International Journal of Computing and Digital Systems
ISSN (2210-142X)
Int. J. Com. Dig. Sys. 14, No.1 (Aug-2023)

http://dx.doi.org/10.12785/ijcds/140145

SVDroid: Singular Value Decomposition with CNN for Android Malware Classification

Franklin Tchakounté[1,2,5,6], Raïssa Edwige Feuyo Manfouo[1], Jean Louis Fendji Kedieng Ebongue[3], Didier Bassolé[4] and Marcellin Atemkeng[5]

[1]*Department of Mathematics and Computer Science, Faculty of Science, University of Ngaoundéré, Ngaoundéré, Cameroon*
[2]*SFTI, National School of Chemical Engineering and Mineral Industries, University of Ngaoundéré, Ngaoundéré, Cameroon*
[3]*Department of Computer Engineering, University Institute of Technology, University of Ngaoundéré, Ngaoundéré, Cameroon*
[4]*Laboratory of Mathematics and Computer Science, University of Joseph Ki-Zerbo, Ouagadougou, Burkina Faso*
[5]*Department of Mathematics, Rhodes University, Grahamstown 6140, South Africa*
[6]*Cybersecurity with Computational and Artificial Intelligence Research Group, Faculty of Science, University of Ngaoundéré, Ngaoundéré, Cameroon*

Received 28 Nov. 2022, Revised 10 Jul. 2023, Accepted 13 Jul. 2023, Published 01 Aug. 2023

Abstract: During the last decade, Android malware detection has ever preoccupied researchers. The rhythm of creation of sophisticated malicious techniques obliges researchers to look for robust countermeasures. Singular Value Decomposition (SVD) is a powerful in signal processing for image compression. Unlike image-based deep learning detection that directly take images made of .dex properties, SVD-based processing of images is investigated in this work to recognize maliciousness. For that, we associate n-gram for completeness of properties extraction and Simhash for uniqueness of application signatures. SVDroid is proposed to transform applications based on n-gram and Simhash into 32 x 32 grayscale images, then to apply SVD to only remain with valuable features. Machine and deep learning algorithms are applied to automatically extract knowledge to profile applications. Experiments have been conducted on 135 malware and 135 benign applications. Results reveal that the association n-gram, Simhash along with the learning algorithm process positively contributes to the profiling. CNN outperforms six machine learning algorithms with an accuracy of 88.55% and an AUC of 93.45% on average. A study demonstrates that SVDroid is able to improve CNN-based image processing approaches. Exploitation of compression techniques such as SVD should be further studied in mobile malware detection.

Keywords: Dex, Deep Learning, Malware, Images, SVD, Sim-hash, n-gram, CNN

1. Introduction

Android still dominates the smart phone market and is predicted to remains popular next years. According to Statista [1], its market share is about 71.93%. Android was reported in 2019 as the most vulnerable operating system.

As a consequence, this system gained a lot of attention from attackers. They design sophisticated techniques to bypass Google filtering measures. They also mislead popular applications to redirect sensitive resources requests such as location, credit card, and contact information [2]. Malicious people have so far, created malware nests that they maintain or evolve through advanced techniques.

Protection countermeasures are on demand with a specific attention in the industry [3]. Authors increasingly rely on designing feature engineering to be applied on artificial intelligence algorithms such as machine learning and deep learning to retrieve knowledge [4], [5]. These algorithms perform on structures based on static features [6], [7] such as permissions [8], [9], Application Programming Interface (API) [10], [11] and dynamic features such as system calls [12], [13], network traffic [14] as well as combination of static and dynamic features [15], [16].

However, these attempts although interesting, generally lack generalization meaning that related models easily recognize malware as benign. The concern is therefore to find reliable structure of application which provides kernel features representing the malicious facets of malware. To this end, authors propose application transformation into image due to two reasons: (i) image is an object which easily keeps representative properties related to its pixels [17], [18], [19], (ii) there are several mathematical tools to

E-mail address: f.tchakounte@cycomai.com, feuyo.manfouo@aims-cameroon.org, lfendji@gmail.com, dbassole@gmail.com, m.atemkeng@ru.ac.za

process images.

Following this direction, authors propose approaches that transform an .apk into images based on elements inside the packages such as manifest, byte code, and source code. They transfer these images as inputs to Convolutional Neural Network (CNN), a renowned deep learning technique in image recognition [20]. Most of these authors perform simple matrix transformations and directly apply the learning algorithms[21]. However, pre-processing on these images can be very useful to extract the relevant characteristics of the pixels.

In this work, SVDroid, is proposed to take advantage of the SVD compression subtleties for the extraction representative structure and key properties [22], [18]. The proposed approach first applies n-gram and Simhash to .dex opcodes to obtain images which are then processed with SVD to capture relevant features.

This study contributes into the two following points:

- We introduce SVDroid, a novel feature engineering approach relying on two key processes: image transformation based on n-gram and Simhash and image processing to extract key features relying on SVD. The proposed approach does not take directly converted .dex into image to CNN classification. Unlike, SVDroid suggests to feature extracts properties from that images based on robust image processing such as SVD.
- Experiments have been conducted on 135 malware and 135 benign transformed in samples of 32 x 32 grayscale images. Results reveal that the variation of the n-gram hyperparameter impacts the detection performance and that CNN is the most efficient algorithm reaching on average Area Under Curve (AUC) of 93.45% against with six machine learning algorithms. A study with similar works reveals that SVDroid improves performance.

The remaining of this paper is structured as follows. Feature engineering approaches for Android malware detection are presented in section 2. Notions about Android application structure, n-gram, Simhash and SVD are described in section5. In section 4, the proposed approach architecture and components are presented. In section 6, experiments are performed and results are discussed. We conclude and provide perspectives at the end the document.

2. Related works

In general, solutions against malware fall into static, dynamic and hybrid categories. The first category does not run the application. It extracts static features [23] such as permission, code related information, API and based on them, identifying the nature of application. The second category observes activities during run-time to make decision about the application nature[24]. The third category combines properties from static and dynamic[25]. Whatever the category is, authors structure applications into objects manipulable by Artificial Intelligence AI techniques. We are interested in studying works based on image processing and not image processing, to capture malware knowledge.

Permissions controlling access to resources, is the most popular feature in malware detection [6]. Several authors use permissions to facilitate application reconnaissance [26], [27]. Features related to malware installation processes such as repackaging, updating, privilege escalation are exploited in [28]. In [6], additional features are considered: financial charges features such as Short Message Service (SMS) and phone calls as well as personal information features such as phone number. Code-based features such as byte-code frequency, opcode, or opcode sequence are also used for the same purposes. Additional features are exploited. In [29], authors exploit data-flow graph (DFG) and control-flow graph (CFG) whereas in [30], [31], authors identify sequence of opcodes to profile families of applications. Dynamic features include features which require runtime of the application to be retrieved, such as network traffic. In [32], the authors collect system calls during application execution. HTTP network traffic is utilized by Aresu et al. [33] to profile an application. DroidBox [34] is used in [35] to generate dynamic information exploited as features. Authors in [36], classify applications based on consumption of resources. Combination of features can be exploited. Mantoo and Khurana [24] use this principle on permissions, system calls and intrinsic features with linear discriminant analysis as feature engineering technique. In [37], the authors associate log features related to file I/O, network flows, and cryptographic usage. In [38], the authors exploit the DroidBox tool [34] to have dynamic features related to network traffic.

The aforementioned approaches are subject to some constraints. They rely on the right selection of features. This capacity belongs which belongs to the expert who should have enough knowledge to reliably select features. In this case, the high number of data often tends to provide poor generalized models. On the contrary, an image is a structure from matrices that have enough features (pixels as an example) and proven theories to manipulate. Authors start investigating on this direction.

In [39], a system is proposed for identifying Android malware. Ten types of images are created by inserting domain knowledge to the image transformation. The process of conversion the Dalvik opcode and API information then transforms .dex classes into fractal shaped Hilbert curve images. Using CNN model including two layers, the system was accurate with 92%. In [40], authors rely on color images. They applied different CNN techniques that have been successfully exploited in ImageNet Large-Scale Visual Recognition Challenge (ILSVRC). Their system is accurate with 98.428%. Unlike, the generation of images did not consider the structure of .dex file.

