IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 29 September 2023, accepted 20 October 2023, date of publication 30 October 2023, date of current version 6 November 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3328314

RESEARCH ARTICLE

Android Ransomware Analysis Using Convolutional Neural Network and Fuzzy Hashing Features

HORACIO RODRIGUEZ-BAZAN, GRIGORI SIDOROV, AND PONCIANO JORGE ESCAMILLA-AMBROSIO, (Senior Member, IEEE)
Centro de Investigación en Computación (CIC), Instituto Politécnico Nacional (IPN), Mexico City 07738, Mexico

Corresponding author: Ponciano Jorge Escamilla-Ambrosio (pescamilla@cic.ipn.mx)

This work was supported in part by the Mexican Government under Grant A1-S-47854; and in part by Secretaría de Investigación y Posgrado of Instituto Politécnico Nacional under Grant SIP-20230990, and Grant SIP-20232782.

ABSTRACT Most of the time, cybercriminals look for new ways to bypass security controls by improving their attacks. In the 1980s, attackers developed malware to kidnap user data by requesting payments. Malware is called a ransomware. Recently, they have demanded payment in Bitcoin or any other cryptocurrency. Ransomware is one of the most dangerous threats on the Internet, and this type of malware could affect almost all devices. Malware cipher device data, making them inaccessible to users. In this study, a new method for Android ransomware classification was proposed. This method implements a Convolutional Neural Network (CNN) for malware classification based on images. This paper presents a novel method for transforming an Android Application Package (APK) into a grayscale image. The image creation relies on using Natural Language Processing (NLP) techniques for text cleaning and Fuzzy Hashing to represent the decompiled code from the APK in a set of hashes after preprocessing using NLP techniques. The image is composed of *n* fuzzy hashes that represent the APK. The method was tested using a dataset of 7,765 Android ransomware samples obtained from external researchers and public sources. The accuracy of the proposed method was higher than that of other methods in the literature.

INDEX TERMS Android ransomware, convolutional neural network, deep learning, fuzzy hashing, malware classification, ransomware.

I. INTRODUCTION

Since 2008 with the delivery of the first Android version, this year marked before and after on mobile devices. Over time, Android has become the most widely used mobile operating system worldwide.

As a consequence of Android philosophy (''Android is designed to be open.'' ''Android is designed for developers.'' and ''Android is designed for users''). [1] is the primary target of the attack. Cybercriminals develop and release apps daily with malware, mainly in unofficial app stores or websites, owing to a lack of security controls for publishing apps. Therefore, the amount of malware released has increased. Although it was determined that more than 3 million malware could run on the Android operating system in 2014, 279 thousand new malware were released every month [2]. According to the Kaspersky Security Network [3], in Q3 2022, over 5.5 million mobile malware were blocked, which shows an exponential increase in mobile malware.

Ransomware is a type of malware that encrypts user data. The first ransomware variants were created during the late 1980s. This class of malware is potentially malicious because the files are encrypted with unknown keys, making them inaccessible, and recovery is a complex task unless the user pays the required payment, mainly in cryptocurrencies such as Bitcoin. It is difficult to prevent or detect ransomware before it encrypts data on a mobile device.

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

Recently, Android malware has been studied as an emergency, and several researchers have focused on dealing with this type of threat by developing models to detect and classify malware using Artificial Intelligence (AI). On the other hand, cybercriminals continuously seek new ways to bypass security controls to achieve their goals, improve their techniques, and make malware more sophisticated and complex to detect using traditional methods, for instance, based on signatures.

Researchers have proposed various approaches to detect and/or classify Android malware. Researchers have proposed using n-grams, API calls, sandbox outputs, and other features in combination with Machine Learning (ML) and Deep Learning (DL) algorithms. Recently, a new method of malware analysis was explored, in which samples were converted to images, and ML and DL algorithms were trained for malware classification and/or detection using these images.

The motivation behind this research is to make a meaningful contribution to the field of Android malware analysis by explicitly focusing on ransomware. Ransomware is widely regarded as one of the most severe threats to the internet. To address this issue, we employed model-based AI and Natural Language Processing (NLP) techniques to bolster defense against cybercriminals. This paper proposes a new method for Android ransomware classification that transforms an Android Application Package (APK) sample into a grayscale image composed of fuzzy hashing of decompiled and preprocessed code. The main contributions of this study are as follows.

- A novel method for transforming an Android application into a grayscale image composed of fuzzy hashes was presented. First, decompiled code is preprocessed using Natural Language Processing (NLP) techniques. Subsequently, a fuzzy hashing technique was used to calculate the hashes per block of code, considerably reducing the image size. The image size is important because variations in the size of the images can affect feature extraction.
- For the first time, using Natural Language Techniques (NLP) for text cleaning and extraction was introduced as per our state-of-the-art research. These techniques help maintain and standardize the data and remove useless information that could add noise to the images, thereby reducing the accuracy.
- An experimental evaluation of the proposed approach on a ransomware dataset was performed to demonstrate the feasibility of our approach in comparison to those found in the literature.

The remainder of this paper is organized as follows. Section II describes the concepts related to this investigation. Section III presents an analysis of the state-of-the-art methods. Section IV presents the proposed method. Section V provides details of the experimental evaluation and results. Section VI presents limitations, future work, and conclusions.

II. RELATED CONCEPTS

A. FUZZY HASHING

Fuzzy Hashing (FH), also known as Context Triggered Piecewise Hashing (CTPH), is a combination of Cryptographic Hashes (CH), Rolling Hashes (RH), and Piecewise Hashes (PH). This can be expressed as $FH = CTPH = PH + RH$. Unlike traditional hashes, in which the 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. Fuzzy hashing 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 indicates low similarity and 100 indicates high similarity [4].

File comparison tools, such as MD5, SHA1, and SHA256, are commonly utilized to determine whether two files are identical. However, it is essential to determine whether they are the same or different and understand their similarities. To accomplish this, fuzzy hashing tools, such as SSDEEP or SDHASH, can be employed.

SSDEEP and SDHASH are fuzzy hashing algorithms used to compare files for similarity. Other fuzzy algorithms, including SimHash, TLSH, and LZJD, were utilized to measure similarities. An evaluation of these fuzzy hashing algorithms was conducted by Daojing et al. [5].

The SSDEEP algorithm sequentially divides a file into equal groups of bytes and calculates the hash for each group. A new hash is calculated to represent the entire file. The generated hash can be utilized to determine the similarity between the file and others.

The similarity digest hash (SDHASH) fuzzy hashing 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 [6]. Generally, a feature is a 64-byte string determined using entropy calculations [7]. It employs the cryptographic hash function SHA-1 and Bloom filters to calculate the SDHASH fuzzy hash value [8].

The SDHASH is a robust algorithm for fuzzy hashing. This algorithm provides high accuracy compared to its predecessor, SSDEEP. The algorithm can compute SDHASH using options that generate different SDHASH lengths, and the results can be compared. Two or more SDHASH strings can be compared, even if their lengths differ.

Bloom filters have predictable probabilistic properties that allow for directly comparing two filters 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. To compare the two digests for each filter in the first digest, the maximum match among the filters of the second digest is found. The resulting matches are averaged [7].

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

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

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 1,000 equivalence classes to address this. Statistics are gathered using the approximation method.

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

The significance in the range is the 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* results (range: 1-10) are generally weak; typically, most would be false positives. *Negative* (range:0), the correlation between the targets is statistically comparable to that of the two blobs of random data. *Unknown* (range:-1) is a rare occurrence for files above 4KB unless they contain large regions of low-entropy data. However, in all cases, the significance depends on the amount and type of the data [9].

SSDEEP and SDHASH differ because the hash length is always the same for SSDEEP. In SDHASH, the hash length depends on the input (amount of data). As mentioned previously, SDHASH is more robust and yields options that can be used during computation. For instance, the segment size by default is 128MB, but this setting can be changed.

