Received December 14, 2021, accepted December 25, 2021, date of publication December 30, 2021,
date of current version January 14, 2022.

Digital Object Identifier 10.1109/ACCESS.2021.3139334

Efficient Deep Learning Network With Multi-Streams for Android Malware Family Classification

HYUN-IL KIM[1], MOONYOUNG KANG[2], SEONG-JE CHO[1,2], (Member, IEEE), AND SANG-IL CHOI[1,3], (Senior Member, IEEE)
[1]Department of Computer Science and Engineering, Dankook University, Jukjeon-ro, Sugi-gu, Yongin-si, Gyeonggi-do 16890, Republic of Korea
[2]Department of Software Science, Dankook University, Jukjeon-ro, Sugi-gu, Yongin-si, Gyeonggi-do 16890, Republic of Korea
[3]Department of Computer Engineering, Dankook University, Jukjeon-ro, Sugi-gu, Yongin-si, Gyeonggi-do 16890, Republic of Korea

Corresponding authors: Seong-Je Cho (sjcho@dankook.ac.kr) and Sang-Il Choi (choisi@dankook.ac.kr)

This work was supported in part by the National Research Foundation of Korea (NRF) Grant by the Korean Government through the MSIT under Grant 2021R1A2B5B01001412; and in part by the Ministry of Science and Information and Communication Technology (ICT) (MSIT), South Korea, under the ICT Challenge and Advanced Network of Human Resource Development (HRD) (ICAN) Program, supervised by the Institute of Information and Communications Technology Planning and Evaluation (IITP), under Grant IITP-2021-2020-0-01824; and in part by the Basic Science Research Program through the National Research Foundation of Korea (NRF) by the Ministry of Science and ICT under Grant 2018R1A2B2004830.

ABSTRACT It is important to effectively detect, mitigate, and defend against Android malware attacks, because Android malware has long represented a major threat to Android app security. Characterizing and classifying similar malicious apps into groups plays a particularly crucial role in building a secure Android app ecosystem. The classification of malware families can efficiently enhance the malware detection process and systematically elucidate malware patterns. In this paper, we propose a novel efficient deep learning network with multi-streams for Android malware family classification. We first obtain the input data for a convolutional neural network (CNN) in string format from some main files or sections contained in each Android malicious app. We then classify malware families by applying a 1-dimensional convolution filter-based network for the files or sections. Further, by using gradient analysis to visualize the important files and sections in malicious apps, we attempt to intuitively grasp which files or sections are the most significant for malware family classification. To validate the effectiveness of our approach, we conduct extensive experiments with the well-known DREBIN and AMD malware datasets, and we compare our approach with existing methods. Our experimental results show that the 1D CNN model is more accurate than the 2D CNN model, and that the `code_item` part in the classes.dex is the most relevant feature for malware classification, as it is more relevant than other parts such as AndroidManifest.xml and certificate. The proposed method achieves the best accuracy of 93.2% by using 1D convolution filters with multi-streams for the main files and sections of the malware samples.

INDEX TERMS Android malware family, 1D convolution filter, multi-streams, explainable analysis, class separability, gradient analysis.

I. INTRODUCTION

Android malware (malicious apps) can hack users' smartphones without their knowledge, then steal sensitive personal information stored in the smartphone, lock the user out from important user data, or mine cryptocurrency. At present, the number of Android malware samples (malicious Android apps) and their variants are continue to spread widely. According to the McAfee mobile threat report issued in Q1 2020 [1], more than 35 million malware attacks in total and nearly 800 thousand new malware attacks were detected during the fourth quarter of 2019. One of these new attacks, `MalBus` spyware, hides a targeted attack in a legitimate app by hacking the original developer's Google play account. It phishes for the victim's Google user ID and password with a fake login page, scans the user's device for sensitive

The associate editor coordinating the review of this manuscript and approving it for publication was Mostafa Rahimi Azghadi.

military and political keywords, and can even run commands. In addition, a new Android malware family called `LeifAccess` has emerged that abuses accessibility features to create accounts, download apps, and post fake reviews. After `LeifAccess`, also known as `Shopper`, is installed, it advertises click fraud without displaying an icon or a shortcut.

Due to the increasing number of mobile malware instances and the emergence of new malware families, malware has become a serious threat to Android ecosystem security. To defeat this threat and protect mobile users and systems, many studies have investigated the detection and classification of Android malware samples [2]–[17]. Malware detection is a binary classification problem that involves attempting to determine whether a suspicious app is malicious or benign. On the other hand, malware family classification is a multi-class classification problem, which involves attempting to categorize detected malicious apps into one of the known families or one of the new families based on the characteristics of the samples. If malware samples are classified into the correct family and accurately characterized, then malware researchers can focus on highly dangerous families rather than individual malware samples or less risky families [2]. Therefore, an effective malware family classification can help malware analysts identify more malware samples by recognizing other malware samples in that family and grasping their characteristics. Compared to malware detection, malware family classification is more challenging because it is a multi-class classification problem, and because the number of malware samples varies across families [2]–[4], [7].

A malware family is a group of malware samples that shares the same or similar functionality, behavior, and purpose. Because many Android malware variants belong to the same malware family, identifying and categorizing malware variants into their relevant family is very crucial for understanding their typical behavior patterns and conducting further analysis [2], [3], [5]. Thus, Android malware researchers and anti-malware vendors are currently struggling to classify each malicious app into a family of related malware and then provide the appropriate countermeasures [2]–[11].

Many malware classification studies have been conducted on the Microsoft Windows platform [8], [15], [16], [18], [19]. However, the executable file structure of the Windows platform is very different from the executable file structure of the Android platform, methods used to classify Windows malware families cannot be directly applied to Android malware families [5], [6]. Moreover, due to the rapid proliferation of Android malware samples and their variants, Android malware family classification has recently attracted substantial interest from researchers as well as the industrial world [2].

Another recent challenge in machine learning-based malware classifiers is the evolution of malware to change its malicious behavior over time, thus leading to the deterioration of the classifiers [20]–[22]. Several researchers have attempted to address this sustainability problem by defining it as deterioration [21], [23], or model aging [22], [24]. In [21], [23], sustainability metrics were proposed and compared with the latest five Android malware detectors. The classifier, DroidSpan [23], used a new behavioral profile for apps and outperformed the five detectors in sustainability. DroidEvolver [24] performed a necessary update using a model pool that contained five linear online learning algorithms and delayed classifiers. APIGraph [22] enhanced the latest malware classifiers using API semantic similarity from a relation graph of Android APIs among evolved malware samples.

Therefore, in this paper, we propose a novel 1D convolution filter-based classification network with multi-streams suitable for classifying Android malware families. Every Android app is distributed in the form of an Android application package (APK); an APK has several folders and files, where a file consists of several sections (see Figure 1). Among the files and sections of each malware sample, we focus on the `classes.dex` file (CL) and its sections, `AndroidManifest.xml` file (AM), and certificate files (CR). Considering such sample properties, we used 1D convolution filters to extract features for malware family classification. We also construct multi-streams that contain different networks for each part (file or section) in a malware sample to account for the individual characteristics of different parts. Further, we improve malware family classification performance with efficient operations by extracting composite features from only the selected streams based on an analysis of the amount of discriminative information in each part of the malware sample. We finally classify malware samples into their proper families using the composite features.

We conduct extensive experiments for the DREBIN and AMD datasets [4], [25], which are well-known in malware family classification. We compare our approach with the existing methods, which are six 2D convolution filter-based models (SARVOTAM, Ef- ficientNetB0, Vgg16, ResNet50, Inception-V3, LeNet), and two types of proposed models (Basic-1D-CNN with single stream and multi-streams). The experimental results show that the proposed 1D convolution filter-based network with multi-streams achieves the highest accuracy and F1-score. To handle the sustainability challenge, we have also conducted intra-, and inter-dataset experiments on the AMD dataset [25] as well as the DREBIN. The AMD dataset was another well-known Android Malware Dataset, which was collected over longer years than the DREBIN dataset. The AMD dataset is more suitable for longitudinal and evolutionary studies than the DREBIN dataset in the inter-dataset experiment. The generalization performance is improved when the variation information of malware samples of the AMD dataset is learned.

The main contributions of this paper are as follows:

- We propose an effective 1D convolution filter-based classification network for automatically classifying Android malware families from raw malware samples without any data preprocessing.
- We investigate which parts (files or sections) in the malware sample are more effective for the classification

of Android malware families through an expandable analysis for 1D convolution filter-based networks.
- We extract composite features by designing a network with multiple streams specialized for each part of the malware sample. In particular, by selectively using only streams corresponding to parts containing a large amount of discriminant information, the separability of the composite feature is increased, and efficient operation is performed. We achieve the highest accuracy and F1-score when applying the 1D CNN model with six streams.
- Our approach does not necessitate extraction of specific features such as application programming interface (API) calls, strings, opcodes, permissions, components, intents, or system calls from the APKs. Further, it is not necessary to decrypt, de-obfuscate or execute the code in the CL file.

The rest of this paper is organized as follows. Section II offers background knowledge and some related work. Section III explains our model. Section IV interprets the experimental results and presents the performance evaluation. In Section V, we discuss limitations and future directions. Finally, we provide a conclusion in Section VI.

II. BACKGROUND AND RELATED WORK

A. STRUCTURE OF APK FILES

An APK file is a zip file that contains all the contents of an Android app, as depicted in Figure 1. An APK comprises of the executable code (**classes.dex**s file), manifest (**AndroidManifest.xml** file), meta information (**META-INF** folder), resources (**resources.arsc** file and **res** folder), assets (**assets** folder), and library(**lib** folder) [26]–[29].

