Technique/Framework/Model
[11]	97	Deep learning
[14]	96.7	Deep learning and API calls
[39]	98.26	CNN and adjacency matrix
[45]	93.7	System calls and API sequences
[46]	93.7	Machine learning and topic modelling
[This article]	98.3	Transfer learning and topic modelling

Table 7: Comparison of training time and computational power overhead.

Data set	Model	Mean time to detect (MTTD) in seconds	Computational power requirements
Hybrid	CNN	90000	Processor: Intel Xeon, GPU: Nvidia RTX 3080, RAM: 128 GB
Malgenome	Transfer learning	61200	Processor: Intel Core i9 10900K, GPU: Nvidia RTX 3060, RAM: 96 GB
AAGM	Transfer learning	55944	Processor: Intel Xeon, GPU: Nvidia RTX 3060, RAM: 96 GB

detectors depend on training time and computational overhead. Kozik et al. [47] showed the comparison of training time and computational power overhead for ELM (Extreme Learning Machine) classifier. The authors showed that the computational resource requirement increases with the data set size and complexity. In this work, we utilized the metric mean time to detect (MTTD) to evaluate the effectiveness of malware detectors. MTTD is a key metric widely used in cybersecurity. It provides the average amount of time taken to detect the threats. From Table 7, it can be observed that the transfer learning model outperforms the traditional CNN model as the MTTD value for the CNN model is nearly around 90000 seconds which is higher than the transfer learning model. The main reason behind this is that the transfer learning model need not be trained from the scratch unlike the CNN model. Moreover, the computational power overhead is higher for the CNN model as compared to the transfer learning model because the CNN model contains each and every layer to be trained on the complex data set. Figure 11 shows the comparison of the average time taken to detect the malware using the CNN model and the transfer learning model.

Due to the faster convergence of the transfer learning approach, the hyperparameters are effectively utilized as compared to other approaches. The performance plots of the detection techniques are shown in Figures 12 and 13, respectively.

As depicted in Figure 12, the F-scores of traditional models are lower as compared to the transfer model. The main reason for this is that traditional models require significant computational power during the training phase,

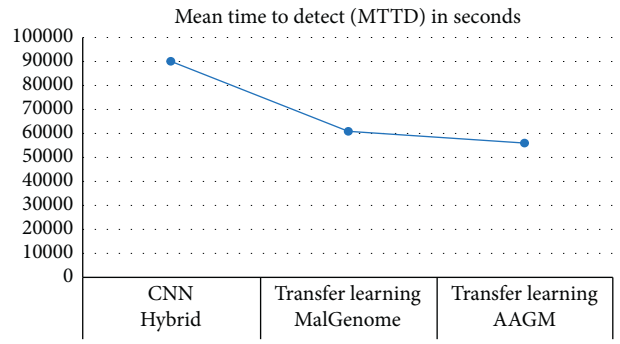

Figure 11: MTTD for CNN model and transfer learning model.

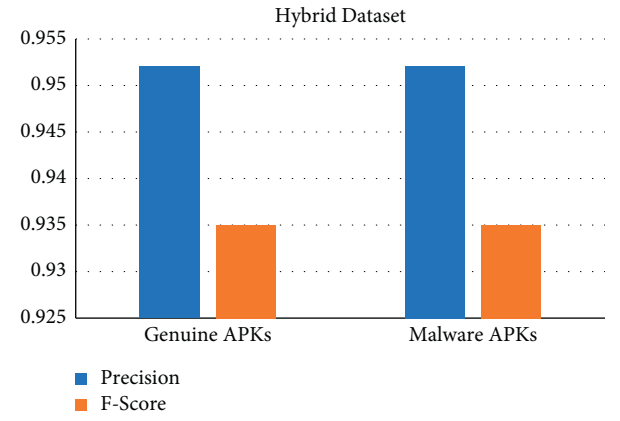

Figure 12: Precision and F-Scores for CNN model.

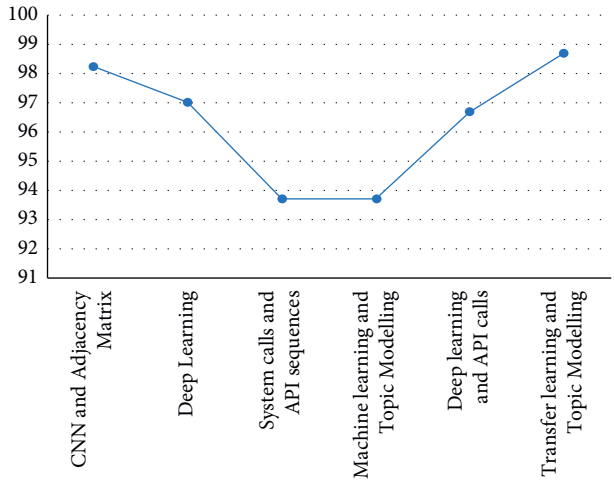

Figure 13: Comparison among different techniques.

whereas the transfer learning model requires less power and time without compromising the accuracy, as only a few layers need to be updated. The overfitting problem is reduced by the objective function defined in Equation 2. Hence, the overall detection rate improved significantly with negligible false positives.

In Table 6, the accuracy of the proposed technique is compared with similar techniques or models. The proposed method has achieved a good accuracy. The comparative results are shown in Figure 13. Moreover, the familial classification process also achieved a good accuracy which is described in the next section.

Table 8 shows the comparative analysis of various state-of-the-art Android malware detectors based on objective, i.e., detection or categorization or both, familial clustering, robustness, and computational cost. The various detectors are good at some points; however, most of the detectors lack robustness in terms of detecting zero-day and obfuscated malware. Our proposed method detects and categorizes malware with less computational cost requirement. However, the robustness of the method needs to be enhanced in terms of obfuscation, especially, resource obfuscation.

3.5. Limitations of the Proposed Method. The proposed approach aims to detect and categorize Android malware with less computational power requirements. However, several other important factors such as sustainability and performance deterioration need to be considered. The basic goal of our approach is to complement the other detection approaches that require a lot of computation resources for training and testing the models. In our study, we focused on reducing the need for high-end GPUs and RAM and overcoming the problem of overfitting in the case of smaller data sets.