In [41], image similarity is combined with CNN to recognize unknown malware. Authors have experimented on two datasets of 128 × 128 images. The first contains 9458 grayscale images made of 25 malware families and the second contains 3000 benign software. Their approach has been accurate with 98%. In [42], CNN is applied on gray-scale images converted from executable files of malicious code. Then, Non-dominated Sorting Genetic Algorithm II (NSGA-II) is applied to overcome data imbalance in malware families. In [43], authors take out the byte-code file from the APK. The byte-code file is then translated into a two-dimensional matrix which is taken as input in convolution neural network (CNN). DeepVisDroid is proposed in the paper [44]. Authors construct four image datasets from four elements: the Manifest.xml file, the .dex code files, association of manifest and Resources.arsc files and Manifest, association of Resources.arsc and .dex files. They provide hybrid approach considering each dataset in CNN.

Table shows a summary of papers which are based on CNN classification based on image features, in terms of dataset sizes and accuracy.

Table I shows that authors mainly rely on image processing because they rely on CNN. Due to that, they directly convert the byte-code into matrix, and then apply CNN on matrices. Unlike, this work proposes that images require prior deep processing to extract the relevant matrix. According to [21], SVD is a renowned tool used in image compression. In linear algebra, SVD converts a matrix into different sub-matrices while giving out geometric structure and relevant properties of the input matrix. This study investigates whether SVD is able to bring robustness to proposals based on image-based processing. To obtain original matrices for each sample, we use n-gram on opcode sequences for completeness of representation and then Simhash for uniqueness of hashes to each sample.

3. Background

In this section, notions about application internals[51], n-gram[52], Simhash[53], and SVD[22] are presented.

A. *Basics of application structure*

As depicted in Figure 1, an Android application is composed of several files and folders all grouped together in an archive file called .apk. This file is used to distribute and install the application on Android. More specifically, an application contains the manifest file called *AndroidManifest.xml* which contains application metadata such as package names, access rights, definition of components including activities, services, broadcast receivers and content providers. It also contains a folder called *res* that contains static resources of such as images, icons, texts and user interfaces. The directory *lib* contains the compiled code of different libraries used in the application. There is an important file called *.dex* meaning Dalvik executable (DEX) referring to the code that will be executed by the virtual machine. Finally, there is *META-INF*, a directory

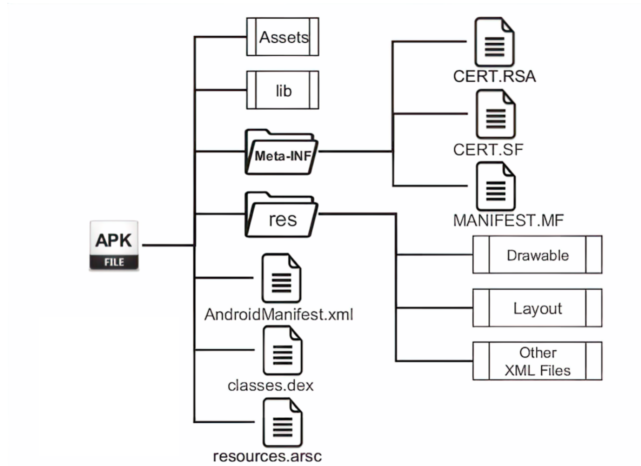


Figure 1. Application Structure [54]

that contains cryptographic signatures used to verify and certify developer identities.

As shown in Figure 2, the .dex comprises classes compiled to fit into the environment of limited resources. The .dex file has four sections: header section, ids sections, class_defs section and data section. The header section contains top-level information including file size, signature, the offset and size of the other sections. The ids sections has the list of identifiers including strings, types, fields and methods. The class_defs section has the definition of classes including class, superclass, and interface types as well as the source file name. The data section has the class data and runtime source codes which describe application behavior. code_item includes certain information about methods as well as corresponding byte codes. map_list saves the following data: size and offset of sections and DEX contents. Each method contains human-readable Dalvik bytecode (i.e. statements). Each statement has one opcode and several operands. Attackers usually target this file to insert malicious payloads that launch bad activities. They therefore exploit the data section.

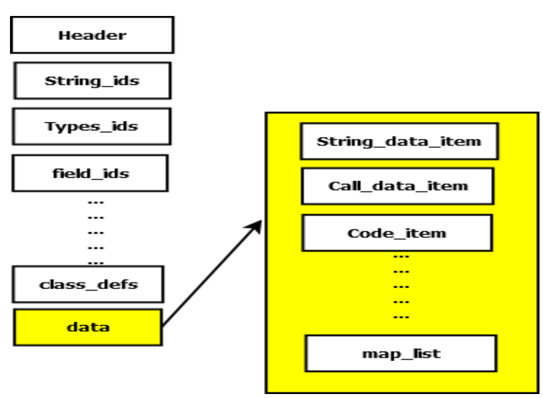


Figure 2. DEX file Format

TABLE I. Research dealing with image processing and techniques of machine learning and deep learning. NA: Not available

Author	Feature and algorithm	Dataset (malware; goodware)	Accuracy
[45]	Grayscale image, CNN	(5.377 ; 6.249)	98.02%
[46]	Grayscale images, K-NN, grabar kernel	9.459	98%
[47]	Grayscale images, CNN	(10.805 ; *NA*)	99.260 %
[48]	Images (*type NA*), CNN based on VGG-16	(9.339 and 21.741 (malware))	98.52 %; 99.97%
[49]	Images processing, CNN	(12000)	96%
[50]	Grayscale images (32x32 pixel),DT	(9342;*NA*)	88%
	RF		91.6%
	Perceptrons		90.5 %
[43]	Grayscale images (*size NA*), CNN	(3962;1000)	86%
[44]	Grayscale images, CNN	04 datasets of (4850 benign and 4850 malware) each	98%

B. N-gram

N-gram is a popular approach for feature engineering in Natural language processing [52]. N-gram derives contiguous chains of items with length *n*, such as sequence of words, characters etc. The popular n-gram models in text mining are word-based n-grams. An 1-gram refers to *unigram*, 2-gram refers to *bigram* or *digram* and 3-gram refers to *trigram*. N-gram is useful for malware detection since the manifest has text contents. In this case, N-gram represents given data into several words. Then, a new word is made as a composition of n-size serial words to have the context information. For example, 2-gram in the sentence *God makes Franklin happy* gives *God makes*, *makes Franklin* and *Franklin happy*. N-gram is exploited in this work due to the following reason. Malware take advantage of sequence of opcodes to be harmful in a specific order [55]. Therefore, n-gram will retrieve the possible sequences including the malicious ones.

C. SVM

Binary classification can be realized by support vector machines (SVM)[56]. Its objective is to find the optimal discriminating hyperplane which separates both classes. In simple words, SVM finds the separation hyperplane which maximizes the margins from both classes. The margin refers to the distance between the separator and some close points. The latter are called support vectors since they control the separator. An infinite number of separators can be obtained but the optimal one is kept. Figure 3 shows a case with two line separators. The one in red is not selected since the margin is not maximal. The one in black is therefore selected.

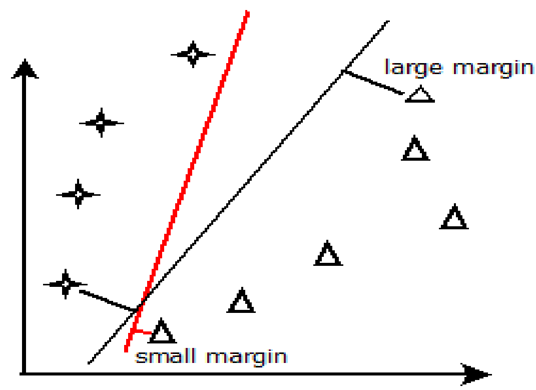


Figure 3. SVM illustration

D. Sim-hash

Simhash is a hashing mechanism designed by Moses Charikar[57], used for identification of text structures and identification of duplicated web pages [58]. Using Simhash guarantees that similar contents will have similar hash values with very low collision. This technique includes the following steps: (i) structuring given data (ii) hashing all structured data (iii) computing weighted vectors into a calculated value and (iv) Translating the calculated value into a binary value.