B. APK FILE STRUCTURE

An APK file is a compiled application for the Android operating system. The package contains all files needed for a single application and is organized in a particular structure. Figure 1 shows the APK file structure and 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 application. The folder contained multiple directories, one for each supported CPU architecture (armeabi-v7a, x86, etc.).
- ***res/***: Directory with resources not compiled into *resources.arsc*. This directory contains all resources except files in the *res/values*. The resource files are in a binary XML format, and all image files are optimized (crunched) to save space and improve the run-time performance when inflating these files.
- ***assets/***: The directory with application assets can be retrieved by *AssetManager*.
- ***AndroidManifest.xml***: The application manifests in binary XML file format. It contains application metadata such as app name, version, permissions, and the minimum SDK version.
- ***classes.dex***: The classes are compiled in Java language that will be executed on the device by the virtual machine.
- ***resources.arsc***: This contains metadata about resources. The ARSC file is an Android resource table file that contains a list of application resources in table format.

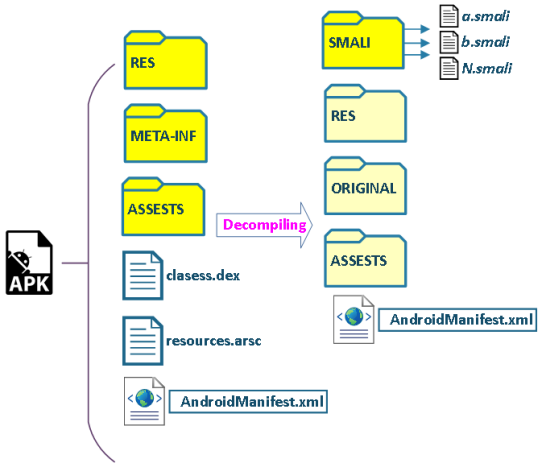

FIGURE 1. APK structure before and after decompiling.

There are multiple tools available for decompiling the APK. APKTool [10] was used in this study. In the decompiling process, multiple types of files are generated. All the APKs have Java code compiled into a classes.dex file. On the other hand, the application properties are declared in AndroidManifest.xml, both compiled in a binary format. Additional resources are included in the APK structure, such as images, app signatures, and Cascading Style Sheets (CSS). The most essential part of an APK is the source code and its properties. Therefore, these essential files were chosen to analyze the applications.

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.

C. NLP TECHNIQUES

In Natural Language Processing, many techniques can be used for text cleaning and extraction. This section describes the techniques employed in this study.

1) PUNCTUATION

In Natural Language Processing (NLP), punctuation refers to using marks or symbols in text analysis to 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 the speech of 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 emotions of a text.

Moreover, punctuation can pose challenges in NLP, such as dealing with informal or unstructured text data on social media posts or chat messages that may contain unconventional or inconsistent punctuation. Therefore, it is essential to consider the specific context and characteristics of text data when using punctuation in NLP.

2) TOKENIZATION

Tokenization is the process of breaking down a text document or string of text into smaller units called ''tokens.'' In natural language processing (NLP), these tokens usually correspond to words but can also be phrases, symbols, or characters.

Tokenization is essential in many NLP tasks, such as text classification, sentiment analysis, and language translation. It allows machine learning algorithms to better understand the structure and meaning of text data by breaking it down into smaller, more manageable pieces.

There are different approaches to tokenization, depending on the specific task needs and characteristics of the data text. Common methods include whitespace tokenization, which splits text into tokens based on spaces or other whitespace characters, and word-level tokenization, which breaks the text into individual words.

3) LEMMATIZATION

Lemmatization is the process of reducing a word to its base or dictionary form, known as the ''lemma.'' In other words, it is the process of converting words into canonical forms. 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. This helps group the different inflected forms of a word to be analyzed as a single item.

Lemmatization is often used in natural language processing (NLP) and text analysis tasks such as language translation, information retrieval, and sentiment analysis. Search engines also use these to improve the accuracy of search results by reducing search queries and indexing documents to their base forms before matching them.

4) STEMMING

Stemming is the process of reducing a word to its root or stem form by removing its 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.'' Note that 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 than lemmatization. It is often used in information retrieval and search engines, where speed is essential, and the exact meaning of the words is less critical. However, stemming can also lead to errors or the loss of meaning, particularly in languages with complex word forms and irregular verbs.

5) KEYWORD EXTRACTION

Term Frequency (TF) is a metric used in Natural Language Processing (NLP) and information retrieval to quantify the importance or frequency of a term within a document or corpus. It measures the frequency with which 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 the document. This 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 zero and one.

Term frequency is a technique used to determine the significance or relevance of a term in a document. Generally, words or phrases that appear more frequently are considered more important or relevant to the content of the document.

TF measures the frequency of a term within a document, *tf* (term frequency). This indicates how often a term appears in a document relative to the total number of words. A higher TF value indicates that a term is more significant within a document. More specifically, it is denoted by tf_{ij}, the number of times word i appears in document j [11].

III. RELATED WORK

Traditionally, there are two approaches to malware analysis: static and dynamic [12]. The main difference is that the sample was executed in a controlled environment during dynamic analysis. The resulting features were used to train Machine Learning (ML) and Deep Learning (DL) algorithms to build classification and detection models.

A. IMAGE VISUALIZATION

This section reviews studies related to ML and DL for malware analysis using image visualization. The reviewed works used the approach of converting samples into images, regardless of the platform. Moreover, Android-related studies were used for comparison purposes.

Geremias et al. [13] presented a method for Android malware analysis that first extracts the features in feature vectors, and PCA is applied to generate a matrix of $M \times M$ size. Then, three grayscale images were generated using different data types (API calls, OPCodes, and Dex). The final image was composed of three images as layers to build a colored image (multi-view), and the resulting images were used to train a CNN model. The method was evaluated using the CICMalDroid dataset, which contained 11,598 samples and achieved an accuracy of 98.70%.

Shiva et al. [14] proposed a Windows malware detection method based on CNN. A Portable Executable (PE) was executed on the Cuckoo Sandbox. The CAT-API features were extracted from Malware Instruction Set (MIST) files to generate n-grams. Subsequently, the n-gram was transformed into a grayscale image to feed the CNN. The experimental results showed that the proposed approach effectively uncovered malware PE files by utilizing the significant behavioral features suggested by the Relief Feature Selection Technique. The method was evaluated using the Malheur dataset containing 3,282 benign and 4,151 malware. The method achieved a detection accuracy of 97.96%.

Gulmez et al. [15] investigated a method for Windows malware that, without the need for unpacking or dynamic analysis, consists of creating a graph or sub-graph based on Opcode Sequences (opcode) obtained from the disassembly process. An opcode sequence contains crucial information about the executable file action without running it. These images (graph) were used in machine learning models for classification, achieving the highest accuracy with the Random Forest algorithm of up to 97.00%. The dataset consists of 7,500 malware samples from VX-Heaven and 7,500 benign samples from Download.com.

Kural et al. [2] presented a framework called Apk2Img4AndMal. The tool directly transforms the APK into a grayscale image without any preprocessing or reverse engineering process. No feature extraction from the static or dynamic analysis was required. The images were generated by reading the APK as binary and transforming them into grayscale images. The images were analyzed using a CNN model, achieving an accuracy of up to 94.00%. The framework was tested using 24,588 Android malware applications and 3,000 benign applications.

Jaiteg et al. [16] proposed a method that combines features from the decompiled APK file, searching for the optimal combination (Android manifest (AM), certificate (CR), classes.dex (CL), and resource (RS)), and 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 are used to feed the CNN. This method was evaluated using the DREBIN dataset, which contains multiple classes. This method achieved a high accuracy of 93.24% for the malware image combination CR + AM using the Feature Fusion-SVM classifier.

Xu et al. [17] presented a method for analyzing Android malware using DEX (bytecode) files. 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 study selected 5,000 Android apps, including 2,500 benign and 2,500 malicious applications. The research attained a high accuracy of 98.7%.

Li et al. [18] presented a method for Windows binary that combines image segmentation and a deep learning residual network (ResNet) based on image segmentation using the watershed algorithm to address the issue of malware families sharing similarities. The original gray image was transformed into more distinctive sample data using image segmentation technology, which makes the dataset increase the distance between classes and reduces the distance within classes. The training was performed on the residual network. The results showed an increase in accuracy compared to those without segmentation preprocessing. An unbalanced dataset comprising 9,339 samples was used in this study. The accuracy was improved by up to 98.94% in detection and up to 81.48% in detecting similar families.