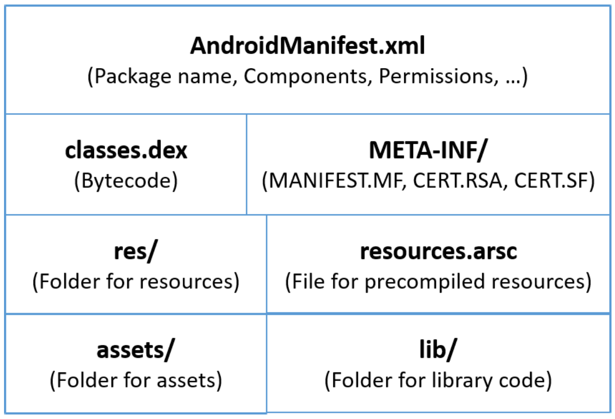

AndroidManifest.xml
(Package name, Components, Permissions, ...)
classes.dex
(Bytecode)
META-INF/
(MANIFEST.MF, CERT.RSA, CERT.SF)
res/
(Folder for resources)
resources.arsc
(File for precompiled resources)
assets/
(Folder for assets)
lib/
(Folder for library code)

FIGURE 1. Structure of APK files.

`AndroidManifest.xml` (AM) contains the package name and version of the app, components, permissions, and referenced library files for the app. `META-INF` is a folder that contains `MANIFEST.MF`, `CERT.RSA`, and `CERT.SF` files. `MANIFEST.MF` is a manifest file that contains a list of all files in the APK along with their SHA-1 digests. `CERT.SF` is a signature file that consists of a list of all files in the APK along with their SHA-1 digests again; however, each digest of the CERT.SF is the digest of the three-line entry of that file. `CERT.RSA` is the real signature file of the APK which contains the certificate of the public key for the app to verify the signature [27]. In this paper, a certificate (CR) represents the combination of the `CERT.RSA` and `CERT.SF` files of each app.

The `resources.arsc` file contains precompiled resources such as strings, colors, or styles in binary XML. The `res` folder contains all resources not compiled into `resources.arsc`. The `assets` folder is an optional folder that contains the app's assets, such as media files. The `lib` folder, which holds the native code libraries, may contain several sub-folders, each of which will have platform dependent-compiled code for specific hardware architectures.

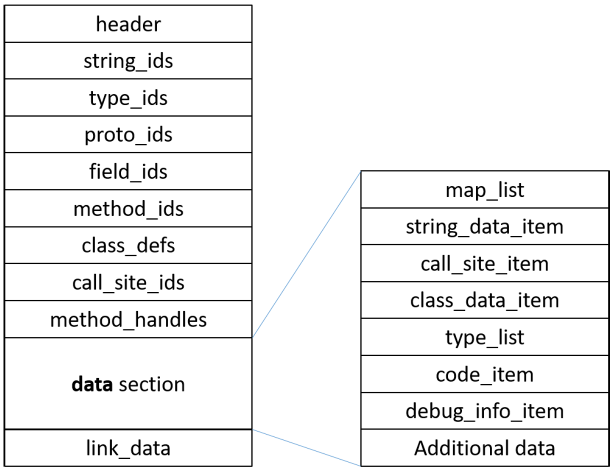


FIGURE 2. Layout of a DEX file inside an APK.

A DEX file, **classes.dex** (CL), contains all of the app bytecode compiled in the dex file format. The DEX file structure is described on the official Dalvik Executable format page [30]. As illustrated in Figure 2, a DEX file may consist of several sections: a *header*; lists for *strings*, *types*, *method prototypes*, *fields*, *methods* identifiers (ids), *class definitions*, *call site identifiers*, and *method handles*; a *data section*; and the *link_data* used in statically linked files [30], [31].

On the right-hand side of Figure 2, we show a more detailed representation of the **data** section. The *map_list* is a list of the entire contents of a file. The *string_ids* section and *string_data_item* contain all the data about strings including the string length, where the 'string' refers to the parts of operations and definitions represented by string labels (e.g., string constants as well as type and class names). The *call_site_item* is an *encoded_array_item* whose elements correspond to the arguments provided to a bootstrap linker method. The *type_list* contains the size and elements of the list. The *code_item* contains all code instructions including the array indicating where in the code exceptions are caught and how to handle them, the bytes representing a list of lists of catch types and associated handler addresses, the number of registers, words, *try_items*, and the size of list of instructions.

B. RELATED WORK

Although malware detection [12], [16], [32] is one of the most important topics for strengthening the security and stability of the mobile app ecosystem, malware detection is beyond the scope of this paper. To date, there have been many studies on malware family classification [2]–[11], [13]–[15], [17]. According to a survey paper [2], some studies have used image classification methods to classify Android malware families. In these methods, the features of Android apps are represented in the form of grayscale or color images, and Android malware samples are then classified based on these image features.

Fang *et al.* [5] classified Android malicious apps into their corresponding families using the DEX file section features. They first transformed the DEX file into a Red/Green/Blue (RGB) image and plain text based on the section features. Then, they extracted three features: color, texture, and text. The texture and text features were extracted by using the GIST algorithm and the Simhash algorithm, respectively. The features were fed into a feature fusion algorithm using multiple kernel learning (MKL) to classify malware families. Their dataset consisted of fifteen malware families from the Android Malware Dataset (AMD) [25]. They used three classifiers: Support Vector Machine (SVM), k-Nearest Neighbor (kNN), and Random Forest (RF). The feature fusion method achieved better classification performance than the cascade of all features directly.

Singh *et al.* [6] proposed a system called **S**umming of neur**A**l a**R**chitecture and **V**isualizati**O**n **T**echnology for **A**ndroid **M**alware identification (`SARVOTAM`). This system first converts malicious Android apps into fingerprint images, then uses a fine-tuned CNN to extract features from the grayscale images, along with a total of fifteen different combinations of the malware image sections for malware classification. The flatten layer in the CNN architecture flattens the output from the previous layer into a one-dimensional tensor. The softmax layer of CNN is replaced by traditional classifiers such as kNN, SVM, and RF to classify the images. In the SARVOTAM system, the CNN-SVM model achieved better performance than the original CNN, CNN-kNN, and CNN-RF. The SARVOTAM system was evaluated with twenty families in the DREBIN dataset. The CNN-SVM model achieved the highest accuracy of 92.59% when using the malware images generated from the combination of the certificates (CR) and manifest (AM), and it achieved 90.57% accuracy when using the malware images generated from only the classes.dex (CL).

Sun *et al.* [7] also used the grayscale malware images converted from the bytecodes of malicious apps' bytecode. The malware code-images were given to the input of a pre-trained CNN. They chose `GoogLeNet Inception-v3` as the deep learning network architecture and used 2048-dimensional feature vectors of the images. They set the optimizer and learning rate of the neural network to `AdamOptimizer` and 0.0005 respectively. They performed some experiments with fourteen families as well as twenty families of varying sizes in the DREBIN dataset. The average F1-scores for the fourteen families and the twenty families were 95.2% and 90.5%, respectively. They mentioned that the malware image-based classification method is resilient to code obfuscation techniques.

Kang *et al.* [11] used two types of grayscale images for Android malware family classification. The first image was converted from the entire DEX file, while the second image was converted from only the data section inside the DEX file. They chose 64 × 64 as the normalized size of images, and they used a CNN model with `LeNet` to classify the two types of images. The top 20 families in the DREBIN dataset were used for the experiments. Their method obtained an accuracy of around 91% for both types, but the size of the data section images was reduced by an average of 18% compared to the size of the DEX images.

Zhao and Qian [33] decompiled each APK; extracted the opcodes, sensitive API packages and risky API functions; and mapped the three different extracted features to the R channel, G channel, and B channel of an RGB image, respectively. They then detected the feature images of malware families by using a CNN modified from LeNet5. They also conducted several experiments with the 14 representative families from the DREBIN dataset. Their classification achieved up to 96.91% accuracy.

Darwaish and Naït-Abdesselam [34] converted several elements inside an APK file into an RGB image, and they fixed the image using the nearest neighbour interpolation. They extracted permissions, intents, activities and services from the manifest, then mapped the extracted information to the green channel. They also extracted opcodes and API calls from the DEX, and mapped them to the red channel. Lastly, they mapped all the malicious components gathered from the manifest and DEX files on the blue channel. They transformed large APK file (typically consisting of multiple MBs) into small RGB image of 3-4KB. Finally, conducted experiments on the top 10 families in the AndroZoo dataset [35], where each family had more than 5K samples, and they classified the malware samples with an accuracy of 99.37%.

Chen *et al.* [36] converted the DEX files into grayscale images, and they extracted image texture feature using the GIST algorithm. They used XGBoost as the classifier, and they conducted experiments on the 10 families containing anywhere from 208 to 1824 samples. They achieved up to 99.14% accuracy.

While some research groups have used visualization and image processing methods to classify Microsoft Windows malware families [8], [15], [16], [18], [19], they have not examined Android malware families.

For Android malware family classification, many studies have used features other than malware images. For example, Alswaina and Elleithy [3] used the permissions declared in malicious Android apps as the static features. Arp *et al.* [4] used requested permissions, suspicious or restricted API calls, filtered intents, hardware components, network addresses, etc. Taheri *et al.* [9] extracted permissions and intents as static features, and API calls and network traffic as dynamic features. The dynamic features were used for malware family classification while the static features were used for malware detection.

Suarez-Tangil *et al.* [10] developed a static malware classification system called `DroidSieve` which relies on API calls, permissions, code structure, invoked components, native code, obfuscation artifacts, and obfuscation-invariant features. Its performance was evaluated on the datasets from the DREBIN and MalGenome. The authors showed only the results for the DREBIN dataset because it included all MalGenome samples and was larger and more recent. DroidSieve achieved a high accuracy with resilience against reflection by reducing code analysis and using resource-centered features.

Xu *et al.* [13] used control-flow graphs and data-flow graphs as static features on the instruction level of malware. They also used the n-gram sequences extracted from the graphs. Qiu *et al.* [14] proposed a multi-label classification model to annotate the malicious capabilities of suspicious malware samples. The model used API calls, permissions and network addresses extracted from APKs as the features. Massarelli *et al.* [17] employed resource consumption metrics from the `proc` file system through dynamic analysis, and they extracted the features by processing them with *detrended fluctuation analysis* and *Pearson's correlation*.