Simhash takes a specified dimension bits vector V to model a document. More specifically, the n-th bit of V is determined based on the calculation of the hashed value of the whole keywords in the document. If the number of hashed values whose n-th bit is 1 is greater than the number of hashed values whose n-th bit is 0, then the n-th bit of V is 1. Otherwise the n-th bit of V is 0. In this work, we rely on opcodes extracted from each .apk. Due to the fact that the size of opcode chains is variable, it is difficult to compare similarity of sequences of distinct length. Simhash is exploited to compare sequence similarity.

Algorithm 1 presents the Simhash process and Figure 4 illustrates Simhash with an example.

Algorithm 1 Simhash Algorithm

Require: opcodes sequence Seq, n-bit hash algorithm;
Ensure: n-bit Simhash values;

```
Begin ;
Initialize a n-bit vector V as zero vector;
Initialize a binary number S as zero;
for each opcode to Seq do
    b =hash(opcode);
    for i = 1 to n do
        if b[i] == 1 then
            v[i]+ = 1;
        else
            v[i]− = 1;
        end if
    end for
end for
for i = 1 to n do
    if v[i] > 0 then
        s[i Give by the formula :]= 1;
    else
        s[i] = 0
    end if
end for
End
```

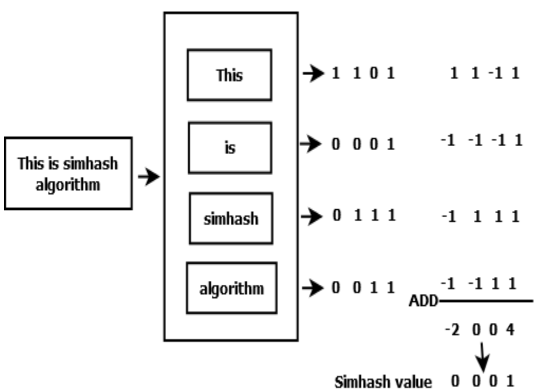


Figure 4. Simhash Algorithm

E. Singular Value Decomposition

Singular Value Decomposition (SVD) is a mathematical theory used in various applications, which is used in various domains based on image processing[22]. SVD has been popular due to its strength to find key properties in images. This is the main justification that this work targets SVD. The operations on images making SVD powerful are: packing maximum energy, finding least square solutions, computing pseudo-inverse of a matrix as well as multivariate analysis [59], [60].

SVD relies on matrix factorization which is the representation of a matrix into a product of matrices. SVD takes a matrix A and decomposes it into the product of three matrices. Suppose that an image I refers to a matrix A of $m \times n$ real with rank r such that $r < m, r < n$. An SVD of A is identified as follows:

$$A = U_A S_A V_A^T \tag{1}$$

where

- A : $m \times n$ matrix;
- U_A : $m \times m$ orthogonal matrix;
- S_A : $m \times n$ diagonal matrix;
- V_A : $n \times n$ orthogonal matrix.

According to [61], SVD is popular in several of applications due to the following properties:

- The singular values (S_A) indicate the intensity of the image whereas the singular vectors (U_A and V_A) accentuate geometrical characteristics of that image.
- S_A has a good stability means that the singular values of the image keep constant with time independently from any small perturbation.
- Updating singular values during the reformation step has an incidence on the image quality. They are therefore useful to represent kernel part of the image.

4. SVDroid approach

As shown in Figure 5, SVDroid is made up of three main modules: Data preprocessing, feature engineering, and CNN based training. The first module takes the dataset of malicious and benign applications and transform them into $32x32$ grayscale images from decompiling into .dex to the serial applications of n-gram and hashing. During feature engineering which requires as input the dataset of images, we generate the representative singular matrix based on SVD exploited as the profile for each sample. CNN based training aims at learning from these images to predict the class of an unknown application.

A. Preprocessing

In this module, the transformation of the applications into grayscale images is realized. We describe this process which is carried out in several sub-processes represented in Figure 6.

1) Data collection

SVDroid requires datasets of malware and goodware .apk collected from reliable various repositories. A script is necessary to ensure that there are no duplicates within each category of samples. Since we seek models to assign the predicted class to any sample, we therefore proceed to labelling of the whole samples as 0 for malware and 1 for goodware.

2) Extraction of .dex features and conversion into images

The aim is to take out .dex opcodes from different .apk. It is realized as follows: a package is decompressed and a

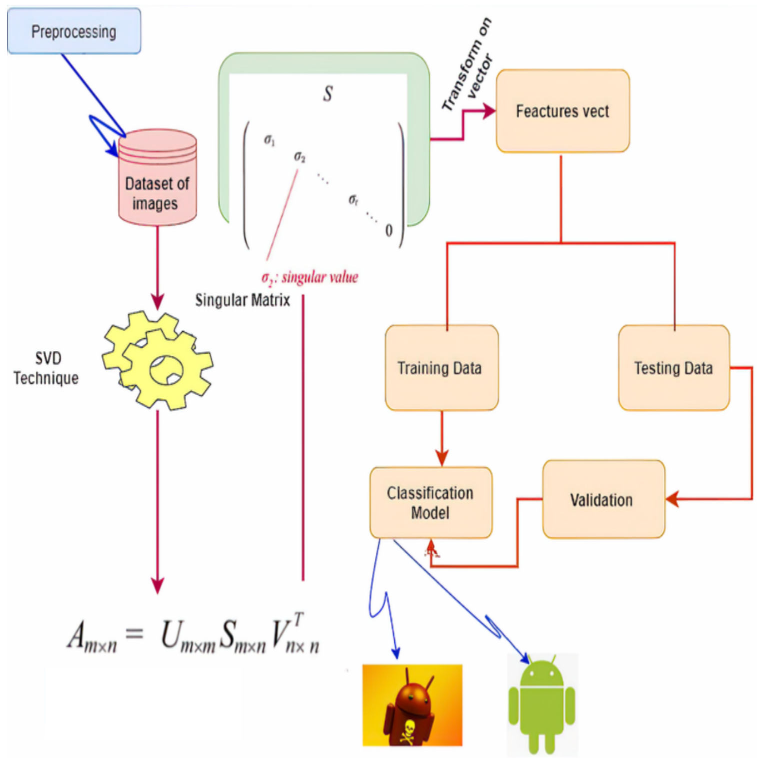


Figure 5. General procedure of SVDroid

readable .dex is extracted using Smali[62]. The sequence of opcodes have been generated from the .dex and are split based on separators using n-gram. Unlike [63], opcodes are extorted as a single sequence as a whole. Each chain is then encoded to a m-bit vector of 0 and 1 using Simhash. It is worth mentioning that the hyperparameter n in n-gram and the hashing vector size m are experimentally set.

Each sample is therefore encoded into binary Simhash values with identical size. Each Simhash bit is converted to a pixel value such that 0 gives 0 and 1 gives 255. Based on this arrangement of the n pixels into a matrix, the Simhash structure is therefore converted into a grayscale image. Figure 7 illustrates an example.

The image size highly depends on the Simhash structure size. Table II presents certain corresponding sizes in some hash algorithms.

TABLE II. Hashing techniques versus image sizes

Algorithm	Image size
MD5 (64 bits)	8×8
MD5 (128 bits)	8×16
SHA-256	16×16
SHA-512	16×32

The image size parameter is chosen such that related experiments provide better performance. The images obtained from n-gram and Simhash constitute the new datasets.

B. Feature extraction

The preprocessing module provides as output the set of grayscale images. Unlike existing works which directly consider these images for learning, we intuitively believe that it is possible to extract interesting properties from images before the training process. In this work we rely on SVD, that powerful image processing which extracts three key matrices from an original one. We transform the dataset of images into raster images consisting of a matrix of juxtaposed colored points. An example of such images is shown in Figure 8. Each raster image is transformed into matrix and SVD is fitted to that matrix to obtain its factorisation. The singular matrix extracted is retained to profile an application.