Radifa et al. [19] proposed converting Windows binary into grayscale images and then using a CNN (ResNet-50) in the classification process. Preprocessing was not required in this study. The samples were directly converted into grayscale images to feed the neural network. They used a binary file conversion process by converting it to an 8-bit vector and finally producing a grayscale image with dimensions $[0, 255]$. The MalImg dataset, consisting of 25 malware classes and 9,435 samples, was used. Additionally, 295 samples without malware were used, and an accuracy of up to 94.03% was achieved.

Xiang et al. [20] investigated an Android malware detection model based on deep learning using autoencoders to detect malware. This study aims to determine whether autoencoders can reconstruct malware images with low loss and detect malware by determining 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. The Andro-dumpsys dataset used contains 906 malicious binaries from 13 different malware families and 1,776 benign files downloaded from the Google Play store. An accuracy of up to 93.00% was achieved.

Jinrong et al. [21] proposed a classification method for Windows malware that uses a sequence of assembly instructions to generate RGB images, combined with the instruction part of the Intel manual and the malware dataset (BIG2015) provided by Microsoft. The RGB images preserve as much information as possible by removing the machine code corresponding to each instruction. Subsequently, a CNN for malware classification was tested. The experimental dataset was obtained from the Malware Classification Challenge on Kaggle in Microsoft 2015. It contains 10,868 malware samples and includes nine large malware families with binary and disassembly files. The research attained a high accuracy of 97.73%.

Jun-Seob et al. [22] presented a method for analyzing Windows malware based on icon similarities. The authors tested

their hypothesis using four types of Hamming distances (aHash, pHash, dHash, and wHash), and the results were documented based on icon distance. However, the authors did not document any evaluation metric. They collected a dataset of 15,400 samples and filtered 7,312 samples containing icons for a total of 2,572 unique icons.

Naït-Abdesselam et al. [23], [24] proposed transforming the APK into an RGB by 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) images to use them in a CNN and ResNet for classification of malware. The method was evaluated using the AndroZoo dataset, which increases the number of samples over time. The authors chose n samples per time-frame, and the method attained a detection accuracy of up to 99.37%.

Yong et al. [25] presented a new method for analyzing Android malware based on DEX files. They proposed converting DEX files into RGB images and then applying text and color features. In addition, plain text was filtered from the DEX file to obtain text features, GIST was used to obtain texture features, and color moments were used to feed multiple kernels as input data for malware classification. The resulting images show that the samples from the same family have similar colors and textures. The authors used a Support Vector Machine (SVM) in the classification phase by applying multiple kernels for testing. The AMD dataset contains 24,553 samples, categorized into 135 varieties among 71 malware families from 2010 to 2016, with a classification accuracy of up to 96.00%.

Peng et al. [26] presented a method for Android malware classification based on 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 was used to generate a grayscale image representing the traffic characteristics of each session. Finally, the images were analyzed using a deep learning model (1.5D-CNN). The CICAndMal2017 dataset contains over 1,700 benign and 400 malware samples. The proposed model achieved an accuracy of up to 98.5%.

Jianguo et al. [27] focused on Android malware analyses. 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, which is represented as a matrix to feed the HG-CNN classifier. The dataset was collected from diverse sources and used a known dataset, such as Drebin. In total, using 11,423 benign and 14,546 malware samples, the authors achieved an accuracy of over 97%.

Yoo et al. [28] presented a method for detecting exploit kits by using images. The exploit was directly transformed into a grayscale image without preprocessing. For the detection phase, the images were processed using a Recursive Convolutional Neural Network (RCNN). The model was tested using 36,863 real-world datasets provided by an antivirus company, achieving 98.2% accuracy in exploit kit detection and family classification.

Baptista et al. [29] proposed a method that uses binary visualization with colored RGB images (red if the character is extended, green if the character is controlled, and blue if the character is printable). The method focuses on two types of files: *pdf* and *doc* files infected with malware. Other types of files were collected and analyzed using the same technique. Documents were collected from the VirusShared website. The corpus contains 2,000 benign and 2,000 malware samples. For malware detection, up to 94.1% was achieved using self-organizing incremental neural networks (SOINN).

Peng et al. [30] converted Windows malware into RGB malware using a PE file structure, thereby enhancing the traditional grayscale image model. In each color channel, the researchers added specific data from the extraction process (Red: malware binary data, Green: malware ASCII character data, and Blue: malware PE structure data). In the classification phase, a CNN was applied. Documents were collected from the Cyberspace Security Institute of the Beijing University of Posts and Telecommunications. The corpus contained 10,000 malware samples. In malware detection, up to 87.75% was achieved using a neural network (Spp-Net).

Shaojie et al. [31] investigated malware-infected Microsoft documents. The researchers converted the document into a grayscale image using data and table sections. The corpus contained 1,796 unique documents, including 978 malicious files collected from VirusTotal and 818 benign files. In a detection study, the authors tested the method using three CNN models. VGG obtained the best results, and the experimental results showed that the detection accuracy rate for the test dataset reached 94.09%. In the simulated zero-day malware detection experiment, the average accuracy rate reached 94.70% (8 malware and 12 benign samples).

O'Shaughnessy [32] proposed a novel method for visualizing and classifying Windows malware using Space-Filling Curves (SFCs). The binary image was converted using SFC, and the images were trained using the KNN-HOG and GIST algorithms. This research is different from the rest because they applied this type of algorithm to generate images. The dataset comprises 9,235 Windows 32-bit executable samples from 28 families. The results showed that KNN-HOG obtained better results than the GIST. The detection accuracy rates in the validation phase reached 83.00% and 91.30% during the training phase, respectively.

Yang et al. [33] presented an algorithm that transforms APK into a grayscale image based on a Portable Executable

(PE) file format. The images had APK data (CERT.RSA, AndroidManifest.xml, resources.arsc, classes.dex) converted directly without preprocessing. In this investigation, the researchers used the Drebin corpus, which contained 5,560 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 accuracy, reaching up to 95.51% in accuracy.

Tingting et al. [34] in their research proposed binary malware detection by converting opcode sequences into images in combination with PCA to extract the features, and SVM was used for malware classification by applying multiple kernel functions to increase the accuracy. The highest accuracy achieved was 97.62% using the SVM and RBF kernel functions. The dataset contained 9,168 malware samples and 8,640 benign programs.

Ajit et al. [35] presented 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, and HSL) to test which format worked better in combination with machine learning algorithms (Decision Tree, Random Forest, KNN) for the classification task. They proved that grayscale achieved the highest accuracy of 91.00% using a Random Forest.

Yajamanam et al. [36] presented a method for Windows malware analysis that transforms a sample into a grayscale image. The samples were classified based on their GIST features. Unlike previous related works, the researchers tested the robustness of GIST features by adding noise to the images. Based on the results, as expected, the accuracy of classifying images with noise decreased compared to images without noise. Two datasets were used in this research: Malimg data consists of more than 9,000 malware samples belonging to 25 families, and Malicia data contains 11,363 malware samples, primarily composed of three types of malware.

Huang et al. [37] presented a method for robust hashing by treating samples as two-dimensional images. The authors performed multiple tests to compare the SVM with robust hashing techniques. They found that some classes were correctly classified by robust hashing, and the results were comparable with those of the SVM. The Malimg dataset, which consists of more than 9,000 malware samples belonging to 25 families, was used in this investigation.

On the other hand, other researchers have proposed Android malware analysis based on static features such as permission [38]. However, without transforming the features into an image format, the permissions were represented as a vector in a binary sequence, and the analysis was based on a feature vector to train a machine learning model. The accuracy was 96%.

The studies listed presented malware detection and classification methods by transforming the samples into images. The proposed method highlights that the sample is transformed into a grayscale image. Unlike the studies reviewed, a new way of converting the sample into an image is presented. The image is composed of fuzzy hashes generated by the decompiled code and preprocessed. A summary of the related work is presented in Table 1.

IV. PROPOSED METHOD

This study aims to determine whether converting a sample into a grayscale image composed of fuzzy hashes of decompiled and preprocessed code can be used in a deep learning model (CNN) for ransomware classification. Therefore, our malware classification method was divided into two stages: data preprocessing (APK to a grayscale image) and malware classification (ransomware family). Figure 2 shows the proposed workflow at a high level, and its architecture is detailed in Figure 3.