The robustness of the proposed method is not as good as compared to some previous studies such as DroidCat [25], DroidSieve [26], and DroidSpan [51]. The main reason for this is that we have considered static feature sets along with dynamic feature sets in our work. The static features lack runtime behavior attributes, and new malware variants dynamically change their behavior and form to evade detection mechanisms. Our proposed approach works well for detecting existing malware; however, the sustainability of the detection approach requires the reselection of various feature sets and layers to be transferred to the target model for training purposes. Though the less computational cost will decrease the training time, the target transfer learning model needs to be retrained for new malware samples in contrast to some prior detectors mentioned above. Model updation depends on several factors such as novel malware behavior, dynamic permissions, resource obfuscation, and system call obfuscation. The problem of retraining the target model can be resolved by considering the overall behavioral features rather than static features. Moreover, the deterioration problem of our proposed learning model can be reduced by considering the evolutionary categorization of Android applications. Fu and Cai [31] focused on the deterioration problem in their study and were able to achieve the highest average F1 score over seven years. The authors have also compared their work with state-of-the-art baselines and outperformed them. Our proposed approach provides good detection accuracy; however, we will try to overcome the above-mentioned limitations in our next study to increase the efficiency of our detector. Our next step will focus on sustainability and performance deterioration issues concerning the transfer learning approach.

4. Familial Clustering Using Topic Modelling

In this section, the malicious APKs that are detected during the classification stage are grouped together to form a family. To achieve this, we utilized LDA (Latent Dirichlet Allocation) [52] and hierarchical clustering [53] techniques which are based on topic modelling [54]. In topic modelling, the apps are grouped and combined into their respective families on the basis of functionality of malicious applications, for example, applications that show malicious ads are grouped into Android.adware family. Similarly, the applications that require access to system calls and critical permissions are combined into the virus family and so on. To implement this, we created a database of detected APKs and converted their features into topic vectors. Furthermore, vectors with common attributes are combined to form a cluster. Each cluster represents an Android malware family.

4.1. Preprocessing APK Database. To process the database, we applied a filtering technique which is used to eliminate stop words. To achieve this, we utilized the *tensorflow* python library. After the data gets filtered, we executed the stemming process on topic vectors. Stemming is used to find the core word from a set of words. For example, the malicious applications "com.android.spy.fake," "com.android.spy.install," and "com.android.spy.worm" can be stemmed into spyware. Similarly, other malicious applications can also be stemmed into their respective root topics. This helps to minimize the processing overhead for a large set of applications. The stemming process for malicious applications is shown in Figure 14.

Table 8: Comparative analysis of various Android malware detectors on the basis of objective, clustering, robustness, and computational cost.

Technique/Method	Year	Approach	Objective	Familial clustering	Detection robustness				Computational cost
					Reflection	Resource obfuscation	System call obfuscation	Dynamic permissions	
DroidCat [25]	2018	Dynamic	Detection and categorization	✓	✓	✓	✓	✓	N/A
AOMDroid [41]	2020	Dynamic	Detection	×	×	✓	✓	✓	High
MamaDroid [33]	2017	Static	Detection	N/A	×	×	✓	✓	N/A
DroidSieve [26]	2017	Static	Detection and categorization	✓	✓	×	✓	×	N/A
Android-SEM [48]	2022	Dynamic	Detection and categorization	✓	×	×	✓	×	Moderate
DroidScribe [49]	2016	Dynamic	Categorization	✓	✓	×	×	×	Moderate
MalDozer [11]	2018	Dynamic	Detection and categorization	✓	×	✓	✓	✓	High
DL-Droid [50]	2020	Static and dynamic	Detection	×	×	×	×	✓	High
Ad-Mat [39]	2021	Static	Detection and categorization	✓	×	×	✓	×	High
Proposed method	This study	Static and dynamic	Detection and categorization	✓	✓	×	✓	✓	Low

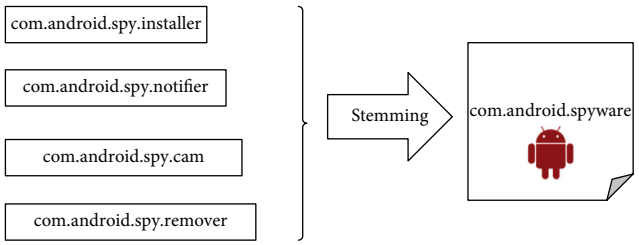

Figure 14: Stemming process.

After the stemming process, we created an array of topic vectors based on similar features. For example, the malicious APKs having the same functionality and feature set are grouped into a common cluster. Furthermore, we applied the LDA algorithm and hierarchical clustering algorithm to create clusters.

4.1.1. Latent Dirichlet Allocation. In our experiment, we utilized the LDA algorithm to generate the probability of topic vectors. The likelihood of each malicious APK can be identified by analyzing the feature set vector. If the features of two or more applications are similar in the context of behavior and control flow, the probability that these applications belong to a similar malware family will be higher. The APKs having same probability assignment are combined into a common cluster with the help of the SVM (Simple Vector Machine) [55–57] clustering technique. In place of SVM, KNN (K-Nearest Neighbour) [58] can also be used, but SVM gives good results as it considers outliers effectively. We labeled the malicious APKs that have similar probability assignments and generated clusters as depicted in Figure 15.

The topic-wise likelihood distribution of data set APKs is listed in Table 9. In Table 9, the applications "spydetector," "trojanfinder," "adsdetector," and so on have the highest probability of being malicious as these applications need access to critical permissions and generate malicious ad links. Similarly, the other applications are distributed topically on the basis of static feature sets.

After distributing the applications topic-wise, the clusters of topics are formed. The cluster with the highest probability value includes maximum topics inside it. The sample cluster-forming process is presented in Table 10.

Cluster formation for distinct topics is demonstrated in Table 10. In this table, large number of applications lies in the "news" category, i.e., cluster 8, and very few applications belong to the "medical" category, i.e., cluster 6. One of the drawbacks of classifying the apps category-wise is that most of the apps do not provide relevant information and fall into unknown categories. This problem can be solved by using the hierarchical clustering technique discussed in the next section.

