Research Article

A Modified ResNeXt for Android Malware Identification and Classification

Marwan Ali Albahar 1, Mahmoud Said ElSayed, 2 and Anca Jurcut 2

1 School of Computer Science, Umm Al-Qura University, Mecca, Saudi Arabia
2 School of Computer Science, University College Dublin, Belfield, Dublin, Ireland

Correspondence should be addressed to Marwan Ali Albahar; marwanalialbahar@gmail.com

Received 23 February 2022; Revised 15 March 2022; Accepted 28 April 2022; Published 20 May 2022

Copyright © 2022 Marwan Ali Albahar et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

It is critical to successfully identify, mitigate, and fight against Android malware assaults, since Android malware has long been a significant threat to the security of Android applications. Identifying and categorizing dangerous applications into categories that are similar to one another are especially important in the development of a safe Android app ecosystem. The categorization of malware families may be used to improve the efficiency of the malware detection process as well as to systematically identify malicious trends. In this study, we proposed a modified ResNeXt model by embedding a new regularization technique to improve the classification task. In addition, we present a comprehensive evaluation of the Android malware classification and detection using our modified ResNeXt. The nonintuitive malware’s features are converted into fingerprint images in order to extract the rich information from the input data. In addition, we applied fine-tuned deep learning (DL) based on the convolutional neural network (CNN) on the visualized malware samples to automatically obtain the discriminatory features that separate normal from malicious data. Using DL techniques not only avoids the domain expert costs but also eliminates the frequent need for the feature engineering methods. Furthermore, we evaluated the effectiveness of the modified ResNeXt model in the classification process by testing a total of fifteen different combinations of the Android malware image sections on the Drebin dataset. In this study, we only use grayscale malware images from a modified ResNeXt to analyze the malware samples. The experimental results show that the modified ResNeXt successfully achieved an accuracy of 98.25% using Android certificates only. Furthermore, we undertook extensive trials on the dataset in order to confirm the efficacy of our methodology, and we compared our approach with several existing methods. Finally, this article reveals the evaluation of different models and a much more precise option for malware identification.

1. Introduction

Malware has many different definitions specified by different scholars and researchers depending on the attack vector deployed or harm caused. However, all researchers agreed on the same meaning in general, that is, malware applications have an evil intent [1]. Malicious software (malware) is any software with malicious intent. Malicious code is designed to disrupt normal functioning, display unwanted advertising, control the user’s device without their awareness or knowledge, steal or gather sensitive information, and delete or encrypt important data [2]. Unintentionally, harmful software and malware are collectively referred to as “bad ware.” The newly developed malware, which is very sophisticated, can obstruct emulators and elude deep static analysis. Malware can also be spread through metamorphic techniques such as instruction permutation, registry modification, encryption, antidebugging, multipacket, virtual machines, and code transformation. It has the capability to launch the payload intelligently to evade detection techniques [3–7]. Many new variants of malware can be generated using automation and reused development modules [8–10]. Like computer systems, malware systems have evolved enormously to be smarter, more intelligent, and more decisive. The main categories of malware are botnets, ransomware, viruses, rootkits, worms, and Trojans [10].
Malware can avoid detection by using polymorphic and metamorphic techniques [3, 11–13]. Malware developers frequently modify minor portions of the original source code in order to create new variants and avoid detection [9, 14, 15]. This makes it extremely difficult to distinguish malware variants from the same family [16, 17].

Malicious Android applications can infiltrate smartphones to be able to do anything without the user’s knowledge, such as stealing information, blocking access to critical information from the device, or even mine cryptocurrency. Currently, the rate shows an incredibly high increase in Android malware samples (malicious Android apps) and their variants keep proliferating. The McAfee mobile threat report in Q1 2020 [18] announced that the size of new detected malware attacks reached 800,000 in the 4th quarter of 2019, exceeding the previous quarter of 35 million malware attacks.

Some of these attacks are not easy to detect since the attacker mimic the same normal behavior. For example, the attackers develop a new malware, named MalBus, to avoid any detection by using the original developer’s Google Play account.

Moreover, the attacker can collect sensitive military and political information using scannable devices as well as target Google account login information using a phishing-style fake login page. Besides, a new Android malware family (i.e., LeifAccess), which is also known as Shopper, works by exploiting accessibility advantages to create accounts and post fake reviews on the Play Store. After installing it, it promotes click fraud without displaying an icon or a shortcut. As a result of the rising threat from an ever-increasing number of mobile malware instances and new malware families emerging, the Android ecosystem’s security will be affected. In order to combat this threat and protect mobile users and systems [19, 20], many studies have been conducted to look for ways to detect and classify Android malware samples. As follows, there are many problems related to the classification and detection of malware: the problems of binary classification in which an app’s malignancy is attempted to be determined, multiclass classification problems that include attempting to classify detected malicious apps into a known or unknown family, which is related to identifying malware families, and many others. So, malware researchers should be focusing their attention on the most dangerous families instead of focusing on individual malware samples or lower-risk families if samples are correctly classified and accurately characterized [19]. As a result, an effective malware family classification can assist malware analysts in identifying more malware samples by recognizing and understanding the characteristics of other malware samples in a family. On the other hand, there is a more challenging task than malware detection, which is the classification of malware families. The reason for that is that the numbers of malware samples vary extremely between different families [19–22].

Windows malware families have been the subject of several malware classification studies [9, 17, 23–25]. Due to the different structures and characteristics between Windows malware and Android malware, it is not applicable to utilize the same methodologies to categorize Windows malware families for Android malware families [25, 26]. Consequently, Android malware samples and variations have recently received a lot of attention from academics and the industry.

A recent publication [27] sheds light on the advancement of malware detection through the use of the ResNeXt model. This advancement is due to the architecture of the model, which combines the features of the ResNet and InceptionNet architectures. In addition, it requires low flops and applies the skip concept from ResNet architecture. Motivated by this fact, we proposed a modified ResNeXt model for Android malware classification. The proposed model operates on raw bytes, obviating the need for decryption, disassembly execution of code, and reverse engineering in order to identify malware. To extract the quality information, we converted the malware’s nonintuitive features into fingerprint images. Seeing through malware binary, we evaluated the performance and generation of the proposed model to view the capability of discovering and extracting insights necessary for malware analysis and to pave the path for the development of effective malware classification systems.

The main contributions of this work are enumerated as follows:

(i) Proposes an effective modified ResNext classification network for automatically classifying Android malware families from raw malware samples. The Drebin dataset was used to test and validate the proposed system. This dataset contains 5560 applications from 179 different malware families.

(ii) Investigates fifteen different combinations of Android malware file structures in order to classify and generate malware images. In addition, we observed that CR combination of malware image is the most suited feature for malware identification and classification.

(iii) We extract composite features by designing a modified ResNext with a new regularization technique. In particular, we used the standard deviation of the weight matrix to create an adaptive weight decay form in order to prevent the model from taking values.

The rest of this study is organised as follows: Section 2 provides a brief background about the structure of the API files, CNN, deep residual networks, and regularization techniques. Section 3 provides an in-depth view of the existing promising countermeasures that have been produced to monitor and detect Android malware categories. The dataset analysis, methodology, and proposed model are described in Section 4. Section 5 discusses the experimental results and analysis, while an overall discussion of this research and the limitations are introduced in Section 6. Finally, the study’s conclusion is discussed in the last section.
2. Background Theory

This section provides a brief background to introduce the structure of the API files and other classifiers and regularization techniques, which are applied for our classification problem.

2.1. Structure of APK Files. Android has become one of the most popular operating systems for smartphones. Since the Android operating system is open source, cybercriminals are attracted to using it. This section goes over the Android application, common Android malware families, Android malware analysis, and Android malware detection techniques. Generally, Android applications are mainly written using the Java programming language and then gather data and source files into an archive file called the Android application package (APK) (Android package) [28]. The APK is shared in the application market and is used for the installation of applications. However, the APK is a ZIP file consisting of multiple files, making it necessary to unzip it before use [29]. The Android APK structure is shown in Figure 1, and the details of each file and folder component are given in Table 1.

Malware families are groups of malwares with similar characteristics, behavior, and capabilities, such as stealing information from a location or a remote server, sending paid or malicious SMS messages, and so on. Malicious behavior uses the same package names as the attack for injecting a payload. In addition, the identity (signature) of a group of malwares (family) can be determined by repeating the use of package names (or other common characteristics) [19]. The most common malware families are given in Table 2 [18].

