machine learning & knowledge extraction

MDPI

Article

Android Malware Classification Based on Fuzzy Hashing Visualization

Horacio Rodriguez-Bazan †, Grigori Sidorov † and Ponciano Jorge Escamilla-Ambrosio *,†

Centro de Investigación en Computación (CIC), Instituto Politécnico Nacional (IPN), Av. Juan de Dios Batiz, s/n, Mexico City 07320, Mexico; hrodriguezb2019@cic.ipn.mx (H.R.-B.); sidorov@cic.ipn.mx (G.S.)

* Correspondence: pescamilla@cic.ipn.mx

† These authors contributed equally to this work.

Abstract: The proliferation of Android-based devices has brought about an unprecedented surge in mobile application usage, making the Android ecosystem a prime target for cybercriminals. In this paper, a new method for Android malware classification is proposed. The method implements a convolutional neural network for malware classification using images. The research presents a novel approach to transforming the Android Application Package (APK) into a grayscale image. The image creation utilizes natural language processing techniques for text cleaning, extraction, and fuzzy hashing to represent the decompiled code from the APK in a set of hashes after preprocessing, where the image is composed of n fuzzy hashes that represent an APK. The method was tested on an Android malware dataset with 15,493 samples of five malware types. The proposed method showed an increase in accuracy compared to others in the literature, achieving up to 98.24% in the classification task.

Keywords: android malware; convolutional neural network; deep learning; fuzzy hashing; malware classification; natural language processing

Citation: Rodriguez-Bazan, H.; Sidorov, G.; Escamilla-Ambrosio, P.J. Android Malware Classification Based on Fuzzy Hashing Visualization. *Mach. Learn. Knowl. Extr.* **2023**, *5*, 1826–1847. https://doi.org/10.3390/make5040088

Academic Editors: Antonio Fernandez-Caballero and Byung-Gyu Kim

Received: 30 September 2023
Revised: 17 November 2023
Accepted: 20 November 2023
Published: 28 November 2023

1. Introduction

In the contemporary digital age, Android has become the most widely used mobile operating system in the world, powering billions of smartphones and other connected devices. The open and adaptable nature of the Android ecosystem [1] has not only facilitated innovation and user empowerment but has also inadvertently opened the floodgates to a relentless wave of cyber threats. Among these threats, Android malware is a pervasive and evolving menace. According to Kaspersky Security Network [2], in Q3 2022, over 5.5 million mobile malware was blocked, which shows an exponential increase in mobile malware lately.

Android malware encompasses malicious software designed to exploit vulnerabilities, steal sensitive information, disrupt operations, and sometimes extort users. These malicious programs, often cloaked in seemingly innocuous applications, pose a significant threat to personal privacy, data security, and the integrity of the Android ecosystem.

In light of this looming threat, the necessity to detect and classify Android malware has never been more critical. As the Android malware landscape continues to evolve with increasing sophistication, more than traditional security measures are needed. Thus, the development of robust malware detection and classification techniques is paramount. By understanding the inner workings of malware and employing advanced analysis methods, we gain the capacity to mitigate the threats they pose, safeguard user data, and fortify the resilience of the Android ecosystem.

Researchers have faced new opportunities and challenges in the landscape of Android malware analysis. Traditional methods for detecting and classifying Android malware often relied on features such as n-grams, API calls, and sandbox outputs, combined with machine learning and deep learning algorithms. While these techniques have been somewhat effective, they have limitations. As Android malware becomes increasingly sophisticated, it is not only about the static analysis of code or the behavior of applications but also about how they visually appear to the human eye.

A novel approach gaining attention involves transforming malware samples into images and leveraging machine learning and deep learning algorithms to classify and detect malware based on visual content. This innovative technique adds a new dimension to malware analysis and presents new challenges. Converting APK files into images introduces concerns about the loss of information, the potential for adversarial attacks, and the need for preprocessing techniques.

The research community is exploring these image-based techniques as Android malware evolves with more complex and polymorphic characteristics. This shift opens to deeper insights and improved detection but also emphasizes the significance of robust preprocessing, comprehensive datasets, and resilient machine learning and deep learning models.

This work proposes a new method for Android malware classification, transforming the APK sample into a grayscale image composed of the fuzzy hashing of decompiled and preprocessed code. The main contributions of this work can be described as follows.

- A novel method is introduced for transforming Android applications into grayscale images of fuzzy hashes. The decompiled code is initially subjected to preprocessing using Natural Language Processing (NLP) techniques. Subsequently, a fuzzy hashing technique is employed to compute hashes for each code block, significantly reducing image size. The importance of image size is underscored due to the potential impact on feature extraction, particularly in light of the variations in image dimensions.
- For the first time, it is introduced to the utilization of NLP techniques for text cleaning and extraction in our cutting-edge research. These techniques prove invaluable in preserving and standardizing data while eliminating extraneous information that could introduce noise into the images, ultimately enhancing accuracy.
- An experimental evaluation of the proposed method on a public multiclass malware dataset to demonstrate its feasibility compared to existing literature.

This paper is organized as follows. Section 2 describes concepts related to the investigation. Section 3 shows an analysis of the state-of-the-art. Section 4 presents the proposed method. Section 5 provides details of the experimental evaluation and results. Section 6 contains the limitations, future work, and conclusions.

2. Related Concepts

This section describes the concepts related to this investigation, such as fuzzy hashing and NLP techniques used in the research.

2.1. Fuzzy Hashing

Fuzzy Hashing (FH), which is also called Context Triggered Piecewise Hashing ($CTPH$), is a combination of Cryptographic Hashes (CH), Rolling Hashes (RH), and Piecewise Hashes (PH). It can be perceived as $FH = CTPH = PH + RH$. Unlike traditional hashes, in which their hashes (checksum) can be seen more as right or wrong and as black or white, $CTPH$ is more like the gray hash type, as it can identify two files that may be near copies of one another that generally may not be located using traditional hashing methods. FH allows two arbitrary blobs of data to be compared for similarity based on common strings of binary data using a score percentage between 0 and 100, where 0 is low similarity, and 100 is high similarity [3].

To determine if two files are the same, algorithms such as MD5, SHA1, and SHA256, to name a few, are generally used. However, it is necessary to know how similar they are, not only if they are the same or different. *FH* algorithms such as SSDEEP or SDHASH can be used. The SSDEEP algorithm sequentially divides a file into equal groups of bytes, and on each of these groups, it calculates a hash. Then, a new hash is calculated to represent the entire file. The resulting hash can be used to compare the similarity with other files.

The SDHASH (similarity digest hash) method finds common and rare features in a file and matches the rare features in another file to determine the similarity between the two files [4]. Generally, a feature is a 64-byte string found using an entropy calculation [5]. It employs the cryptographic hash function SHA-1 and Bloom filters to calculate the SDHASH fuzzy hash value [6].

SDHASH is a robust algorithm for FH. This algorithm provides high accuracy compared to the predecessor SSDEEP. The algorithm can compute SDHASH with some options that generate different SDHASH lengths, and the results can be compared. Two or more SDHASH can be compared even if the length is different.

Bloom filters have predictable probabilistic properties, which allow for two filters to be directly compared using a Hamming distance-based measure $D(.)$. The result estimates the fraction of features the two filters have in common that are not due to chance. The maximum match among the second filters is found to compare two digests for each of the filters in the first digest. The resulting matches are then averaged [5].

Formally, the similarity distance $SD(F, G)$ for digests $F = f_1 f_2 \dots f_n$ and $G = g_1 g_2 \dots g_m$, $n < m$, is defined as:

$$SD(F, G) = \frac{1}{N} \sum_{i=1}^{n} \max_{j=1 \dots m} D(f_i, g_i), \quad (1)$$

An important point to consider is that directly estimating the empirical probability of encountering a 64-byte feature is not feasible, nor is it practical to store and retrieve such observations. SDHASH calculates a normalized Shannon entropy measure and assigns features to 1000 equivalence classes to address this. The statistics are gathered using this approximation method.

The similarity between two files has a threshold from 0 to 100, with 100 being the highest similarity detected.

The significance in the range is a confidence value that the two data objects have non-trivial amounts of commonality. *Strong* (range: 21–100) these are reliable results with very few false positives. *Marginal* (range: 11–20), the significance of resemblance comparisons in this range depends substantially on the underlying data. *Weak* (range: 1–10) are generally weak results; typically, most would be false positives. *Negative* (range: 0), the correlation between the targets is statistically comparable to that of two blobs of random data. *Unknown* (range: -1) is a rare occurrence for files above 4 KB unless they contain large regions of low-entropy data. However, in all the cases, the significance depends on the amount of data and the type of data [7].

The difference between SSDEEP and SDHASH is that the hash length is always the same; in SDHASH, the length depends on the input (amount of data). As mentioned, SDHASH is more robust, which yields the options that can be used at the compute time. For instance, the segment size by default is 128 MB, but the developer preferences define this setting.

2.2. APK File Structure

An Android Application Package (APK) file is a compiled application for the Android operating system. The package contains all the files needed for a single application and is organized in a particular structure. Figure 1 shows the APK file structure and a list of the most prominent files and directories:

- META-INF/: Directory with the APK metadata, such as its signature.