Sample clusters are shown in Figure 16. The malware applications "com.android.mp3player," "com.android.covidfinder," "com.android.addtector," etc., are grouped into cluster 2 as shown in Figure 15. This cluster is named as "com.android.virus," and it depicts one of the malware family "virus." Similarly, cluster 1, cluster 3, and cluster 4 depict "spyware," "trojan," and "backdoor" families, respectively.

4.1.2. Hierarchical Clustering Technique. LDA clustering provides a static approach for clustering the malicious applications. To improve the clustering mechanism and achieve dynamic clustering, we executed the hierarchical clustering technique. In this technique, prior topic information is not necessary. Therefore, by obtaining the hierarchy of similar feature vectors, the clustering can be done

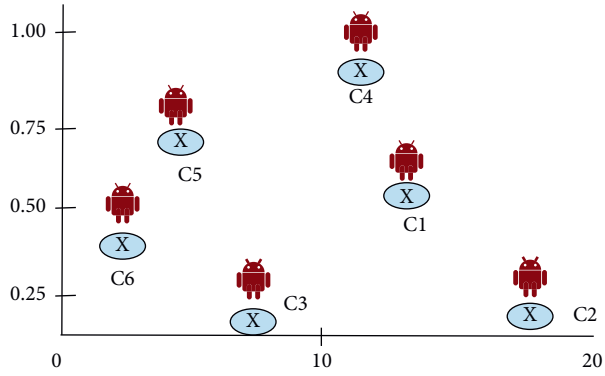

Figure 15: Probability assignment of malicious APK clusters.

Table 9: Probability assignment for each application topic-wise.

Android APK	Topic 1	Topic 2	Topic 3	Topic 4	Topic 5
com.android.mp3player	0.052	0.076	0.054	0.045	0.077
com.android.addetector	0.068	0.078	0.082	0.036	0.069
com.android.spy.installer	0.088	0.066	0.058	0.071	0.083
com.android.fakeantivirus	0.037	0.055	0.049	0.084	0.063
com.android.rootkitfinder	0.059	0.057	0.052	0.083	0.072
com.android.covidinfo	0.088	0.076	0.058	0.068	0.022
com.android.memorybooster	0.033	0.017	0.029	0.088	0.055
...	...	...	...	...	...

Table 10: Clusters formation category-wise.

Cluster	Type	APK Probability
Cluster 1	Device	0.171
Cluster 2	Music	0.152
Cluster 3	Phone	0.177
Cluster 4	Game	0.164
Cluster 5	Utility	0.134
Cluster 6	Medical	0.129
Cluster 7	E-commerce	0.178
Cluster 8	News	0.189
Cluster 9	Forecast	0.156
Cluster 10	Health	0.162

dynamically and efficiently. The hierarchical clustering technique is shown in Figure 17.

In Figure 17, a similarity or relationship matrix is created from the APK feature set. Below are the steps involved in generating the clusters:

(i) Step 1: each topic is used to form a topic cluster.

(ii) Step 2: the relationship matrix among topics is generated, and the topics which are similar are grouped together into a novel topic T.

(iii) Step 3: the relationship between novel topic T and the remaining topics is computed, and this process is repeated until a threshold "T_h" is achieved such that the relationship between two topics becomes dissimilar and final topics are obtained.

The size of the similarity matrix decreases for every iteration as the clustering process continues. In our experiment, the threshold value T_h is set to a value of 0.3, and it is computed on the basis of the maximum similarity percentage among topics. This threshold value provides faster convergence and optimal clustering.

The relationship matrix is computed using topic vectorization. The computational formula is as follows:

$$\text{Relation}(T_X, T_Y) = \vec{X} \cdot \vec{Y} = |\vec{X}| \cdot |\vec{Y}| \cdot \cos\Theta, \quad (8)$$

where T_X is topic X, T_Y is topic Y, X′ is the topic vector of X, Y′ is the topic vector of Y, and Beta is the angle between X′ and Y′. Identifying the similarity between two topic vectors