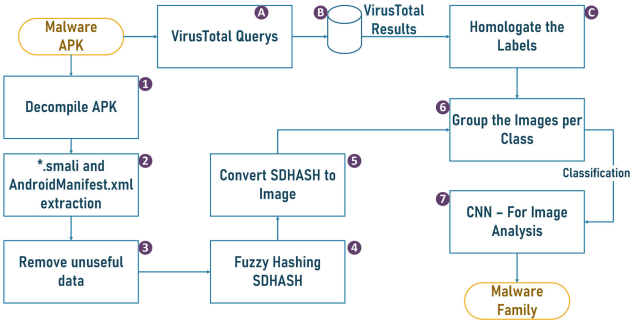

FIGURE 2. Workflow for APKs analysis.

First, it performs a novel transformation of the APK file into a lightweight grayscale image using fuzzy hashing of the decompiled smali code and the manifest file. Second, it trains a Convolutional Neural Network (CNN) on the obtained images for ransomware family classification. Furthermore, VirusTotal [39] was used to label the corpus, as described in Section V-B.

A. ANDROID PACKAGE TO IMAGE

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

The substeps transform the APK file into a grayscale image during this phase. The first step is decompiling the APK, as described in Section II-B. Multiple files are generated once the sample is decompiled (An APK has n smali files and one AndroidManifest.xml). At this point, only the smali and AndroidManifest.xml files were selected. It is feasible to determine the *family* of the sample by analyzing the smali code and application property file.

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

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

TABLE 1. Summary of the related work. DT - Decision Tree, GIST - Global Image descriptors, GLCM - Gray Level Co-occurrence Matrix-based, KNN - k-Nearest Neighbors, LBP - Local Binary Pattern, RF - Random Forest, NLP - Natural Language Processing, PCA - Principal Component Analysis, RCNN - Recursive Convolutional Neural Network, SFC - Space-Filling Curves, SVM - Support Vector Machine.

Ref./Year	Type of Malware	Samples	Type of Images	CNN	Other Algorithm	Other Features	Accuracy
[13], 2022	Android	APK	RGB	Yes	PCA	Multi-view image	98.70%
[2], 2021	Android	APK	Grayscale	Yes	-	APK read as binary	94.00% - CNN
[16], 2021	Android	APK	Grayscale	Yes	GLCM, GIST LBP	Feature fusion	93.24% - SVM
[17], 2021	Android	APK	Grayscale	Yes	GIST	dex file read as binary	98.70% - GIST
[20], 2020	Android	APK	Grayscale	Yes	Auto-enconders	Reconstruction error	93.00%
[23], 2020	Android	APK	RGB	Yes	ResNet	Decompiled data to RGB	99.37% - ResNet
[27], 2020	Android	APK	RGB	Yes	SVM, KNN, RF, HS-resNet,	Static and dynamic features	97.00%
[26], 2020	Android	APK	Grayscale	Yes	1.5D-CNN	Network traffic as image	98.50%
[25], 2020	Android	APK	RGB	No	SVM, GIST, Multiple Kernels	dex as image and plain text	96.00% - SVM
[24], 2020	Android	APK	RGB	Yes	ResNet	Decompiled data to RGB	99.37% - ResNet
Proposal	Android	APK	Grayscale	Yes	NLP Techniques	Fuzzy Hashing as image	98.97%

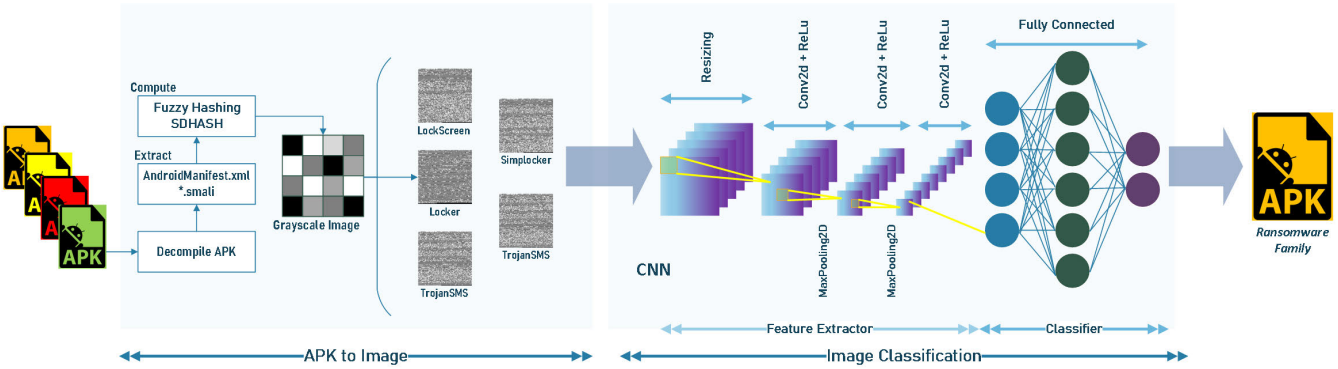

FIGURE 3. Proposed method: (A) APK to image conversion. This step transforms the APK into a grayscale image, (B) Grayscale images are analyzed in CNN for Android ransomware classification.

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

AndroidManifest.xml was also preprocessed at this stage. Furthermore, unlike the smali file, helpful information is extracted from the manifest file, such as app components (which include all activities, services, broadcast receivers, and content providers), permissions, and hardware and software features. The app requires discarding XML tags [40], using Equation (3)

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

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

Unlike related studies, the significance of the NLP techniques applied in this research lies in their ability to preprocess decompiled data, remove useless information, and standardize the data. This is essential because the fuzzy hashing technique measures similarity, and standardized data enhances these similarities. Furthermore, it helps avoid noise in the images and reduces the image size, which is beneficial during training.

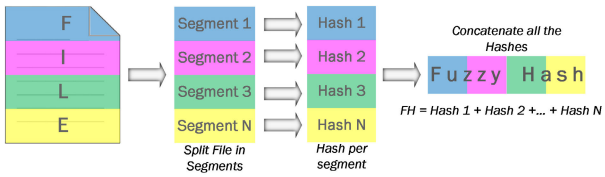

FIGURE 4. Fuzzy hashing description.

During the second stage, fuzzy hashing is computed from the smali and AndroidManifest.xml files using the same amount of data to force the same hash length in the output. The size of the input data (grayscale) is a matrix of $N \times M$, where N is the length of the SDHASH (fuzzy hash), which is 344 (344 pixels), and M, depends on the number of fuzzy hashes plus AndriodManifest data. No standard image size can be defined because each APK is different. The resize was performed in the CNN. The grayscale image structure has AndroidManifest data at the top and all the fuzzy hashes after. At this stage, each malware sample has n fuzzy hashes, as shown in Figure 4, which is a graphical description of how a fuzzy hash is computed. SDHASH is a string encoded in Base64, and each byte of the fuzzy hash is converted to a scale from 0 to 255, corresponding to one pixel in the grayscale

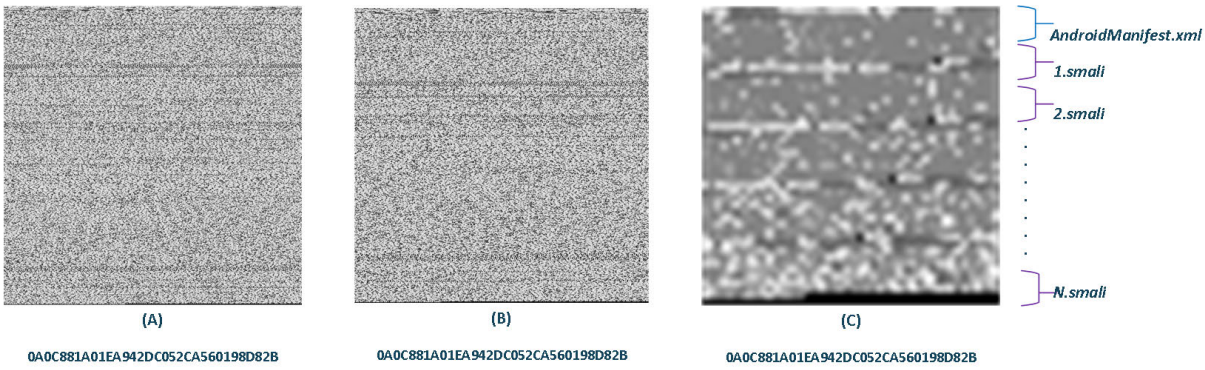

FIGURE 5. Android sample MD5 "0A0C881A01EA942DC052CA560198D82B" converted in a grayscale image using three different input data after decompiled sample: (A) Image created after decompiling process without any preprocessing, (B) Image created after preprocessing decompiled code using NLP, (C) Image created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

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

B. CONVOLUTIONAL NEURAL NETWORK

An image itself has specific properties. Convolutional Neural Networks (CNN) have been widely used for image analysis. Researchers have transformed the input data into images (grayscale or RGB). However, it is not common for them to preprocess the data before generating the images. In the architecture depicted in Figure 3, the APK is transformed into a grayscale image at the first stage after preprocessing the decompiled data. 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 epoch values in the experimental phase to increase the accuracy at the learning stage.