- lib/: Directory with compiled native libraries used by the app. The folder contains multiple directories, one for each supported CPU architecture (armeabi-v7a, x86, etc.).
- res/: Directory with the resources not compiled into *resources.arsc*. This directory contains all resources except the files in *res/values*. The resource files are in a binary XML format, and all image files are optimized (crunched) to save space and improve run-time performance when inflating these files.
- assets/: The directory with application assets can be retrieved by *AssetManager*.
- AndroidManifest.xml: Application manifest in the binary XML file format. It contains application metadata, for example, app name, version, permissions, minimum SDK version, etc.
- classes.dex: The classes are compiled in Java language that will be executed on the device by the virtual machine.
- resources.arsc: It contains all the metadata information about the resources. An ARSC file is an Android Resource table file that contains the list of the application resources in a table format.

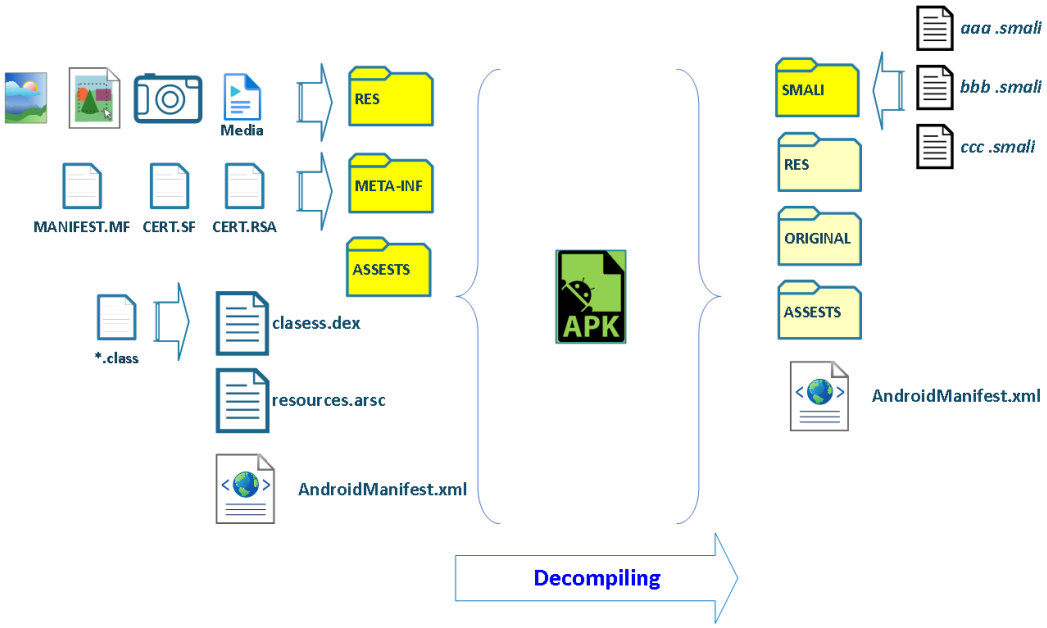

Figure 1. Android Application Package structure before and after the decompiling process. The diagram was generated based on the input and output generated by APKTool [8].

There are multiple tools to decompile an APK. An APKTool [8] was used in this research. In the decompiling process, multiple types of files are generated. All the APKs have Java code compiled into *classes.dex* file. On the other hand, the application properties are declared in AndroidManifest.xml, both compiled in binary format. Additional resources are included in the APK structure, such as images, app signature, and Cascading Style Sheets (CSS). Furthermore, the most essential part of the APK is the source code and the properties. Therefore, this investigation chose those essential files to analyze the application.

This investigation focuses on two types of files: ***smali*** files, which are generated after decompiling ***classes.dex***, and ***AndroidManifest.xml*** which contains the properties of the application in XML format.

2.3. Natural Language Processing Techniques

In NLP, many techniques can be used for text cleaning or extraction that are beneficial during machine learning training [9]. The following section will describe some techniques used in this investigation.

- ***Punctuation***: In NLP, punctuation refers to using marks or symbols in text analysis to help identify and structure sentences, paragraphs, and other text units. Punctuation

can be used as a feature in various NLP tasks, such as part-of-speech tagging, sentence boundary detection, and sentiment analysis. For example, in part-of-speech tagging, punctuation marks can be used as context clues to help determine the correct part of speech for a word. In sentence boundary detection, punctuation marks such as periods, question marks, and exclamation points can be used to identify the boundaries between sentences. In sentiment analysis, punctuation can be used as a feature to help identify the tone and emotion of a text.

- ***Tokenization***: It is the process of breaking down a text document or string of text into smaller units called "tokens". In NLP, these tokens usually correspond to words but can also be phrases, symbols, or individual characters. Tokenization is essential in many NLP tasks, such as text classification, sentiment analysis, and language translation.
- ***Lemmatization***: In NLP, lemmatization refers to the process of reducing a word to its base or dictionary form, known as the "lemma". In other words, it is converting words to their canonical form. For example, the lemma of the word "running" is "run", the lemma of "went" is "go", and the lemma of "better" is "good". The main goal of lemmatization is to reduce inflectional forms and sometimes derivationally related forms of a word to a common base form. Lemmatization is often used in NLP and text analysis tasks such as language translation, information retrieval, and sentiment analysis.
- ***Stemming***: It is the process of reducing a word to its root or stem form by removing any suffixes or prefixes. The resulting stem may not necessarily be a valid word in the language, but it still captures the essential meaning of the original word. For example, the stem of the word "jumping" is "jump", the stem of "cats" is "cat", and the stem of "happiness" is "happi". In this example, "happi" is not a valid English word, but it still captures the essential meaning of "happiness". Stemming is a simpler and faster approach to normalizing words compared to lemmatization, and it is often used in information retrieval.
- ***Keyword Extraction***: Term Frequency (TF) is a metric used in NLP and information retrieval to quantify the importance or frequency of a term within a document or a corpus. It measures how frequently a specific term appears in a document or text. TF is calculated by dividing the number of times a term occurs in a document by the total number of terms in that document. It is often normalized to prevent bias towards longer documents. One common normalization approach is to divide the raw term frequency by the maximum term frequency in the document, which results in a value between 0 and 1.

TF measures the frequency of a term within a document tf (term frequency). It indicates how often a term appears in a document relative to its total number of words. A higher TF value indicates a term is more significant within the document. Specifically, it is denoted by tf_{ij}, i.e., how often the word i appears in the document j [10].

3. Related Work

There have historically been two primary avenues in malware analysis: static analysis and dynamic analysis [11]. These two approaches diverge mainly in executing the malware sample, as dynamic analysis involves running it within a controlled environment. Subsequently, the features extracted from this execution are harnessed to train Machine Learning (ML) and Deep Learning (DL) algorithms, forming the foundation for constructing classification and detection models. These models are widely used for multi-platform malware analysis [12–14].

Image Visualization

This section reviews the works related to ML and DL for malware analysis using image visualization. The reviewed works have the approach of converting samples into images no matter the platform. Moreover, for comparison purposes, Android-related works are used.

Geremias et al. [15] presented a method for Android malware analysis that first extracts the features in feature vectors, and PCA was applied to generate a matrix of $M \times M$ size. Then, generate three grayscale images using different data types (API calls, OPCodes, and Dex). The final image is composed of the three images as layers to build a colored image (multi-view), and finally, the resulting images are used to train a CNN model. The method was evaluated using the CICMalDroid dataset, which contains 11,598 samples, and achieved an accuracy of 98.70%.

Kural et al. [16] presented a framework called Apk2Img4AndMal. The tool transforms the APK into a grayscale image without any preprocessing or reverse engineering process. No feature extraction from the static or dynamic analysis is needed. The images are generated by reading the APK as binary and transformed into a grayscale image. The images are analyzed using CNN, achieving an accuracy of up to 94.00%. The framework was tested with 24,588 Android malware and 3000 benign applications.

Jaiteg et al. [17] proposed a method that combines features from the APK file decompiled, looking for the optimal combination (Android manifest (AM), certificate (CR), classes.dex (CL), and resource (RS)), the handcrafted features were extracted from the image sections using multiple algorithms such as Gray Level Co-occurrence Matrix-based (GLCM), Global Image deScripTors (GIST), and Local Binary Pattern (LBP). The authors generated 15 sets of images using different combinations of files as input data for image generation. The resulting images were used to feed a CNN. The method was evaluated using the DREBIN dataset, which contains multiple classes. The method attained a high accuracy of 93.24% for malware image combination CR + AM using the Feature Fusion-SVM classifier.

Xu et al. [18] presented a method to analyze Android malware using DEX (bytecode) files. The DEX files were transformed into grayscale images using an interpolation algorithm to generate uniform-sized images. During the detection process, CNN was improved to extract and normalize the features using the GIST algorithm (lightGBM + LR) used to extract texture features. This article selects 5000 Android application software, including 2500 benign and 2500 malicious applications. The research attained a high accuracy of 98.7%.

Xiang et al. [19] investigated an Android malware detection model based on deep learning using autoencoders to detect malware. This paper aimed to study whether the autoencoders can reconstruct malware images with low loss and detect malware by judging the error value and reducing the risk of data confusion and redundant API injection (NOP no-operation instruction). They proposed using a neural network model to exclusively learn the features of malware instead of malware and benign features. Andro-dumpsys dataset was used, which contains 906 malicious binaries from 13 different malware families and 1776 benign files downloaded from the Google Play store. An accuracy of up to 93.00% was achieved.

Naït-Abdesselam et al. [20,21] proposed transforming the APK into an RGB leveraging the three channels to store different data on them (Green Channel: Conversion of Permissions and app components from AndroidManifest.xml, Red Channel: Conversion of API calls and unique opcode sequences from DEX file, Blue Channel: Conversion of protected strings, suspected permissions, app components, and API calls) image to use it in a CNN and ResNet for classification of malware. The method was evaluated using the AndroZoo dataset, which increases the number of samples across time. The authors chose n samples per time time-frame, and the method attained a detection accuracy of up to 99.37%.