C. CNN training

In this paper, deep learning with a convolutional neural network has been provided for profiling Android applications as malware and benign. It includes two processes as depicted in Figure 9: mapping of malicious codes to grayscale image and CNN formalization on grayscale images for detection.

In the first process, binary files are converted in grayscale images based on the feature engineering approach. CNN is then used to recognize applications based on those images. Relying on image classification, an automatic recognition is realized and malicious apps are classified.

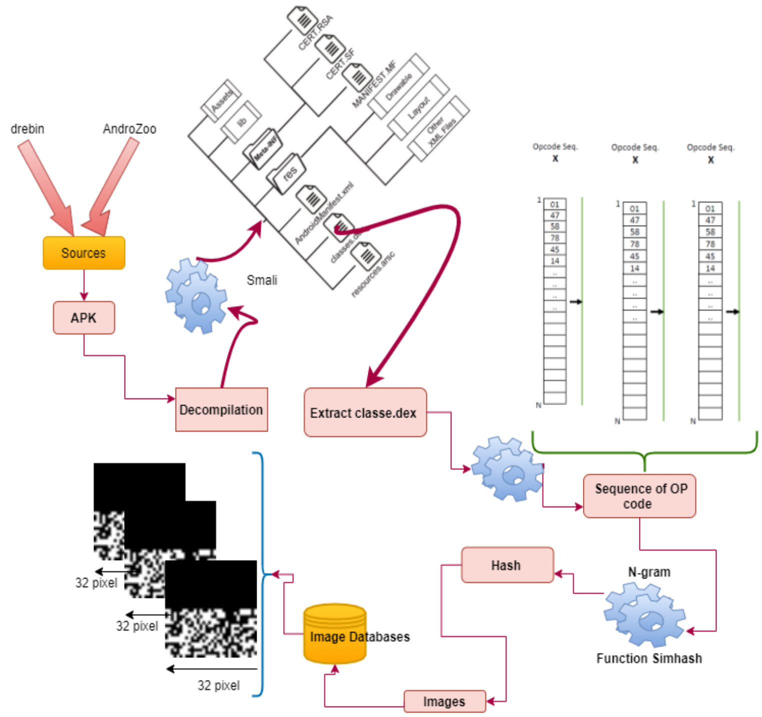


Figure 6. Preprocessing.

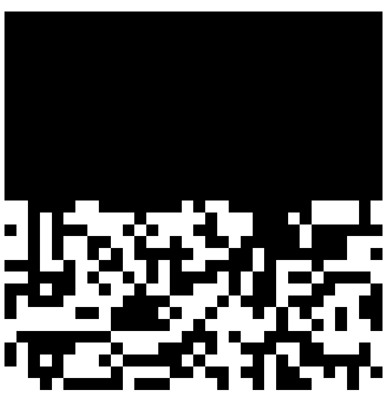


Figure 7. Malware images

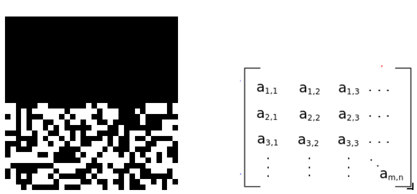


Figure 8. Application image into to matrix

CNN provides the convolution layer (C) and the pooling layer (P) (see Figure 9). The neural network in our case, has four C with a **Relu activation function** and four P with a *dropout layer* of 0.5. The next step is that the output from Cs is flattened and the **fully connected layers** are used.

By establishing a local connectivity pattern between units of neighbor layers, CNNs can use spatially local correlation because each unit is connected to a limited number of the input units.

Each trainable kernel in a convolutional layer is made up of a layer of connection weights with an input that is the size of a small two-dimensional patch.

One unit is produced as the result. Each kernel is convolved across the input patch's width and height during the forward phase to provide a two-dimensional feature map of the kernel. Whether or not the kernel is convolved through all places relies on the step size (stride). As an illustration, when the stride is 2, the kernel will jump two pixels at a time as we move it. This will result in geographically reduced output quantities[43].

One unit is produced as the result. Each kernel is convolved across the input patch's width and height during the forward phase to provide a two-dimensional feature map of the kernel. Whether or not the kernel is convolved through all places relies on the step size (stride). As an illustration, when the stride is 2, the kernel will jump two pixels at a time as we move it. This will result in geographically reduced output quantities[43].

The CNN algorithm has thick layers after convolution

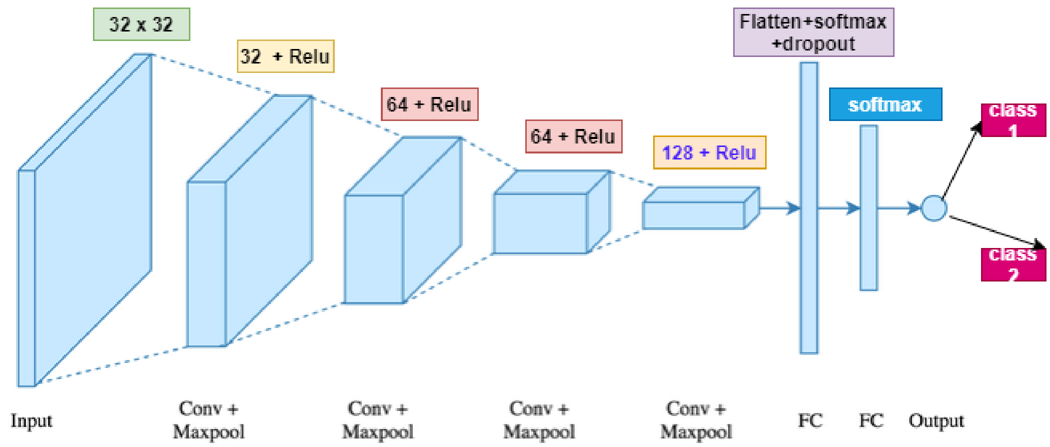


Figure 9. Overview of the proposed CNN method

and pooling. All of the nodes in the preceding layer are completely connected to all of the nodes in the dense layer. The convolutional layer or pooling layer uses the dense layer to integrate local information with category discrimination. A Dropout layer that enhances the model's ability to generalize by avoiding overfitting typically follows this kind of layer. The *activation function* of the dense layer used in this work is *ReLU* which is defined as $f(x) = max(0, x)$. The output of the last layer used is the *softmax* regression. Due to its popularity for multi-class classification and for the purpose of optimization, both the *Adam* optimizer and the *Cross entropy* loss function are employed. The overall CNN training has been automated using Keras to implement and manipulate the deep learning models.

5. Experimentations, results and discussions

We put our strategy, SVDroid, into practice and ran various tests to gauge how well it would detect malware. We describe several components of our experiment in this section and, based on the results, discuss its reliability.

A. Datasets

We collected 270 samples including 135 malware and 135 goodware from various malware and goodware repositories.The malware was collected through the Drebin database[64], which is popular set of malware families. The dataset of goodware was collected through the AndroZoo platform [65] while exploiting available API according to guidelines specified in to the official Androzoo website[1].

B. Evaluation metrics

In order to evaluate efficiency of SVDroid, five metrics are exploited [20]: confusion matrix, accuracy, and AUC.

Confusion matrix: The confusion matrix is the basic structure used to highlight the classification performance of the labeled samples. In other words, it indicates whether a classifier works well and how reliable it is. Each class in the classifier is represented by a column and a row. The row indicates the number of elements belonging to the class and the column indicates how many elements this class C is assigned to. More specifically, True positive (TP) refers to fact that a malwareis correctly classified as malicious. False positive (FP) refers to the fact that a benign application is incorrectly classified as malicious. $Truenegative(TN)$ refers to the fact that a normal application is correctly profiled as normal. False negative (FN) refers to the fact that the a malware application is mistakenly profiled as benign.