Therefore, the neural network uses a rectified linear unit (ReLu) as an activation function and a 'sigmoid' as an activation function in the output layer. Figure 6 summarizes the CNN model.

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

V. EXPERIMENTS

This section will conduct experiments to evaluate the proposed malware classification approach using CNN (Convolutional Neural Network). The primary purpose of these experiments was to prove the hypothesis that a malware classification model (deep learning) using grayscale images based on fuzzy hashing (similarities) can help classify malware samples into the right family. This section details the experimental setup, dataset description, experimental evaluation, and results.

```
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)        18496

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

 flatten (Flatten)           (None, 50176)             0

 dense (Dense)               (None, 128)               6422656

 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.

A. EXPERIMENTAL SETUP

As described above, the method was tested on a ransomware corpus (7,765 samples from multiple ransomware 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 @ 2.60GHz, 8 Core Processors, 32GB RAM, 300GB SSD, and GPU NVIDIA GeForce GTX 960M).

B. CORPUS DESCRIPTION

The corpus (database) comprises 2,288 Android ransomware samples shared by Wuhan University for research purposes [41], along with an additional 5,477 ransomware samples sourced from publicly available datasets, including CIC-AndMal2017 [42], CCCS-CIC-AndMal-2020 [43], [44], and Androzoo [45]. The dataset was labeled using VirusTotal [39] considering the results from the most popular antiviruses such

as AhnLab-V3, McAfee, CrowdStrike, Symantec Mobile Insight, F-Secure, and others. For instance, the sample with MD5 hash "0A0C881A01EA942DC052CA560198D82B" has multiple labels, per the VirusTotal report, as shown in Table 2.

TABLE 2. Sample MD5 "0A0C881A01EA942DC052CA560198D82B", partial results from VirusTotal.

AntiVirus	Malware
AegisLab	Trojan.AndroidOS.Generic.C!c
AhnLab-V3	Android-Trojan/Koler.b45b
Alibaba	Trojan:Android/Koler.729cc57f
ESET-NOD32	A Variant Of Android/Koler.O
F-Secure	Malware.ANDROID/Koler.O.Gen
Kaspersky	HEUR:Trojan-Ransom.AndroidOS.Svpeng.af
MaxSecure	Android.Svpeng.b
McAfee-GW-Edition	Artemis!Trojan
Symantec Mobile Insight	Trojan:Lockdroid.E
BitDefender	Undetected

The reports were standardized for the remaining samples because each antivirus had labels. Some were similar, but others were very different. Moreover, some antiviruses do not have labels per malware family. In other words, the number of samples is the number of families. Table 3 shows the distribution of Android ransomware families after standardization.

TABLE 3. Ransomware malware dataset, labels simplified after VirusTotal results.

No. Class	Class Name	No. Samples	Percentage
1	Jitsu	2,790	35.93 %
2	SMForw	1,156	14.88 %
3	SMSSpy	759	9.77 %
4	LockScreen	739	9.51 %
5	Locker	704	9.06 %
6	Koler	478	6.15 %
7	Simplocker	407	5.24 %
8	Torec	392	5.04 %
9	Congur	340	4.37 %
	Total	**7,765**	**100 %**

C. CLASSIFICATION ACCURACY

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

D. EXPERIMENTAL RESULTS

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

The first stage of the proposal involves transforming the APK into grayscale images. In the image analysis field, there is 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 4 shows the distribution using three different input data after generating grayscale images. The fuzzy hashing approach reduced the image size compared with the other two input data. Figure 7 presents a set of samples from the ransomware families converted to grayscale images using fuzzy hashing and normalized in terms of height and width.

The tests were designed mainly in two ways: the first used all the classes in the dataset, although some classes did not have the same number of members, and the second selected only classes with a high number of samples.

TABLE 4. Image size after conversion in 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 any preprocessing.

File Size	SDBF	Src NLP	All Src
0KB – 20KB	4,587	1,826	1,020
20KB – 40KB	1,029	2,393	622
40KB – 60KB	276	201	2
60KB – 80KB	851	320	667
80KB – 100KB	223	137	1,142
101KB – 500KB	413	1,134	960
500KB – 1MB	298	389	678
1MB – 2MB	78	620	2,238
2MB – 3MB	10	601	271
3MB – 4MB	0	102	98
> 4MB	0	42	67
Total	**7,765**	**7,765**	**7,765**

The CNN was evaluated using three generated images to prove that the fuzzy hashing approach is more accurate than the other images. Section IV-B describes the CNN settings as part of the second stage of the proposal.

Table 5 summarizes the 12 scenarios tested. The parameters were varied during the evaluation of CNN to determine the optimal values. Moreover, because of the dataset described in Table 3, the classes did not have the same number of samples, and the learning process was impacted. Based on the distribution shown in Table 3, the top five comprised hold 79.17% (6,148 samples) of the dataset. Meanwhile, the

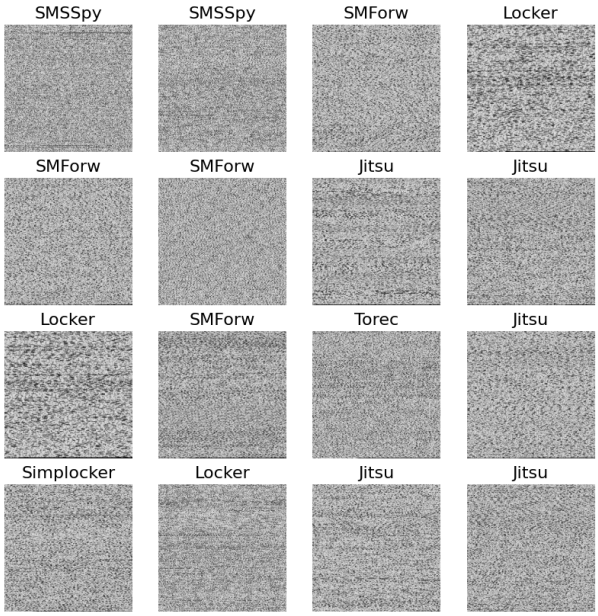

FIGURE 7. Grid of ransomware families converted to grayscale images using fuzzy hashing.

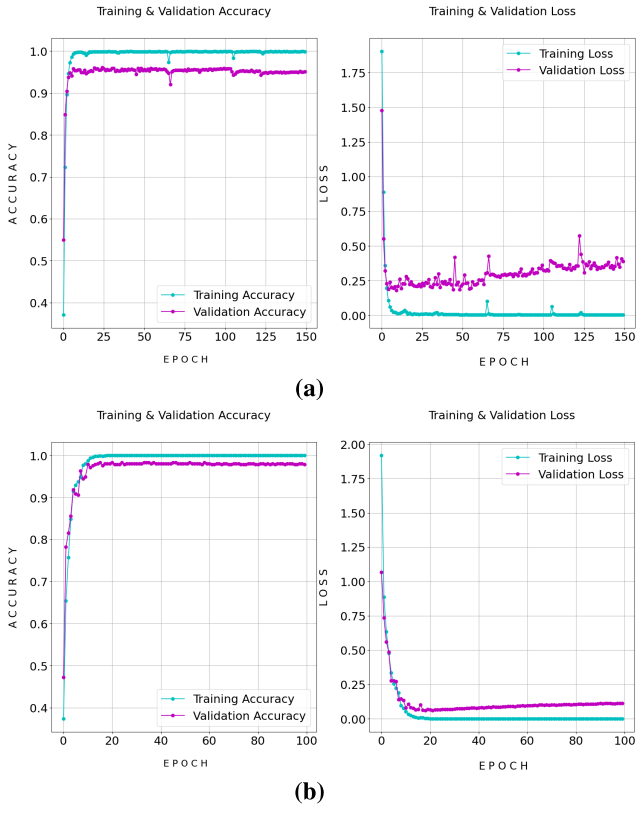

FIGURE 8. Experimental metrics using grayscale images based on fuzzy hashing, the graphs show the accuracy and loss during training and evaluation. (a) CNN training and validation results using all the classes in the dataset with 150 epochs, accuracy 95.62% (b) CNN training and validation results using the top 5 classes in the dataset with 100 epochs, accuracy 98.97%.

other four classes provided 20.83% (1,617 samples) of the dataset.

The experimental results presented in Table 5 summarize the metrics of the CNN model using different types of images and CNN parameters. Each experiment was executed ten times (training and evaluation), and the results were averaged. For instance, using the entire dataset (nine classes), the accuracy was 94.37%, and 98.16% using the five most representative classes. There was an increase in the model accuracy by leaving out the non-significant classes from the dataset. In both cases, using the fuzzy hashing dataset (images).

Figure 8 describes training and validation curves for loss and accuracy for the best test results shown in Table 5. The curves represent one execution using all the classes and the most representative. The learning curves represent a compelling performance in classifying Android ransomware.