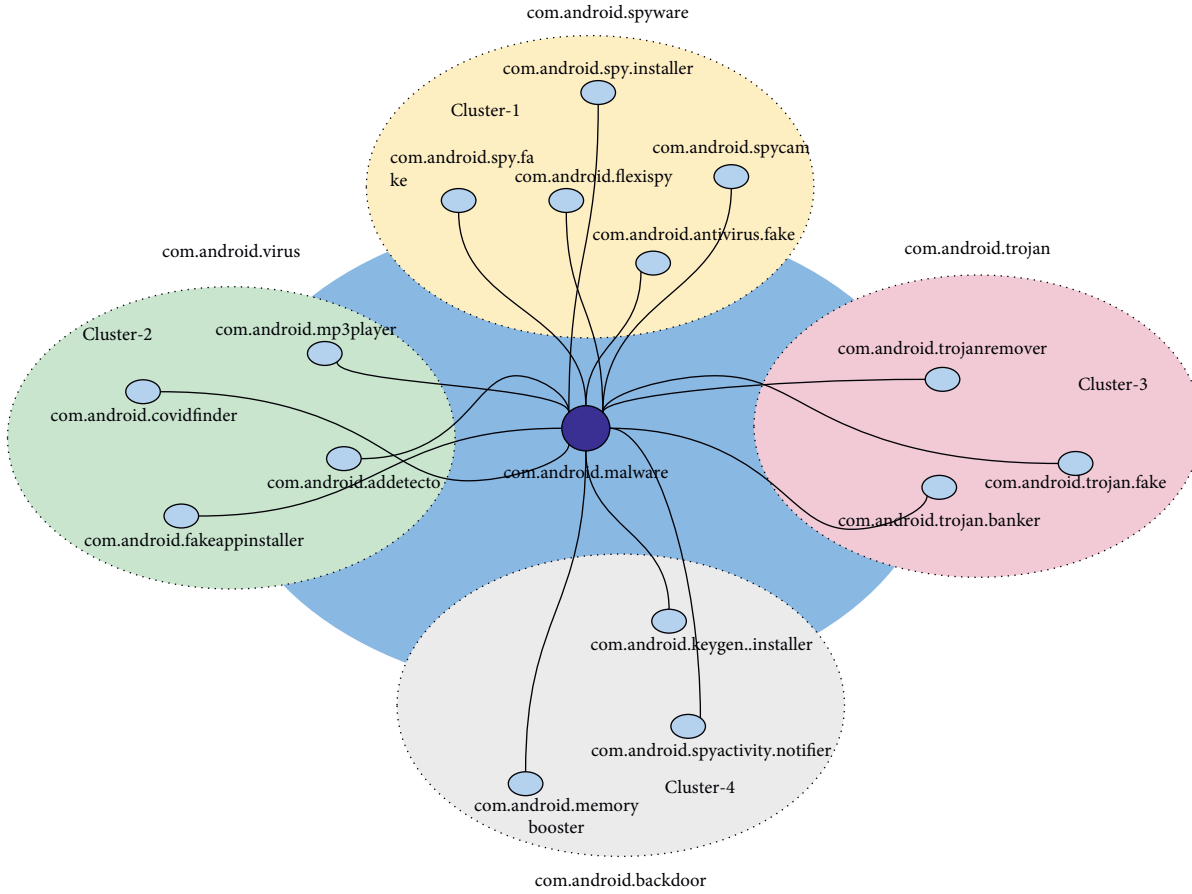

Figure 16: Sample malware family clusters.

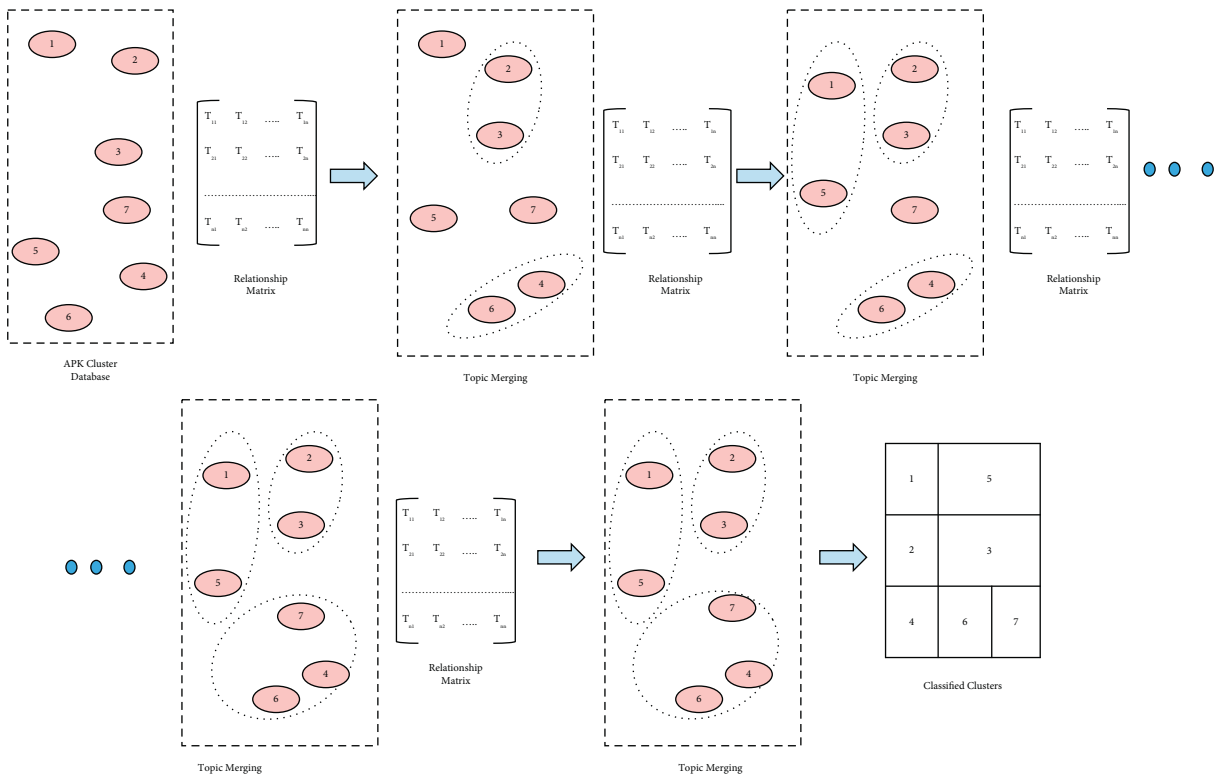

Figure 17: Hierarchical clustering process.

Table 11: Performance outcomes for distinct topics quantity.

Data set	Core topics quantity	F-Score	Precision	Recall
Drebin	10	0.977	0.987	0.969
	25	0.950	0.956	0.945
	45	0.880	0.947	0.823
Malgenome	10	0.880	0.981	0.798
	25	0.971	0.988	0.955
	45	0.947	0.988	0.910
AAGM	10	0.878	0.948	0.818
	25	0.956	0.961	0.952
	45	0.966	0.988	0.946

Table 12: Topic words extracted using LDA and hierarchical clustering.

Topic content	LDA clustering	Hierarchical clustering
Android Ransomware detected in 2020 were mostly themed on COVID-19	Malware, security, threat, and specific	Attack, ransomware, malware, security, and malicious
Novel system update virus hides itself in victim's phone	Virus, security, attack, charge, and music	Spyware, malware, attack, security, system, virus, and version
Mobile trojans can access critical permissions	Threat, virus, charge, target, and mobile	Trojan, detection, virus, device, and permissions
DDoS attacks are able to control device resources	Virus, attack, device, and news	Virus, malware, device, attack, and resources