A significant amount of time is required to manually create features throughout the Android package (APK) structure for Android malware family classifications [3, 4, 30, 34, 35]. These safety mechanisms require significant computer resources, and their deployment in a restricted smartphone environment is challenging [31]. Android malware traces have been studied through classes.dex (CL), resource (RS), manifest, and Android application certificate (CR) files. Malware detection technology, as well as malicious code, have both been developed over time. It is necessary to analyze malware in order to be able to detect it. There are several methods for analyzing and classifying malware, including static, dynamic, hybrid, visualization (image), and audio [32, 33]. Static analysis and classification are the most common methods. Anti-malware signatures and behavioral techniques such as static and dynamic analyses are the most important techniques for identifying malware. Intelligent malware, on the other hand, employs dynamic analysis in conjunction with antiemulation technology [28, 31, 36]. In order to use dynamic and static techniques on such files, a significant amount of manual effort or human intervention is required. To reverse engineer or analyze an application, it is necessary to have prior knowledge of the domain [29, 37–40]. Table 3 provides a comparison between two types of analysis, which are static and dynamic analyses.

2.2. Convolutional Neural Network (CNN). CNN is one of the neural network (NN) algorithms widely used for computer vision. However, in the traditional NN, the hidden layers are fully connected with each other, which
Table 1: Structure of APK files.

| Reference | APK folders/files | Responsibilities |
|-----------|-------------------|------------------|
| AndroidManifest.xml | It is one of the most important files in the Android application, which stores the basic information for the applications and includes the app components, such as activities, services, broadcast receivers, content providers, and others, in addition to package information, such as permissions, package name, and app ID. It also reveals the SDK version. |
| Assets/folder | The assets include the assets of an application directory, like images and files, which can be put in this folder and accessed by the asset manager object to retrieve the application assets detailed in the assets folder. |
| Lib/folder | This folder contains the native code libraries. The software layer of a processor relates to a specific type of gather inside in this folder. |
| META-INF/folder | This includes three main files, which are the signatures certifications, and manifest files for the APK such as MANIFEST.MF, SF, and *.RSA. |
| Res/folder | This folder includes a description of the resources such as icons, music, images, string, resources, and layouts. These resources are not compiled in resources, arsc folder. |
| Classes.dex | Dex code represents bytecode for Android applications which is generated after the compilation of the Java code. which contains multiple constructs for all classes composed like file header, string table, local variable list, class definition table, and method list and can be understandable by the Dalvik virtual machine. Any change in the dex file will affect the APK. |
| Resources.arsc | This includes an application’s resources in a binary format, like strings, styles, and the paths of images or layouts files, which are a part of this content. However, the data can only be processed in an XML format. |

Table 2: Common android malware families.

| Common android malware families |
|-------------------------------|
| Accu Track | Counterclank | FakeTaoBao | Kidlogger | Placms | SpyOO |
| Ackposts | Crusewind | FakeTimer | KMIN | Plankton | Steek/Fatak |
| Acnetdoor | Dogowar | FakeUpdate/Apkqug | Kapp | Podec | Ssucl |
| Adsms | Dougalek | Fakevertu | LeNa | PoisonCake | Tascudap |
| Airpush/StopSMS | DroidDeluxe | Find and Call/Fidall | Lien/ | Proxy/Trojan/NotCompatible/NioServ | TapSnake/Droisnake |
| Anserver/Answerbot | DroidDream | Finspy | Locker/SLocker/ | Qiccomos | TGloader/Stimer |
| Antares/ | DroidJack/ | Fjcon | Raden | Radium | TigerBot |
| Antammi | SandoRAT | | | | |
| Arsam | DroidDreamLight | Flexispy | Loozion | Repane | Tetus |
| AVpass | DroidKungfu | Foncy | Lovetrapp/ | Roidsec/Sinpon | Titan |
| BackFlash/Crosate | Droidsheep | Fokange/Fokonge | Luckycat | RootSmart/ | Tonclank |
| Badaccents | DSEncrypt | Fonflee/Feefar | Maister | Bmaster | Tracer |
| Badnews | Extension/Modad | Gamex | Malap | RuFrad | TypStu |
| BankBot | FaceNiff | Gazon | Mania | Samsapo | UpdBot |
| Basebridge | FakeAngry | Geinimi | MMMarketPay | Scavir | UpdKiller |
| BeanBot | FakeApp.AL | GGTracker | MobiDash | Scipiex | Uracto |
| Beita | FakeAV | GingerBreak | MobileTx | SaveMe/SocialPath | USBeleaver |
| Binv | FakeDaum/Vmwo | GingerMaster/ | Mobinauten | SMMsniffier | Uten |
| BgServ | FakeBank | GingerBreaker | | Spambot | Uxipp |
| Biige | FakeDefender | GoldenEagle/GlodEagl | GODwon | Moghava | SeaWeth |
| | | | | Nandrobox | Seltmte |
| | | | | Netisend | Smack |
| | | | | Nickisp | Vloader |
| | | | | Obad | |
| | | | | Oldboot/MouaBad | |
| | | | | OpFake | |
| | | | | SSM Silence/SMSCatcher | |
significantly increases the number of training parameters, and this can increase the complexity of the classifier. The CNN produces the concept of parameter sharing in each layer to solve the limitations of the traditional NN and reduce the explosion of weight vectors. The weight sharing concept of the CNN can reduce the computational cost and the training time of the model classifier compared to the other DL models. The CNN is constructed from three core layers, i.e., convolution, pooling, and fully connected layers [48]. The simple architecture of the CNN is shown in Figure 1. The convolution layer is produced as a result of the linear operation of the kernel or filter with the previous output layer. The Relue activation function is widely used in the CNN to increase the degree of nonlinearity and to remove also all negative values of feature maps. The CNN can include more than a convolutional layer. The first layer is used to capture the simple features such as corners or edges. While, the higher layers are mainly used to learn the high-level features. The CNN does not only have the capability to extract the discriminatory features from the input data but can also reduce the spatial size of the convolved feature through the pooling layers. The CNN is implemented in several available architectures, such as ZFNet [49], ResNet [50], GoogLeNet [51], VGGNet [52], AlexNet [53], and LeNet [54]. Motivated by the success of the CNN in various application domains, we utilize the CNN for Android malware detection. The parameters’ sharing and the concept of dimensionality reduction are the factors that inspired us to use the CNN for the detection problem.

2.3. Deep Residual Networks. In 2016, Microsoft Research Lab [55] released the deep residual network to solve the inherent problems of traditional deeper networks. The traditional models are more difficult to train and exposed to the degradation problem (of training accuracy). Adding new

| Table 2: Continued. |
|---------------------|
| Bosster | FakeDoc | Gomeln60seconds |
| Boxer | FakeFlash | GP spy |
| Cajino | FakeInst | HefHe |
| Carberp | FakeJobOffer | Hidelcon |
| Cawitt | FakeMarket | HippoSMS |
| Code4hk/xRAT | FakeMart | HongTouTou/Ard |
| Chuli Cellspy | FakeNeflx | Iconsys |
| Coogos | FakeNotify | Imlog |
| CopyCat Cosha | FakePlay | Jifake |
| Cosha | FakePlayer | JollyServ |
| FakeRegSMS | Jismshider/X sider |

| Table 3: Compare between static and dynamic analyses. |
|-----------|
| Static | Dynamic |
| How it works | The suspected code is analyzed without the application being run during static analysis. This method involves disassembly of source code and analyzing it to check the presence of malware without executing the source code and depend only on malware abstraction characteristics and application byte code. Mostly, reengineering is applied [41–43]. |
| The suspected code is analyzed during the runtime execution. It focuses on the characteristics and traces of suspicious use during implementation [24–26, 44–46] and also focuses on system calls and application programs. It is a cybernetic environment used for the execution of code. |
| Advantage | Harmful applications are not needed to be installed on the device. |
| Do not execute or run the malware code. |
| Applications are in format APK or archive in a zip package [41–43]. |
| It can detect dependencies that are impossible to detect in the static method. |
| Collects temporal instructions. |
| Deals with real data, whereas in the static analysis, you cannot know input files to be passed for analysis. |
| It can overcome string detection issues, such as malware fitting and pleomorphism [41–43]. |
| Disadvantage | This technique does not take into consideration the analysis of unknown malware. |
| The source codes used are not directly available, and it must be disassembled to extract the features [16–21]. |
| Harmful applications cannot appear until the code has been run. |
| Suffers from code obfuscation [9, 17, 22, 23]. |
| Can have a negative performance impact on the application. |
| Requires better mobile security at critical monitoring stages. |
| It can give incorrect results for similar behavior of the malicious applications with starting applications. |
| It is a complex and time-consuming technique that requires high resource usage and storage capacity [22, 47]. |
layers to the deeper networks not necessarily improves the training accuracy. In most cases, the accuracy becomes saturated and will degrade rapidly with increasing of the network depth, and this can lead to high training errors. On the other hand, the deep residual network works to overcome the aforementioned limitations by using the residual blocks. The simple structure of residual building block is shown in Figure 2.