Precision ($Prec$): Precision represents the proportion of samples correctly predicted as positive among all samples correctly predicted (positive and negative). It is expressed as follows:

$$Prec = \frac{TP}{TP + FP} \tag{2}$$

Accuracy (Acc): Accuracy is the total number of correctly classified instances (positive and negative) among all available instances. It is expressed as follows:

$$Acc = \frac{TP + TN}{TP + TN + FP + FN} \tag{3}$$

Error Rate (Err) Error is the proportion of misclassifications in the whole dataset.

$$Err = \frac{FP + FN}{TP + TN + FP + FN} \tag{4}$$

Recall (R): The ratio of all positive samples that were accurately predicted to all positive samples is known as recall.

$$R = \frac{TP}{TP + FN} \tag{5}$$

Receiver Operating Characteristic (ROC): The rate of true positives versus the rate of false positives is shown by the ROC curve. In the event that the test results are

[1] https://androzoo.uni.lu/api_doc

uneven, it is in its best interest to minimize their size. This illustration emphasizes an indicator called AUC. The closer it is to 1, the more powerful the classifier is.

C. Experiments and results

We ran a number of tests to determine the accuracy-based effectiveness of SVDroid. Each experiment was asked to address one of the following factors.

Hyperparameter variation: This aspect concerns how far the variation of *n* in n-gram impact the final performance. Here we seek the best variation of n-gram based on five learning algorithms explained in [66]: 5-Nearest Neighbours (NN), 10-NN, LR (Logistic Regression), RF (Random Forests), DT (Decision Tree), and SVM (Support Vector Machine).

Algorithms investigation: This aspect is about studying performance of SVDroid on several machine and deep learning algorithms. Here, we look for the best learning algorithm independently from n-gram variations.

Partitioning: In this point, we vary the partitions of the original training set in (80%-20%, 75%-25%, 70%-30%) to see whether performance change is considerable.

1) Hyperparameter variation

The question which sustains this part is : *Which variation of n-gram is the most accurate while considering learning algorithms?*. To achieve this objective, we selected six classification algorithms of different variants such as 5-NN, 10-NN, LR, RF, DT, and SVM. The variations are 2-gram, 3-gram, 5-gram and 10-gram.

As shown in Table III, accuracy varies when *n* changes. But the variation is not growing depending on the accuracy. One important result is that SVDroid performance firmly is impacted based on *n*. We also observe that in certain cases, algorithms do not provide results with the variation of *n*. This means that in certain situations, profiling of sequence of opcodes is not of importance. Meanwhile, we see that 5-gram is the only which can provide results with more than one algorithm with 87.33% of accuracy on average. This means sequences of 5-opcodes fit the best with image processing approaches in general and SVDroid in particular. We can already observe 10-NN provides the best accuracy trend (with 88%).

TABLE III. Hyperparamater variation results

	5NN	10NN	LR	RF	DT	SVM
1-gram		81%				
2-gram	80%					
3-gram				85%		
5-gram	88%	87%	87%			
7-gram					87%	
10-gram		87%				

2) Cross partitioning, n-gram variation vs. algorithms

The question which underlines this part is: *Which partitioning is the most accurate given the algorithms and each variation?*. We randomly selected three splitting and applied each one on these algorithms: 80% (as training)-20% (as testing), 75% (as training)-25% (as testing) and 70% (as training)-30% (as testing). We run each algorithm on every splitting to have accuracy results available in Figure 10-15. As shown in Figure 10, 5-NN is the most accurate algorithm with 5-gram and with 88%. This is observed under the splitting of 75% – 25%. Figure 10 presents some results

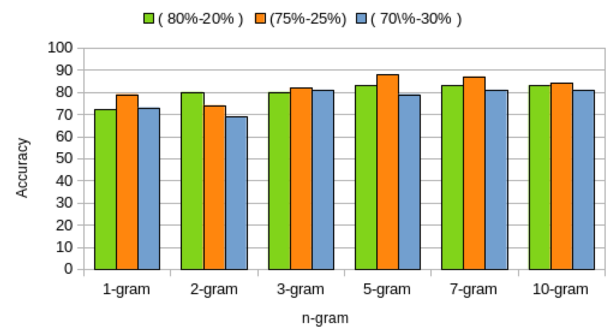


Figure 10. Results with 5-NN

with 10-NN. When k grows to 10, 10-gram fits the most under the partition 80% – 20% and the accuracy increases to 87%. The model in this case has offered an AUC of 86% average, which confers that this model tends to excellence. In summary, 10-NN fits better with 10-gram and 80%−20%.

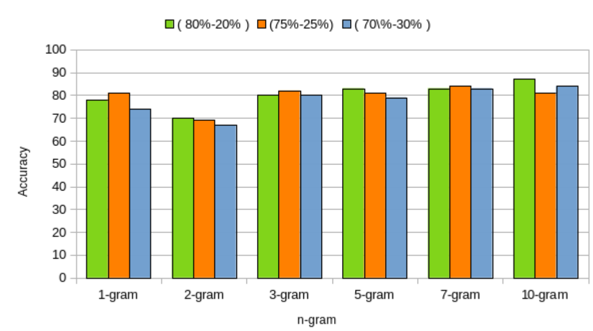


Figure 11. Results with 10-NN

As depicted in Figure 12, LR stands most accurate with 5-gram providing an accuracy of 87%, and an AUC of 86%. This model is proud to be robust concerning the class prediction of an unknown application. In sum, RL adapts better to 5-gram=5 and 80% – 20%.

Figure 13 illustrates results about DT. This figure shows that DT is the best with 3-gram with an accuracy of 85% and an AUC with the same value. The partitioning scheme which fits is 80% – 20%.

Using RF (Figure 14), 5-gram is accurate with a proba-

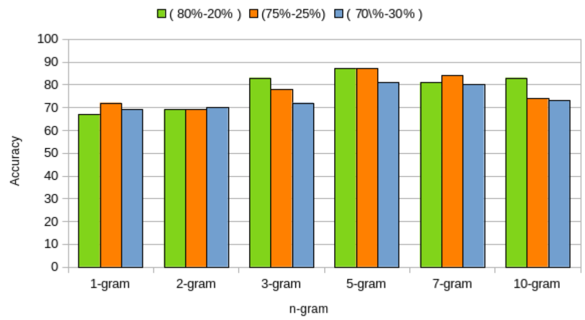


Figure 12. Results with Logistic Regression

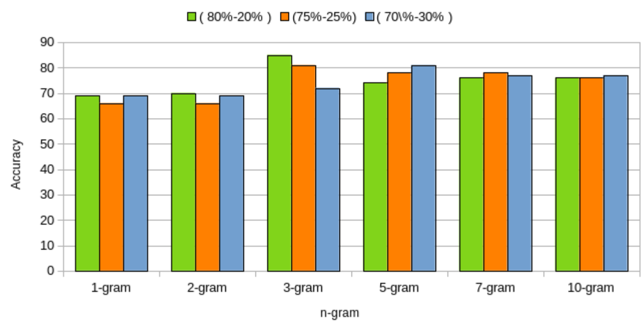


Figure 13. Result with Decision Tree

bility of 0.87% with an AUC of 86%. The properties of the previous algorithms therefore remain. But the partitioning scheme for that is 75% – 25%.

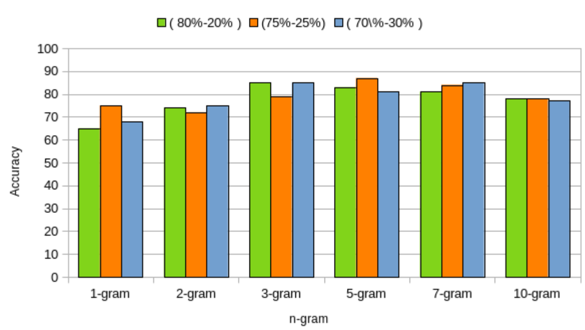


Figure 14. Result with Random Forest

Figure 15 shows that association with 7-gram worth the most with an accuracy of 81%, and a AUC of 81%, demonstrating that SVDroid brings a certain prediction robustness. This property is verified under the 80% – 20% partitioning.