Chakraborty *et al.* [37] classified malicious apps into large and small families by using the combined features obtained through static and dynamic analyses. They collected author information, app components, and permissions as the static features, as well as *n*-gram representation from generated dynamic logs. The significant features they considered were the re-use of signatures, requesting permissions, and the use of encryption. They also evaluated the performance of supervised classification and unsupervised clustering.

Atzeni *et al.* [38] clustered malware samples in families and developed rules of family signatures to allow for the accurate identification of samples. They used both static and dynamic features. The static features contained permissions, activities, receivers and filters extracted from the `AndroidManifest.xml` of each app as well as suspicious API calls extracted from the code. The dynamic features contained the characteristics of the app that interacted with the operating system along with the network information generated during the app's execution.

Garcia *et al.* [39] proposed a method called `RevealDroid` for malware family identification which was shown to be accurate, efficient and obfuscation resilient. They employed static features such as API calls, reflective calls, and invocations in native binaries. They then evaluated the method by comparing it with the state-of-the-art methods.

DroidCat [40] detected and classified Android malware using dynamic features based on a behavioral app profile consisting of method calls and inter-component communication(ICC) intents. It achieved 97% F1-score accuracy for classifying apps evolving over the nine years. It was robust to attacks targeting system calls or specific sensitive APIs and malware samples adopting obfuscation schemes. It achieved higher performance compared with the two state-of-the-art techniques with the datasets from the AndroZoo, VirusShare [41], DREBIN, and MalGenome [42]. DroidCat adopted several learning algorithms, including Random Forest(RF), Support Vector Machine(SVM), Naive Bayes, etc. The RF with 128 trees performed better than all the alternatives.

Ficco [43] proposed a dynamic malware analysis technique resilient to specific evasion methods by combining generic and specialized detectors during the analysis process. The proposed technique dealt with the evolution of malware and employed an alpha-count mechanism that explored how the length of the observation time window during runtime could affect detectors' accuracy and speed. He evaluated the technique with 27 families from the DREBIN dataset. He also used another validation dataset downloaded from the VirusShare dataset from June 2013 to March 2014.

The previous studies [3], [4], [9], [10], [13], [14], [17], [37]–[40], [43] require domain expert knowledge and feature engineering such as binary disassembling, assembly language, and runtime information, while our approach does not require any of these factors.

Recently, researchers have presented evolution-based approaches for long-span malware classifiers [20]–[24]. Suares-Tangil and Stringhini [20] found that the type of activity performed by malware and the level of obfuscation used by malware has remarkably changed. Malicious payloads changed their behavior significantly over time. These findings mean that a sustainable approach is necessary for malware classifiers.

Cai [21], [23] proposed sustainability metrics and compared them among the latest five Android malware detectors. The malware classifier, DroidSpan [23], tried to find sophisticated and distinguishable features in behavioral evolution patterns of three ecosystem elements (the mobile platform, apps, and users) and information flow and then build sustainable models. He showed that DroidSpan outperformed the five detectors in sustainability. In [23], the authors focused on malware detection and did not address malware family classification.

DroidEvolver [24] an Android malware detector, updated itself and delayed classifier aging through online learning based on the majority voting of five linear learning models. It utilized API occurrence information for static features and updated its feature set incrementally by including new API calls used in the unknown apps. DroidEvoler showed a good performance and was robust against typical code obfuscation techniques.

APIGraph [22] enhanced the sustainability of the latest malware classifiers using API semantic similarity from a relation graph of Android APIs among evolved malware samples. It captured semantic similarity among evolved malware samples in an Android malware family.

III. DEEP LEARNING NETWORK BASED ON EXPLAINABLE ANALYSIS

The proposed method consists of a feature extractor with multi-streams and a classifier that receives the extracted features as input and finally determines the malware family. We first construct Basic-1D-CNN with whole malware data samples (Figure 4). Then, from the classification result of this network, the class separability of 15 sections composing the malware data sample was evaluated using Grad-CAM. The heatmap, the result of Grad-CAM, reflects the class separability information of each section. Each stream of the classification network with multi-streams corresponds to a section composing a malware sample. The individual stream contains Basic-1D-CNN that can effectively extract features suitable for the properties of malware data. Based on the heatmaps, we select the sections in which helpful information to classify malware families were concentrated and then constructed the feature extractor only with six streams corresponding to those sections (Figure 7).

A. 1-DIMENSIONAL CONVOLUTION FILTERS FOR MALWARE FAMILY CLASSIFICATION

Since the introduction of CNN-based deep learning networks [44]–[48], which perform very well in image classification, several studies [49] have also used CNNs to solve various classification problems involving items other than images. These methods converted a domain data into a 2D array in image form to apply the CNN-based network structure for image classification to problems in that same domain. Similar attempts have been made in malware family classification [5]–[7], [11], [33], [34], [36]. They convert a malware sample in string form into a 2D array form and apply 2D convolution filters to it. Most of these 2D convolution filter-based networks involve the fine-tuning of pre-trained models with large image databases such as ImageNet [50]. However, when utilizing these models in domains other than image data, there are several constraints caused by the differences in attributes between image data and domain data.

For example, for string-type data converted from malware samples, the correlation between adjacent data elements can be arbitrarily distorted in a 2D array. Moreover, considering the size of the data used when the network was pre-trained, it is necessary to resize the malware data to be used as the input data, and this resizing can also lead to distortion of the data. Figure 3 shows the classification performance depending on the interpolation method used in the resizing process when Inception-V3 [45], a representative 2D convolution filter-based classification network, was used for classification of DREBIN dataset [4]. In Figure 3, it can be seen that the classification performance was dependent on the process used to convert malware data to a 2D array data, which means that the 2D convolution filter-based network is not reliable in this classification problem.

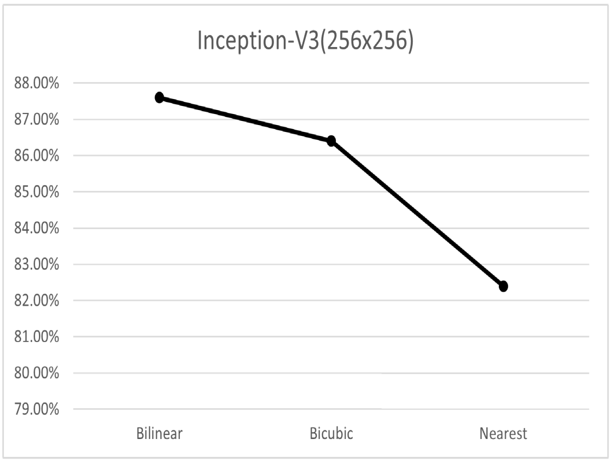


FIGURE 3. Classification performance of Inception-V3 according to the interpolation method used.

Therefore, rather than using a 2D convolution filter, we design Basic-1D-CNN, which is a 1D convolution filter-based classification network that can exploit the spatial relationships between data elements without distortion by retaining the original structure of the data. The proposed network was designed simply for the efficient computation of high-dimensional malware data. Considering the characteristics of malware data, where each data sample is of a different size, we use Global Average Pooling(GAP) [51] to enable the use of the original data samples as inputs to the network without the need for resizing processing. Further, by designing the network to have a small size, it can be learned using only malware data samples without pre-training, and it can therefore be specialized in malware family classification.

Figure 4 and Table 1 respectively present the structure of Basic-1D-CNN and the information on the filters used. Basic-1D-CNN consists of a feature extractor containing four convolution blocks and a classifier that uses these extracted features as input. Each convolution block consist of 1D convolution filters with a kernel size of 5 to extract features, an activation function (ReLU) [52], and a max-pooling layer to reduce computation. The feature extractor extracts the features that are useful for classifying malware families from the parts corresponding to the AM, CR, and CL file in malware data samples. The i-th feature map, which is the output value of the i-th convolution block y_i of the feature extractor, is as follows.

$$\begin{aligned} y_1 &= f_1(x|W_1) \\ y_i &= f_i(y_1|W_{i-1}), \quad for\ i = 2, 3, 4 \end{aligned} \tag{1}$$

x refers to a string corresponding to the AM, CR, and CL files in a malware data sample, which is an input to the network. Each sample of malware data has a different size, so the size of the feature map y_i from the convolution block can also vary depending on the size of the input value x. f_i is the i-th convolution block, and W_i is the parameter of the i-th convolution block. The number of filters used for each convolution block is $\{c_0, 2 * c_0, 4 * c_0, 8 * c_0\}$, respectively,

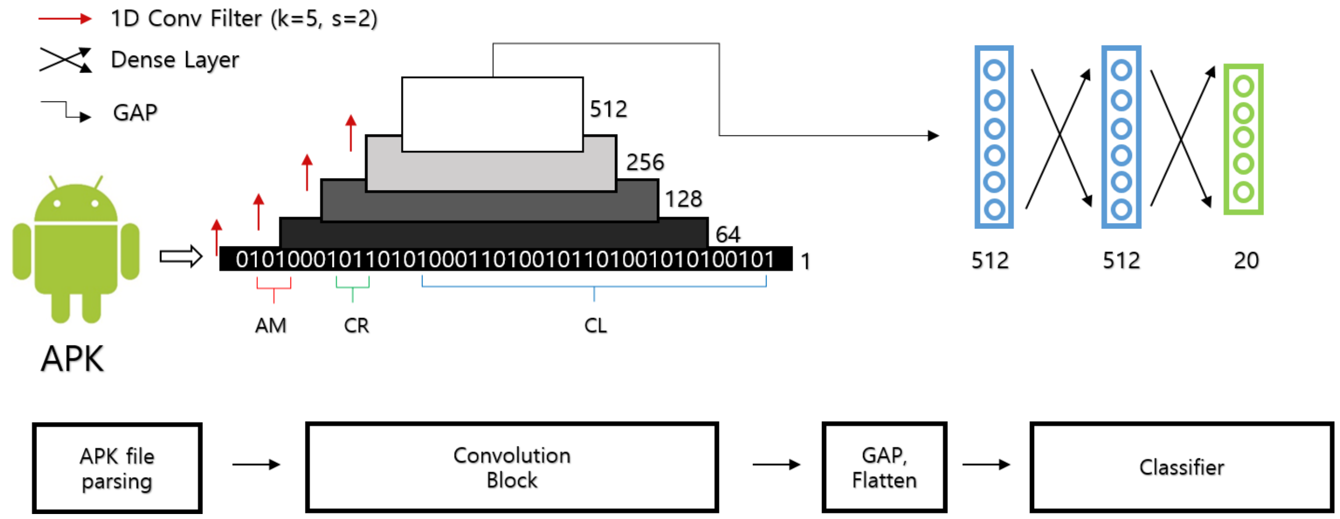


FIGURE 4. Configuration of the proposed Basic-1D-CNN. The proposed network uses malware sample in string form as input data for classifying malware families.