The potential of this type of networks lies on the concept of "skip connections" to improve the accuracy of the models. Instead of the consecutive connections of the layers in the neural network, some of the layers are skipped and feed the output of one layer as the input to the next layers. The residual blocks are stacked together in a sequential way to consist the residual. The identity mapping of deep residual is descried according to the following formula:

$$X_{i+1} = X_i + F(X_i, W_i),$$

where $F$ is the residual function, $W_i$ is the weight parameters of the block, and $X_i$ and $X_{i+1}$ represent the input and output of the $i^{th}$ unit in the network. In recent days, the deep residual approved its robustness in several computer vision applications. Motivated by its successfully achieved results in different tasks, we also utilized deep residual in this article for Android Malware detection.

### 2.4. Regularizer Technique

#### 2.4.1. Regularization: A Method of Controlling the Model from Complexity.

One of the big challenges in machine learning is how to build a more robust model that can perform effectively in the training data and the new testing data as well. However, the overfitting is one of the significant problems that can hinder the normal operation of machine learning techniques. The model can perform very well during the training, but unfortunately, it performs very poor in the new testing samples, which eliminates its wild implementation for zero-day attacks. The reason behind that returns to the complexity of the training model, resulting from large number of training parameters. As a result, the model can learn the noise in the input data as specific features to discriminate between different attack classes. To eliminate this problem and reduce the error of the prediction model, we can use regularization techniques. The key idea behind the regularization methods is to penalize the model by dropping some of its weight parameters, and this can increase the model’s performance on unseen data detection. There are many several techniques widely used for the regularization process. L1 and L2 are the most two popular regularizer methods and comprehensively used in the domain area of machine learning [56]. In the following subsections, we will discuss these two methods in detail.

#### 2.4.2. L1 or Lasso Regularizer.

In the L1 regularizer, the absolute value of the magnitude is added to the new loss function. Penalizing the model with the absolute value will make the weight parameters of the insignificant features to reach to zero. As a result, these features will be ignored totally from the model training, i.e., are not contributing any more for the classifier’s boundaries. So, we can find the L1 regularizer is widely used for feature selection purposes to selectively reduce or eliminate unnecessarily features [56]. The L1 regularizer is denoted mathematically as follows.

$$\lambda \sum_{i=1}^{n} |w_i|,$$

where $\lambda$ is the new hyperparameter used for regularization, $n$ represents the number of features in dataset, and $w$ gives the weight values of each feature.

#### 2.4.3. L2 or Ridge Regularizer.

In the L2 regularizer [56], we added the square value of the magnitude to the new loss function instead of the absolute value like in L1. Thus, the weight of less important features will reach close to zero but never reach digit zero itself, i.e., the weight matrix will remain greater than zero. L2 regularizer provides better performance with low loss compared to L1. Since, it considers all features during the training process [57]. The mathematical notation of the L2 regularizer is discussed in the following equation:

$$\lambda \sum_{i=1}^{n} w_i^2,$$

where $\lambda$ is the hyperparameter parameter and used to impose an additional penalty on the corresponding weight values and $n$ and $w$ represent the number of features and the coefficient value of each feature, respectively.

#### 2.4.4. New Regularizer.

As we explained in the aforementioned section above, the L1 regularizer is widely used for feature selection, while the L2 regularizer gives less importance to less significant features. However, both regularizers have substantial drawbacks and inherit some limitations, which hinder their broad use to detect zero-day threats. Unfortunately, L1 and L2 regularizers do not take into account the relationship between entries in a weight
matrix. In other words, they only deal with individual weight values. However, any change in the feature attributes, even a small change, can cause a big difference in the model performance. To solve this problem, we developed a new regularizer method (SD-Reg) based on the standard deviation to efficiently deal with the weight values’ dispersion, i.e., the SD-Reg regularizer restrains the learning model of using a diaphraged range of weight space. It works to create a weight-decay adaptive form by considering the standard deviation of a weight matrix and multiplies it by $\lambda$ parameter to obtain its regularization term.

It is shown in equations (3)–(5) how to formulate the new regularizer.

$$\lambda \sigma (w),$$

where $\sigma$ indicates the standard deviation of the following weight values:

$$\sigma (w) = \sqrt{\frac{1}{nk} \left( \sum_{i=1}^{nk} w_i^2 - \frac{1}{nk} \left( \sum_{i=1}^{nk} w_i \right)^2 \right)}.$$ (5)

For each row of the weight matrix, there are $k$ rows; each row corresponds to an individual weight. Whereas, $\sigma$ stands for the standard deviation’s weighted averages. $n$ is the number of columns in each $i^{th}$ row of the weight matrix, which is controlled by the parameter $\lambda$. The weight vector has a length of $n$. In our case, the loss function is

$$\min_w \{ f(X, y; w) + \lambda \sigma (w) \}.$$ (6)

Thus, the standard deviation of $w$ is used to minimize the loss function for $w$ in order to select a specific range of values.

3. Literature Review

Recent years have witnessed a significant increase in the number of Android malware cyberattacks. On the other hand, there are substantial efforts by malware researchers and cybersecurity scholars to develop new reliable techniques in order to identify and mitigate the frequent development of these attacks. As a result, the automated Android malware detection (AMD) to deal with this critical cybersecurity challenge has increased too and cannot be neglected.

On the other hand, comprehensive AMD approaches have witnessed a significant increase in the use of ML and DL techniques to ensure the security of the Android ecosystem. This section represents the most widely employed ML and DL approaches for Android malware detection.

3.1. Malware Detection Based on ML. Several ML algorithms such as logistic regression (LR), random forest (RF), k-nearest neighbor (KNN), and support vector machine (SVM) have been used for Malware attack detection.

In [58], three ML classifiers, namely, KNN, RF, and DT were used for AMD on APK samples, which consisted of 300 benign samples and 183 malwares. The Dalvik opcode was extracted from the classes.dex file and then converted into an 8-bit grayscale image. The GIST descriptor was used later to extract the input attributes from the created images. The experimental results showed that RF provided better accuracy of 84.14%.

Li et al. [59] produced a lightweight model for AMD using the SVM algorithm on a dataset downloaded from Google Play store. The authors used less number of permissions for malware classification instead of using all the requested permissions, i.e., selected the top permissions that are more specific for benign and the top permissions for malware to build the malware detection system.

Wang and Li [60] utilized three different ML algorithms, including NB, DT and K-nearest neighbors, for AMD. The feature selection and reduction techniques were used at the first stage to reduce the number of training features and to create a lightweight model with less number of features.

In the study by Massarelli et al. [20], the dynamic analysis was used to find the resource consumption metrics, including CPU, memory, and network usage, as input features of the training classifier. The SVM was utilized to identify several Android malware families on the Drebin dataset.

Chen et al. [61] proposed TinyDroid for AMD. The authors extracted the opcode sequence from Dalvik Executable files and then used the n-gram to extract the trained features from the opcode data. Four ML algorithms, RF, KNN, SVM, and NB, were used for the classification process to identify the malicious malware from normal APKs.

However, most of the aforementioned methods mainly used feature engineering in order to find the best features of the input data. However, selecting the best features is not an easy task, since the features that can work efficiently for one malware class are not necessarily important for other classes. Besides, the input data almost has a high degree of non-linearity, and shallow learners have a poor ability to learn the complex and nonlinear structure of the data.

3.2. Malware Detection Based on DL. In recent days, deep learning (DL) techniques have been conducted in several application domains, such as image processing and speech recognition [62]. The high potential of DL in several applications returns to its capability to extract the representation features from the input data automatically without any human intervention. It has the good capability to work in data, which has a high degree of nonlinearity, in contrast to shallow learners, almost experienced with high false alarms as they require hand-crafted features as input. The high performance of DL in intensive domain areas encourages many organizations and enterprises such as Facebook, Google, and Microsoft to deploy DL in various applications [63]. Researchers are also starting to leverage DL for cybersecurity tasks and malware detection as well.

Huang and Kao [64] proposed a new model, namely, R2D2 for detecting the Android malware. The authors first obtained fixed size colored images from the classes.dex bytecode of the Android archive file. Then, different network models based on the CNN algorithm were used for the
training and classification processes. A realistic dataset with more than 2 million of benign and malicious malware samples was collected in the period from January 2017 to August 2017 for the research work.

Hardy et al. [65] used the stacked autoencoders- (SAEs-) based DL model for malware detection. The model composed of two phases. In the first phase, unsupervised learning was used to extract the discriminatory features of the input data. The API calls extracted from the Portable Executable (PE) files were used as input features for the DL model. In the second stage, fine-tuning based on supervised learning was used to adapt the weights and offset vectors. A dataset collected from the Comodo Cloud Security Center is used for a comprehensive experimental study to compare various malware detection models.

Kim et al. [66] proposed an Android malware detection model using a multimodal deep learning method. Seven diverse features have been extracted from the unzipped APK, i.e., shared library, dex, and manifest files. The collected features are merged together to create a fixed feature vector. In the classification process, the DNN was used on malware samples produced from three different sources, i.e., VirusShare, MalGenome project, and Google Play app store.