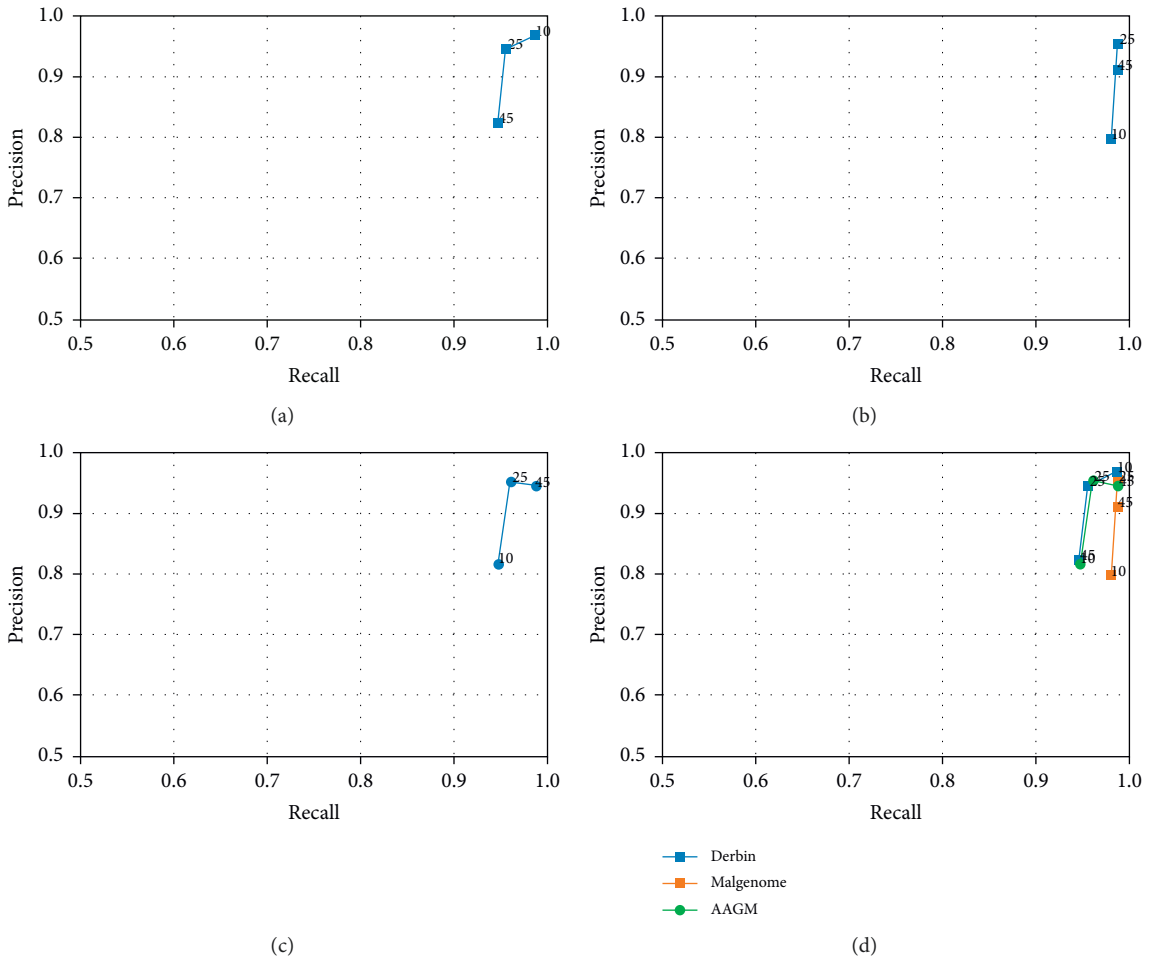

Figure 18: P-R curves for core topics quantity. (a) Drebin dataset. (b) Malgenome dataset. (c) AAGM dataset. (d) Performance comparison.

Table 13: Performance metric outcomes for topic words quantity.

Data set	Topic words quantity	F-Score	Precision	Recall
Drebin	5	0.728	0.788	0.677
	10	0.840	0.887	0.799
	15	0.681	0.711	0.655
Malgenome	5	0.743	0.794	0.699
	10	0.667	0.722	0.621
	15	0.721	0.712	0.732
AAGM	5	0.717	0.774	0.668
	10	0.669	0.662	0.678
	15	0.786	0.798	0.776

using the above formula provides a fast merging rate, and irrelevant topics are avoided. This provides less complexity and high robustness in the cluster-forming process.

4.1.3. Assessment Methods. To evaluate the process, we utilized three famous parameters,"$\text{Recall}_{\text{rate}}$," "$F_{\text{Score}}$," and "Precision." These parameters are defined in clustering methodology with the help of the following formulas:

$$\begin{aligned} \text{Recall}(x,i) &= \frac{N_{xi}}{N_x}, \\ \text{Precision}(x,i) &= \frac{N_{xi}}{N_i}, \end{aligned} \tag{9}$$

where N_x and N_i are the quantity of topics of class x and cluster i, respectively. N_{xi} is the quantity of topics consisting of class x in cluster i. To determine the F-score, the following formula is used.

$$F_{\text{Score}}(x,i) = 2 * \left[\frac{\text{Recall}(x,i) * \text{Precision}(x,i)}{\text{Recall}(x,i) + \text{Precision}(x,i)}\right]. \tag{10}$$

The value of $F_{\text{Score}}(x)$ of class x is the maximum value among $F_{\text{Score}}(x,i)$ values of class x and groupings i. F_{Score} is defined in the following formula:

$$F_{\text{Score}} = \arg_i \max\left(F_{\text{Score}}(x,i)\right). \tag{11}$$

If R_x denotes recall rate and P_x denotes precision P_x of class x, then the overall precision, recall rate, and F_{Score} values will be the average of precision, recall, and F_{Score} for every cluster, respectively. The following formulas define the final values.

$$\begin{aligned} \text{Precision} &= \sum \frac{N_x}{N} P_x, \\ F_{\text{Score-Overall}} &= \sum \frac{N_x}{N} F_{\text{Score}}(x), \\ \text{Recall}_{\text{rate}} &= \sum \frac{N_x}{N} R_x. \end{aligned} \tag{12}$$

4.1.4. Evaluation Results. The clustering experiment is performed on the topic database of malicious APKs obtained by traditional CNN and transfer learning classification techniques. The performance metric parameters are evaluated for 10, 15, 25, 30, and 45 core topic word features. For "T" topic features with $T = 10$, 25, and 45, the performance achieved is better for Drebin, Malgenome, and AAGM data sets, respectively, as compared to other values of T. We chose maximum values of T as 10, 25, and 45 for these data sets because we achieved good results on these thresholds. By increasing this value further, the complexity of the clustering process increases and the FPR (False Positive Rate) value also increases. For T values less than 10, the accuracy tends to decrease. Table 11 lists the outcome values for various parameters.

In Table 8, the precision achieved is the highest for $T = 10$ for the Drebin data set. Similarly, the precision for Malgenome and AAGM data sets is high for T values 25 and 45, respectively. Sample topic words extracted by LDA and hierarchical clustering are presented in Table 12.

The P-R curves according to the quantity of core topics are shown in Figure 18.