From the previous results, one fundamental notice is that based on partitioning algorithms provides slightly a gap in performance since different metrics turn around 85%. Their impact as well as n-gram variations are less considerable based on gaps between accuracy and AUC. The two

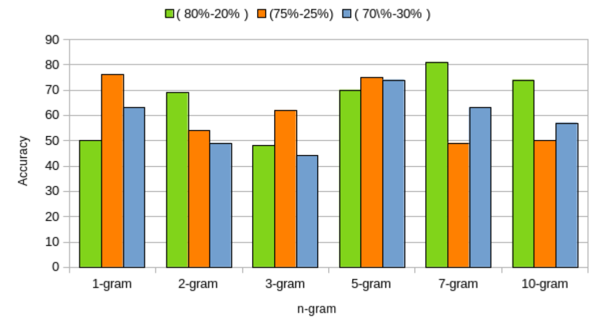


Figure 15. Result with SVM

partitioning under n-gram variations were 80% – 20% and 75% – 25%.

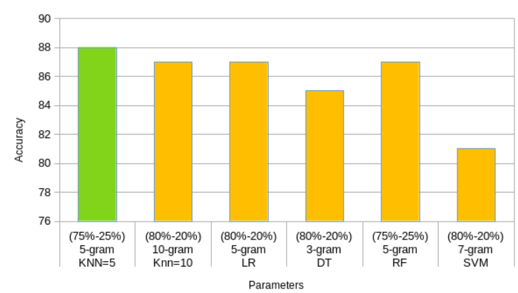
Now we come to deep learning, specifically CNN to study whether strengths can be improved. The results are presented in Table IV.

We observe that is much more improved in terms of accuracy i.e. 88.38% on average and a precision of 90% on average. With on average 93.45%, AUC proves that SVDroid is powerful than machine learning to profile unknown instances. Moreover, the best partitioning is once 80% – 20%.

3) Algorithms investigation

The question which sustains this part is : *Which (machine/deep) learning algorithm provides the best accuracy?*. Figure 16 depicts accuracy results of machine learning algorithms. The x-axis holds the algorithms and the y-axis holds accuracy values. This figure reveals that 5-NN with 5-gram (highlighted in red) under conditions presented is the most accurate with 88%.

Figure 16. Machine learning algorithm investigation

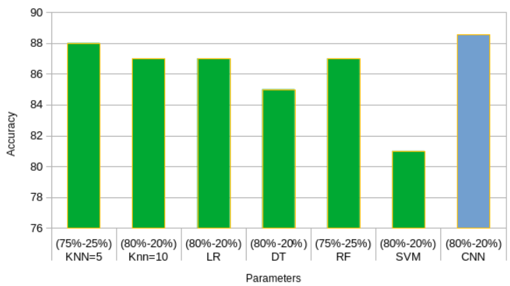


Now we study the performance of CNN compared to machine learning algorithms. It highlighted in blue in Figure 17 that CNN is the best compared with the machine learning algorithms in terms of accuracy. Moreover, as presented previously CNN is robust in terms of AUC. This result is independent from the partitioning as shown in Table IV. Different accuracy values are greater than 88%.

TABLE IV. Results with CNN

Metrics	AUC	Precision	Accuracy	TP	TN	FP	FN
(split 80% – 20%)	93.27%	89%	88.55%	91%	6.96%	84%	11%
(split 75% – 25%)	93.48%	90%	88.15%	91%	6.96%	90%	10%
(split 70% – 30%)	93.59%	91%	88.45%	92%	6.45%	83%	13%

Figure 17. Machine learning vs. CNN



D. Comparison with a similar work

In this section, we compare SVDroid to [43], which directly extracts byte-code file from .apk, and directly converts the byte-code file into a matrix of byte-code of two-dimensions, to exploit CNN to detect malware. Since the authors worked on different datasets, we completely reproduced their approach on our dataset. The authors splitted the whole byte stream into m sub-sequences with $m = [\sqrt{n}] + 1$ and n is the size of the byte stream (here the DEX file).

The classes.dex file is represented as a $m \times m$ matrix, with each sub-sequence being viewed as a row of matrix M. We add zeros to the end of any sub-sequence that is shorter than m in length. Since the lengths of different .dex files vary, so do the widths of the two-dimensional matrices created for byte-code files.

TABLE V. SVDroid vs. Ding et al. [43]

Criteria/Work	Ding et al.[43]	**SVDroid**
Datasets	APK	APK
File name	.dex	.dex
Feature	byte-code	byte-code
Image processing	convert directly byte code to image based on $m = [\sqrt{n}] + 1$ sub-sequences	convert a opcodes to image based on n-gram for Sim-hash
Image type	grayscale	grayscale
Image size	m × m	32 × 32
Accuracy	86%	88.39%
Precision	89.5%	on average 90%
AUC	91.7%	on average 93.45%

We observe in Table V that SVDroid is able to improve CNN based image processing approaches. CNN models resize input images while they are being trained, according to the sensitivity analysis. By the way, data loss often results from resizing. Notably, our approach uses data portions rather than the entire DEX file. As a result, our proposal's image size is less than the standard one, leading to reduced data loss, which raises accuracy. We also observe that SVD retains key properties of the image that even while reducing it, offers a great impact on data, regardless of the effect of splits on the data. Moreover, n-gram is able to stabilize the CNN overall process.

Results considerably decrease in [43] with our dataset. This shows that their approach is not scalable to new data. SVDroid comes on the contrary, with that property because by nature it is independent to the dataset nature. This situation explains the AUC and precision results.

Despite these interesting results from SVDroid, we can not guarantee that the variation of the image size and the nature of image (from grayscale to color) will keep SVDroid efficient. Additional experiments are required for these doubts.

As an overall result, we see that CNN incorrectly profiles 6.76% of malicious applications and 11.33% of normal applications. This is because SVD processing does not capture all the properties or unknowingly discards other properties that can improve results. This is a limitation of SVDroid that can be covered by complementing with other features related to resources and the manifest [44].

6. Conclusion and future works

The objective of the study was to investigate on the contribution of SVD to the detection of Android malware. SVDroid has been proposed to improve feature processing in CNN. This approach has been shown reliable than exploiting classical machine learning algorithms.

The following aspects are considered in future works: (i) the variation of image sizes (ii) an investigation of other deep learning methods and (iii) the tuning of different deep learning features.

7. Acknowledgements

This research has benefited from the implication of the members of the research group Cyber Security With Computational and Artificial Intelligence (www.cycomai.com) in terms of proof-reading and brainstorming sessions. We would like to thank all of them as a whole.

References

[1] Statista, "Mobile operating systems' market share worldwide from January 2012 to January 2021," feb 2020. [Online]. Available: https://www.statista.com/statistics/272698/global-market-share-held-by-mobile-operating-systems-since-2009/

[2] X. Liu, J. Liu, S. Zhu, W. Wang, and X. Zhang, "Privacy risk analysis and mitigation of analytics libraries in the android ecosystem," *IEEE Transactions on Mobile Computing*, vol. 19, no. 5, pp. 1184–1199, 2019.

[3] İ. A. Doğru and Ö. KİRAZ, "Web-based android malicious software detection and classification system," *Applied Sciences*, vol. 8, no. 9, p. 1622, 2018.

[4] K. Liu, S. Xu, G. Xu, M. Zhang, D. Sun, and H. Liu, "A Review of Android Malware Detection Approaches Based on Machine Learning," *IEEE Access*, vol. 8, pp. 124 579–124 607, 2020.

[5] Z. Wang, Q. Liu, and Y. Chi, "Review of Android Malware Detection Based on Deep Learning," *IEEE Access*, vol. 8, pp. 181 102–181 126, oct 2020.

[6] J. Sedano, S. González, C. Chira, Á. Herrero, E. Corchado, and J. R. Villar, "Key features for the characterization of Android malware families," *Logic Journal of IGPL*, vol. 25, no. 1, pp. 54–66, feb 2017. [Online]. Available: https://academic.oup.com/jigpal/article-lookup/doi/10.1093/jigpal/jzw046

[7] W. Wang, M. Zhao, Z. Gao, G. Xu, H. Xian, Y. Li, and X. Zhang, "Constructing Features for Detecting Android Malicious Applications: Issues, Taxonomy and Directions," *IEEE Access*, vol. 7, pp. 67 602–67 631, 2019.