Another detection approach, i.e., MalDozer was proposed by Karbab [67] for Android malware classification. The CNN with one convolutional layer followed by another softmax layer and one fully connected layer was used to detect samples of unknown malware families. The classifier approach has been applied on three different datasets, including MalGenome, Drebin, and merged datasets. While, API calls that appear in the DEX file were extracted for input attributes.

Nix and Zhang [68] investigated the CNN for Android application/malware classification. The code included within the classes.dex was examined to obtain API calls as an input attributes. The CNN approach was compared with LSTM and other n-gram-based methods on a dataset collected from the Contagio Mobile repository. The results showed that the CNN outperformed the other classifier techniques.

Suarez-Tangil et al. [3] proposed DroidSieve to classify the malware samples using a static solution. Several features have been used to identify the normal samples from the malicious malware, such as API calls, native code, invoked components, code structure, and permissions. The authors used Drebin and MalGenome datasets for their experimental evaluation. However, the reported results relied on the Drebin dataset during its large size and covered all MalGenome samples. The obtained results approved that the DroidSieve approach successfully achieved high accuracy when using resource-centered features and reducing code analysis.

Along with dynamic features derived from an application’s behavioral profile, such as method calls and inter-component communication (ICC) intents, DroidCat [69] identified and classified Android malware. Using apps that have evolved over the last nine years, it classified them with 97% accuracy. It was able to defeat attacks that targeted system calls or sensitive APIs, as well as malware samples that used obfuscation schemes. It outperformed two state-of-the-art techniques in terms of detection accuracy, using MalGenome, Drebin, AndroZoo, and VirusShare datasets [70]. It relied on a variety of machine learning algorithms, including SVM, Naive Bayes, and RF. The RF with 128 trees outperformed all other methods. Because the dynamic malware analysis technique proposed by Picco [71] is composed of a combination of generic and specialized detectors that are used throughout the analysis process, it is resistant to specific evasion techniques. To address malware evolution, the proposed technique utilized an alpha-count mechanism to investigate the effect of varying the length of the observation time window during run-time on the accuracy and speed of detectors. He demonstrated the technique’s efficacy using data from 27 DREBIN families. Additionally, a second validation dataset, spanning the period June 2013 to March 2014, was used as a validation dataset taken from the VirusShare dataset.

In this research work, we propose an accurate and automated vision-based AMD model to deal with the critical cybersecurity challenges, which are difficult to ignore. A fine-tuned DL-based CNN algorithm is developed to efficiently detect malware attacks on Android OS.

On the Drebin dataset, we test several combinations and compare our approach to some state-of-the-art works, such as LeNet, Inception V3, ResNet50, Vgg16, EfficientNetB0, and SARVOTAM. All these methods are considered 2D convolution filter-based models. Also, we conduct extensive experiments and include the basic-1D-CNN with single and multisreams in our computations. The results indicate that the proposed ResNeXt achieved significant results in terms of accuracy and F1-score. In addition, the extensive computations of the ResNext model are significantly lower since it requires a smaller number of features to be analyzed compared to other methods. As a result, the modified ResNext demonstrates its effectiveness by quickly distinguishing Android malware from benign apps with the fewest recorded errors.

4. Materials and Methods

This section describes the methodology and the used dataset for our proposed model. The architecture and detailed explanation of the learning model are discussed in detail, considering the file size and other used parameters for the model tuning.

4.1. Dataset. Most of the studies that have been published between the periods of 2014 and 2020 use the Drebin dataset for training and evaluating their developed models. It is considered one of the most widely used datasets for malware family classification purposes [21]. So, for that reason, it is used in this experiment setup. The Drebin dataset has the most popular Android malware families, which were collected in the period between August 2010 and October 2012. It contains more than 5,560 files belonging to 179 special malware families like Fake Installer, GoldDream [72], GingerMaster [73], DroidKungFu [74], and many others. Table 4 provides the outlines of different malware datasets that have been used by the research community.
4.2. Transforming Malware APK into Images. The fundamental files considered for visualization in APK are classes.dex, resources, manifest, and certificates. In this work, these four types from the malware APK files are employed to extract the malware images, which are used for our model training.

First, the binaries are transformed into 8-bit vectors, and in the next stage, these vectors are converted to grayscale images. The detailed procedure is discussed. Initially, a malware substring consists of a sequence of numerous substrings where each substring is 8-bit long and is called a pixel. The 8-bit substring is converted to a decimal number in the next step, ranging from 0 to 255. Furthermore, all the malware substrings were transformed into a one-dimensional vector and converted to a two-dimensional matrix of a specific width. We called it a “malicious code matrix.” This matrix is considered the two-dimensional grayscale image. The conversion process of APKs to grayscale images is shown in Figure 3. The width of the images was fixed based on the size of the APK files given in Table 5. Hence, the height also depends on the file size. CNN-based models require inputs to be of the same shape. Therefore, instead of trying varying sizes of APK files, we use the dimensions proposed by [26]. The main reason behind the chosen sizes is to retain as much information as required along with keeping the size compact. However, it is empirically decided by [17, 26]. Therefore, to avoid the trial-and-error method for finding the proper sizes, this work follows their proposed procedure. A complete APK can be represented by grayscale images with an underlying structure that follows certain divisions. Fifteen different file structure combinations were used to generate the Drebin Android malware images, each containing at least one image of a distinct malware family. Some images constructed from the files are shown in Figure 4. Classes.dex (CL), AndroidManifest (AM), certificate (CR), and resources (RS) are among the files included.

4.3. Proposed Model. In this work, we use the same ResNeXt classification model, which was proposed by Xie et al. [27] in order to categorize malware families. The basic idea behind ResNeXt is to use an aggregated residual block instead of the basic residual block. This strategy is called “split-transform-merge,” and it was implemented in the inception architecture [27].

The inner product of a synthetic neural network is the weighted sum of the primary neurons in each layer, which is calculated for each layer separately. When viewed through the lens of equation (7), the inner product can be thought of as a type of aggregate transformation.

\[
\sum_{i=1}^{D} w_i x_i, \tag{7}
\]

where \(w_i\) represents the filter weight for the \(i^{th}\) channel of the neuron, and \(x_i\) describes the D-channel input vector of the neuron.

A more inclusive function, which can perform as a network itself, has been developed by Xie at el. instead of a simple aggregating transformation [27]. They demonstrated aggregated transformations as

\[
F(x) = \sum_{i=1}^{C} t_i(x), \tag{8}
\]

where \(t_i(x)\) is an arbitrary function. Analogous to a simple neuron, here \(t_i\) projects \(x\) into an (optionally, low-dimensional) embedding and then transforms it. \(C\) represents the size of the set of transformations to be aggregated, while Xie at el.’ study used \(C\) to represent the cardinality. Their study claimed that the dimension of cardinality can control great numbers of complex transformations. Figure 5 [75] shows 32 cardinality blocks of ResNeXt.

The aggregated transformation in (8) serves as the residual function, as shown in (9).

\[
y = x + \sum_{i=1}^{C} t_i(x). \tag{9}
\]

ResNeXt is designed using ResNet’s skip concept and cardinality with better accuracy than a wide and deep network. In experiments using the ImageNet dataset, ResNeXt outperformed existing models in terms of accuracy. Due to these advantages, we utilize the ResNeXt model.
for malware image classification. The architecture of ResNeXt-50 is shown in Figure 6.

A new regularization technique was incorporated into our model in order to control individual weight values and the relationship between weight matrix entries. This is done by taking the weight matrix’s standard deviation and multiplying it by $\lambda$ to create an adaptive weight decay form. Thus, the regularizer prevents the learning model from taking values from the weight space that are too widely distributed. In fact, the new regularizer has been extensively tested on various tasks with different datasets and proved to be more effective than other regularization methods [57, 76–80].

Our model uses the new regularizer technique to control the relationship between entries using weight values from the weight matrix. To create a weight decay adaptive form, we multiplied the standard deviation of the weight matrix by $\lambda$. This reduces model complexity by removing unnecessary data and keeping only data that are useful for classification. The explained regularizer was tested extensively in different domains, including computer security, and provided better performance than the other regularization techniques.

5. Results

In this section, we briefly discuss the experimental setup and explain the classification results of the proposed approach.

5.1. Classification Module Training and Validation. In this research, we used the Drebin dataset with a focus on the top 20 malware families as follows: BaseBridge, Plankton, DroidDream, SMSreg, FakeInstaller, OpFake, SendPay, FakeRun, ExploitLinuxLotoor, Iconosys, DroidKungFu, Adrd, Glodream, Gappusin, Kmin, MobileTx, GinMaster, and Geinimi. Table 6 provides Drebin malware class combinations and associated instances. The modified ResNeXt has 50 layers except for the input layer and the fully connected layer. 48 of the 50 layers are divided into 16 blocks. Each block contains three layers, with a total of 32 cardinalities.