Yong et al. [22] presented a new way to analyze Android malware based on DEX files. They proposed converting DEX files into RGB images, then applied text and color features. Also, from the DEX file, plain text was filtered to obtain text features, the GIST was used to get texture features, and the color moments were used to feed multiple kernels as input data for malware classification. The resulting images showed that samples from the same family have similar colors and textures. The authors used a Support Vector Machine (SVM) in the classification phase, applying multiple kernels for testing. AMD dataset contains

24,553 samples, categorized into 135 varieties among 71 malware families ranging from 2010 to 2016, with a classification accuracy of up to 96.00%.

Peng et al. [23] showed a method for Android malware classification based on the traffic generated (PCAP). The traffic was filtered by removing third-party traffic, and the resulting flows (malicious traffic) were split into sessions. The first part of the session is used to generate a grayscale image representing the traffic characteristics for each session. Finally, the images were analyzed using a deep learning model (1.5D-CNN). CICAndMal2017 dataset, which contains over 1700 benign and 400 malware samples. The model proposed achieved an accuracy of up to 98.5%.

Jianguo et al. [24] focused on Android malware analysis. The authors transformed the APK data into nodes and edges combining features from static and dynamic analysis (for instance: code region–invoke–sensitive API, sensitive API–invoked by–code region, code region–belongs to–package, package–contains–code region, code region–included in–signature MD5, signature md5–includes–code region). This transformation is called a heterogeneous graph, and the graph is represented into a matrix to feed the HG-CNN classifier. The dataset was collected from diverse sources using a known dataset like Drebin. With 11,423 benign and 14,546 malware samples, the authors achieved an accuracy of over 97%.

Yajamanam et al. [25] showed a method for Windows malware analysis, transforming the sample into a grayscale image. The samples were classified based on their GIST features. Unlike the previous related work, the researchers tested the robustness of the GIST features by adding noise to the images. Based on the results, as expected, the accuracy of classifying images with noise decreased compared to the images without noise. A couple of datasets were used in this research: The Malimg dataset consists of more than 9000 malware samples belonging to 25 families, and the Malicia dataset contains 11,363 malware samples, primarily composed of 3 types of malware.

Huang et al. [26] presented work for robust hashing, treating the samples as two-dimensional images. The authors performed multiple tests to compare SVM with robust hashing techniques. They found that some classes are correctly classified by robust hashing, and the results can be comparable with SVM. Malimg dataset was used in this investigation, which consists of more than 9000 malware samples belonging to 25 families.

Yang et al. [27] presented an algorithm that transforms the APK into a grayscale image based on Portable Executable (PE) file format. The images have APK data (CERT.RSA, AndroidManifest.xml, resources.arsc, and classes.dex) converted directly without preprocessing. In this investigation, the researchers used the Drebin corpus, which contains 5560 samples from 178 classes. The images were used in different machine-learning algorithms for testing. Random Decision Forest (RDF) is the algorithm with the highest results, reaching up to 95.51% in accuracy.

Tingting et al. [28] in their research proposed a binary malware detection by converting opcode sequences into images in combination with PCA to extract the features, and then, SVM was used for the malware classification applying multiple kernel functions to increase the accuracy. The highest accuracy achieved was 97.62% using SVM and RBF kernel function. The dataset used contains 9168 malware samples and 8640 benign programs.

Ajit et al. [29] showed a method without feature extraction, decompiling, preprocessing, and static or dynamic analysis output. They converted the APK sample directly into four image formats (grayscale, RGB, CMYK, HSL) to test which format worked better with machine learning algorithms (Decision Tree, Random Forest, KNN) for the classification task. They proved that using grayscale achieved the highest accuracy of 91.00% using Random Forest.

On the other hand, other researchers have proposed Android malware analysis based on static features such as permission [30]. Still, without transforming the features into image format, the permissions were represented as a vector in a binary sequence, and the analysis was based on a feature vector, training a machine learning model. The accuracy was 96%.

The works listed presented malware detection and classification methods by transforming the samples into images. The proposed method highlights transforming the sample into a grayscale image. Unlike the works reviewed, a new way to convert the sample into an image was presented. The image is composed of the fuzzy hashes generated by the decompiled code and preprocessed utilizing NLP techniques for text cleaning and extraction, which reduces the image size and the time-consuming of the CNN. A summary of the related work is presented in Table 1.

Table 1. Summary of the related work.

Ref./Year	Samples	Type of Images	CNN	Other Algorithm	Other Features	Accuracy
[15], 2022	APK	RGB	Yes	PCA	Multi-view image	98.70%
[16], 2021	APK	Grayscale	Yes	-	APK read as binary	94.00%—CNN
[17], 2021	APK	Grayscale	Yes	GLCM, GIST LBP	Feature fusion	93.24%—SVM
[18], 2021	APK	Grayscale	Yes	GIST	dex file read as binary	98.70%—GIST
[19], 2020	APK	Grayscale	Yes	Auto-enconders	Reconstruction error	93.00%
[20], 2020	APK	RGB	Yes	ResNet	Decompiled data to RGB	99.37%—ResNet
[24], 2020	APK	RGB	Yes	SVM, KNN, RF, HS-resNet,	Static and dynamic features	97.00%
[23], 2020	APK	Grayscale	Yes	1.5D-CNN	Network traffic as image	98.50%
[22], 2020	APK	RGB	No	SVM, GIST, Multiple Kernels	dex as image and plain text	96.00%—SVM
[21], 2020	APK	RGB	Yes	ResNet	Decompiled data to RGB	99.37%—ResNet

4. Proposed Method

This paper aims to study the feasibility of utilizing a deep learning model based on a CNN architecture for malware classification by converting a sample into a grayscale image composed of fuzzy hashes derived from decompiled and preprocessed code. Therefore, our malware classification method is divided into two stages: data preprocessing (APK to a grayscale image) and malware classification (multiple families). Figure 2 shows the proposal workflow at a high level, and the architecture is detailed in Figure 3.

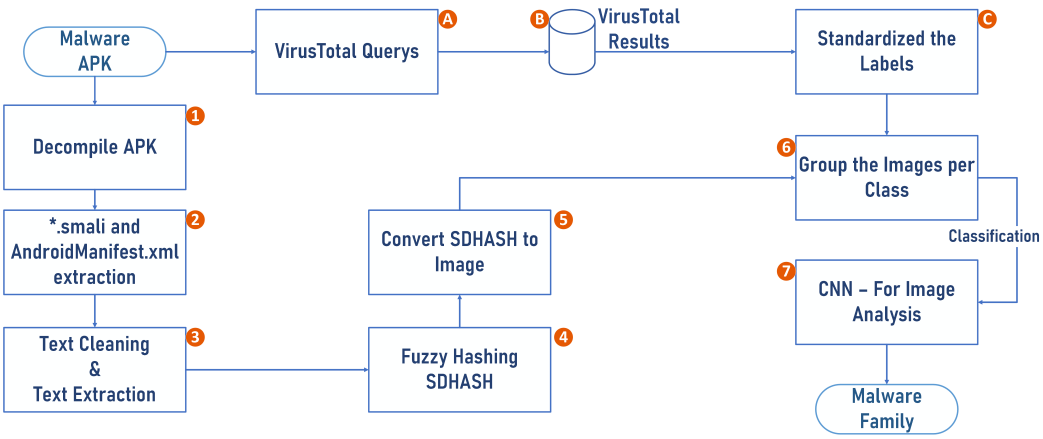

Figure 2. Workflow for APKs analysis of the proposed method.

First, it performs a novel transformation of the APK file into a lightweight grayscale image utilizing a fuzzy hashing of the decompiled smali code and manifest file. Secondly, it trains a CNN model on the obtained images for malware family classification. Furthermore, VirusTotal [31] is used to label the corpus, as described in Section 5.2.

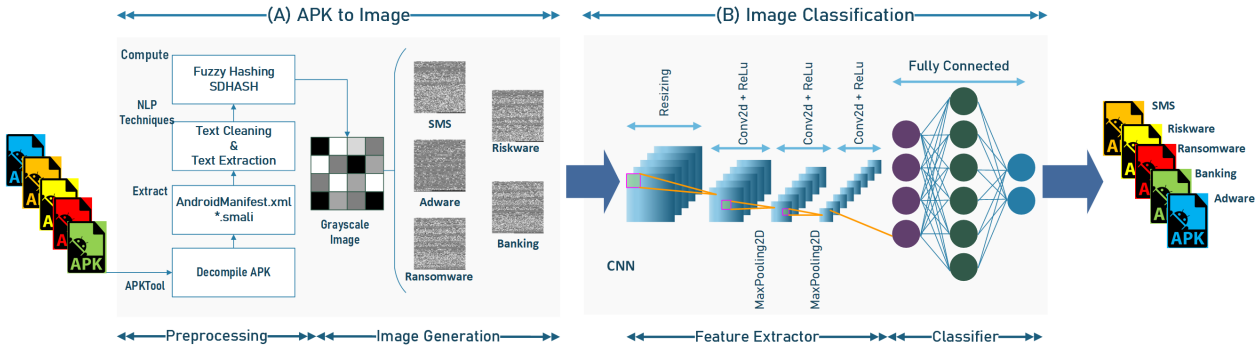

Figure 3. Proposed architecture: (**A**) APK to image conversion. This step transforms the APK into a grayscale image. (**B**) Grayscale images are analyzed in CNN for Android malware classification.

4.1. Android Binary to Visual Representation

The scope of this phase is to transform Android binary into image data that the classification model can handle, as shown in Figure 3.