TABLE 1. Configuration of the proposed Basic-1D-CNN structure.

Stage	Operator	Input Channels	Output Channels
1	Conv1D, k=5, s=2	1	64
2	Conv1D, k=5, s=2	64	128
3	Conv1D, k=5, s=2	128	256
4	Conv1D, k=5, s=2	256	512
5	Fully Connected Layer	512	512
6	Output Layer	512	20

where c_0 is set to 64 by default.

$$G(y_4) = -\frac{1}{N}\sum_{d=1}^{N} y_4^d$$
$$y^{out} = D(G(y_4)|W_D) \tag{2}$$

where y_4^d is the d-th feature map of y_4. Since the size of y_4^d depends on the size of x, we adjust it to a fixed size (512 in this experiment) using GAP (G) prior to D. The size of $G(y_4)$ is equal to the number of channels in y_4, and its elements are the average value for each channel, which reflects how activated the channel is. Classifier D consists of a Fully Connected layer(FC) and an output layer. W_d is a parameter set for FC that constructs D, and the number of parameters in each layer is set to 512 and 20, respectively. y^{out} is the predicted malware family for x as the final output of D. The structure of Basic-1D-CNN is summarized in Table 1.

B. ANALYSIS OF CLASS SEPARABILITY FOR FILES AND SECTIONS IN MALWARE DATA

As previously mentioned in Subsection II-A, each malware sample has AM, CR, and CL files, each of which can be divided into multiple sections (or subsections). Each part (file or section) of the malware sample is connected in series, and the convolution filter moves all parts sequentially to extract features. However, not all parts of the sample contain useful information for malware family classification, and filtering the entire malware sample of around one million dimensions on average is not efficient for computational dimensions. Therefore, we used Grad-CAM(Gradient Weighted Class Activation Map) [53] to analyze how discriminative information is distributed for family classification in each part of a malware sample. Grad-CAM is an explainable analysis method used in the interpretation of predictive results for each class in Table 1 summarizes the structure of Basic-1D-CNN a convolution filter-based network. Grad-CAM generates a heatmap of the same size as the input data for each class (family) to be classified. Each element of a heatmap is the degree to which the corresponding element in the data sample affects the classification result, which is represented by a normalized value between 0 and 1. We obtain heatmaps (H_c) of Grad-CAM for C classes with validation data samples ($\{x_i^v, c\}, i = 1, .., N; c = 1, .., C$) to investigate which parts of malware data samples have useful information for family classification.

Figure 5 shows examples of H_c for the three classes. For visualization, we transform each heatmap into a two-dimensional form and display it with pseudo colors.

The closer the value of the heatmap element was to 1, the more red it was, and the closer it was to 0, the more blue it was. As shown in Figure 5, the information that is useful for classification within a malware sample is concentrated on the particular sections instead of being evenly distributed across the entire region. Based on the heatmaps for the validation samples, we plot the size of the values of the heatmap elements in each part to determine how the discriminant information is distributed in the malware sample.

Figure 6 is a boxplot for the heatmap elements values for two sections of AM, CR (`Cert.RSA`, `Cert.SF`), and twelve sections (subsections) of CL (`header`, `string_ids`, `type_ids`, `proto_ids`, `field_ids`, `method_ids`, `class_defs`, `map_list`,

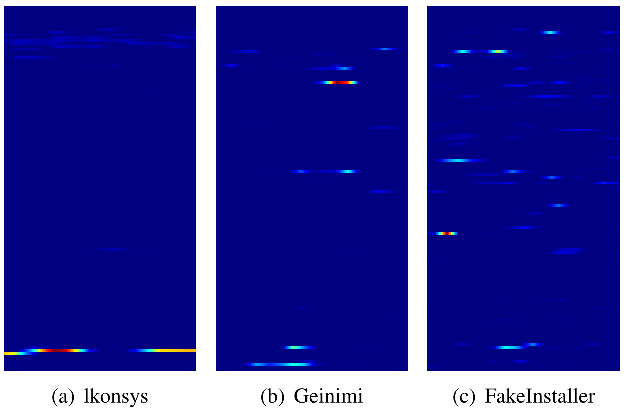


FIGURE 5. Grad-Cam results H_C from Basic-1D-CNN for the three classes (lkonsys, Geinimi, FakeInstaller) of the DREBIN dataset.

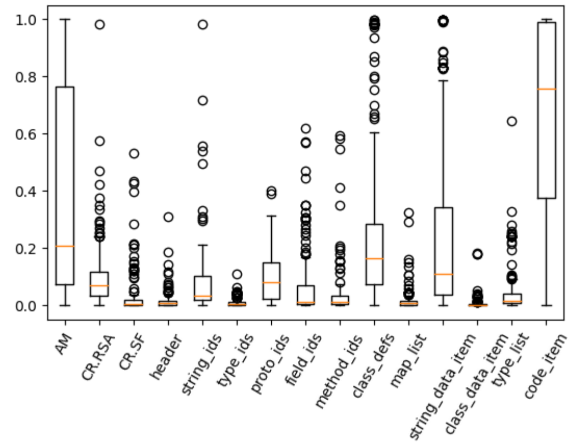


FIGURE 6. Heatmap element values for each part.

`string_data_item`, `class_data_item`, `type_list`, `code_item`). In Figure 6, the orange bar represents the median of the max values of the heatmap elements of the parts in N^v validation samples, while the height of the box represents the standard deviation.

As shown in Figure 6, discriminant information is concentrated on the AM file and the subsection `code_item` of the CL file, whereas there is little distribution in the `class_data_item` subsection and the `type_ids` section of the CL file.

C. DESIGN OF DEEP LEARNING NETWORK WITH MULTI-STREAMS

Based on the results in Figure 6, we chose six parts AM, `Cert.RSA`, `proto_ids`, `class_defs`, `string_data_item` and `code_item` with a large amount of discriminative information to build an efficient deep learning network for malware family classification. This allows us to focus more on critical information while reducing the computation volume by eliminating parts that are unnecessary for classification. Figure 7 shows the configuration of the proposed classification network with multi-streams. The network largely consist of feature extractors and classifiers. Since each part has different dimensions (lengths) of data as well as different informational properties, we construct a different feature extractor for each individual part to extract features specific to each part. The composite feature, which is created by combining feature maps generated from individual streams, is used as an input to a classifier consisting of two dense layers to obtain final classification results.

1) FEATURE EXTRACTOR

The feature extractor of the proposed model has a separate stream for each part of the malware data. Individual streams are designed with four convolution blocks, consisting of a 1D convolution filter with a kernel size of 5 to extract feature information, a ReLU function for activating nodes, and maxpooling to reduce computation. In total, 24 convolution blocks for six streams included in the feature extractor were trained to extract different feature information for each part by initializing them with different parameters. The feature map $y_s, s = 1, .., 6$ extracted from the s-th stream is as follows.

$$\begin{aligned} y_1^s &= f_1^s(x^s|W_1^s) \\ y_i^s &= f_i^s(x^s|W_{i-1}^s), \quad for\ i = 2, 3, 4 \end{aligned} \tag{3}$$

x^s means the part of the data sample corresponding to the s-th stream. f_i^s is the i-th convolution block for the s-th stream, and W_i^s is the parameters of f_i^s. The numbers of filters used for f_1^s through f_4^s were set to $\{c_0^s, 2 * c_0^s, 4 * c_0^s, 8 * c_0^s\}$, respectively, where c_0^s is set to 64 by default. We apply a ReLU activation function after passing the convolution layer to learn nonlinearities, then use max-pooling with a kernel size and stride of 2 to exclude less important feature information for efficient computation.

2) CLASSIFIER

Aside from the output layer, the fully connected layer of the classifier was composed of two layers. A composite feature vector (F) combined with feature maps from six streams in the feature extractor is used as an input to the classifier's first FC. Before passing values to the output layer, we apply a dropout method that learns with only the left node, while excluding nodes randomly, to avoid overfitting and improve generalization performance (in this paper, we experiment by changing the dropout rate from 25% to 50%). The output layer consists of as many nodes as the classes to be classified. Each node in the output layer computes the final output value by applying the weighted sum of nodes in the previous layer to the softmax function.