5.2. Experimental Setup. In this work, all experiments have been executed using Python programming language. Several
machine learning and deep learning libraries such as Keras and TensorFlow are used to build the ResNeXt model.

The simulations were run on a system with 20 GB of RAM and an Intel® CoreTM i3 processor, as well as an NVIDIA GeForce GTX 1080ti graphics processing unit with a frame buffer of 11 GB.

5.3. Experimental Results and Analysis. In our work, the modified ResNext model with an embedded new regularization performed well for approximately 100 epochs. The simulation results were recorded for the Drebin dataset after converting the attributes of raw malware binary executable files to grayscale images. We compared our observed performance measures with various state-of-the-art models, which indicates that the performance of our model is higher (Figure 7). The evaluation is compared for various combinations of the image types as given in Table 7.

The highest accuracy is achieved for the CR combination, as given in Table 7. As a result of the observations and simulation results, it is clear that the maximum amount of relevant information about malicious types is contained in the CR file, resulting in satisfactory classification performance. Apart from classification measures, Table 8 provides a time-based comparison for each combination used in the study and the number of images processed per second that belong to the corresponding class. The relative execution time is within a satisfactory range, which makes it possible to use it in real-time applications.

Thus, after training the model with high-quality classification measures, it can be used for testing in various applications. As given in Table 8, the average processing time for a single image is comparable to previous work [26]. Consequently, once integrated and deployed in software systems, the model’s execution performance will be the same as the state-of-the-art method.

The new regularized technique was implemented in order to control individual weight values as well as the relationship between weight matrix entries in order to eliminate unnecessary data while selectively using only data useful for classification. This demonstrates the effectiveness of our model in extracting better features while also maintaining a reasonable overall running time. The detailed confusion matrix for the top 20 malware families is given in Table 9. Table 10 provides the results of family-specific classifications for the original ResNeXt and our modified ResNeXt. As observed, the new regularization method was adaptive in order to avoid overfitting and to improve the CNN’s ability to predict whether a new observation of the data was not trained on the model. As a result, it enables a more adaptable method of weight loss. As a result, the regularizer prevents the learning model from using global values from the weight space as input. This reduces the complexity of the model and removes unnecessary data, while keeping only the data that are useful for classification. As shown in Figure 8, the proposed model outperformed all the other methods of family-specific classification in terms of F1-score.

5.4. Malware Family Classification Performance Evaluation. It has recently become a problem for machine learning-based malware classifiers to deal with the evolution of malware, which changes its malicious behavior over time, resulting in the deterioration of the classifiers. It has been suggested that deterioration [20, 22] and model aging [17, 21] are better terms to describe this issue of long-term sustainability. Sustainability is defined as the ability of the classifier to sustain its capabilities over time without frequent retraining. Recently, the sustainability challenge associated with machine learning-based malware detection has been discussed, but with limited investigation depth and solutions.

In the same context, the authors in [20, 22] proposed and compared sustainability metrics with the five most recent Android malware detectors. Another study [81]
| Combination | CR | AM | RS | CL | CR+AM | CR+RS | CR+CL | AM+RS | AM+CL | RS+CL | CR+AM+RS | CR+AM+CL | CR+RS+CL | AM+RS+CL | CR+AM+RS+CL |
|-------------|----|----|----|----|-------|-------|-------|-------|-------|-------|----------|----------|----------|----------|-------------|
| No. of      | 1826 | 4659 | 4659 | 4660 | 4659 | 4659 | 4660 | 4659 | 4660 | 4660 | 4659 | 4660 | 4660 | 4660 | 4660 |
Table 7: Generic and augmented CNN accuracies on 15 different grayscale malware image combinations.

| Image combination | CNN (%) | CNN-SVM (%) | CNN-KNN (%) | CNN-RF (%) | VGG16 (%) | GoogLeNet (%) | ResNeXt (%) | Modified ResNeXt (%) |
|-------------------|---------|-------------|-------------|------------|----------|--------------|-------------|----------------------|
| 1 CR              | 83.58   | 82.92       | 77.11       | 83.42      | 78.27    | 88.86        | 92.96       | 98.25                |
| 2 AM              | 89.79   | 90.18       | 83.94       | 84.85      | 85.76    | 90.76        | 92.51       | 95.50                |
| 3 RS              | 86.86   | 88.56       | 86.02       | 84.53      | 82.12    | 89.37        | 95.21       | 96.50                |
| 4 CL              | 89.46   | 90.57       | 89.40       | 87.58      | 87.23    | 91.16        | 94.74       | 95.63                |
| 5 CR + AM         | 91.48   | 92.59       | 86.93       | 87.52      | 90.57    | 89.81        | 92.74       | 96.88                |
| 6 CR + RS         | 87.12   | 89.47       | 86.80       | 85.89      | 88.91    | 89.16        | 94.08       | 97.38                |
| 7 CR + CL         | 89.33   | 90.25       | 89.01       | 88.43      | 89.34    | 90.01        | 93.85       | 96.94                |
| 8 AM + RS         | 88.29   | 89.47       | 87.78       | 84.98      | 86.78    | 90.07        | 93.86       | 96                    |
| 9 AM + CL         | 89.33   | 90.83       | 89.79       | 88.69      | 84.43    | 90.07        | 92.06       | 95.57                |
| 10 RS + CL        | 88.49   | 90.96       | 89.34       | 87.58      | 84.37    | 85.77        | 94.98       | 96.07                |
| 11 CR + AM + RS   | 89.46   | 90.77       | 88.75       | 85.50      | 87.67    | 89.66        | 93.56       | 96.75                |
| 12 CR + AM + CL   | 89.33   | 90.51       | 88.49       | 88.82      | 86.81    | 90.26        | 93.40       | 96.46                |
| 13 CR + RS + CL   | 89.53   | 90.90       | 89.66       | 88.17      | 84.56    | 89.80        | 94.30       | 96.49                |
| 14 AM + RS + CL   | 88.55   | 90.70       | 89.86       | 87.97      | 89.29    | 90.43        | 94.15       | 95.88                |
| 15 CR + AM + RS + CL | 89.33 | 90.70       | 89.60       | 87.84      | 84.32    | 90.04        | 93.86       | 96.47                |

Figure 7: Comparison of combination classification accuracy for each model.

Table 8: A comparison of execution time and images processed per second by the proposed model.

| S/no. | Combination | Execution time (s) | Images processed/second |
|-------|-------------|---------------------|-------------------------|
| 1     | CR          | 231.2               | 6.57                    |
| 2     | AM          | 663.8               | 5.1                     |
| 3     | RS          | 787.4               | 4.25                    |
| 4     | CL          | 1102.1              | 4.21                    |
| 5     | CR + AM     | 790.2               | 4.23                    |
| 6     | CR + RS     | 1004.4              | 4.64                    |
| 7     | CR + CL     | 1109.7              | 4.2                     |
| 8     | AM + RS     | 850.5               | 4.34                    |
| 9     | AM + CL     | 1120.4              | 4.12                    |
| 10    | RS + CL     | 1093.3              | 4.26                    |
| 11    | CR + AM + RS| 624.7               | 5.04                    |
| 12    | CR + AM + CL| 1139.4             | 4.04                    |
| 13    | CR + RS + CL| 1235.5             | 3.78                    |
| 14    | AM + RS + CL| 1203.9             | 3.83                    |
| 15    | CR + AM + RS + CL | 1513.7 | 3.08                    |
Table 9: Confusion matrix for the top 20 malware families in the proposed model.