The first step is decompiling the APK, as mentioned in Section 2.2; during this process, multiple files are generated (An APK has n smali files and one AndroidManifest.xml). At this point, only smali files and AndroidManifest.xml are selected. It is feasible to determine the malware family by analyzing the smali code and the application properties file.

The second step consists of preprocessing the smali files generated by each APK using NLP techniques for text cleaning to remove useless information by the Equation (2)

$$TC_{smali} = [PR, TK, ST, LM], \tag{2}$$

where Text Cleaning (TC) is a sequence of the Punctuation Removal (PR), Tokenization (TK), Stemming (ST), and Lemmatization (LM) techniques applied to smali files.

The AndroidManifest.xml is also preprocessed in this stage. Furthermore, unlike the smali file, from the manifest file, helpful information is extracted, such as components of the app (which include all activities, services, broadcast receivers, and content providers), permissions, and hardware and software features the app requires discarding XML tags [32], by the Equation (3)

$$TE_{AndroidManifest} = [PR, TK, IR], \tag{3}$$

where Text Extraction (TE) is a sequence of the Punctuation Removal (PR), Tokenization (TK), and Information Retrieval (IR) techniques applied to the AndroidManifest.xml file. For information retrieval, Term Frequency (TF) was applied.

In contrast to prior studies, the importance of incorporating NLP techniques in this research stems from their capacity to preprocess decompiled data by eliminating useless information and ensuring data standardization. This plays an essential role as the fuzzy hashing technique measures similarity, and standardized data enhances these similarities. Moreover, it prevents the incorporation of noise into the images and facilitates a reduction in image size, ultimately proving advantageous during training.

As part of the second stage, fuzzy hashing is computed from the smali and AndroidManifest.xml files, using the same amount of data to obtain the same hash length in the output. The size of input data (grayscale) is a matrix of $N \times M$, where N is the length of SDHASH (fuzzy hash), which is 344 (344 pixels), and M, depends on the number of fuzzy hashes plus AndriodManifest data. Defining a standard image size for every APK is impossible because each is unique. The resize is performed into the CNN. The grayscale image structure has AndroidManifest data at the top and all the fuzzy hashes after. Each malware sample has n fuzzy hashes at this stage, as shown in Figure 4, a graphical description of how a fuzzy hash is computed. The SDHASH is a string encoded in Base64; each byte of the fuzzy hash is converted to a scale from 0 to 255, corresponding to one pixel in the

grayscale image. Figure 5 shows three grayscale images generated by applying SDHASH to different input data obtained from the same APK sample.

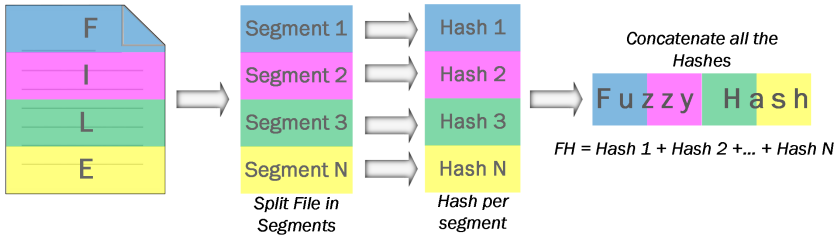

Figure 4. SDHASH fuzzy hashing description for the compute stage. Source: The image was generated based on the research from Naik [33].

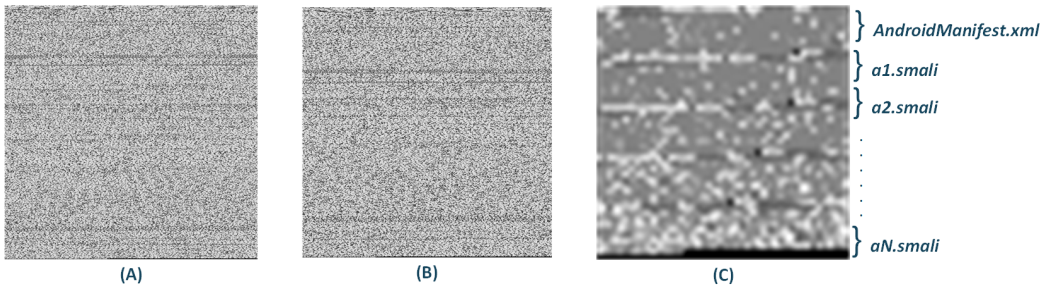

Figure 5. Android sample MD5 "0A0C881A01EA942DC052CA560198D82B" converted in a grayscale image using three different input data after decompiled sample: (**A**) Image created after decompiling process without any preprocessing, (**B**) Image created after preprocessing decompiled code using NLP, (**C**) Image created using fuzzy hashing technique after preprocessing with NLP.

4.2. Convolutional Neural Network

An image itself has specific properties. CNN models are widely explored for image analysis. In the architecture depicted in Figure 3, the APK is transformed into a grayscale image at the first stage. CNN is suitable for image classification at the second stage.

The CNN model was built with three convolution layers, three pooling layers, a ReLU activation function, an Adam optimizer, and varying the epoch values in the experimental phase to increase accuracy at the learning stage Figure 6 describes the CNN model summary.

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 rescaling_1 (Rescaling)     (None, 224, 224, 3)       0

 conv2d (Conv2D)             (None, 224, 224, 16)      448

 max_pooling2d (MaxPooling2D  (None, 112, 112, 16)     0
 )

 conv2d_1 (Conv2D)           (None, 112, 112, 32)      4640

 max_pooling2d_1 (MaxPooling  (None, 56, 56, 32)       0
 2D)

 conv2d_2 (Conv2D)           (None, 56, 56, 64)        18,496

 max_pooling2d_2 (MaxPooling  (None, 28, 28, 64)       0
 2D)