IV. EXPERIMENTAL RESULTS

A. DATASET AND IMPLEMENTATION DETAILS

The DREBIN dataset used in this experiment is one of the representative datasets used in malware family classification studies [2]. The DREBIN dataset was collected in the period from August 2010 to October 2012 and includes popular Android malware families such as Fake Installer, GoldDream,

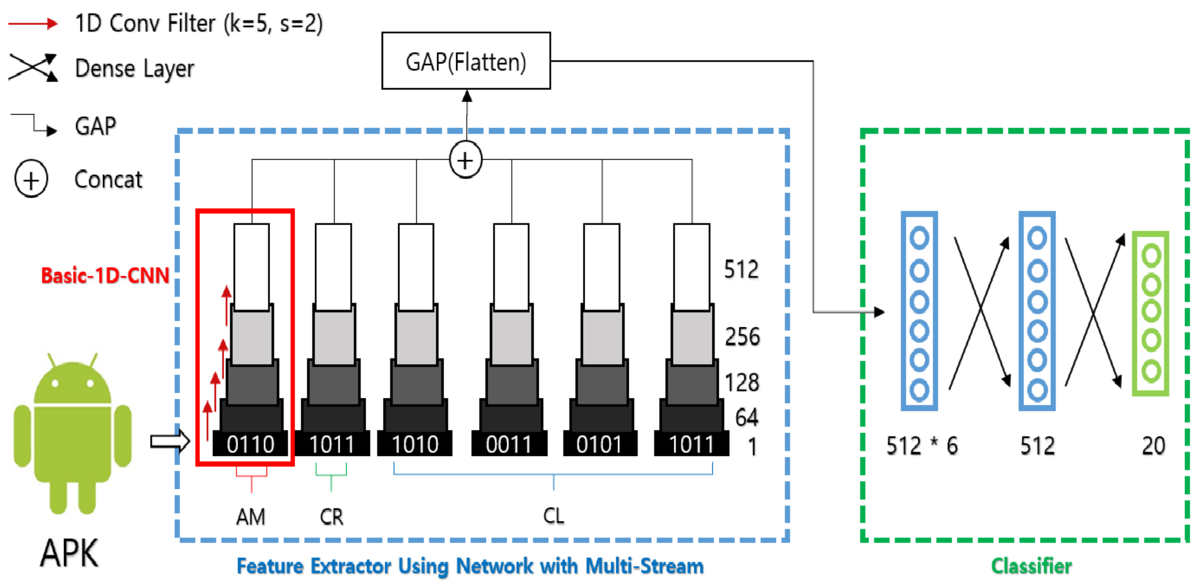


FIGURE 7. Overall structure of the proposed classification network with multi-streams.

TABLE 2. Comparison of publication counts for various malware datasets.

Dataset	Publications
DREBIN	18
MalGenome	16
Repository	6
Collection	6
AMD	3
Contagio	2
UpDroid	2
AndroZoo	2
AndroMalShare	1
Marvin	1

GingerMaster, DroidKungFu, etc. Most of the research literature from the years 2014–2020 has used the DREBIN dataset as the standard dataset for malware related experiments. Table 2 presents a summary of the various malware datasets used by the research community to date [2].

The DREBIN dataset contains 5,560 files from 179 different malware families. Among them, we experiment only on the top 20 malware families (SendPay, BaseBridge, ExploitLinuxLotoor, Geinimi, Gappusin, Kmin, SMSreg, FakeRun, DroidKungFu, DroidDream, FakeInstaller, Adrd, FakeDoc, Opfake, MobileTx, GinMaster, Plankton, Imlog, Iconosys, Glodream) with the largest number of samples. In total, 4,659 malware samples for 20 families were separated into training, validation, and test datasets at a respective ratio of 6:2:2.

For experiments, the libraries of Keras (v2.2.0) and PyTorch (v1.7.0) were used on a computer with NVIDIA-V100 32GB GPU and an Intel Xeon Silver 4210 processor. Learning of the classification network was carried out for 500 epochs, and 'early stopping' [54] was used to limit the epochs to 20 to prevent overtraining. For effective learning, we set the learning rate l as $0.1 * b/256$ depending on the size of the batch while, using the linear scaling learning rate method [55] (in this experiment, $l = 0.0007$ and 0.0125 for the 1D and 2D convolution filter-based models, respectively). Since the proposed method uses high-dimensional data as it does not involve the resizing process, we set the batch size to 2 so that the computational burden is not too large. In 2D convolution filter-based methods [5]–[7], [11], [33], [34], [36] for performance comparison with the proposed method, we set the batch size to 32 because they reduce the size of the data sample in the process of converting input data into 2D array form. For learning stability, Radam [56] was used as the optimizer, and categorical cross entropy [57] was used as the loss function.

B. PERFORMANCE EVALUATION FOR MALWARE FAMILY CLASSIFICATION

The main metrics for performance evaluation are as follows: (1) accuracy: the ratio of the number of correctly predicted observations to the total number of observations; (2) recall: the ratio of the number of correctly predicted positive observations to the total number of observations actually belonging to the positive class; (3) precision: the ratio of the number of correctly predicted positive observations to the total number of predicted positive observations; (4) F1-score: the weighted average of precision and recall, i.e., F1-score = 2 * (recall * precision)/(recall + precision).

To verify the effectiveness of the proposed Basic-1D-CNN, we compare the performance of malware family classification using deep learning networks such as SARVOTAM [6], Inception-V3 [45], LeNet [11], ResNet50 [46], and EfficientNetB0 [47]. Since data resizing is required to classify malware data samples using 2D convolution filter-based methods, we performed down sampling using Opencv and Fill library, which are widely used in the Python library.

TABLE 3. Comparison of classification performance by each model for DREBIN dataset.

Model	Acc	F1-score
SARVOTAM (108x108) [6]	86.6%	85.8
EfficientNetB0 (256x256) [47]	81.8%	81.3
Vgg16 (256x256) [48]	85.8%	84.9
Resnet50 (256x256) [46]	86.9%	86.4
Inception-V3 (256x256) [45]	87.6%	86.6
LeNet (64x64) [11]	90.2%	90.1
LeNet (128x128) [11]	88.1%	87.8
LeNet (256x256) [11]	87.9%	87.4
Basic-1D-CNN (single stream)	91.3%	91.4
Basic-1D-CNN (6-streams)	93.2%	93.2

The downsampling option uses 'bilinear', which is the best performance in the results previously presented in Subsection III-A (Figure 3). For SARVOTAM, we present the results of the image combination (CL+AM), which shows the best performance among various kinds of 'image combinations' [6].

Table 3 lists the classification results obtained using various methods examining 20 families in the DREBIN dataset. In Table 3, the number in parenthesis next to the name of each method is the size of the data sample when the malware data sample is resized to the 2D array form. To evaluate the performance of the Basic-1D-CNN itself, we present the classification results of Basic-1D-CNN when using the entire malware data sample without separating the data sample according to the part (file or subsection). In many studies on image classification, Inception-v3, ResNet50, and EfficientNetB0 have shown higher classification performance than LeNet, but the malware family classification results presented in Table 3 show that LeNet's performance is higher than those of Inception-v3, ResNet50, and EfficientNetB0. In particular, LeNet showed the highest performance when the size of the data sample was reduced to 64×64, which is also different from the results of several image classification studies. This suggests that even the same network can vary in performance depending on the domain, and it demonstrates the limitations of transfer learning for the use of pre-trained networks in different domains. The results of in Table 3, show that Basic-1D-CNN, which is designed for malware family classification, has the best classification performance among all the methods considered.

Table 4 demonstrates the effectiveness of a network with multi-stream and the performance of a network using only useful streams based on explainable analysis. In Table 4, the classification rates of networks configured multi-stream for all 15 parts of the data sample (Basic-1D-CNN(15-streams)) were 0.8% and 0.7, respectively, which were higher in accuracy and f1-score than in networks configured with a single stream (Basic-1D-CNN(single)) for the whole malware data sample. This is because in a network with multi-stream, the different characteristics of each part are effectively learned in each stream.

TABLE 4. Comparison of performance according to the number of streams of the proposed model.

Model	Acc	F1-score	Inference time(ms)	Training time (GPU hours)
Basic-1D-CNN (single stream)	91.3%	91.4	82.8	6.6
Basic-1D-CNN (15-streams)	92.1%	92.1	49.4	4.4
Basic-1D-CNN (6-streams)	93.2%	93.2	35.5	2.0

TABLE 5. Classification performance for each family in the proposed model.

Family	Acc	F1-score
SendPay	0.600	0.750
BaseBridge	0.833	0.901
ExploitLinuxLotoor	0.500	0.500
Geinimi	0.857	0.765
Gappusin	0.545	0.666
Kmin	0.968	0.968
SMSreg	0.500	0.666
FakeRun	1.000	1.000
DroidKungFu	0.959	0.921
DroidDream	0.875	0.903
FakeInstaller	0.968	0.978
Adrd	0.888	0.761
FakeDoc	0.964	0.964
Opfake	0.992	0.978
MobileTx	1.000	1.000
GinMaster	0.892	0.865
Plankton	1.000	1.000
Imlog	1.000	0.833
Iconosys	1.000	1.000
Glodream	0.545	0.631

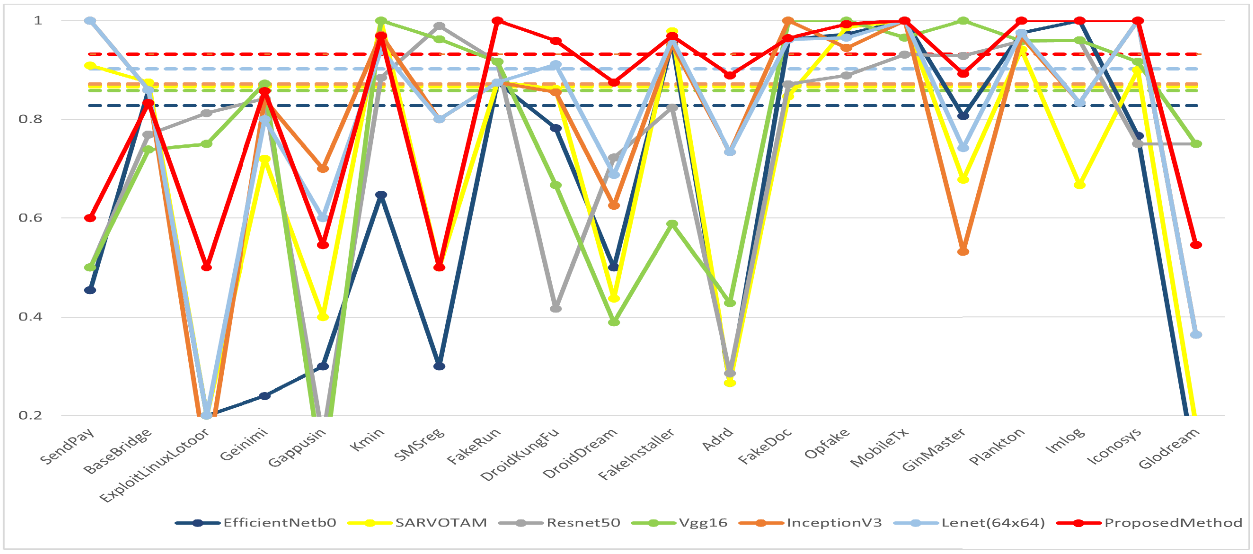
Moreover, since the network with multi-stream divides the data samples into several parts and processes them in parallel, the data classification time is reduced. Moreover, the model consisting of only the six selected streams, which are based on the analysis of discriminant information distribution using Grad-Cam heatmaps, showed the best performance. This shows that by eliminating unnecessary data and selectively using only data useful for classification, it is effective to extract better features while also reducing the overall running time. Tables 5 and 6 respectively present family-specific classification results and confusion matrix for 20 malware family classifications. As shown in Figure 8, the proposed model achieved the best performance with both accuracy and f1-scores exceeding those of other methods that considered all families.