To verify the accuracy of the method, actual and predicted labels were compared using a confusion matrix (CM), which is a commonly used metric for evaluating the performance of classification models.

Figure 9 shows the results of the evaluation using the nine classes, which achieve an overall accuracy of 95.62%. It should be noted that the five classes with the most samples achieved good accuracy at 98.97%, which is expected because larger samples usually result in better accuracy, as shown in Figure 10. However, the test using all classes with few members in some affected the accuracy of the model. This indicates that the performance of the model was affected by an imbalance in the distribution of samples across different classes.

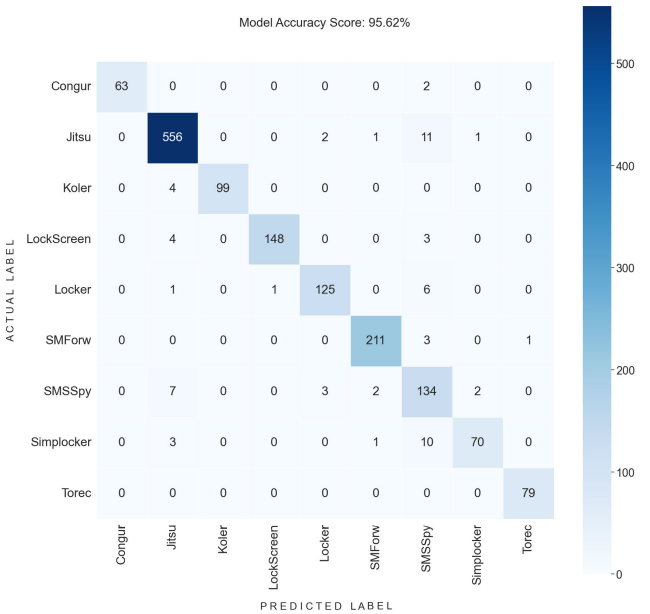

FIGURE 9. Confusion matrix using nine classes.

The metrics per class are described in Table 6 and Table 7 for the nine and five classes, respectively, for one of the ten executions.

TABLE 5. Metrics summary of the test performed using all the classes and using the top classes, each experiment was executed ten times, and the results represent the average.

No. Test	Image Type	Precision	Recall	F1-score	Accuracy	No.Classes	Epochs
1	Fuzzy Hashing	0.94	0.96	0.94	**95.62**	9	150
2	Fuzzy Hashing	0.93	0.95	0.94	94.51	9	100
3	Source Code - NLP	0.95	0.92	0.94	94.01	9	150
4	Source Code - NLP	0.93	0.93	0.93	93.18	9	100
5	All Source Code	0.95	0.95	0.92	94.42	9	150
6	All Source Code	0.89	0.90	0.91	91.33	9	100
7	Fuzzy Hashing	0.98	0.98	0.96	97.28	5	150
8	Fuzzy Hashing	0.98	0.98	0.98	**98.97**	5	100
9	Source Code - NLP	0.97	0.97	0.96	95.33	5	150
10	Source Code - NLP	0.95	0.95	0.95	95.17	5	100
11	All Source Code	0.94	0.94	0.96	95.02	5	150
12	All Source Code	0.94	0.94	0.94	94.41	5	100

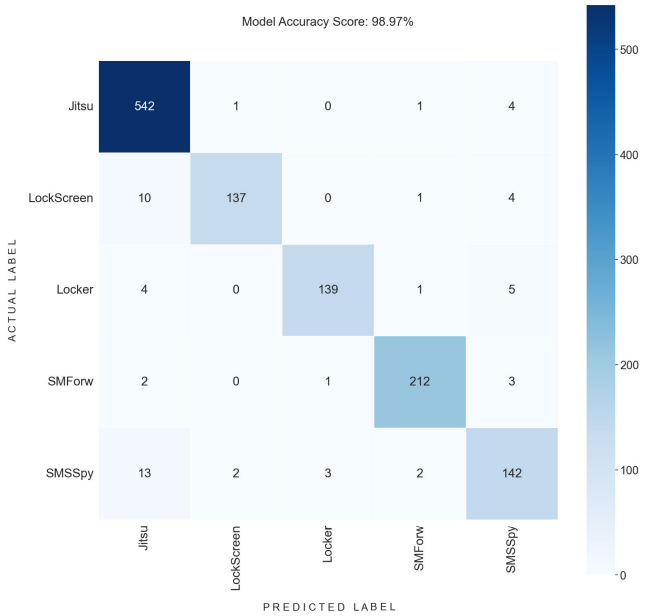

FIGURE 10. Confusion matrix using the top five classes.

TABLE 6. Metrics summary of the test performed using all the classes.

Class	Precision	Recall	F1-score	Support
Congur	1.0000	0.9692	0.9844	65
Jitsu	0.9670	0.9737	0.9703	571
Koler	1.0000	0.9612	0.9802	103
LockScreen	0.9933	0.9548	0.9737	155
Locker	0.9615	0.9398	0.9506	133
SMForw	0.9814	0.9814	0.9814	215
SMSSpy	0.7929	0.9054	0.8454	148
Simplocker	0.9589	0.8333	0.8917	84
Torec	0.9875	1.0000	0.9937	79
accuracy			0.9562	1553
macro avg	0.9603	0.9465	0.9524	1553
weighted avg	0.9587	0.9562	0.9568	1553

Researchers used multiple algorithms for Android malware classification in a state-of-the-art review. For example, SVM,

TABLE 7. Metrics summary of the test performed using the top classes.

Class	Precision	Recall	F1-score	Support
Jitsu	0.9592	0.9872	0.9730	548
LockScreen	0.9930	0.9342	0.9627	152
Locker	0.9650	0.9262	0.9452	149
SMForw	0.9907	0.9725	0.9815	218
SMSSpy	0.9791	0.9659	0.9874	162
accuracy			0.9897	1229
macro avg	0.9734	0.9692	0.9860	1229
weighted avg	0.9831	0.9826	0.9825	1229

KNN, and RF are some of the most commonly used algorithms in image classification.

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

A deep learning model (CNN) was trained for image-based classification. As discussed, the principal approach of this research is malware image-based classification using a CNN. The CNN model achieved up to 98.97% accuracy with an average of 98.16% (10 executions). It is expected that CNN will work well with images. Additional neural network models were evaluated using the images generated in those tests ResNet50 and VGG16, achieving 96.21% and 93.68% during the validation executed ten times, respectively. The three CNN models achieved nearly the same accuracy during the evaluation phase.

However, an additional machine learning model was trained with the same images, and the Support Vector Machine (SVM) achieved an average of 95.24%. Subsequently, the K-NN classification scheme was tested. However, no explicit training phase is required because the classification is computed based solely on the nearest neighbor in the training set [36]. The K-nearest neighbor (k-NN) algorithm with $k = 1$ using the generated images achieved an accuracy of 92.09%. 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 decompiled data (smali and AndroidManifest) in machine learning models such as k-NN, Random Forest (RF), Multilayer Perceptron (MLP), and SVM. In NLP, text is a document that can be used for text classification. Documents were used in the classifiers mentioned to classify malware. The accuracy obtained using k-NN, RF, MLP, and SVM are 89.78%, 92.63%, 92.76%, and 95.33%, respectively.

The cited works focused on malware analysis for Android and Microsoft Windows platforms for context in the field of malware analysis based on image visualization. Table 1 shows the studies related to Android malware analysis for comparison purposes. Table 8 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.

For the comparison tests, the models that required training and validation steps were executed ten times (CNN, SVM, RF, and MLP). Meanwhile, the k-NN was run once. Additional tests proved that the CNN model works well with images achieving up to 98.97% in accuracy metric and an average of 98.16%, which shows an accuracy over the other classifiers, also demonstrating that the method used to generate the images composed of fuzzy hashes was also accurate. Furthermore, k-NN and SVM attained an accuracy of over 90%, using the same images, although these algorithms are not effective in image classification. Moreover, the algorithms tested for text classification showed an increase in accuracy compared with SVM and k-NN using images but were not closer to the results obtained with CNN.

The comparison shows an increase in the accuracy metric of the CNN over the SVM and KNN using the same input data. Regarding state-of-the-art papers, the results reveal an increase in the classification task using the fuzzy hashing approach, converting fuzzy hashing into grayscale images.

Moreover, once the CNN is trained, the average time required to classify a new sample is 40 seconds. The decompiling process is time-consuming, which takes 50% (20 seconds), and text cleaning and text extraction take 10% (4 seconds). Data to grayscale image: 10% (4 seconds). The CNN used for the classification was 30% (12 seconds).

TABLE 8. 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)
CNN	Images	97.32	98.97	1.56
ResNet50	Images	97.02	96.21	3.23
VGG16	Images	93.12	93.68	3.92
KNN	Images	92.09	—	—
SVM	Images	95.45	95.24	2.91
KNN	Code/Text	89.78	—	—
RF	Code/Tex	96.88	92.63	2.27
MLP	Code/Tex	95.27	92.76	2.19
SVM	Code/Tex	94.17	95.33	3.61
CNN [2]	Images	94.00	—	—