 flatten (Flatten)           (None, 50176)             0

 dense (Dense)               (None, 128)               6,422,656

 dense_1 (Dense)             (None, 4)                 516

=================================================================
Total params: 6,446,756
Trainable params: 6,446,756
Non-trainable params: 0
_________________________________________________________________
```

Figure 6. CNN model summary.

This CNN model takes an input grayscale image, processes it through convolutional and pooling layers to extract features, flattens these features, and then passes them through fully connected layers for classification. The number of filters, filter size, activation functions, and units in the dense layers were adjusted for this research.

- Rescaling: The layer scales the input pixel values to the range $[0, 1]$.
- Conv2D: This is a 2D convolutional layer with 16 filters of size 3×3. It applies convolution to the input image, preserving the input shape with a rectified linear unit (ReLU) activation function. This layer extracts various features from the image.
- MaxPooling2D: After each convolution, a max-pooling layer downsamples the output. It reduces the spatial dimensions, helping the model focus on the most important features.
- Conv2D: This is a 2D convolutional layer with 32 filters of size 3×3. It applies convolution to the input image, preserving the input shape with a rectified linear unit (ReLU) activation function. This layer extracts various features from the image.
- MaxPooling2D: After each convolution, a max-pooling layer downsamples the output. It reduces the spatial dimensions, helping the model focus on the most important features.
- Conv2D: This is a 2D convolutional layer with 64 filters of size 3×3. It applies convolution to the input image, preserving the input shape with a rectified linear unit (ReLU) activation function. This layer extracts various features from the image.
- MaxPooling2D: After each convolution, a max-pooling layer downsamples the output. It reduces the spatial dimensions, helping the model focus on the most important features.
- Flatten: This layer flattens the output from the previous layers. It converts the 2D feature maps into a 1D vector.
- Dense: A fully connected (dense) layer with 128 units and ReLU activation. This layer learns complex relationships between the extracted features.
- Dense: The final dense layer with the number of units equal to the number of classes in your classification problem. It provides the output of the model, which is used for classification.

The CNN model splits the corpus using 80% of the data as the training set (K-fold validation was used to split the training dataset. In this case, $K = 10$, 10-fold validation), and 20% of the data was selected as the validation set. The corpus description is presented in Section 5.2.

5. Experiments

This section aims to validate the proposed malware classification approach using a CNN model. The experiments test the hypothesis that a deep learning model can effectively classify malware samples into their respective families using grayscale images generated through fuzzy hashing. This section covers the experimental setup, dataset description, evaluation, and results.

5.1. Experimental Setup

As described, the method was tested on a ransomware [34] and CICAndMal2017 [35,36] corpus (15,493 samples of multiple Android malware families). The algorithm was programmed in Python 3 (Jupyter Notebook), Shell-Scripting, and running Linux on DELL XPS 15 9550 natively (Ubuntu 18.02, CPU Intel(R) Core(TM) i7-6700HQ CPU @ 2.60 GHz, 8 Core Processors, 32GB RAM, 500 GB SSD, and GPU NVIDIA GeForce GTX 960M).

5.2. Corpus Description

The corpus (dataset) contains 2288 Android ransomware samples shared by Wuhan University for research purposes [34], and 13,205 samples of Adware, Banking, Riskware and SMS from CICMalDroid 2020 [35,36]. Table 2 shows the corpus distribution.

Table 2. Malware dataset distribution.

Malware Type	Reference	No. Samples	Percentage	No. Classes
Adware	[35,36]	1515	9.78%	1
Banking	[35,36]	2506	16.18%	1
Ransomware	[34]	2288	14.77%	1
Riskware	[35,36]	4362	28.15%	1
SMS	[35,36]	4822	31.12%	1

The dataset was labeled with the usage of VirusTotal [31] considering the results from the most popular antivirus. For instance, the sample with MD5 hash "1F6D3E6A3F3186D5 FD23E937B159B922" has multiple labels, per the VirusTotal report, as shown in Table 3.

Table 3. Sample MD5 "1F6D3E6A3F3186D5FD23E937B159B922", partial results from VirusTotal.

AntiVirus	Malware
AegisLab	Trojan.AndroidOS.Generic.C!c
AhnLab-V3	Trojan/Android.WipeLocker.33939
Alibaba	Trojan:Android/Soceng.408e621d
ESET-NOD32	A Variant Of Android/Wipelock.F
F-Secure	Undetected
Kaspersky	HEUR:Trojan.AndroidOS.Soceng.f
MaxSecure	Undetected
McAfee-GW-Edition	Artemis!Trojan
Symantec Mobile Insight	Trojan:Habey
BitDefender	Undetected

The reports were standardized for the remaining samples since each antivirus uses its labels, some of which are similar, while others are quite distinct. Additionally, one antivirus lacks labels for malware families. In essence, the number of samples matches the number of families. Table 4 shows the top Android malware family distribution after standardized per class.

Table 4. Malware dataset labels standardized from VirusTotal results (top twenty).

No. Class	Class Name	No. Samples	Percentage (%)
1	Trojan_FakeInst	2991	19.39
2	Trojan_Opfake	1168	7.57
3	Trojan_ASMalwAD	1097	7.11
4	Ransomware_LockScreen	739	4.79
5	AdWare_SMSreg	711	4.61
6	Trojan_SmsSend	706	4.58
7	Ransomware_Locker	704	4.56
8	Trojan_Bankun	672	4.36
9	PUP_Wapsx	439	2.85
10	Trojan_Simplelock	437	2.83
11	TrojanDropper_SmsPay	414	2.68
12	PUP_Dowgin	372	2.41
13	Trojan_HiddenApp	353	2.29
14	Trojan_FakeApp	329	2.13
15	RiskWare_Dnotua	304	1.97
16	Trojan_Koler	263	1.70
17	Trojan_Spitmo	226	1.46
18	Trojan_Rootnik	205	1.33
19	Trojan_SMSstealer	199	1.29
20	Trojan_Marcher	187	1.21

5.3. Classification Accuracy

The accuracy of the method was evaluated using F1-scores for each ransomware family, whose formula requires true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN) to measure the effectiveness of the proposed method (4)–(7).

$$Precision = \frac{TP}{TP + FP}, \tag{4}$$

$$Recall = \frac{TP}{TP + FN}, \tag{5}$$

$$Accuracy = \frac{TN + TP}{TP + FP + TN + FN}, \tag{6}$$

$$F1 = \frac{2 \times precision \times recall}{precision + recall}. \tag{7}$$

A confusion matrix (CM) is a graphical representation to draw the performance of a classification model. The CM uses TP, FP, TN, and FN; hence, they can be used to calculate precision, recall, and accuracy metrics per class as well as global metrics (4)–(7).

5.4. Experimental Results

This section describes the tests performed to demonstrate the feasibility of Android malware classification using fuzzy hashing converted to a grayscale image in combination with the CNN model.

The first stage of the proposal consists of transforming the APK into grayscale images. In the image analysis field, there exists a problem that must be adequately addressed, which is the image size. In this case, as the malware samples are different, they have different sizes. Table 5 shows the distribution using three different input data after generating the grayscale images. The image size was reduced using the fuzzy hashing approach compared to the other two input data.

Table 5. Image size after converted in a grayscale format using three different input data: (a) SDBF—Image generated using fuzzy hashing technique after preprocessing with NLP, (b) Src NLP—Image created after preprocessing decompiled code using NLP, (c) All Src—Image generated after decompiling process without preprocessing.

File Size	(a)-SDBF	(b)-Src NLP	(c)-All Src
0 KB–20 KB	9041	4515	4593
20 KB–40 KB	1129	3389	3106
40 KB–60 KB	533	1180	1048
60 KB–80 KB	508	575	897
80 KB–100 KB	592	997	226
101 KB–500 KB	3158	2676	3395
500 KB–1 MB	304	999	1177
1 MB–2 MB	220	764	634
2 MB–3 MB	4	186	183
3 MB–4 MB	4	73	92
>4 MB	0	139	142
Total	**15,493**	**15,493**	**15,493**

The CNN was evaluated using three types of images generated to prove that the fuzzy hashing approach has an accuracy higher than the other types of images, reducing the time-consuming. Section 4.2 described the CNN settings as part of the second stage. Table 6 summarizes the twelve scenarios tested. The parameters varied during the evaluation of CNN to find the optimal values.