The words obtained by LDA clustering contain less information than the words obtained by hierarchical clustering. Moreover, some words do not convey the gist of the topic. For example, the words "news," "music," "charge," and "specific" have no relevance to the topics. On the other hand, the words obtained from the hierarchical clustering technique are more relevant to the topics. The topic words for the three data sets with performance metrics are listed in Table 13. The P-R curves according to topic words quantity are shown in Figure 19.

As shown in Table 13, for T values 10, 5 and 15, the results are good for Drebin, Malgenome, and AAGM data set, respectively. For T values greater than 15, highly irrelevant topic words were extracted which produces errors. Hence, the optimal topic threshold for topic quantity is restricted to 10. In our experiments, we considered UPGMA (Unweighted Pair Group Method with Arithmetic Mean) method [59] of hierarchical clustering technique to evaluate the performance because it provides high similarity index for related terms.

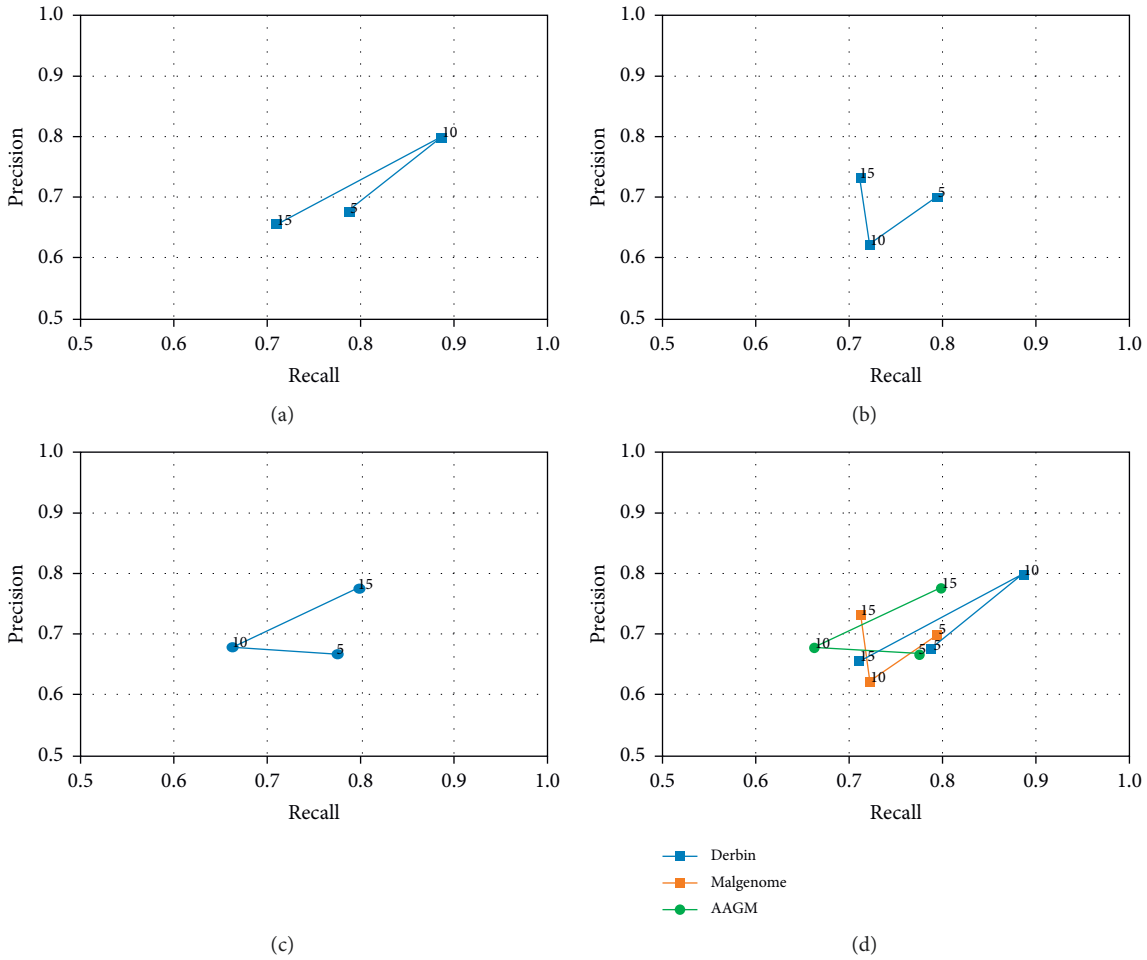

Figure 19: P-R curves according to topic words. (a) Drebin dataset. (b) Malgenome dataset. (c) AAGM dataset. (d) Performance comparison.

5. Conclusion and Future Work

Cell malware has been present since the launch of smartphones. With the increase in Android popularity, malware applications continue to succeed in escaping security models. In this article, we discussed traditional CNN and transfer learning techniques to detect and classify Android malware. Due to the extensive use of CNN in image processing, the application of CNN on malware images becomes important. We proposed a two-stage framework which converts Android APKs into binary gray-scale images. These images are used as an input to the traditional CNN model. To overcome the problems of overfitting, complexity, and computational cost, we applied the transfer learning approach on the trained model by freezing the initial layers of the pretrained model. The evaluation outcomes show that the transfer learning approach provides better accuracy of 98.3% with very few false positives as compared to the traditional CNN model. We also compared the evaluation results with some similar techniques. The results show that transfer learning outperforms the traditional techniques and also reduces the computational cost. Further, we combined the detected malicious APKs into their respective malware families using LDA and hierarchical clustering techniques based on topic modelling. We compared the clustering results of the LDA technique with the hierarchical clustering technique and concluded that hierarchical clustering provides better familial classification. In the upcoming study, we would like to extend the utilization of hybrid classification techniques such as LSTM combined with control-flow graphs. We expect that our future study will provide us in-depth fine-grained feature sets for more better results. Moreover, we will try to overcome the limitations of the proposed model mentioned in the limitations segment under Section 3 of this paper. Moreover, we will also perform the tests of our proposed model for sustainability and performance deterioration issues.

Data Availability

The data sets used in this research are available publicly at the following links: https://www.unb.ca/cic/datasets/android-adware.html and https://www.sec.cs.tu-bs.de/~danarp/drebin/.