SVM [16]	Images	93.24	—	—
SVM [27]	Images	97.00	—	—
SVM [25]	Images	96.00	—	—
ResNet [19]	Images	94.03	—	—
VGG16 [31]	Images	94.09	—	—

VI. DISCUSSION

A. CONCLUSION

This research presented a malware classification mechanism that converts malware files into images and uses CNN to distinguish between ransomware families. A new method for malware classification was proposed by computing the fuzzy hashing of decompiled source code into images. Images based on fuzzy hashing show good performance in ransomware classification.

This study tested whether ransomware families could be found by focusing on the features extracted from the visual representation. Experimental tests were performed using different ransomware families and revealed that the proposed method, whose accuracy is improved with the available samples, can be successfully used for malware classification. The algorithm achieved an overall average classification rate of 98.97% using five representative classes.

B. LIMITATIONS AND FUTURE WORK

1) DATASET

Due to the experimental conditions (corpus), a small dataset of just over 7,700 ransomware samples was used, a significant drawback of our project to find ransomware samples. More complex datasets with almost the same amount of samples per class will be applied for multiclass malware in future work.

2) SAMPLES

The method works with decompiled data if the samples are incomplete (not including all the files required by the application) or at least classes.dex and AndroidManifest.xml, the method will not work. On the other hand, the method has the limitation that corrupted or broken samples cannot be processed.

3) ROBUSTNESS

The proposed method achieved promising results by converting the APK into a grayscale image based on fuzzy hashing. Future work will perform perturbation in the images to prove the robustness of the method and whether the CNN can classify the samples correctly despite the noise.

ACKNOWLEDGMENT

The authors would like to thank Wuhan University for sharing the dataset used for research.

REFERENCES

[1] Google. *Secure an Android Device | Android Open Source Project*. Accessed: Mar. 2, 2023. [Online]. Available: https://source.android.com/docs/security/overview

[2] O. E. Kural, D. Ö. Sahin, S. Akleylek, E. Kiliç, and M. Ömüral, "Apk2Img4AndMal: Android malware detection framework based on convolutional neural network," in *Proc. 6th Int. Conf. Comput. Sci. Eng. (UBMK)*, Sep. 2021, pp. 731–734.

[3] *It Threat Evolution in Q3 2022. Mobile Statistics Securelist*. Accessed: Jun. 3, 2023. [Online]. Available: https://securelist.com/it-threat-evolution-in-q3-2022-mobile-statistics/107978/

[4] N. Sarantinos, C. Benzaïd, O. Arabiat, and A. Al-Nemrat, "Forensic malware analysis: The value of fuzzy hashing algorithms in identifying similarities," in *Proc. IEEE Trustcom/BigDataSE/ISPA*, Aug. 2016, pp. 1782–1787.

[5] D. He, X. Yu, S. Zhu, S. Chan, and M. Guizani, "Fuzzy hashing on firmwares images: A comparative analysis," *IEEE Internet Comput.*, vol. 27, no. 2, pp. 45–50, Mar. 2023.

[6] V. Roussev, "Data fingerprinting with similarity digests," in *Advances in Digital Forensics VI*, K.-P. Chow and S. Shenoi, Eds. Berlin, Germany: Springer, 2010, pp. 207–226.

[7] V. Roussev, "An evaluation of forensic similarity hashes," *Digital Investigation*, vol. 8, pp. S34–S41, Aug. 2011.

[8] N. Naik, P. Jenkins, N. Savage, L. Yang, T. Boongoen, and N. Iam-On, "Fuzzy-import hashing: A malware analysis approach," in *Proc. IEEE Int. Conf. Fuzzy Syst. (FUZZ-IEEE)*, Jul. 2020, pp. 1–8.

[9] V. Roussev and C. Quates. (2013). *The Sdhash Tutorial*. New Orleans, LA, USA, Accessed: May 20, 2023. [Online]. Available: http://roussev.net/sdhash/tutorial/sdhash-tutorial.html

[10] B. Valosek. (2010). *Apktool*. [Online]. Available: https://ibotpeaches.github.io/Apktool/

[11] G. Sidorov, *Vector Space Model for Texts and the TF-IDF Measure*. Cham, Switzerland: Springer, 2019, pp. 11–15.

[12] A. Damodaran, F. D. Troia, C. A. Visaggio, T. H. Austin, and M. Stamp, "A comparison of static, dynamic, and hybrid analysis for malware detection," *J. Comput. Virol. Hacking Techn.*, vol. 13, no. 1, pp. 1–12, Dec. 2015.

[13] J. Geremias, E. K. Viegas, A. O. Santin, A. Britto, and P. Horchulhack, "Towards multi-view Android malware detection through image-based deep learning," in *Proc. Int. Wireless Commun. Mobile Comput. (IWCMC)*, May 2022, pp. 572–577.

[14] S. L. S. Darshan and C. D. Jaidhar, "Windows malware detector using convolutional neural network based on visualization images," *IEEE Trans. Emerg. Topics Comput.*, vol. 9, no. 2, pp. 1057–1069, Apr. 2021.

[15] S. Gülmez and I. Sogukpinar, "Graph-based malware detection using opcode sequences," in *Proc. 9th Int. Symp. Digit. Forensics Secur. (ISDFS)*, Jun. 2021, pp. 1–5.

[16] J. Singh, D. Thakur, T. Gera, B. Shah, T. Abuhmed, and F. Ali, "Classification and analysis of Android malware images using feature fusion technique," *IEEE Access*, vol. 9, pp. 90102–90117, 2021.

[17] X. Ke and Y. X. Hui, "Android malware detection based on image analysis," in *Proc. IEEE 2nd Int. Conf. Inf. Technol., Big Data Artif. Intell. (ICIBA)*, vol. 2, Dec. 2021, pp. 295–300.

[18] L. Xin, L. Chao, and L. He, "Malicious code detection method based on image segmentation and deep residual network RESNET," in *Proc. Int. Conf. Comput. Eng. Appl. (ICCEA)*, Jun. 2021, pp. 473–480.

[19] R. A. Abhesa, Hendrawan, and S. J. I. Ismail, "Classification of malware using machine learning based on image processing," in *Proc. 15th Int. Conf. Telecommun. Syst., Services, Appl. (TSSA)*, Nov. 2021, pp. 1–4.

[20] X. Jin, X. Xing, H. Elahi, G. Wang, and H. Jiang, "A malware detection approach using malware images and autoencoders," in *Proc. IEEE 17th Int. Conf. Mobile Ad Hoc Sensor Syst. (MASS)*, Dec. 2020, pp. 1–6.

[21] J. Chen, X. Jia, C. Zhao, W. Zhang, and Q. Huang, "Using the RGB image of machine code to classify the malware," in *Proc. IEEE 5th Int. Conf. Cloud Comput. Big Data Analytics (ICCCBDA)*, Apr. 2020, pp. 542–549.

[22] J.-S. Kim, W. Jung, S. Kim, S. Lee, and E. T. Kim, "Evaluation of image similarity algorithms for malware fake-icon detection," in *Proc. Int. Conf. Inf. Commun. Technol. Converg. (ICTC)*, Oct. 2020, pp. 1638–1640.

[23] F. Naït-Abdesselam, A. Darwaish, and C. Titouna, "An intelligent malware detection and classification system using apps-to-images transformations and convolutional neural networks," in *Proc. 16th Int. Conf. Wireless Mobile Comput., Netw. Commun. (WiMob)*, Oct. 2020, pp. 1–6.