Table 6. Metrics summary of the tests performed using five and twenty classes, each experiment was executed ten times, and the results represent the average for the two epochs with higher accuracy during the training process.

No. Test	Image Type	Precision	Recall	F1-Score	Accuracy	No. Classes	Epochs	Time
1	Fuzzy Hashing	0.84	0.84	0.84	84.67	5	100	10:43
2	Fuzzy Hashing	0.86	0.85	0.86	**85.23**	5	70	07:24
3	Source Code—NLP	0.82	0.82	0.83	83.24	5	100	14:12
4	Source Code—NLP	0.85	0.83	0.85	84.18	5	70	11:27
5	All Source Code	0.81	0.83	0.82	82.36	5	100	18:38
6	All Source Code	0.70	0.23	0.81	81.12	5	70	16:19
7	Fuzzy Hashing	0.97	0.98	0.98	**97.62**	20	100	11:47
8	Fuzzy Hashing	0.97	0.95	0.97	96.83	20	70	09:43
9	Source Code—NLP	0.96	0.96	0.96	96.15	20	100	16:41
10	Source Code—NLP	0.96	0.97	0.97	96.42	20	70	13:29
11	All Source Code	0.95	0.96	0.96	95.88	20	100	20:15
12	All Source Code	0.95	0.95	0.95	95.25	20	70	17:22

The experiments were performed in two ways: the first was using the five types of malware (adware, banking, ransomware, riskware, and SMS) utilizing these labels, and the other way was using the top classes of each type of malware after label standardization. Based on the distribution shown in Table 4, the top twenty of the classes hold 81.12% (12,516 samples) of the dataset (15,493). Meanwhile, the other 195 classes hold 18.88% (2778 samples) of the dataset.

The experimental results presented in Table 6 summarize the metrics of the CNN model using the different types of images and CNN parameters. Each experiment was executed ten times for each stage (training and validation), and the results represent the average. For instance, using the whole data for five classes, the accuracy was 85.23%, and 97.62% using the most representative twenty classes. There was an increase in the model accuracy by leaving out non-significant classes from the dataset from the standardized labels. In both cases, with the dataset of fuzzy hashing images.

Figure 7 describes training and validation curves for loss and accuracy for the best test results shown in Table 6. The curves represent one execution using five and twenty classes. The learning curves represent a compelling performance in classifying Android malware.

To assess the accuracy of the method, the actual and predicted labels were compared using a confusion matrix (CM), a commonly utilized metric for evaluating the performance of classification models. Figure 8 presents the confusion matrix for the test using only five classes, showing poor accuracy of less than 90%. This suggests that the corpus was not correctly labeled and can improve the accuracy of the model using appropriate family labels.

Figure 9 shows the evaluation results using the twenty classes. It is noted that the twenty classes with the most samples achieved good accuracy after labeling utilizing VirusTotal, which is expected as larger samples usually result in better accuracy. However, the test using the classes without label validation impacted the accuracy of the model. This indicates that the performance of the model is affected by the imbalanced classes and incorrect labels.

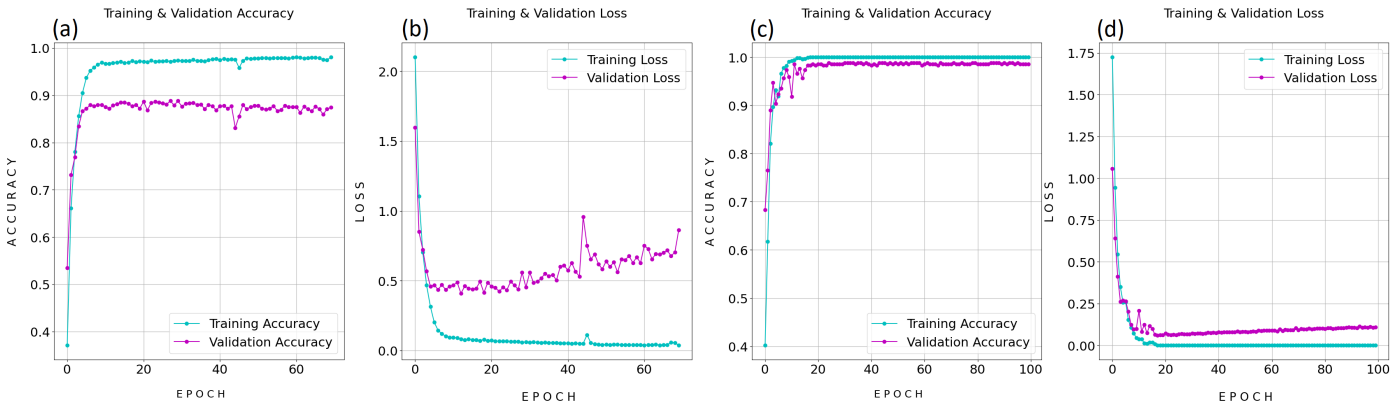

Figure 7. Experimental metrics using grayscale images based on fuzzy hashing, the graphs show the accuracy and loss during training and evaluation. (**a**,**b**) CNN training and validation results using five classes with 70 epochs, accuracy 85.51% (**c**,**d**) CNN training and validation results using the top twenty classes in the dataset with 100 epochs, accuracy 98.24%. Source: It was obtained through Python execution of the CNN model using the *seabron* library.

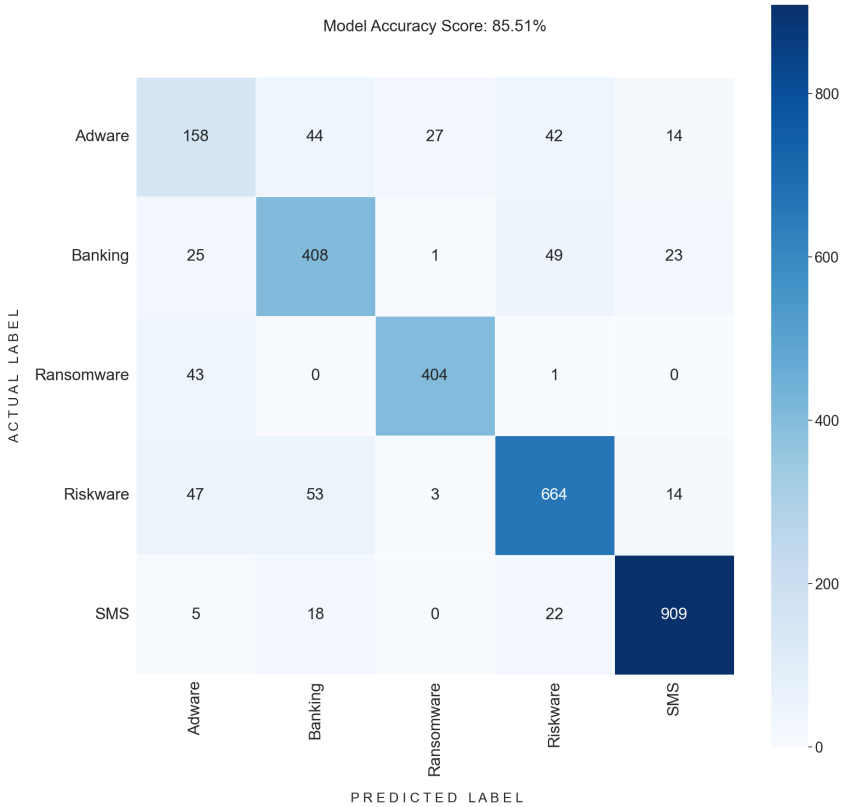

Figure 8. Confusion matrix for the five classes. Source: It was obtained through Python execution of the CNN model using the *seabron* library.

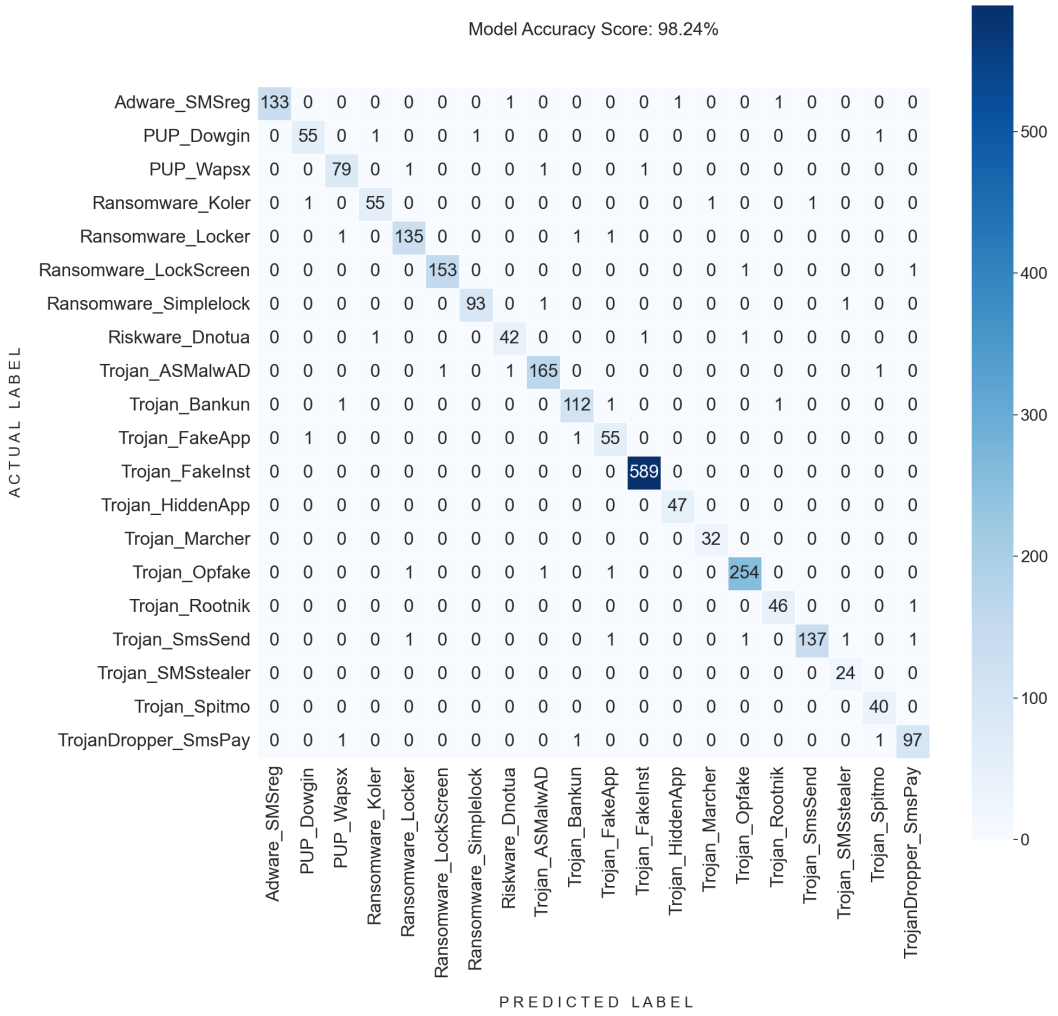

Figure 9. Confusion matrix for the twenty classes. Source: It was obtained through Python execution of the CNN model using the *seabron* library.

The metrics per class are described in Tables 7 and 8 for the five and twenty classes, respectively, for one of the ten executions that achieve the highest accuracy. highest

Table 7. Summary of metrics for the test performed with the five classes. The metrics represent the highest accuracy achieved for one of the ten executions, with the five classes trained for 70 epochs. Source: Output of the CNN model for validation data.

Class	Precision	Recall	F1-Score	Support
Adware	0.5683	0.5544	0.5613	285
Banking	0.7801	0.8063	0.7930	506
Ransomware	0.9287	0.9018	0.9151	448
Riskware	0.8535	0.8502	0.8518	781
SMS	0.9469	0.9528	0.9498	954
Accuracy			0.8551	2974
Macro avg	0.8551	0.8131	0.8142	2974
Weighted avg	0.8550	0.8551	0.8549	2974

Table 8. Summary of metrics for the test performed with the twenty classes. The metrics represent the highest accuracy for one of the ten executions, with the twenty classes trained for 100 epochs. Source: Output of the CNN model for validation data.

Class	Precision	Recall	F1-Score	Support
Adware_SMSreg	1.0000	0.9779	0.9888	136
PUP_Dowgin	0.9649	0.9483	0.9565	58
PUP_Wapsx	0.9634	0.9634	0.9634	82
Ransomware_Koler	0.9649	0.9483	0.9565	58
Ransomware_Locker	0.9783	0.9783	0.9783	138
Ransomware_LockScreen	0.9935	0.9871	0.9903	155
Ransomware_Simplelock	0.9894	0.9789	0.9841	95
Riskware_Dnotua	0.9545	0.9333	0.9438	45
Trojan_ASMalwAD	0.9821	0.9821	0.9821	168
Trojan_Bankun	0.9739	0.9739	0.9739	115
Trojan_FakeApp	0.9322	0.9649	0.9483	57
Trojan_FakeInst	0.9966	1.0000	0.9983	589
Trojan_HiddenApp	0.9792	1.0000	0.9895	47
Trojan_Marcher	0.9697	1.0000	0.9846	32
Trojan_Opfake	0.9883	0.9883	0.9883	257
Trojan_Rootnik	0.9583	0.9787	0.9684	47
Trojan_SmsSend	0.9928	0.9648	0.9786	142
Trojan_SMSstealer	0.9231	1.0000	0.9600	24
Trojan_Spitmo	0.9302	1.0000	0.9639	40
TrojanDropper_SmsPay	0.9700	0.9700	0.9700	100
accuracy			0.9824	2385
macro avg	0.9703	0.9769	0.9734	2385
weighted avg	0.9826	0.9824	0.9824	2385

There is a misclassification issue based on the results shown in the confusion matrix. Resuming the scope of this research, the proposed model uses images composed of fuzzy hashing, which allows us to measure the similarities. The misclassification is related to wrong labeling, as it was noticed in the experiments using the five types of malware compared with the top twenty classes labeled utilizing VirusTotal. The samples misclassified share more similarities with other classes than the ones assigned by the antivirus.

The tests showed that the images of fuzzy hashes achieved a good performance based on the extracted features; one reason is that the image size was reduced by applying NLP and fuzzy hashing techniques, eliminating useless data. Consequently, CNN classified the images faster than those containing more data. The preprocessing applied was beneficial for increasing the accuracy and reducing the time-consuming of the CNN.

6. Discussion

In their comprehensive review of the state-of-the-art Android malware classification, the researchers employed a range of algorithms. Notably, Support Vector Machines (SVM), k-Nearest Neighbors (KNN), and Random Forest (RF) stand out as some of the prevalent algorithms frequently employed in image classification tasks.

Deep learning and machine learning algorithms were tested in our evaluation stage using different data types. The tests were designed using images and the decompiled data (smali and AndroidManifest code) for a reasonable comparison.

For image-based classification, a deep learning model based on CNN was trained. As discussed in the document, the primary approach of this research is image-based malware classification using CNN. The CNN model achieved up to 98.24% in accuracy metric and an average of 97.62% (10 executions), such as it is expected that a CNN works well with images. Additional neural network models were evaluated using the images generated in those tests ResNet50 and VGG16, achieving 96.67% and 95.88% during the validation executed ten times, respectively. The three CNN models achieved almost the same accuracy during the evaluation phase.

On the other hand, an additional machine learning model was trained with the same images, and the Support Vector Machine (SVM) achieved an average of 94.91%. The k-NN classification scheme was tested. However, no explicit training phase is required, as the classification is computed based solely on the nearest neighbor in the training set [25]. K-nearest neighbor (k-NN) algorithm, with $k = 1$ using the images generated, achieved an accuracy of 78.19%. The comparison shows an increase in the accuracy metric of CNN over SVM and k-NN using the same input data (grayscale images).