[8] O. Yildiz and I. A. Doğru, "Permission-based Android Malware Detection System Using Feature Selection with Genetic Algorithm," *International Journal of Software Engineering and Knowledge Engineering*, vol. 29, no. 2, pp. 245–262, feb 2019.

[9] D. Ö. Şahin, O. E. Kural, S. Akleylek, and E. Kılıç, "A novel permission-based Android malware detection system using feature selection based on linear regression," *Neural Computing and Applications*, pp. 1–16, mar 2021. [Online]. Available: https://link.springer.com/article/10.1007/s00521-021-05875-1

[10] M. Alazab, M. Alazab, A. Shalaginov, A. Mesleh, and A. Awajan, "Intelligent mobile malware detection using permission requests and API calls," *Future Generation Computer Systems*, vol. 107, pp. 509–521, jun 2020.

[11] A. Pektaş and T. Acarman, "Learning to detect Android malware via opcode sequences," *Neurocomputing*, vol. 396, pp. 599–608, jul 2020.

[12] T. Franklin and D. Paul, "System calls analysis of malware on android," *International Journal of Science and Technology*, vol. 9, no. 2, pp. 669–674, 2013.

[13] A. Ananya, A. Aswathy, T. R. Amal, P. G. Swathy, P. Vinod, and S. Mohammad, "SysDroid: a dynamic ML-based android malware analyzer using system call traces," *Cluster Computing*, vol. 23, no. 4, pp. 2789–2808, dec 2020. [Online]. Available: https://doi.org/10.1007/s10586-019-03045-6

[14] J. Feng, L. Shen, Z. Chen, Y. Wang, and H. Li, "A Two-Layer Deep Learning Method for Android Malware Detection Using Network Traffic," *IEEE Access*, vol. 8, pp. 125 786–125 796, 2020.

[15] R. B. Hadiprakoso, H. Kabetta, and I. K. S. Buana, "Hybrid-Based Malware Analysis for Effective and Efficiency Android Malware Detection," in *Proceedings - 2nd International Conference on Informatics, Multimedia, Cyber, and Information System, ICIMCIS 2020*. Institute of Electrical and Electronics Engineers Inc., nov 2020, pp. 8–12.

[16] F. Tchakounté, R. C. N. Ngassi, V. C. Kamla, and K. P. Udagepola, "LimonDroid: a system coupling three signature-based schemes for profiling Android malware," *Iran Journal of Computer Science*, pp. 1–20, jul 2020. [Online]. Available: https://doi.org/10.1007/s42044-020-00068-w

[17] O. Sinan and S. Divya, *Feature Engineering Made Easy: Identify unique features from your dataset in order to build powerful machine learning systems*. Packt Publishing, jan 2018.

[18] F. Tchakounté, P. Kamdem, J. Kamgang, H. Tchapgnouo, and M. Atemkeng, "An Efficient DCT-SVD Steganographic Approach Applied to JPEG Images," *EAI Endorsed Transactions on Industrial Networks and Intelligent Systems*, vol. 7, no. 24, p. 166365, oct 2020.

[19] M. Pak and S. Kim, "A review of deep learning in image recognition," in *Proceedings of the 2017 4th International Conference on Computer Applications and Information Processing Technology, CAIPT 2017*, vol. 2018-Janua. Institute of Electrical and Electronics Engineers Inc., mar 2018, pp. 1–3.

[20] K. Huang, A. Hussain, Q.-F. Wang, and R. Zhang, Eds., *Deep Learning: Fundamentals, Theory and Applications*, ser. Cognitive Computation Trends. Cham: Springer International Publishing, 2019, vol. 2.

[21] M. B. Bahador, M. Abadi, and A. Tajoddin, "Hpcmalhunter: Behavioral malware detection using hardware performance counters and singular value decomposition," in *2014 4th International Conference on Computer and Knowledge Engineering (ICCKE)*. IEEE, 2014, pp. 703–708.

[22] M. M. Bejani and M. Ghatee, "Theory of adaptive SVD regularization for deep neural networks," *Neural Networks*, vol. 128, pp. 33–46, aug 2020.

[23] Y. Pan, X. Ge, C. Fang, and Y. Fan, "A Systematic Literature Review of Android Malware Detection Using Static Analysis," *IEEE Access*, vol. 8, pp. 116 363–116 379, 2020.

[24] B. A. Mantoo and S. S. Khurana, "Static, dynamic and intrinsic features based android malware detection using machine learning," in *Lecture Notes in Electrical Engineering*, vol. 597. Springer, 2020, pp. 31–45.

[25] A. T. Kabakus and I. A. Dogru, "An in-depth analysis of Android malware using hybrid techniques," *Digital Investigation*, vol. 24, pp. 25–33, mar 2018.

[26] J. Qiu, J. Zhang, W. Luo, L. Pan, S. Nepal, Y. Wang, and Y. Xiang, "A3CM: Automatic Capability Annotation for Android Malware," *IEEE Access*, vol. 7, pp. 147 156–147 168, 2019.

[27] S. Turker and A. B. Can, "AndMFC: Android malware family classification framework," in *2019 IEEE 30th International Symposium on Personal, Indoor and Mobile Radio Communications, PIMRC Workshops 2019*. Institute of Electrical and Electronics Engineers Inc., sep 2019.

[28] R. Vega Vega, H. Quintián, J. L. Calvo-Rolle, Á. Herrero, and E. Corchado, "Gaining deep knowledge of Android malware families through dimensionality reduction techniques," *Logic Journal of the IGPL*, vol. 27, no. 2, pp. 160–176, mar 2019. [Online]. Available: https://academic.oup.com/jigpal/article/27/2/160/5092722

[29] Z. Xu, K. Ren, and F. Song, "Android malware family classification and characterization using CFG and DFG," in *Proceedings - 2019 13th International Symposium on Theoretical Aspects of Software Engineering, TASE 2019*. Institute of Electrical and Electronics Engineers Inc., jul 2019, pp. 49–56.

[30] J. Jiang, S. Li, M. Yu, G. Li, C. Liu, K. Chen, H. Liu, and W. Huang, "Android Malware Family Classification Based on Sensitive Opcode Sequence," in *Proceedings - International Symposium on Computers and Communications*, vol. 2019-June. Institute of Electrical and Electronics Engineers Inc., jun 2019.

[31] A. Pektaş and T. Acarman, "Deep learning for effective Android malware detection using API call graph embeddings," *Soft Computing*, vol. 24, no. 2, pp. 1027–1043, jan 2020. [Online]. Available: https://doi.org/10.1007/s00500-019-03940-5

[32] S. K. Sasidharan and C. Thomas, "ProDroid — An Android malware detection framework based on profile hidden Markov model," *Pervasive and Mobile Computing*, vol. 72, p. 101336, apr 2021.

[33] M. Aresu, D. Ariu, M. Ahmadi, D. Maiorca, and G. Giacinto, "Clustering android malware families by http traffic," in *2015 10th International Conference on Malicious and Unwanted Software, MALWARE 2015*. Institute of Electrical and Electronics Engineers Inc., feb 2016, pp. 128–135.

[34] DroidBox, "Droidbox – Android Application Sandbox – The Honeynet Project," 2021. [Online]. Available: https://www.honeynet.org/projects/active/droidbox/

[35] A. Martín, V. Rodríguez-Fernández, and D. Camacho, "CANDYMAN: Classifying Android malware families by modelling dynamic traces with Markov chains," *Engineering Applications of Artificial Intelligence*, vol. 74, pp. 121–133, sep 2018.

[36] L. Massarelli, L. Aniello, C. Ciccotelli, L. Querzoni, D. Ucci, and R. Baldoni, "Android malware family classification based on resource consumption over time," in *Proceedings of the 2017 12th International Conference on Malicious and Unwanted Software, MALWARE 2017*, vol. 2018-Janua. Institute of Electrical and Electronics Engineers Inc., mar 2018, pp. 31–38.