We conducted experiments on the AMD dataset to evaluate the generalization performance of the proposed method. The AMD dataset is another representative dataset used for malware family classification research. The AMD data set contains 24,650 labeled Android malware samples classified into 135 varieties within 71 product families, dated from 2010 to 2016.

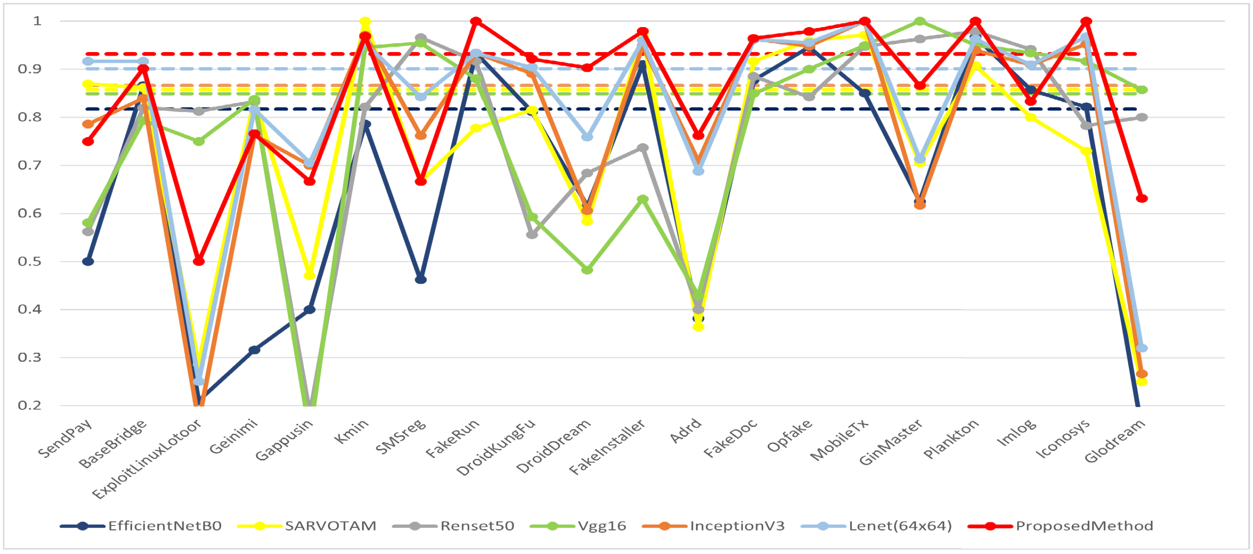
We performed intra-dataset experiments and inter-dataset experiments on the DREBIN and AMD datasets, respectively. In the intra-dataset experiment, samples from the

TABLE 6. Confusion matrix for the top 20 malware families in the proposed model.

	SendPay	Base Bridge	Exploit LinuxLotoor	Geinimi	Gappusin	Kmin	SMSreg	FakeRun	Droid KungFu	Droid Dream	Fake Installer	Adrd	FakeDoc	Opfake	MobileTx	GinMaster	Plankton	Imlog	Iconosys	Glodream	samples
SendPay	6	0	3	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	10
BaseBridge	0	55	0	1	0	0	0	0	4	0	0	0	0	2	0	4	0	0	0	0	66
ExploitLinuxLotoor	0	0	5	0	1	0	0	0	2	0	1	0	0	0	0	0	0	1	0	0	10
Geinimi	0	0	0	18	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	21
Gappusin	0	0	0	0	6	0	0	0	3	0	0	0	0	0	0	1	0	0	0	1	11
Kmin	0	0	0	0	0	31	0	0	0	0	0	0	0	0	0	1	0	0	0	0	32
SMSreg	0	0	0	0	0	0	5	0	1	0	1	1	0	1	0	1	0	0	0	0	10
FakeRun	0	0	0	0	0	0	0	11	0	0	0	0	0	0	0	0	0	0	0	0	11
DroidKungFu	0	0	0	2	0	0	0	0	117	0	0	0	0	0	0	3	0	0	0	0	122
DroidDream	0	0	0	0	0	0	0	0	1	14	0	0	0	0	0	1	0	0	0	0	16
FakeInstaller	0	0	2	0	0	0	0	0	1	0	186	0	0	2	0	0	0	1	0	0	192
Adrd	0	0	0	0	0	0	0	0	0	1	0	16	1	0	0	0	0	0	0	0	18
FakeDoc	0	0	0	1	0	0	0	0	0	0	0	0	27	0	0	0	0	0	0	0	28
Opfake	0	0	0	0	0	0	0	0	1	0	0	0	0	138	0	0	0	0	0	0	139
MobileTx	0	0	0	0	0	0	0	0	0	0	0	0	0	0	14	0	0	0	0	0	14
GinMaster	0	1	0	2	0	1	0	0	2	0	0	0	0	0	0	58	0	0	0	1	65
Plankton	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	123	0	0	0	123
Imlog	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	5	0	0	5
Iconosys	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	28	0	28
Glodream	0	0	0	2	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	6	11



(a) Comparison of family classification accuracy for each model



(b) Comparison of family classification F1-Score for each model

FIGURE 8. Comparison of classification performance for each family.

same dataset were separated into training, validation, and test datasets at a respective ratio of 6:2:2. In the inter-dataset experiment, we learned and evaluated the classifier using a dataset different from the dataset used to train the feature extraction model. In the inter-dataset experiment, we used 30 malware families, which do not overlap with the families of the DREBIN dataset and have many samples per family, among a total of 71 malware families included in the AMD dataset to evaluate generalization performance.

Table 7 shows the experimental results of intra-dataset and inter-dataset. In Table 7, AMD(4570) randomly selected a total of 4,570 malware samples as much as the total number of samples in the DREBIN dataset, and AMD(23755) used all malware samples from the AMD dataset for 30 families. As shown in Table 7, in the intra-dataset, the accuracy for the DREBIN dataset was higher than for the AMD dataset. This is because the number of families in the DREBIN dataset is smaller than that of the AMD dataset and the variation of samples in the AMD dataset collected over a more extended period is greater than that of the DREBIN dataset. In the inter-dataset experiment, the feature extraction model learned with the AMD dataset performed better than the model learned with the DREBIN dataset. This shows that the generalization performance is improved when the variation information of malware samples of the AMD dataset is learned. In particular, the experimental results for AMD(23755)-DREBIN datasets show that when learning the proposed model with sufficient training data, we can expect that the accuracy performance of inter-dataset is comparable to that of intra-dataset.

V. DISCUSSION

The advantage of using deep learning networks for malware family classification is that they can be used to classify data without the need for separate processes such as reverse engineering, decryption, de-obfuscation, or code execution. To use existing deep learning-based classification networks to classify malware samples with an average of one million dimensions, the size of the malware samples should be reduced when learning the network due to computational limitations. However, we experimentally confirm that the downsampling process is not only prone to unintended loss of information, but also unstable network classification performance according to the interpolation option (see Figure 3). Further, when classifying malware families using the widely used 2D convolution filter-based network in image classification problems, the results tended to be different from those of image classification problems (see Table 3), which only show the limitations of transfer learning and the need for networks designed suitably for the specific properties of the data. In this paper, we designed a 1D convolution filter-based network for malware family classification and identified the distribution of discriminant information in malware samples through gradient analysis using Grad-CAM. Based on our analysis of the distribution of the discriminant information, we built a network with multi-stream for each part that constructs the sample. As a result, we extracted composite features that were effective for malware family classification, thus achieving better performance in terms of accuracy and f1-score than conventional 2D convolution filter-based networks. Furthermore, by using the malware sample as it is without preprocessing, we prevented unexpected information distortion and performed effective classification over simple networks.

In this paper, we have found that the malware samples in the DREBIN dataset possess two important properties that are not typically observed in the image domain. As the first property, because the DEX files of malicious apps come in a very wide range of sizes, an image-based malware family classification loses local features through resizing and the model's performance is sensitively changed depending on the type of interpolation. Secondary, the models that worked well in the image domain performed worse than the simplest model in the Android malware family domain. Thus, the 1D convolution filters achieves better performance than the 2D convolution filters.

To classify Android malware families, many existing studies [9], [17], [37], [38], [40], [43] have adopted dynamic features to represent the malicious app behaviors. DroidCat [40] a dynamic malware family classifier, achieved a high F1-score for the datasets from AndroZoo, VirusShare, DREBIN, and MalGenome. Ficco [43] proposed a dynamic analysis-based ensemble detector resilient to the malware evolution, and evaluated the detector with the datasets from DREBIN and VirusShare. However, these dynamic approaches are not scalable due to the difficulty and overhead of tracing many malicious apps. They can suffer from a high false-positive rate [58].