|        | FakeInstaller | DroidKungFu | Plankton | OpFake | GinMaster | BaseBridge | Iconosys | Kmin | FakeDoc | Geinimi | Adrd | DroidDream | ExploitLinux | Mobile Tx | Gladream | FakeRun | SendPay | Gappusin | Imlog | SMSreg |
|--------|---------------|-------------|----------|--------|-----------|------------|----------|------|---------|---------|------|------------|-------------|-----------|----------|--------|--------|---------|-------|--------|
| FakeInstaller | 904 | 0 | 0 | 17 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2 |
| DroidKungFu | 0 | 579 | 3 | 2 | 13 | 10 | 0 | 0 | 0 | 0 | 34 | 0 | 6 | 0 | 3 | 0 | 0 | 17 | 0 | 0 |
| Plankton | 1 | 3 | 373 | 0 | 4 | 20 | 0 | 0 | 0 | 11 | 11 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| OpFake | 0 | 0 | 560 | 51 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2 | 0 | 0 | 0 | 0 | 0 |
| GinMaster | 0 | 1 | 2 | 3 | 315 | 0 | 0 | 0 | 0 | 1 | 12 | 0 | 0 | 0 | 4 | 0 | 0 | 1 | 0 | 0 |
| BaseBridge | 0 | 5 | 0 | 1 | 1 | 396 | 0 | 0 | 0 | 0 | 3 | 0 | 0 | 0 | 4 | 0 | 0 | 0 | 0 | 0 |
| Iconosys | 0 | 0 | 0 | 0 | 0 | 0 | 132 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Kmin | 0 | 0 | 0 | 0 | 0 | 0 | 135 | 0 | 0 | 12 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| FakeDoc | 3 | 0 | 0 | 0 | 2 | 0 | 0 | 0 | 120 | 0 | 2 | 0 | 0 | 0 | 4 | 0 | 0 | 1 | 0 | 0 |
| Geinimi | 0 | 0 | 0 | 0 | 3 | 2 | 0 | 0 | 0 | 16 | 86 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Adrd | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 88 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| DroidDream | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 78 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| ExploitLinux | 0 | 1 | 0 | 0 | 3 | 4 | 0 | 0 | 0 | 1 | 0 | 60 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| Mobile Tx | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 69 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Gladream | 0 | 2 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 2 | 0 | 0 | 0 | 60 | 0 | 0 | 0 | 0 | 0 |
| FakeRun | 0 | 0 | 21 | 10 | 0 | 0 | 0 | 1 | 3 | 0 | 1 | 0 | 4 | 27 | 0 | 3 | 0 | 0 | 0 |
| SendPay | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 69 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Gappusin | 0 | 3 | 0 | 0 | 1 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 51 | 0 | 0 | 0 |
| Imlog | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2 | 41 | 0 |
| SMSreg | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 39 | 0 |
outperformed five detectors in sustainability by employing a new behavioral profile for apps. In particular, the authors proposed DroidSpan, which surpassed the five detectors in terms of sustainability. However, their study was limited to malware detection and did not include any discussion of malware family classification. DroidEvolver [82] used a model pool with five linear online learning algorithms and delayed classifiers to perform the necessary updates. API-Graph [21] used API semantic similarity from an Android API relation graph to improve the latest malware classifiers.

Therefore, we propose a modified ResNext-based classification network with new regularization for Android malware family classification. Each Android application is distributed via an Android application package (APK). An APK contains multiple folders and files, each of which contains multiple sections; an APK contains multiple folders and files; and an APK contains multiple sections. Among other files and sections, we pay close attention to the AndroidManifest.xml file (AM), classes.dex file (CL), and the certificate files included with each malware sample (CR).

### Table 10: F1-score comparisons between the modified ResNeXt and the original ResNeXt in the Drebin dataset.

| Family              | ResNeXt  | Modified ResNeXt |
|---------------------|----------|------------------|
| Adrd                | 0.67433  | 0.908108         |
| BaseBridge          | 0.921283 | 0.961832         |
| DroidDream          | 0.962963 | 0.981132         |
| DroidKungFu         | 0.916865 | 0.935737         |
| ExploitLinuxLotoor  | 0.875912 | 0.827068         |
| FakeDoc             | 0.948617 | 0.988593         |
| FakeInstaller       | 0.613636 | 0.99675          |
| FakeRun             | 0.761194 | 0.97561          |
| Gappusin            | 0.858311 | 0.866667         |
| Geinimi             | 0.794702 | 0.891192         |
| GinMaster           | 0.993464 | 0.939691         |
| Glodream            | 0.97619  | 0.78481          |
| Iconosys            | 0.957447 | 0.996721         |
| Imlog               | 1        | 0.988506         |
| Kmin                | 0.934891 |                  |
| Mobile Tx           | 0.93551  |                  |
| OpFake              | 0.991597 | 0.995938         |
| Plankton            | 0.95122  | 0.994378         |
| SMsreg              | 0.986361 | 0.886076         |
| SendPay             | 0.895833 | 0.944            |

**Figure 8:** Comparison of family classification F1-score for each model in the Drebin dataset.
As a result of these sample characteristics, we chose the ResNeXt block because of its simplicity and performance. Following that, we modify ResNeXt to include a distinct block for each component (section or file) of a malware sample in order to account for the differences in characteristics between the components. A new regularization technique is utilized to improve the efficiency of malware family classification by extracting discriminatory features from the malware sample. This enables us to ultimately classify malware samples according to their correct families.

The modified ResNeXt model is divided into two distinct phases: training and testing. During the training phase, it builds a prediction model using a set of labeled samples from the Drebin dataset. Then, the trained model is used to classify samples from the Drebin and AMD datasets during the testing phase. Specifically, the primary motivation and goal for the modified ResNeXt is to demonstrate that this model can improve classification performance once trained on an older dataset and predict new patterns of malware samples from a new dataset without having to retrain on new samples.

We addressed the issue of sustainability by assessing the model’s performance when it is trained on the Drebin dataset (collected between 2010 and 2012) and predicting labels for other datasets such as the AMD dataset [10] (collected between 2010 and 2016). We selected the AMD dataset because it was amassed over a longer time period than the Drebin dataset. Our experiment revealed that the feature extraction model trained on the AMD dataset outperformed the model trained on the Drebin dataset in terms of overall performance. This is because the AMD dataset contains more variation information about malware samples than the Drebin dataset, which makes it more suitable to study the evolutionary patterns of malware.

In particular, we divided the samples into three different groups, each with their own set of samples. The first group is the Drebin dataset, where we separated the malware samples from the same year into training, validation, and test sets. The goal is to assess the performance of our model with training and testing malware samples collected in the same period of time. The second group is the AMD dataset. We selected 3,460 malware samples randomly (S-AMD). By doing so, we intend to test the capability of our model in classifying apps that have never been used in training. For the third group (T-AMD), we used the entire AMD dataset, in which we utilized malware samples from different time periods than the training set. The goal is to focus on evaluating the stability of our model performance when it is trained on older datasets and predicting the labels of newer ones, spanning one to four years. Figure 9 shows the experimental results.

In the first experiment, we compared the feature extraction model learned from the Drebin and AMD datasets (same period) to observe the performance of our model. It is clear from Table 11 that the model’s generalization is better for Drebin than (S-AMD) and (T-AMD), which showed the suitability of the (S-AMD) and (T-AMD) sets in terms of containing more discriminatory features of malware samples. Next, we compared the accuracy of training and testing on the Drebin and the AMD datasets (same period and over-time). We observed that the accuracy of the Drebin dataset is better than that of the AMD dataset. This is because the Drebin dataset contains fewer malware families than the AMD dataset. In other words, the AMD dataset has sufficient variation.
information about malware samples, which improves the generalization performance. Thus, the generalization performance improved when the variation information of malware samples in the AMD dataset was learned.

Based on the results from Table 11, it was revealed that, even with a span of four years (the difference between Drebin and AMD datasets collection span), our model detection accuracy dropped noticeably over time from about 98% to below 80% in terms of accuracy for testing samples from year one. Over time, our model tended to be much more stable (with minimal fluctuation) in terms of detecting malware samples. It achieved an average accuracy of 87%, despite the evolution of Android malware. Despite the fact that our results are promising, we cannot claim that our model will continue to perform as well as it has in the case of future malware. The unpredictable evolution of our app in the future would also serve as a trigger for retraining our model in the future.

However, previous studies used a variety of different benchmarks cast doubt on the measurement degree of our findings' validity. As a result, despite the fact that we collected samples from a variety of sources, our datasets may not be representative of the app population during the pertinent years. Our findings and conclusions are best understood in light of the benchmarks we analyzed.

6. Discussion and Limitation

There are numerous benefits to utilizing DL networks for malware family classification. The DL techniques have the capability to classify data automatically without the essential requirements of some expensive processes, such as decryption or reverse engineering. However, to successfully build a lightweight classification model and to avoid the high computational cost, the size of training features must be reduced to speed up the training and detection process.

The distribution of the discriminant information is controlled via the new regularizer by constraining the weight values' dispersion. In other words, the standard deviation of the weight matrix was used to obtain the regularization term and then multiplied by $\lambda$. The motivation is to develop a weight-decay adaptive form that helps the regularizer prevent the learning model from extracting values from the weight space that are widely distributed. Thus, it helps the model extract features that are effective for malware family classification. Thus, the modified ResNeXt has the best classification performance among all the methods because we used a new regularization discriminant information distribution to eliminate unnecessary data and selectively use only data useful for classification.

The solutions that were representing the malicious app behaviors using dynamic features (e.g., DroidSieve) suffered from the cost of tracing runtime and scalability, while our approach incurred runtime costs for testing per second.

Additionally, the overall accomplished results for the visual grayscale images may differ from those for the visual color images. This is because color images contain more texture details and visualization features than those included in grayscale images. Also, the tested model used imbalanced Android malware samples for training and validating, so the models need to be tested on balanced Android malware samples. Thus, our proposed model avoided the computational needs of data augmentation and feature-engineering techniques. As a result, we successfully achieved better results and satisfied significant classification performance compared to other existing methods.