Other tests were performed using the decompiled data (smali and AndroidManifest) in machine learning models such as k-NN, Random Forest (RF), Multilayer Perceptron (MLP), and SVM. In NLP, a text is a document that can be used for text classification. The documents were used in the classifiers mentioned to classify the malware. The accuracy attained using k-NN, RF, MLP, and SVM was 93.3%, 92.18%, 93.68%, and 95.42%, respectively.

In Section 3, Table 1. shows related studies for Android malware analysis considered for comparison purposes. Therefore, Table 9 shows the results obtained with the ML and DL algorithms tested using the data generated in the research versus the works documented in the state-of-the-art.

Table 9. Comparison: proposal vs. state-of-the-art. It includes the tests performed using ML and DL algorithms with images and text.

Algorithm	Data	Training (%)	Validation (%)	SD (σ)
Our CNN	Images	97.96	97.62	0.51
ResNet50	Images	97.09	96.67	2.67
VGG16	Images	95.86	95.88	1.29
KNN	Images	78.19	–	–
SVM	Images	95.17	94.91	2.17
KNN	Code/Text	93.30	–	–
RF	Code/Tex	92.18	88.45	5.03
MLP	Code/Tex	93.68	92.40	4.01
SVM	Code/Tex	93.87	95.42	1.28
CNN [16]	Images	94.00	–	–
SVM [17]	Images	93.24	–	–
SVM [24]	Images	97.00	–	–
SVM [22]	Images	96.00	–	–

For the comparative tests, models requiring training and validation steps (CNN, SVM, RF, MLP) were executed ten times, while k-NN was run once. The additional tests reaffirmed the efficacy of the CNN model with images, achieving a remarkable accuracy of up to 98.08% and an average of 97.84%. This superior performance surpasses other classifiers and underscores the accuracy of the fuzzy hash-based image generation method employed.

In contrast, k-NN yielded an accuracy of less than 80%, while SVM achieved over 90% using the same images despite not being optimized for image classification. Furthermore, the algorithms designed for text classification exhibited improved accuracy compared to SVM and k-NN when applied to images, albeit falling short of the results attained with CNN.

The comparison reveals an increase in the accuracy metric of CNN compared to SVM and KNN using the same input data. Regarding the related works, the results indicate an improvement in the classification task using the fuzzy hashing approach by converting fuzzy hashing into grayscale images.

Once the CNN is trained, it takes to classify a new sample is an average of 40 s. The decompiling process is major time-consuming, which takes 50% (20 s); text cleaning and text extraction take 10% (4 s). Data to grayscale image 10% (4 s). And CNN for classification 30% (12 s).

7. Conclusions

This research has introduced a novel mechanism for Android malware classification, transforming malware files into image representation and utilizing Convolutional Neural Network (CNN) to distinguish between various types of malware.

The proposed method in the first stage transforms the APK into a grayscale image composed of hashes and innovatively leverages the sdhash fuzzy hashing technique to represent an APK in similarity hashes. In the preprocessing phase, NLP techniques for text cleaning and extraction, such as *Punctuation Removal, Tokenization, Stemming, Lemmatization, and Information Retrieval*, were used to standardize the data, reducing the image size. The second stage of the method analyzes Android malware analysis based on the images generated; the preprocessing was beneficial because it reduced the time-consuming and, most importantly, increased the accuracy, demonstrating efficacy in classifying multi-class malware.

This study investigated the feasibility of identifying Android malware families by focusing on features extracted from visual representations. Experimental assessments were conducted across various Android malware families, illustrating the utility and reliability of the proposed methodology. Notably, when tested against diverse malware families, the algorithm exhibited an average classification accuracy of up to 98.24% across twenty representative classes.

In conclusion, this research contributes to the malware classification approach, showcasing its potential to advance the field of cybersecurity. The method demonstrated a good performance, particularly in distinguishing diverse malware families, underscoring its practicality and effectiveness in real-world applications.

7.1. Limitations and Future Work

7.1.1. Available Datasets

The model was tested on a single dataset due to the limited availability of open datasets containing APKs. In future work, we intend to apply the model to more complex datasets.

7.1.2. Labeling Dataset

The availability of open datasets sometimes presents challenges related to labeling accuracy, which can adversely affect classification performance. In such cases, the features may become mixed with other classes, leading to confusion for the algorithm. Future work aims to enhance classification by incorporating similarity comparisons to define classes without relying on VirusTotal intervention.

7.1.3. Broken Samples

The method is functional with decompiled data as long as the samples contain at least classes.dex and AndroidManifest.xml. If these essential files are missing, the method may not function correctly. Conversely, it's important to note that the method has limitations; it cannot process corrupted or broken samples.

7.1.4. Imbalanced Classes

In the real world, there is not the same amount of malware samples as benign ones. Therefore, it is challenging to have a balanced dataset with the same number of members per family. Imbalanced classes impact the CNN performance, reducing the accuracy as future work performs hunting to get more samples to balance the classes.

7.1.5. Multiplatform

The model proposed demonstrated a high accuracy for Android malware classification. As future work, the model can be extended for multiplatform malware analysis, which involves Microsoft, IoT, Mobile, and Linux malware.

Author Contributions: These authors contributed equally to this work. All authors have read and agreed to the published version of the manuscript.

Funding: The work was done with partial support from the Mexican Government through the grant A1-S-47854 of CONAHCYT, Mexico, grants SIP-20230990, and SIP-20232782 of the Secretaría de Investigación y Posgrado of the Instituto Politécnico Nacional, Mexico.

Data Availability Statement: Not applicable.

Acknowledgments: We thank Wuhan University for sharing with us the ransomware dataset used for research purposes. We thank Lorenzo Cavallaro and Feargus Pendlebury (Systems Security Research Lab, King's College London) for generously analyzing a large number of Android APKs in CopperDroid.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

APK	Android Application Package
CNN	Recursive Convolutional Neural Network
DT	Decision Tree
GIST	Global Image Descriptors
GLCM	Gray Level Co-occurrence Matrix-based
KNN	k-Nearest Neighbors
LBP	Local Binary Pattern
NLP	Natural Language Processing
PCA	Principal Component Analysis
RF	Random Forest
SFC	Space-Filling Curves
SVM	Support Vector Machine
SDHASH	Similarity Digest Hash
SDBF	Similarity Digest Bloom Filters

References

1. Google. Secure an Android Device | Android Open Source Project. Available online: https://source.android.com/docs/security/overview (accessed on 9 September 2023).
2. It Threat Evolution in q3 2022. Mobile Statistics | Securelist. Available online: https://securelist.com/it-threat-evolution-in-q3-2022-mobile-statistics/107978/ (accessed on 9 September 2023).
3. Sarantinos, N.; Benzaïd, C.; Arabiat, O.; Al-Nemrat, A. Forensic Malware Analysis: The Value of Fuzzy Hashing Algorithms in Identifying Similarities. In Proceedings of the 2016 IEEE Trustcom/BigDataSE/ISPA, Tianjin, China, 23–26 August 2016; pp. 1782–1787. https://doi.org/10.1109/TrustCom.2016.0274.
4. Roussev, V. Data Fingerprinting with Similarity Digests. In *IFIP Advances in Information and Communication Technology*; Advances in Digital Forensics VI. DigitalForensics 2010; Chow, K.P., Shenoi, S., Eds.; Springer: Berlin/Heidelberg, Germany, 2010; Volume 337. https://doi.org/10.1007/978-3-642-15506-2_15.
5. Roussev, V. An evaluation of forensic similarity hashes. *Digit. Investig.* **2011**, *8*, S34–S41. Available online: https://www.sciencedirect.com/science/article/pii/S1742287611000296 (accessed on 27 October 2022).