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

[25] 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.

[26] P. Yujie, N. Weina, Z. Xiaosong, Z. Jie, H. Wu, and C. Ruidong, "End-To-end Android malware classification based on pure traffic images," in *Proc. 17th Int. Comput. Conf. Wavelet Act. Media Technol. Inf. Process. (ICCWAMTIP)*, Dec. 2020, pp. 240–245.

[27] J. Jiang, Z. Liu, M. Yu, G. Li, S. Li, C. Liu, and W. Huang, "HeterSupervise: Package-level Android malware analysis based on heterogeneous graph," in *Proc. IEEE 22nd Int. Conf. High Perform. Comput. Communications; IEEE 18th Int. Conf. Smart City, IEEE 6th Int. Conf. Data Sci. Syst. (HPCC/SmartCity/DSS)*, Dec. 2020, pp. 328–335.

[28] S. Yoo, S. Kim, and B. B. Kang, "The image game: Exploit kit detection based on recursive convolutional neural networks," *IEEE Access*, vol. 8, pp. 18808–18821, 2020.

[29] I. Baptista, S. Shiaeles, and N. Kolokotronis, "A novel malware detection system based on machine learning and binary visualization," in *Proc. IEEE Int. Conf. Commun. Workshops (ICC Workshops)*, May 2019, pp. 1–6.

[30] P. Zhang, B. Sun, R. Ma, and A. Li, "A novel visualization malware detection method based on SPP-net," in *Proc. IEEE 5th Int. Conf. Comput. Commun. (ICCC)*, Dec. 2019, pp. 510–514.

[31] S. Yang, W. Chen, S. Li, and Q. Xu, "Approach using transforming structural data into image for detection of malicious MS-DOC files based on deep learning models," in *Proc. Asia–Pacific Signal Inf. Process. Assoc. Annu. Summit Conf. (APSIPA ASC)*, Nov. 2019, pp. 28–32.

[32] S. O'Shaughnessy, "Image-based malware classification: A space filling curve approach," in *Proc. IEEE Symp. Visualizat. Cyber Secur. (VizSec)*, Oct. 2019, pp. 1–10.

[33] M. Yang and Q. Wen, "Detecting Android malware by applying classification techniques on images patterns," in *Proc. IEEE 2nd Int. Conf. Cloud Comput. Big Data Anal. (ICCCBDA)*, Apr. 2017, pp. 344–347.

[34] T. Wang and N. Xu, "Malware variants detection based on opcode image recognition in small training set," in *Proc. IEEE 2nd Int. Conf. Cloud Comput. Big Data Anal. (ICCCBDA)*, Apr. 2017, pp. 328–332.

[35] 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.

[36] S. Yajamanam, V. R. S. Selvin, F. Di Troia, and M. Stamp, "Deep learning versus gist descriptors for image-based malware classification," in *Proc. 4th Int. Conf. Inf. Syst. Secur. Privacy*, 2018, pp. 553–561.

[37] W.-C. Huang, F. D. Troia, and M. Stamp, "Robust hashing for image-based malware classification," in *Proc. 15th Int. Joint Conf. e-Business Telecommun.*, 2018, pp. 451–459.

[38] N. Chavan, F. D. Troia, and M. Stamp, "A comparative analysis of Android malware," 2019, *arXiv:1904.00735*.

[39] Virustotal. *Virustotal*. Accessed: Oct. 31, 2023. [Online]. Available: https://www.virustotal.com/gui/home/upload

[40] *App Manifest Overview | Android Developers*. Accessed: Mar. 15, 2023. [Online]. Available: https://developer.android.com/guide/topics/manifest/manifest-intro

[41] J. Chen, C. Wang, Z. Zhao, K. Chen, R. Du, and G.-J. Ahn, "Uncovering the face of Android ransomware: Characterization and real-time detection," *IEEE Trans. Inf. Forensics Security*, vol. 13, no. 5, pp. 1286–1300, May 2018.

[42] A. H. Lashkari, A. F. A. Kadir, L. Taheri, and A. A. Ghorbani, "Toward developing a systematic approach to generate benchmark Android malware datasets and classification," in *Proc. Int. Carnahan Conf. Secur. Technol. (ICCST)*, Oct. 2018, pp. 1–7.

[43] D. S. Keyes, B. Li, G. Kaur, A. H. Lashkari, F. Gagnon, and F. Massicotte, "EntropLyzer: Android malware classification and characterization using entropy analysis of dynamic characteristics," in *Proc. Reconciling Data Analytics, Autom., Privacy, Secur., Big Data Challenge (RDAAPS)*, May 2021, pp. 1–12.

[44] A. Rahali, A. H. Lashkari, G. Kaur, L. Taheri, F. Gagnon, and F. Massicotte, "DIDroid: Android malware classification and characterization using deep image learning," in *Proc. 10th Int. Conf. Commun. Netw. Secur.* New York, NY, USA: Association for Computing Machinery, Nov. 2020, pp. 70–82.

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

HORACIO RODRIGUEZ-BAZAN received the B.Sc. degree in computer engineering and the Master of Science (M.Sc.) degree (Hons.) in computer engineering from Instituto Politécnico Nacional (IPN), Mexico City, Mexico, in 2008 and 2019, respectively. He is currently pursuing the Ph.D. degree with Centro de Investigación en Computación (CIC), IPN. His research interests include artificial intelligence (AI) applied to cybersciences, mainly in android malware analysis and computer forensics.

GRIGORI SIDOROV is currently a Full Professor and a Researcher with Centro de Investigación en Computación (CIC), Instituto Politécnico Nacional (IPN), Mexico City, Mexico. He has coauthored more than 190 scientific publications with an H-index of 30. His research interests include computational linguistics, automatic word processing, and the application of machine learning methods to natural language processing tasks. In addition, he is a regular member of the Mexican Academy of Sciences and the National Researcher of Mexico (SNI) Level 3 (highest). He is the Editor-in-Chief of *Computación y Sistemas* (ISI-Thomson Web of Science [SciElo and CORE Collection (Emerging Sources)], Scopus, DBLP, and the Index of Excellence of CONAHCYT).

PONCIANO JORGE ESCAMILLA-AMBROSIO (Senior Member, IEEE) received the B.Sc. degree in mechanical electrical engineering and the M.Sc. degree (Hons.) in electrical engineering from the National Autonomous University of Mexico (UNAM), in 1995 and 2000, respectively, and the Ph.D. degree from the University of Sheffield, U.K., in January 2004. From 2003 to 2010, he was a Research Associate with the Aerospace Engineering Department and the Computer Science Department, University of Bristol, U.K. From 2010 to 2011, he was a Research Associate with the Department of Electronics, National Institute of Astrophysics Optics and Electronics, Mexico. From 2011 to 2013, he was the General Director of Innovation and Development with the Scientific Division, Secretariat of the Interior, Mexico. He is currently a Researcher with the Computing Research Centre, National Polytechnic Institute, Mexico. He has published more than 100 publications in journals, conference proceedings, and book chapters. His research interests include the Internet of Things, smart cities, sensors/data fusion, wireless sensor networks, cybersecurity, neuro-fuzzy networks, and intelligent control.