7. Conclusions

In this work, we proposed a modified ResNeXt model for the classification of Android malware. The ResNeXt is utilized due to its flexibility and requirement for low flops, coupled with a new regularization technique to improve the capability of the model in the classification of Android malware. In the first step, various binary malware files from the Drebin dataset were transformed into 8-bit vectors based on the substrings. In the next step, these vectors are converted to grayscale images. For classification, we adopt a modified ResNeXt with skip concept and cardinality to enhance the detection performance. Furthermore, we embedded a new regularization technique to improve the classification detection rate. Different combinations of the images were used to fine-tune the model to look for those files having the most effect on the model. From simulation results, it is concluded that the certificate (CR) is the most suited feature, containing enough information to be used for the identification and classification of malware. We reported various highest measures, including accuracy, recall, precision, and F1 measures, obtained from using the CR images. 97.07% accuracy is observed, which is the highest accuracy so far achieved by using the Drebin dataset.

For intrinsic evaluation of our DL approach, the proposed model is compared with other state-of-the-art techniques, which widely use DL for classification purposes within the Android malware family. In the future, various models can be trained and studied to adopt a less complex model, while enhancing the performance further for the malware classification problem. In addition, when malware authors distribute each malware instance in both its simple and obfuscated forms concurrently, the images between the simple and obfuscated versions are likely to differ. So, utilizing only malware images may be ineffective in this case. A possible solution to this issue is to include another feature that is resistant to obfuscation, packing, and encryption attacks and then combine it with the malware image approach, for example, the extraction of code features from native app binaries and security-sensitive APIs, including reflection-based features.

Data Availability

The datasets used to support the findings of this study are included within the article.

Conflicts of Interest

The authors declare that they have no conflicts of interest.
References

[1] M. Ashawa and S. Morris, “Analysis of android malware detection techniques: a systematic review,” International Journal of Cyber-Security and Digital Forensics, vol. 8, no. 3, pp. 177–187, 2019.

[2] R. Komatwar and M. Kokare, “A survey on malware detection and classification,” Journal of Applied Security Research, vol. 16, no. 3, pp. 390–420, 2020.

[3] G. Suarez-Tangli, S. K. Dash, M. Ahmadi, J. Kinder, G. Giacinto, and L. Cavallaro, “CODASPY: Data and application security and privacy,” in Proceedings of the Seventh ACM Conference on Data and Application Security and Privacy, Scottsdale, Arizona, USA, March 2017.

[4] J. Garcia, M. Hammad, and S. Malek, “Lightweight, obfuscation-resilient detection and family identification of android malware,” ACM Transactions on Software Engineering and Methodology, vol. 26, no. 3, pp. 1–29, 2018.

[5] V. Rastogi, Y. Chen, and X. Jiang, “Catch me if you can: evaluating android anti-malware against transformation attacks,” IEEE Transactions on Information Forensics and Security, vol. 9, no. 1, pp. 99–108, 2014.

[6] O. Mirzaei, J. M. de Fuentes, J. Tapiador, and L. Gonzalez-Manzano, “AndroDet: an adaptive Android obfuscation detector,” Future Generation Computer Systems, vol. 90, pp. 240–261, 2019.

[7] V. Balachandran, D. J. T. Tan, and V. L. L. Thng, “Control flow obfuscation for Android applications,” Computers & Security, vol. 61, pp. 72–93, 2016.

[8] R. Vinayakumar, M. Alazab, K. P. Soman, P. Poornachandran, and S. Venkatraman, “Robust intelligent malware detection using deep learning,” IEEE Access, vol. 7, Article ID 46717, 2019.

[9] J. Fu, J. Yue, Y. Wang, Z. Liu, and C. Shan, “Malware visualization for fine-grained classification,” IEEE Access, vol. 6, Article ID 14510, 2018.

[10] F. Wei, Y. Li, S. Roy, X. Ou, and W. Zhou, “Deep ground truth analysis of current android malware,” in Proceedings of the Detection of Intrusions and Malware, and Vulnerability Assessment, pp. 252–276, Bonn, Germany, July 2017.

[11] S. Dong, M. Li, W. Diao et al., “Understanding android obfuscation techniques: a large-scale investigation in the wild,” in Proceedings of the Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, pp. 172–192, Singapore, August 2018.

[12] D. Maiorca, D. Ariu, I. Corona, M. Aresu, and G. Giacinto, “Stealth attacks: an extended insight into the obfuscation effects on Android malware,” Computers & Security, vol. 51, pp. 16–31, 2015.

[13] K. Bakour, H. M. Unver, and R. Ghanem, “A deep camouflage: evaluating android’s anti-malware systems robustness against hybridization of obfuscation techniques with injection attacks,” Arabian Journal for Science and Engineering, vol. 44, no. 11, pp. 9333–9347, 2019.

[14] N. Xie, X. Wang, W. Wang, and J. Liu, “Fingerprinting Android malware families,” Frontiers of Computer Science, vol. 13, no. 3, pp. 637–646, 2018.

[15] S. Ni, Q. Qian, and R. Zhang, “Malware identification using visualization images and deep learning,” Computers & Security, vol. 77, pp. 871–885, 2018.

[16] S. Turk and A. B. Can, “Andmdsf: android malware family classification framework,” in Proceedings of the IEEE 30th International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC Workshops), Istanbul, Turkey, September 2019.

[17] D. Vasan, M. Alazab, S. Wassan, H. Naeem, B. Safaei, and Q. Zheng, “Imcfm: image-based malware classification using fine-tuned convolutional neural network architecture,” Computer Networks, vol. 171, 2020.

[18] McAfee, “McAfee mobile threat report q1,” 2020, https://www.mcafee.com/content/dam/consumer/en-us/docs/2020-Mobile-Threat-Report.pdf.

[19] F. Alsawina and K. Elleithy, “Android malware family classification and analysis: current status and future directions,” Electronics, vol. 9, 2020.

[20] L. Massarelli, L. Aniello, C. Ciccotelli, L. Querzoni, D. Ucci, and R. Baldoni, “Android malware family classification based on resource consumption over time,” in Proceedings of the 12th International Conference on Malicious and Unwanted Software (MALWARE), Fajardo, PR, USA, October 2017.

[21] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, and K. Rieck, “Drebin: effective and explainable detection of android malware in your pocket,” in Proceedings of the Network and Distributed System Security Symposium (NDSS), 2014.

[22] Y. Sun, Y. Chen, Y. Pan, and L. Wu, Android Malware Family Classification Based on Deep Learning of Code Images, 2019.

[23] K. Kancherla and S. Mukkamala, “Image visualization based malware detection,” in Proceedings of the IEEE Symposium on Computational Intelligence in Cyber Security (CICS), pp. 40–44, Singapore, April 2013.

[24] L. Nataraj, S. Karthikeyan, G. Jacob, and B. S. Manjunath, “Malware images: visualization and automatic classification,” in Proceedings of the 8th International Symposium on Visualization for Cyber Security - VizSec ’11, Pennsylvania, PA, USA, July 2011.

[25] Y. Fang, Y. Gao, F. Jing, and L. Zhang, “Android malware familial classification based on dex file section features,” IEEE Access, vol. 8, 2020.

[26] J. Singh, D. Thakur, F. Ali, T. Gera, and K. S. Kwak, “Deep feature extraction and classification of android malware images,” Sensors, vol. 20, 2020.

[27] J. H. Go, T. Jan, M. Mohanty, O. P. Patel, D. Puthal, and M. Prasad, “Visualization approach for malware classification with retnex,” in Proceedings of the IEEE Congress on Evolutionary Computation (CEC), Glasgow, UK, July 2020.

[28] T. Vidas and N. Christin, “Evading android runtime analysis via sandbox detection,” in Proceedings of the 9th ACM Symposium on Information, Computer and Communications Security, Kyoto, Japan, June 2014.

[29] H. Gascon, F. Yamaguchi, D. Arp, and K. Rieck, “Structural detection of android malware using embedded call graphs,” in Proceedings of the ACM SIGSAC Conference on Computer and Communications Security, Berlin, Germany, November 2013.

[30] C. Yang, Z. Xu, G. Gu, V. Jegneswaran, and P. Porras, “Droidminer: automated mining and characterization of fine-grained malicious behaviors in android applications,” in Proceedings of the 19th European Symposium on Research in Computer Security, Wroclaw, Poland, September 2014.

[31] P. Farukli, A. Bharma, V. Laxmi et al., “Android security: a survey of issues, malware penetration, and defenses,” IEEE Communications Surveys & Tutorials, vol. 17, 2015.

[32] K. Bakour, H. M. Unver, and R. Ghanem, “The android malware detection systems between hope and reality,” SN Applied Sciences, vol. 1, 2019.