Unlike the dynamic approaches, the static methods, including our approach and DroidSieve [10] do not incur runtime tracing costs and are scalable. Our approach incurs a low overhead on testing (that is, 35 milli seconds was taken for testing on average per app), while DroidCat [40] took 0.01 seconds for testing on average per app. In addition, our approach does not need in-depth analysis, such as parsing the AndroidManifiest.xml and DEX files, while DroidSieve did.

Several studies have extracted the features from the executable code of apps [4], [9], [10], [13], [14], [37], [38]. Those code based-features may introduce excessively detailed information [37]. If malware writers adopt encryption or reflection to evade static analysis, this may introduce substantial noise into the code.

On the other hand, our approach requires only the raw byte data of the `classes.dex`, `Manifest.xml` and certificate in each malicious app, without the need for further analysis. Therefore, even when unknown malware samples newly appear, there is no need to convert the app to 2D array forms of images. In particular, our approach can be used to identify new Android malware families such as `LeifAccess`.

The key idea of our approach can be applied to other platforms such as Microsoft Windows, even though the package structure (APK) and executable file (DEX) format of Android apps are different from the package structure (MSIX) and executable file (PE) format of Microsoft Windows apps. The reason for this is that our string-based approach only uses the data obtained in the form of strings from specific parts of each malicious app. That is, we can expand our approach to other platforms only if the internal structure of each malware

TABLE 7. Classification performance of intra- and inter-dataset experiments for the DREBIN and AMD datasets (recall/precision/accuracy) (%).

Train / Test	DREBIN	AMD(4570)	AMD(23755)
DREBIN	93.4/93.6/93.2	66.3/67.2/65.1	76.8/75.0/73.7
AMD(4570)	87.2/87.9/87.2	82.8/87.0/83.5	-
AMD(23755)	86.9/87.5/86.4	-	92.6/92.8/92.2

sample on the target platform is understood. However, the part of each malware sample that contains the most significant features may vary depending on the specific platform.

Another issue in classifying Android malware families is to handle the malicious apps written with cross-platform mobile app development tools such as `Xamarin`, `Unity`, `Cocos2D`, `PhoneGap`, etc [59], [60]. In these cases, malicious code may be located in the javascript(js) file, or `asset` or `lib` folders of an APK, as opposed to the DEX file. To address this problem, we plan to additionally consider the js file, the `DLL` in the asset folder, and the `so` files in the lib folder of malicious apps.

To classify malware samples into families based on malware images, some studies have used grayscale images [6], [7], [11], [18] while others have used color images [5], [8], [15]. For Microsoft malware family classification, [18] used grayscale images while [8] and [15] used color images. Fu *et al.* [15] expanded grayscale images to color images, and they discovered that the visualization results of their samples were slightly different from the results of [18]. According to the existing study [8] that classified Microsoft malware families based on color images, malware images in color achieved better accuracy than grayscale malware images.

Kumar *et al.* [61] presented an Android malware detection method that transformed APK files into grayscale, RGB, CMYK and HSL images, and they extracted GIST feature from the four types of images respectively. They classified suspicious apps as benign or malicious using the three classifiers of RF, kNN, and decision tree (DT). Among the four image formats and the three classifiers, RF achieved the best performance with the grayscale images. Meanwhile, Fang *et al.* [5] classified Android malware families by converting DEX files into RGB images as well as obtaining plain text from the DEX files. They reported that the grayscale image has single characteristics while the RGB image have more features than the grayscale image [5].

When considering all the above studies, it is not clear which kinds of images are the most effective for Android malware analysis.

However, in our experiments using Grad-Cam, we found that among the various parts constituting the APK, there is a greater amount of identification information in `AM`, `Cert.RSA`, `proto_ids`, `class_defs`, `string_data_item` and `code_item`. We also found that by using only the parts with a lot of discriminant information, we could simultaneously increase the classification performance and reduce the inference time

Table 7 presents a comparison of our work with the state-of-the-art Android malware family classification studies.

TABLE 8. Comparison of our method with previous studies.

Study	Year	Features	Model	Dataset	No. of families
[5]	2020	RGB images, texture, text from DEX files	kNN, RF, SVM	AMD	15
[6]	2020	grayscale images from CR, AM, CL, etc.	CNN, CNN-SVM CNN-kNN, CNN-RF	DREBIN	20
[7]	2019	grayscale images from DEX files	CNN	DREBIN	14 & 20
[11]	2020	grayscale images from DEX and data section	2D CNN	DREBIN	20
[33]	2018	RGBs from opcodes, API pkgs & functions	CNN	DREBIN	14
[34]	2020	RGB images from APIs, opcodes, perms, etc.	CNN	AndroZoo	10
[36]	2018	grayscale images from DEX files	XGBoost	Not specified	10
Our method	2021	grayscale images from CR, AM, CL	1D CNN, 2D CNN	DREBIN	20

In our work, the 1D convolution filter-based model performed well on comparison to various 2D convolution filter-based models.

Meanwhile, the CR+AM image-based CNN-SVM model performed well compared to a generic CNN model. Our findings differ from those of the study with `SARVOTAM` [6] where the CR+AM image-based CNN-SVM model showed better performance than other models such as CNN, CNN-RF, and CNN-kNN.

If malware writers simultaneously distribute each malware instance in a simple form as well as an obfuscated form, then the images between the simple and obfuscated versions can differ. In this case, the malware image-only approach may not be effective. One solution to this is to introduce another features which is resilient against obfuscation/packing/encryption attacks, then combine the features with the malware images. Examples of obfuscation-resilient features are the use of security-sensitive API including reflection-based features as well as the use of code features extracted from native binaries of apps [39].

We dealt with the sustainability problem by conducting intra- and inter-dataset experiments on both DREBIN and AMD datasets. The AMD dataset was collected over more extended periods than the DREBIN dataset. In the inter-dataset experiment, we have found that the feature extraction model learned with the AMD dataset performed better than the model learned with the DREBIN dataset. This shows that the AMD dataset is more suitable for longitudinal and evolutionary studies than the DREBIN dataset. The generalization performance is improved when the variation information of malware samples of the AMD dataset is learned.

VI. CONCLUSION

This paper has adopted a 1D convolution filters for the goal of classifying Android malware families. Our approach receives input data in the form of strings from each part of the manifest file (AM), certificate file (CR), and the sections of executable code (CL) in every malicious app, then connects the data in series and builds a 1D convolution filter-based deep learning model, at which point it can finally classify malicious apps into related families.

We have observed that the Basic-1D-CNN with a low network depth and small parameters show higher performance than 2D CNN based models (SARVOTAM, EfficientNetB0, Vgg16, ResNet50, Inception-V3, and LeNet).

We created a model with 24-convolution layers and 2-dense layers for a total of 26 layers.

According to the experimental results, the 1D convolution filter-based classification network with multi-streams achieved the highest accuracy with 93.2%. In particular, the 1D convolution filter-based model takes over 23 times more inference time than the 2D convolution filter-based model, but its performance is better than those of other models. Specific parts in an app, such as `code_item` and `AM`, contain the most effective features for classifying malware families.

Finally, we have compared our approach with state-of-the-art approaches to Android malware family classification. Our approach does not require domain expert knowledge and feature engineering such as disassembling, de-obfuscating, and decrypting of executable code of Android apps. Further, our approach uses static analysis that, incurs less overhead, compared to dynamic analysis which needs direct execution and as well as expensive computation.

In the future, we intend to consider a case which even includes malware families of very small sizes (e.g., the number of samples in a family < 10), and to explore a data augmentation technique to augment minor class to overcome imbalanced data.

REFERENCES

[1] McAfee. *McAfee Mobile Threat Report Q1, 2020*. Accessed: Jan. 2, 2021 [Online]. Available: https://www.mcafee.com/content/dam/consumer/en-us/docs/2020-Mobile-Threat-Report.pdf

[2] F. Alswaina and K. Elleithy, "Android malware family classification and analysis: Current status and future directions," *Electronics*, vol. 9, no. 6, p. 942, Jun. 2020.

[3] F. Alswaina and K. Elleithy, "Android malware permission-based multi-class classification using extremely randomized trees," *IEEE Access*, vol. 6, pp. 76217–76227, 2018, doi: 10.1109/ACCESS.2018.2883975.

[4] D. Arp, M. Spreitzenbarth, M. Hübner, H. Gascon, and K. Rieck, "DREBIN: Effective and explainable detection of Android malware in your pocket," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, San Diego, CA, USA, Feb. 2014, pp. 23–26.

[5] Y. Fang, Y. Gao, F. Jing, and L. Zhang, "Android malware familial classification based on DEX file section features," *IEEE Access*, vol. 8, pp. 10614–10627, 2020, doi: 10.1109/ACCESS.2020.2965646.

[6] 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, Dec. 2020.

[7] Y. Sun, Y. Chen, Y. Pan, and L. Wu, "Android malware family classification based on deep learning of code images," *IAENG Int. J. Comput. Sci.*, vol. 46, no. 4, pp. 524–533, 2019.

[8] D. Vasan, M. Alazab, S. Wassan, H. Naeem, B. Safaei, and Q. Zheng, "IMCFN: Image-based malware classification using fine-tuned convolutional neural network architecture," *Comput. Netw.*, vol. 171, Apr. 2020, Art. no. 107138.

[9] L. Taheri, A. F. A. Kadir, and A. H. Lashkari, "Extensible Android malware detection and family classification using network-flows and API-calls," in *Proc. Int. Carnahan Conf. Secur. Technol. (ICCST)*, Oct. 2019, pp. 1–8.

[10] G. Suarez-Tangil1, S. K. Dash, M. Ahmadi, J. Kinder, G. Giacinto, and L. Cavallaro, "DroidSieve: Fast and accurate classification of obfuscated Android malware," in *Proc. 7th ACM Conf. Data Appl. Secur. Privacy*, 2017, pp. 309–320.

[11] M. Kang, J. Park, S. Park, S. J. Cho, and M. Park, "Android malware family classification using images from DEX files," in *Proc. 9th Int. Conf. Smart Media Appl. (SMA)*, 2020, pp. 181–186.