Conflicts of Interest

The authors declare that they have no conflicts of interest or personal relationships that could have influenced the work reported in this paper.

References

[1] S. Albawi, T. A. Mohammed, and S. Al-Zawi, "Understanding of a convolutional neural network," in *Proceedings of the 2017 International Conference on Engineering and Technology (ICET)*, pp. 1–6, IEEE, Antalya, Turkey, August 2017.

[2] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, K. Rieck, and C. Siemens, "Drebin: effective and explainable detection of android malware in your pocket," *Proceedings 2014 Network and Distributed System Security Symposium*, vol. 14, pp. 23–26, 2014.

[3] F. A. Narudin, A. Feizollah, N. B. Anuar, and A. Gani, "Evaluation of machine learning classifiers for mobile malware detection," *Soft Computing*, vol. 20, no. 1, pp. 343–357, 2016.

[4] P. Faruki, A. Bharmal, V. Laxmi et al., "Android security: a survey of issues, malware penetration, and defenses," *IEEE Communications Surveys & Tutorials*, vol. 17, no. 2, pp. 998–1022, 2015.

[5] A. Kapratwar, F. Di Troia, and M. Stamp, "Static and dynamic analysis of android malware," in *Proceedings of the 3rd International Conference on Information Systems Security and Privacy*, pp. 653–662, San Jose State University, San Jose, CA, USA, 2017.

[6] M. Garg, A. Monga, P. Bhatt, and A. Arora, "Android app behaviour classification using topic modeling techniques and outlier detection using app permissions," in *Proceedings of the 2016 Fourth International Conference on Parallel, Distributed and Grid Computing (PDGC)*, pp. 500–506, IEEE, Waknaghat, India, December 2016.

[7] Y. Zhao, W. Cui, S. Geng, B. Bo, Y. Feng, and W. Zhang, "A malware detection method of code texture visualization based on an improved faster rcnn combining transfer learning," *IEEE Access*, vol. 8, pp. 166630–166641, 2020.

[8] M. Y. Wong and D. Lie, "Intellidroid: a targeted input generator for the dynamic analysis of android malware," in *Proceedings of the 2016 Network and Distributed System Security Symposium*, vol. 16, pp. 21–24, Toronto, Canada, January 2016.

[9] T. Bhatia and R. Kaushal, "Malware detection in android based on dynamic analysis," in *Proceedings of the 2017 International Conference on Cyber Security and Protection of Digital Services (Cyber Security)*, pp. 1–6, IEEE, London, UK, June 2017.

[10] A. De Lorenzo, F. Martinelli, E. Medvet, F. Mercaldo, and A. Santone, "Visualizing the outcome of dynamic analysis of android malware with vizmal," *Journal of Information Security and Applications*, vol. 50, Article ID 102423, 2020.

[11] E. B. Karbab, M. Debbabi, A. Derhab, and D. Mouheb, "Maldozer: automatic framework for android malware detection using deep learning," *Digital Investigation*, vol. 24, pp. S48–S59, 2018.

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

[13] W. Zhiqiang and L. Jun, "A review of object detection based on convolutional neural network," in *Proceedings of the 2017 36th Chinese Control Conference (CCC)*, pp. 11104–11109, IEEE, Dalian, China, July 2017.

[14] T. Mu, H. Chen, J. Du, and A. Xu, "An android malware detection method using deep learning based on api calls," in *Proceedings of the 2019 IEEE 3rd Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC)*, IEEE, Chongqing, China, October 2019.

[15] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[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 Transactions on Information Forensics and Security*, vol. 14, no. 3, pp. 773–788, 2019.

[17] M. Masum and H. Shahriar, "Droid-nnet: deep learning neural network for android malware detection," in *Proceedings of the 2019 IEEE International Conference on Big Data (Big Data)*, pp. 5789–5793, IEEE, Los Angeles, CA, USA, December 2019.

[18] A. Azmoodeh, A. Dehghantanha, and K.-K. R. Choo, "Robust malware detection for internet of (battlefield) things devices using deep eigenspace learning," *IEEE transactions on sustainable computing*, vol. 4, no. 1, pp. 88–95, 2019.

[19] J. Feng, L. Shen, Z. Chen, Y. Wang, and H. Li, "A two-layer deep learning method for android malware detection using network traffic," *IEEE Access*, vol. 8, pp. 125786–125796, 2020.

[20] A. Alotaibi, "Identifying malicious software using deep residual long-short term memory," *IEEE Access*, vol. 7, pp. 163128–163137, 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 *Proceedings of the 2018 19th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)*, pp. 140–145, IEEE, Busan, Korea, June 2018.

[22] J. Singh, D. Thakur, T. Gera, B. Shah, T. Abuhmed, and F. Ali, "Classification and Analysis of Android Malware Images Using Feature Fusion Technique," *Journals and Magazines*, vol. 9, pp. 90102–90117, 2021.

[23] G. D'Angelo, M. Ficco, and F. Palmieri, "Association rule-based malware classification using common subsequences of api calls," *Applied Soft Computing*, vol. 105, Article ID 107234, 2021.

[24] M. Ficco, "Malware analysis by combining multiple detectors and observation windows," *IEEE Transactions on Computers*, vol. 71, 2021.

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

[26] G. Suarez-Tangil, S. K. Dash, M. Ahmadi, J. Kinder, G. Giacinto, and L. Cavallaro, "Droidsieve: fast and accurate classification of obfuscated android malware," in *Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy*, pp. 309–320, Arizona, USE, March 2017.

[27] G. Suarez-Tangil and G. Stringhini, "Eight years of rider measurement in the android malware ecosystem: evolution and lessons learned," 2018, https://arxiv.org/abs/1801.08115.

[28] H. Cai, "Embracing mobile app evolution via continuous ecosystem mining and characterization," in *Proceedings of the IEEE/ACM 7th International Conference on Mobile Software Engineering and Systems*, pp. 31–35, New Year, NY, USA, July 2020.

[29] H. Cai, X. Fu, and A. Hamou-Lhadj, "A study of run-time behavioral evolution of benign versus malicious apps in android," *Information and Software Technology*, vol. 122, Article ID 106291, 2020.