6. Naik, N.; Jenkins, P.; Savage, N.; Yang, L.; Boongoen, T.; Iam-On, N. Fuzzy-Import Hashing: A Malware Analysis Approach. In Proceedings of the 2020 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE), Glasgow, UK, 19–24 July 2020; pp. 1–8. https://doi.org/10.1109/FUZZ48607.2020.9177636.
7. Roussev, V.; Quates, C. The Sdhash Tutorial—The Sdhash Tutorial. New Orleans, Louisiana. 2013. Available online: http://roussev.net/sdhash/tutorial/sdhash-tutorial.html (accessed on 9 September 2023).
8. Valosek, B. Apktool. 2010. Available online: https://ibotpeaches.github.io/Apktool/ (accessed on 9 September 2023).
9. Oprea, S.V.; Bara, A.; Dobrita, G.; Barbu, D.C. A Horizontal Tuning Framework for Machine Learning Algorithms Using a Microservice-based Architecture. *Stud. Inform. Control.* **2023**, *32*, 31–43. https://doi.org/10.24846/v32i3y202303.
10. Sidorov, G. Vector Space Model for Texts and the tf-idf Measure. In *Syntactic N-Grams in Computational Linguistics*; SpringerBriefs in Computer Science; Springer: Cham, Switzerland, 2019. https://doi.org/10.1007/978-3-030-14771-6_3.
11. Damodaran, A.; Troia, F.D.; Visaggio, C.A.; Austin, T.H.; Stamp, M. A comparison of static, dynamic, and hybrid analysis for malware detection. *J. Comput. Virol. Hack. Tech.* **2017**, *13*, 1–12. https://doi.org/10.1007/s11416-015-0261-z.

12. Gopinath, M.; Sibi, C.S. A comprehensive survey on deep learning based malware detection techniques. *Comput. Sci. Rev.* **2023**, *47*, 100529. https://doi.org/10.1016/j.cosrev.2022.100529.
13. Akhtar, M.S.; Feng, T. Malware Analysis and Detection Using Machine Learning Algorithms. *Symmetry* **2022**, *14*, 2304. https://doi.org/10.3390/sym14112304.
14. Kamran, S.; Suhuai, L.; Vijay, V. A novel deep learning-based approach for malware detection. *Eng. Appl. Artif. Intell.* **2023**, *122*, 106030. https://doi.org/10.1016/j.engappai.2023.106030.
15. Geremias, J.; Viegas, E.K.; Santin, A.O.; Britto, A.; Horchulhack, P. Towards Multi-view Android Malware Detection Through Image-based Deep Learning. In Proceedings of the 2022 International Wireless Communications and Mobile Computing (IWCMC), Dubrovnik, Croatia, 30 May–3 June 2022; pp. 572–577. https://doi.org/10.1109/IWCMC55113.2022.9824985.
16. Kural, O.E.; Şahin, D.Ö.; Akleylek, S.; Kılıç, E.; Ömüral, M. Apk2Img4AndMal: Android Malware Detection Framework Based on Convolutional Neural Network. In Proceedings of the 2021 6th International Conference on Computer Science and Engineering (UBMK), Ankara, Turkey, 13–17 September 2021; pp. 731–734. https://doi.org/10.1109/UBMK52708.2021.9558983.
17. Singh, J.; Thakur, D.; Gera, T.; Shah, B.; Abuhmed, T.; Ali, F. Classification and Analysis of Android Malware Images Using Feature Fusion Technique. *IEEE Access* **2021**, *9*, 90102–90117. https://doi.org/10.1109/ACCESS.2021.3090998.
18. Ke, X.; Hui, Y.X. Android Malware Detection Based on Image Analysis. In Proceedings of the 2021 IEEE 2nd International Conference on Information Technology, Big Data and Artificial Intelligence (ICIBA), Chongqing, China, 17–19 December 2021; pp. 295–300. https://doi.org/10.1109/ICIBA52610.2021.9688179.
19. Jin, X.; Xing, X.; Elahi, H.; Wang, G.; Jiang, H. A Malware Detection Approach Using Malware Images and Autoencoders. In Proceedings of the 2020 IEEE 17th International Conference on Mobile Ad Hoc and Sensor Systems (MASS), Delhi, India, 10–13 December 2020; pp. 1–6. https://doi.org/10.1109/MASS50613.2020.00009.
20. Naït-Abdesselam, F.; Darwaish, A.; Titouna, C. An Intelligent Malware Detection and Classification System Using Apps-to-Images Transformations and Convolutional Neural Networks. In Proceedings of the 2020 16th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob), Thessaloniki, Greece, 12–14 October 2020; pp. 1–6. https://doi.org/10.1109/WiMob50308.2020.9253386.
21. Darwaish, A.; Naït-Abdesselam, F. RGB-based Android Malware Detection and Classification Using Convolutional Neural Network. In Proceedings of the GLOBECOM 2020–2020 IEEE Global Communications Conference, Taipei, Taiwan, 7–11 December 2020; pp. 1–6. https://doi.org/10.1109/GLOBECOM42002.2020.9348206.
22. Fang, Y.; Gao, Y.; Jing, F.; Zhang, L. Android Malware Familial Classification Based on DEX File Section Features. *IEEE Access* **2020**, *8*, 10614–10627. https://doi.org/10.1109/ACCESS.2020.2965646.
23. Yujie, P.; Weina, N.; Xiaosong, Z.; Jie, Z.; Wu, H.; Ruidong, C. End-To-End Android Malware Classification Based on Pure Traffic Images. In Proceedings of the 2020 17th International Computer Conference on Wavelet Active Media Technology and Information Processing (ICCWAMTIP), Chengdu, China, 18–21 December 2020; pp. 240–245. https://doi.org/10.1109/ICCWAMTIP51612.2020.9317489.
24. Jiang, J.; Liu, Z.; Yu, M.; Li, G.; Li, S.; Liu, C.; Huang, W. HeterSupervise: Package-level Android Malware Analysis Based on Heterogeneous Graph. In Proceedings of the 2020 IEEE 22nd International Conference on High Performance Computing and Communications; IEEE 18th International Conference on Smart City; IEEE 6th International Conference on Data Science and Systems (HPCC/SmartCity/DSS), Yanuca Island, Cuvu, Fiji, 14–16 December 2020; pp. 328–335. https://doi.org/10.1109/HPCC-SmartCity-DSS50907.2020.00040.
25. Yajamanam, S.; Selvin, V.; Di Troia F.; Stamp, M. Deep Learning versus Gist Descriptors for Image-based Malware Classification. In Proceedings of the 2018 International Conference on Information Systems Security and Privacy, Funchal-Madeira, Portugal, 22–24 January 2018; pp. 553–561. https://doi.org/10.5220/0006685805530561.
26. Bagga, N.; Troia F.; Stamp, M. On the Effectiveness of Generic Malware Models. In Proceedings of the 15th International Joint Conference on e-Business and Telecommunications (ICETE 2018)—Volume 1: DCNET, ICE-B, OPTICS, SIGMAP and WINSYS, Porto, Portugal, 26–28 July 2018; San José State University: San Jose, CA, USA, 2018; pp. 442–450, ISBN 978-989-758-319-3. https://doi.org/10.5220/0006921504420450.
27. Yang, M.; Wen, Q. Detecting android malware by applying classification techniques on images patterns. In Proceedings of the 2017 IEEE 2nd International Conference on Cloud Computing and Big Data Analysis (ICCCBDA), Chengdu, China, 28–30 April 2017; pp. 344–347. https://doi.org/10.1109/ICCCBDA.2017.7951936.
28. Wang, T.; Xu, N. Malware variants detection based on opcode image recognition in small training set. In Proceedings of the 2017 IEEE 2nd International Conference on Cloud Computing and Big Data Analysis (ICCCBDA), Chengdu, China, 28–30 April 2017; pp. 328–332. https://doi.org/10.1109/ICCCBDA.2017.7951933.
29. Kumar, A.; Sagar, K.P.; Kuppusamy, K.S.; Aghila, G. Machine learning based malware classification for Android applications using multimodal image representations. In Proceedings of the 2016 10th International Conference on Intelligent Systems and Control (ISCO), Coimbatore, India, 7–8 January 2016; pp. 1–6. https://doi.org/10.1109/ISCO.2016.7726949.
30. Neeraj, C.; Di Troia, F.; Stamp, M. A Comparative Analysis of Android Malware. In Proceedings of the 5th International Conference on Information Systems Security and Privacy, ICISSP 2019, Prague, Czech Republic, 23–25 February 2019.
31. Sood, G. *Virustotal: R Client for the Virustotal API*, R Package Version 0.2.2; 2021. Available online: https://www.virustotal.com/gui/home/upload (accessed on 9 September 2023).

32. App Manifest Overview | Android Developers. Available online: https://developer.android.com/guide/topics/manifest/manifest-intro (accessed on 9 September 2023).
33. Naik, N.; Jenkins, P.; Savage, N.; Yang, L.; Boongoen, T.; Iam-On, N.; Naik, K.; Song, J. Embedded YARA rules: Strengthening YARA rules utilising fuzzy hashing and fuzzy rules for malware analysis. *Complex Intell. Syst.* **2020**, *7*, 687–702. https://doi.org/10.1007/s40747-020-00233-5.
34. Chen, J.; Wang, C.; Zhao, Z.; Chen, K.; Du, R.; Ahn, G.-J. Uncovering the Face of Android Ransomware: Characterization and Real-Time Detection. *IEEE Trans. Inf. Forensics Secur.* **2018**, *13*, 1286–1300. https://doi.org/10.1109/TIFS.2017.2787905.
35. Mahdavifar, S.; Abdul Kadir, A.F.; Fatemi, R.; Alhadidi, D.; Ghorbani, A.A. Dynamic Android Malware Category Classification using Semi-Supervised Deep Learning. In Proceedings of the 18th IEEE International Conference on Dependable, Autonomic, and Secure Computing (DASC), Calgary, AB, Canada, 22–26 June 2020.
36. Mahdavifar, S.; Alhadidi, D.; Ghorbani, A.A. Effective and Efficient Hybrid Android Malware Classification Using Pseudo-Label Stacked Auto-Encoder. *J. Netw. Syst. Manag.* **2022**, *30*, 22.