[12] J. Jung, J. Choi, S. J. Cho, S. Han, M. Park, and Y. Hwang, "Android malware detection using convolutional neural networks and data section images," in *Proc. Conf. Res. Adapt. Convergent Syst. (ACM RACS)*, 2018, pp. 149–153.

[13] Z. Xu, K. Ren, and F. Song, "Android malware family classification and characterization using CFG and DFG," in *Proc. Int. Symp. Theor. Aspects Softw. Eng. (TASE)*, Jul. 2019, pp. 49–56.

[14] J. Qiu, J. Zhang, W. Luo, L. Pan, S. Nepal, Y. Wang, and Y. Xiang, "A3CM: Automatic capability annotation for Android malware," *IEEE Access*, vol. 7, pp. 147156–147168, 2019.

[15] J. Fu, J. Xue, Y. Wang, Z. Liu, and C. Shan, "Malware visualization for fine-grained classification," *IEEE Access*, vol. 6, pp. 14510–14523, 2018.

[16] K. Kancherla and S. Mukkamala, "Image visualization based malware detection," in *Proc. IEEE Symp. Comput. Intell. Cyber Secur. (CICS)*, Apr. 2013, pp. 40–44.

[17] L. Massarelli, L. Aniello, C. Ciccotelli, L. Querzoni, D. Ucci, and R. Baldoni, "Android malware family classification based on resource consumption over time," in *Proc. 12th Int. Conf. Malicious Unwanted Softw. (MALWARE)*, Oct. 2017, pp. 31–38.

[18] L. Nataraj, S. Karthikeyan, G. Jacob, and B. S. Manjunath, "MalWare images: Visualization and automatic classification," in *Proc. 8th Int. Symp. Visualizat. Cyber Secur.*, 2011, pp. 1–7.

[19] D. Vasan, M. Alazab, S. Wassan, B. Safaei, and Q. Zheng, "Image-Based malware classification using ensemble of CNN architectures (IMCEC)," *Comput. Secur.*, vol. 92, May 2020, Art. no. 101748.

[20] G. Suarez-Tangil and G. Stringhini, "Eight years of rider measurement in the Android malware ecosystem: Evolution and lessons learned," 2018, *arXiv:1801.08115*.

[21] H. Cai, "Embracing mobile app evolution via continuous ecosystem mining and characterization," in *Proc. IEEE/ACM 7th Int. Conf. Mobile Softw. Eng. Syst.*, Jul. 2020, pp. 31–35.

[22] X. Zhang, Y. Zhang, M. Zhong, D. Ding, Y. Cao, Y. Zhang, and M. Yang, "Enhancing state-of-the-art classifiers with api semantics to detect evolved Android malware," in *Proc. CCS*, 2020, pp. 757–770.

[23] H. Cai, "Assessing and improving malware detection sustainability through app evolution studies," *Trans. Softw. Eng. Methodol.*, vol. 29, no. 2, pp. 1–28, 2020.

[24] K. Xu, Y. Li, R. Deng, K. Chen, and J. Xu, "DroidEvolver: Self-evolving Android malware detection system," in *Proc. EuroS&P*, Jun. 2019, pp. 47–62.

[25] F. Wei, Y. Li, S. Roy, X. Ou, and W. Zhou, "Deep ground truth analysis of current Android malware," in *Proc. Int. Conf. Detection Intrusions Malware, Vulnerability Assessment*. Cham, Switzerland: Springer, 2017, pp. 252–276.

[26] Android Studio. *Analyze Your Build With APK Analyser*. Accessed: Feb. 2, 2021 [Online]. Available: https://developer.android.com/studio/build/apk-analyzer

[27] Signed APK META-INF Files-Digests. *Signature and Certificate*. Accessed: Feb. 2, 2021. [Online]. Available: https://sites.google.com/site/jamestu6166workingtips/gm-git-command/signed-apk-meta-inf-files-digests-signature-and-certificate

[28] Wikipedia. *Android Application Package*. Accessed: Feb. 2, 2021 [Online]. Available: https://en.wikipedia.org/wiki/Android_application_package

[29] *The Way of Ryantzj, Android Application/Package APK Structure Part 1*. Accessed: Feb. 2, 2021. [Online]. Available: http://www.ryantzj.com/android-application-package-apk-structure-part-1.html

[30] *Dalvik Executable Format*. Accessed: Feb. 2, 2021. [Online]. Available: https://source.android.com/devices/tech/dalvik/dex-format

[31] X. Zhang, E. Breitinger, and I. Baggili, "Rapid Android parser for investigating DEX files (RAPID)," *Digit. Invest.*, vol. 17, pp. 28–39, Jun. 2016.

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

[33] Y. L. Zhao and Q. Qian, "Android malware identification through visual exploration of disassembly files," *Int. J. Netw. Secur.*, vol. 20, no. 6, pp. 1061–1073, 2018.

[34] A. Darwaish and F. Nait-Abdesselam, "RGB-based Android malware detection and classification using convolutional neural network," in *Proc. IEEE Global Commun. Conf.*, Dec. 2020, pp. 1–6.

[35] K. Allix, T. F. Bissyandé, J. Klein, and Y. Le Traon, "AndroZoo: Collecting millions of Android apps for the research community," in *Proc. IEEE/ACM 13th Work. Conf. Mining Softw. Repositories (MSR)*, 2016, pp. 468–471.

[36] H. Chen, R. Du, Z. Liu, and H. Xu, "Android malware classification using XGBoost based on images patterns," in *Proc. IEEE 4th Inf. Technol. Mechatronics Eng. Conf. (ITOEC)*, Dec. 2018, pp. 1358–1362.

[37] T. Chakraborty, F. Pierazzi, and V. S. Subrahmanian, "EC2: Ensemble clustering and classification for predicting Android malware families," *IEEE Trans. Dependable Secure Comput.*, vol. 17, no. 2, pp. 262–277, Mar. 2020, doi: 10.1109/TDSC.2017.2739145.

[38] A. Atzeni, F. Díaz, A. Marcelli, A. Sánchez, G. Squillero, and A. Tonda, "Countering Android malware: A scalable semi-supervised approach for family-signature generation," *IEEE Access*, vol. 6, pp. 59540–59556, 2018.

[39] J. Garcia, M. Hammad, and S. Malek, "Lightweight, obfuscation-resilient detection and family identification of Android malware," *ACM Trans. Softw. Eng. Methodol.*, vol. 26, no. 3, pp. 1–29, Jan. 2018.

[40] H. Cai, N. Meng, B. G. Ryder, and D. Yao, "DroidCat: Effective Android malware detection and categorization via app-level profiling," *IEEE Trans. Inf. Forensics Security*, vol. 14, no. 6, pp. 1455–1470, 2019.

[41] *VirusShare*. Accessed: 2011. [Online]. Available: https://virusshare.com/

[42] Y. Zhou and X. Jiang, "Dissecting Android malware: Characterization and evolution," in *Proc. IEEE Symp. Secur. Privacy*, May 2012, pp. 95–109.

[43] M. Ficco, "Malware analysis by combining multiple detectors and observation windows," *IEEE Trans. Comput.*, early access, May 19, 2021, doi: 10.1109/TC.2021.3082002.

[44] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. Adv. Neural Inf. Process. Syst. (NIPS)*, vol. 25, Stateline, NV, USA, Dec. 2012, pp. 1097–1105.

[45] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Feb. 2016, pp. 2818–2826.

[46] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2016, pp. 770–778.

[47] M. Tan and Q. Le, "EfficientNet: Rethinking model scaling for convolutional neural networks," in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 6105–6114.

[48] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2014, *arXiv:1409.1556*.

[49] A. Sharma and D. Kumar, "Classification with 2-D convolutional neural networks for breast cancer diagnosis," 2020, *arXiv:2007.03218*.

[50] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2009, pp. 248–255.

[51] M. Lin, Q. Chen, and S. Yan, "Network in network," 2013, *arXiv:1312.4400*.

[52] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural Netw.*, vol. 61, pp. 85–117, Jan. 2015.

[53] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra, "Grad-CAM: Visual explanations from deep networks via gradient-based localization," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Oct. 2017, pp. 618–626.

[54] R. Caruana, S. Lawrence, and L. Giles, "Overfitting in neural nets: Backpropagation, conjugate gradient, and early stopping," in *Proc. Adv. Neural Inf. Process. Syst.*, 2001, pp. 402–408.

[55] T. He, Z. Zhang, H. Zhang, Z. Zhang, J. Xie, and M. Li, "Bag of tricks for image classification with convolutional neural networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, May 2019, pp. 558–567.

[56] L. Liu, H. Jiang, P. He, W. Chen, X. Liu, J. Gao, and J. Han, "On the variance of the adaptive learning rate and beyond," 2019, *arXiv:1908.03265*.

[57] K. P. Murphy, *Machine Learning: A Probabilistic Perspective*. Cambridge, MA, USA: MIT Press, 2012.

[58] K. N. Khasawneh, N. Abu-Ghazaleh, D. Ponomarev, and L. Yu, "RHMD: Evasion-resilient hardware malware detectors," in *Proc. 50th Annu. IEEE/ACM Int. Symp. Microarchitecture*, Oct. 2017, pp. 315–327.

[59] J. Shim, K. Lim, S.-J. Cho, S. Han, and M. Park, "Static and dynamic analysis of Android malware and goodware written with unity framework," *Secur. Commun. Netw.*, vol. 2018, pp. 1–12, Jun. 2018.

[60] W. Lee and X. Wu, "Cross-platform mobile malware, write once, run everywhere," in *Proc. Virus Bull. Conf.*, Oct. 2015, pp. 1–9.

[61] A. Kumar, K. P. Sagar, K. S. Kuppusamy, and G. Aghila, "Machine learning based malware classification for Android applications using multimodal image representations," in *Proc. 10th Int. Conf. Intell. Syst. Control (ISCO)*, Jan. 2016, pp. 1–6.