[33] M. Farrokhamanesh and A. Hamzeh, “A novel method for malware detection using audio signal processing techniques,”
in Proceedings of the Artificial Intelligence and Robotics (IRANOPEN), Qazvin, Iran, April 2016.

[34] G. Suarez-Tangil, J. E. Tapiador, P. Peris-Lopez, and J. Blasco, “Dendroid: a text mining approach to analyzing and classifying code structures in android malware families,” Expert Systems with Applications, vol. 41, 2014.

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

[36] M. K. Alzaylaee, S. Y. Yerima, and S. Sezer, Emulator vs Real Phone, 2017.

[37] D. Su, J. Liu, X. Wang, and W. Wang, “Detecting android locker-ransomware on Chinese social networks,” IEEE Access, vol. 7, 2019.

[38] F. Idrees, M. Rajarajan, M. Conti, T. M. Chen, and M. K. Alzaylaee, “Pindroid: a novel android malware detection system using ensemble learning methods,” Computers & Security, vol. 68, 2017.

[39] B. Jung, T. Kim, and E. G. Im, “Malware classification using byte sequence information,” in Proceedings of the RACS ’18: International Conference on Research in Adaptive and Convergent Systems, Honolulu Hawaii, October 2018.

[40] S. Wu, P. Wang, X. Li, and Y. Zhang, “Effective detection of android malware based on the usage of data flow apis and machine learning,” Information and Software Technology, vol. 75, 2016.

[41] S. Talukder and Z. Talukder, “A survey on malware detection and analysis tools,” International Journal of Network Security & Its Applications, vol. 12, 2020.

[42] M. A. Omer, S. R. M. Zeebaree, M. A. M. Sadeeq et al., “Efficiency of malware detection in android system: a survey,” Asian Journal of Computer Science & Information Technology, vol. 7, 2021.

[43] R. S. Arslan, “Androanalyzer: android malicious software detection based on deep learning,” PeerJ Computer Science, vol. 7, 2021.

[44] D. Vasan, M. Alazab, S. Wassan, B. Safaei, and Q. Zheng, “Image-based malware classification using ensemble of cnn architectures (imec),” Computer and Security, vol. 92, 2020.

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

[46] X. Zhang, Y. Zhang, M. Zhong et al., Enhancing State-Of-The-Art Classifiers with Api Semantics to Detect Evolved Android Malware, in Proceedings of the ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, USA, November 2020.

[47] H. Cai, “Embracing mobile app evolution via continuous ecosystem mining and characterization,” in Proceedings of the EEE/ACM 7th International Conference on Mobile Software Engineering and Systems, Seoul Republic of Korea, July 2020.

[48] R. Yamashita, M. Nishio, R. K. G. Do, and K. Togashi, “Convolutional neural networks: an overview and application in radiology,” Insights into imaging, vol. 9, no. 4, pp. 611-629, 2018.

[49] M. D. Zeiler and R. Fergus, “Visualizing and understanding convolutional networks,” in Proceedings of the European conference on computer vision, pp. 818–833, Springer, Zurich, Switzerland, September 2014.

[50] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He, “Aggregated residual transformations for deep neural networks,” in Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 1492–1500, July 2017.

[51] C. Szegedy, W. Liu, Y. Jia et al., “Going deeper with convolutions,” in Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 1–9, Boston, MA, USA, June 2015.

[52] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” 2014, https://arxiv.org/abs/1409.1556.

[53] F. N. Landolsi, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, “SqueezeNet: alexnet-level accuracy with 50x fewer parameters and< 0.5 mb model size,” 2016, https://arxiv.org/abs/1602.07360.

[54] F. LeCun, “Lenet-5, convolutional neural networks,” vol. 20, no. 5, 2015, http://yann.lecun.com/exdb/lenet.

[55] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 770–777, Las Vegas, NV, USA, June 2016.

[56] A. Y. Ng, “Feature selection, l1 vs. l2 regularization, and rotational invariance,” in Proceedings of the 21st international conference on Machine learning - ICML ’04, July 2004.

[57] M. S. ElSayed, N.-A. Le-Khac, M. A. Albahar, and A. Jurcute, “A novel hybrid model for intrusion detection systems in sdn based on cnn and a new regularization technique,” Journal of Network and Computer Applications, vol. 191, Article ID 103160, 2021.

[58] F. M. Darus, N. A. A. Salleh, and A. F. M. Ariffin, “Android malware detection using machine learning on image patterns,” in Proceedings of the Cyber Resilience Conference (CRC), pp. 1-2, IEEE, Putrajaya, Malaysia, November 2018.

[59] J. Li, L. Sun, Q. Yan, Z. Li, W. Srisa-An, and H. Ye, “Significant permission identification for machine-learning-based android malware detection,” IEEE Transactions on Industrial Informatics, vol. 14, no. 7, pp. 3216–3225, 2018.

[60] X. Wang and C. Li, “Android malware detection through machine learning on kernel task structures,” Neurocomputing, vol. 435, pp. 126–150, 2021.

[61] T. Chen, Q. Mao, Y. Yang, M. Lv, and J. Zhu, “Tinydroid: a lightweight and efficient model for android malware detection and classification,” Mobile Information Systems, vol. 2018, Article ID 4157156, 9 pages, 2018.

[62] C. Yin, Y. Zhu, J. Fei, and X. He, “A deep learning approach for intrusion detection using recurrent neural networks,” IEEE Access, vol. 5, Article ID 21954, 2017.

[63] A. A. Barakabitze, A. Ahmad, R. Mijumbi, and A. Hines, “5g network slicing using sdn and nfv: a survey of taxonomy, architectures and future challenges,” Computer Networks, vol. 167, Article ID 106984, 2020.

[64] T. Huang and H.-Y. Kao, “R2-d2: color-inspired convolutional neural network (cnn)-based android malware detections,” in Proceedings of the IEEE International Conference on Big Data (Big Data), pp. 2633–2642, IEEE, 2018.

[65] W. Hardy, L. Chen, S. Hou, Y. Ye, and X. Li, “D44md: a deep learning framework for intelligent malware detection,” in Proceedings of the International Conference on Data Science (ICDATA), The Steering Committee of The World Congress in Computer Science, 2016.

[66] 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.

[67] 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.
[68] R. Nix and J. Zhang, “Classification of android apps and malware using deep neural networks,” in Proceedings of the International joint conference on neural networks (IJCNN), pp. 1871–1878, IEEE, Anchorage, AK, USA, May 2017.

[69] 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.

[70] Y. Zhou and X. Jiang, “Dissecting android malware: characterization and evolution,” in Proceedings of the IEEE Symposium on Security and Privacy, May 2012.

[71] M. Ficco, “Malware analysis by combining multiple detectors and observation windows,” IEEE Transactions on Computers, vol. 71, p. 1, 2021.

[72] A. Naway and Y. Li, “A review on the use of deep learning in android malware detection,” 2018, https://arxiv.org/abs/1812.10360.

[73] W. Wang, M. Zhao, Z. Gao et al., “Constructing features for detecting android malicious applications: issues, taxonomy and directions,” IEEE Access, vol. 7, 2019.

[74] O. Aslan and R. Samet, “A comprehensive review on malware detection approaches,” IEEE Access, vol. 8, 2020.

[75] S. Hitawala, “Evaluating resnext model architecture for image classification,” 2018, http://arxiv.org/abs/1805.08700.

[76] M. A. Albahar, “Recurrent neural network model based on a new regularization technique for real-time intrusion detection in SDN environments,” Security and Communication Networks, vol. 2019, Article ID 8939041, 9 pages, 2019.

[77] M. A. Albahar, “Skin lesion classification using convolutional neural network with novel regularizer,” IEEE Access, vol. 7, Article ID 38306, 2019.

[78] M. A. Albahar, M. Binsawad, and L. Maglaras, “Deep autoencoders and feedforward networks based on a new regularization for anomaly detection,” Security and Communication Networks, vol. 2020, Article ID 7086367, 9 pages, 2020.

[79] M. A. Albahar, A. A. Albahr, and M. H. Binsawad, “An efficient person re-identification model based on new regularization technique,” IEEE Access, vol. 8, Article ID 171049, 2020.

[80] A. Albahr, M. Albahar, M. Thanoon, M. Binsawad, and M. Versaci, “Computational learning model for prediction of heart disease using machine learning based on a new regularizer,” Computational Intelligence and Neuroscience, vol. 2021, Article ID 8628335, 10 pages, 2021.

[81] H. Cai, “Assessing and improving malware detection sustainability through app evolution studies,” ACM Transactions on Software Engineering and Methodology, vol. 29, 2020.

[82] K. Xu, Y. Li, R. Deng, K. Chen, and J. Xu, “Droidevolver: self-evolving android malware detection system,” in Proceedings of the IEEE European Symposium on Security and Privacy (EuroS&P), Stockholm, Sweden, Jun 2019.