[30] H. Cai and B. Ryder, "A longitudinal study of application structure and behaviors in android," *IEEE Transactions on Software Engineering*, vol. 47, no. 12, pp. 2934–2955, 2021.

[31] X. Fu and H. Cai, "On the deterioration of learning-based malware detectors for android," in *Proceedings of the 2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, pp. 272-273, IEEE, Montreal, Canada, May 2019.

[32] K. Xu, Y. Li, R. Deng, K. Chen, and J. Xu, "Droidevolver: self-evolving android malware detection system," in *Proceedings of the 2019 IEEE European Symposium on Security and Privacy (EuroS&P)*, pp. 47–62, IEEE, Stockholm, Sweden, June 2019.

[33] E. Mariconti, L. Onwuzurike, P. Andriotis, E. De Cristofaro, G. Ross, and G. Stringhini, "Mamadroid: detecting android malware by building Markov chains of behavioral models," 2016, https://arxiv.org/pdf/1612.04433.pdf.

[34] H. Cai, "Assessing and improving malware detection sustainability through app evolution studies," *ACM Transactions on Software Engineering and Methodology*, vol. 29, no. 2, pp. 1–28, 2020.

[35] H. Cai and J. Jenkins, "Towards sustainable android malware detection," in *Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings*, pp. 350-351, New Year, NY, USA, May 2018.

[36] H. Fereidooni, M. Conti, D. Yao, and A. Sperduti, "Anastasia: android malware detection using static analysis of applications," in *Proceedings of the 2016 8th IFIP International Conference on New Technologies, Mobility and Security (NTMS)*, pp. 1–5, IEEE, Larnaca, Cyprus, November 2016.

[37] M. Bierma, E. Gustafson, J. Erickson, D. Fritz, and Y. R. Choe, "Andlantis: large-scale android dynamic analysis," 2014, https://arxiv.org/abs/1410.7751.

[38] J. Singh, D. Thakur, F. Ali, T. Gera, and K. S. Kwak, "Deep feature extraction and classification of android malware images," *Sensors*, vol. 20, no. 24, p. 7013, 2020.

[39] L. N. Vu and S. Jung, "Admat: a cnn-on-matrix approach to android malware detection and classification," *IEEE Access*, vol. 9, pp. 39680–39694, 2021.

[40] K. Liu, S. Xu, G. Xu, M. Zhang, D. Sun, and H. Liu, "A review of android malware detection approaches based on machine learning," *IEEE Access*, vol. 8, pp. 124579–124607, 2020.

[41] Y. Jiang, R. Li, J. Tang, A. Davanian, and H. Yin, "Aomdroid: detecting obfuscation variants of android malware using transfer learning," in *International Conference on Security and Privacy in Communication Systems*, pp. 242–253, Springer, Berelin, Germany, 2020.

[42] K. Weiss, T. M. Khoshgoftaar, and D. Wang, "A survey of transfer learning," *Journal of Big data*, vol. 3, no. 1, pp. 9–40, 2016.

[43] M. Abadi, P. Barham, J. Chen et al., "Tensorflow: a system for large-scale machine learning," in *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI '16)*, pp. 265–283, Savannah, GA, USA, November 2016.

[44] A. Paszke, S. Gross, F. Massa et al., "Pytorch: an imperative style, high-performance deep learning library," *Advances in Neural Information Processing Systems*, vol. 32, pp. 8026–8037, 2019.

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

[46] S. Lou, S. Cheng, J. Huang, and F. Jiang, "Tfdroid: android malware detection by topics and sensitive data flows using machine learning techniques," in *Proceedings of the 2019 IEEE 2Nd International Conference on Information and Computer Technologies (ICICT)*, pp. 30–36, IEEE, Kahului, HI, USA, March 2019.

[47] R. Kozik, M. Choraś, M. Ficco, and F. Palmieri, "A scalable distributed machine learning approach for attack detection in edge computing environments," *Journal of Parallel and Distributed Computing*, vol. 119, pp. 18–26, 2018.

[48] Y. Huang, X. Li, M. Qiao et al., "Android-sem: generative adversarial network for android malware semantic enhancement model based on transfer learning," *Electronics*, vol. 11, no. 5, p. 672, 2022.

[49] S. K. Dash, G. Suarez-Tangil, S. Khan et al., "Droidscribe: classifying android malware based on runtime behavior," in *Proceedings of the 2016 IEEE Security and Privacy Workshops (SPW)*, pp. 252–261, IEEE, San Jose, CA, USA, May 2016.

[50] M. K. Alzaylaee, S. Y. Yerima, and S. Sezer, "DL-Droid: deep learning based android malware detection using real devices," *Computers & Security*, vol. 89, Article ID 101663, 2020.

[51] H. Cai, "A preliminary study on the sustainability of android malware detection," 2018, https://arxiv.org/abs/1807.08221.

[52] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent dirichlet allocation," *The Journal of Machine Learning Research*, vol. 3, pp. 993–1022, 2003.

[53] F. Kolini and L. Janczewski, *Clustering and Topic Modelling: A New Approach for Analysis of National Cyber Security Strategies*, Association for Information Systems (AIS) eLibrary, Greece, 2017.

[54] X. Lu, X. Zhou, W. Wang, P. Lio, and P. Hui, "Domain-oriented topic discovery based on features extraction and topic clustering," *IEEE Access*, vol. 8, pp. 93648–93662, 2020.

[55] K. Soman, R. Loganathan, and V. Ajay, *Machine Learning with SVM and Other Kernel Methods*, PHI Learning Pvt. Ltd, Delhi, USA, 2009.

[56] S. Winters-Hilt and S. Merat, "Svm clustering," *BMC Bioinformatics*, vol. 8, pp. S18–S12, 2007.

[57] H. Altuwaijri and S. Ghouzali, "Android data storage security: a review," *Journal of King Saud University-Computer and Information Sciences*, vol. 32, no. 5, pp. 543–552, 2020.

[58] Z. Yong, L. Youwen, and X. Shixiong, "An improved knn text classification algorithm based on clustering," *Journal of Computers*, vol. 4, no. 3, pp. 230–237, 2009.

[59] I. Gronau and S. Moran, "Optimal implementations of upgma and other common clustering algorithms," *Information Processing Letters*, vol. 104, no. 6, pp. 205–210, 2007.