[37] T. Chakraborty, F. Pierazzi, and V. S. Subrahmanian, "EC2: Ensemble Clustering and Classification for Predicting Android Malware Families," *IEEE Transactions on Dependable and Secure Computing*, vol. 17, no. 2, pp. 262–277, mar 2020.

[38] K. Aktas and S. Sen, "UpDroid: Updated Android Malware and Its Familial Classification," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 11252 LNCS. Springer Verlag, nov 2018, pp. 352–368.

[39] J. Gennissen, L. Cavallaro, V. Moonsamy, and L. Batina, "Gamut: sifting through images to detect android malware," *Bachelor thesis, Royal Holloway University, London, UK*, 2017.

[40] T. Hsien-De Huang and H.-Y. Kao, "R2-d2: Color-inspired convolutional neural network (cnn)-based android malware detections," in *2018 IEEE International Conference on Big Data (Big Data)*. IEEE, 2018, pp. 2633–2642.

[41] R. Kumar, Z. Xiaosong, R. U. Khan, I. Ahad, and J. Kumar, "Malicious code detection based on image processing using deep learning," in *Proceedings of the 2018 International Conference on Computing and Artificial Intelligence*, 2018, pp. 81–85.

[42] Z. Cui, L. Du, P. Wang, X. Cai, and W. Zhang, "Malicious code detection based on cnns and multi-objective algorithm," *Journal of Parallel and Distributed Computing*, vol. 129, pp. 50–58, 2019.

[43] Y. Ding, X. Zhang, J. Hu, and W. Xu, "Android malware detection method based on bytecode image," *Journal of Ambient Intelligence and Humanized Computing*, pp. 1–10, 2020.

[44] K. Bakour and H. M. Ünver, "DeepVisDroid: android malware detection by hybridizing image-based features with deep learning techniques," *Neural Computing and Applications*, pp. 1–18, mar 2021. [Online]. Available: https://doi.org/10.1007/s00521-021-05816-y

[45] J. Jung, J. Choi, S.-j. Cho, S. Han, M. Park, and Y. Hwang, "Android malware detection using convolutional neural networks and data section images," in *Proceedings of the 2018 Conference on Research in Adaptive and Convergent Systems*, 2018, pp. 149–153.

[46] L. Nataraj, S. Karthikeyan, G. Jacob, and B. S. Manjunath, "Malware images: visualization and automatic classification," in *Proceedings of the 8th international symposium on visualization for cyber security*, 2011, pp. 1–7.

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

[48] M. Kalash, M. Rochan, N. Mohammed, N. D. Bruce, Y. Wang, and F. Iqbal, "Malware classification with deep convolutional neural networks," in *2018 9th IFIP international conference on new technologies, mobility and security (NTMS)*. IEEE, 2018, pp. 1–5.

[49] S. Choi, S. Jang, Y. Kim, and J. Kim, "Malware detection using malware image and deep learning," in *2017 International Conference on Information and Communication Technology Convergence (ICTC)*. IEEE, 2017, pp. 1193–1195.

[50] K. Kosmidis and C. Kalloniatis, "Machine learning and images for malware detection and classification," in *Proceedings of the 21st Pan-Hellenic Conference on Informatics*, 2017, pp. 1–6.

[51] G. Shrivastava, P. Kumar, D. Gupta, and J. J. P. C. Rodrigues, "Privacy issues of android application permissions: A literature review," *Transactions on Emerging Telecommunications Technologies*, vol. 31, no. 12, p. e3773, dec 2020. [Online]. Available: https://onlinelibrary.wiley.com/doi/10.1002/ett.3773

[52] L. D. Subhashini, Y. Li, J. Zhang, A. S. Atukorale, and Y. Wu, "Mining and classifying customer reviews: a survey," *Artificial Intelligence Review*, pp. 1–47, mar 2021. [Online]. Available: https://doi.org/10.1007/s10462-021-09955-5

[53] O. Jafari, P. Maurya, P. Nagarkar, K. M. Islam, and C. Crushev, "A Survey on Locality Sensitive Hashing Algorithms and their Applications," *ACM Computing Surveys*, feb 2021. [Online]. Available: http://arxiv.org/abs/2102.08942

[54] V. Sihag, M. Vardhan, and P. Singh, "A survey of android

application and malware hardening," *Computer Science Review*, vol. 39, p. 100365, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1574013721000058

[55] H. Ahmed, I. Traore, and S. Saad, "Detection of Online Fake News Using N-Gram Analysis and Machine Learning Techniques," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 10618 LNCS. Springer Verlag, oct 2017, pp. 127–138.

[56] C. Cortes and V. Vapnik, "Support-vector networks," *Machine Learning*, vol. 20, pp. 273–297, 1995.

[57] M. S. Charikar, "Similarity estimation techniques from rounding algorithms," in *Conference Proceedings of the Annual ACM Symposium on Theory of Computing*. New York, New York, USA: Association for Computing Machinery (ACM), 2002, pp. 380–388.

[58] S. Ni, Q. Qian, and R. Zhang, "Malware identification using visualization images and deep learning," *Computers and Security*, vol. 77, pp. 871–885, aug 2018.

[59] M. Moonen, P. Van Dooren, and J. Vandewalle, "A singular value decomposition updating algorithm for subspace tracking," *SIAM Journal on Matrix Analysis and Applications*, vol. 13, no. 4, pp. 1015–1038, 1992.

[60] T. Konda and Y. Nakamura, "A new algorithm for singular value decomposition and its parallelization," *Parallel Computing*, vol. 35, no. 6, pp. 331–344, 2009.

[61] N. M. Makbol, B. E. Khoo, T. H. Rassem, and K. Loukhaoukha, "A new reliable optimized image watermarking scheme based on the integer wavelet transform and singular value decomposition for copyright protection," *Inf. Sci.*, vol. 417, no. C, p. 381–400, Nov. 2017. [Online]. Available: https://doi.org/10.1016/j.ins.2017.07.026

[62] Baksmali, "smali/baksmali," Mar 2020. [Online]. Available: https://github.com/JesusFreke/smali

[63] J. Zhang, Z. Qin, H. Yin, L. Ou, and Y. Hu, "Irmd: malware variant detection using opcode image recognition," in *2016 IEEE 22nd International Conference on Parallel and Distributed Systems (ICPADS)*. IEEE, 2016, pp. 1175–1180.

[64] S. M. H. M. H. G. K. R. ARP, Daniel, "Drebin: Effective and explainable detection of android malware in your pocket," in *2014 Network and Distributed System Security Symposium (NDSS)*, 2014, pp. 23–26.

[65] K. Allix, T. F. Bissyandé, J. Klein, and Y. Le Traon, "Androzoo: Collecting millions of android apps for the research community," in *Proceedings of the 13th International Conference on Mining Software Repositories*, ser. MSR '16. New York, NY, USA: ACM, 2016, pp. 468–471.

[66] B. Giuseppe, *Machine learning algorithms*. Packt Publishing, jul 2018.

Raïssa Edwige Feuyo Manfouo is a Master's student in artificial intelligence at the African Masters of Machine Intelligence (AMMI) in Senegal. She held a Master degree in Mathematical Sciences from the African Institute for Mathematical Sciences at AIMS Cameroon and a M.Sc. in Computer Engineering from the University of Ngaoundéré in Cameroon. She is a member of the CyComAI (Cybersecurity with Computational and Artificial Intelligence) research group and she has been a trainer in the Women in Tech program, at Ngaoundéré in Cameroon. Her interests includes Machine Intelligence Related subjects.

Franklin Tchakounté is an Associate Professor and researcher in computer science with more than 10 years of experience in cybersecurity and data science with a strong background in distributed systems. He received his M.Sc. in Computer engineering from the University of Ngaoundéré and then his PhD Degree in Mobile Security from the University of Bremen. He authored books, book chapters and several research papers in the area of cyber security. He is the founder of Cybersecurity with Computational and Artificial Intelligence (CyComAI) research group. He is fellow of DAAD Staff Exchange in Sub-Saharan-Africa, Research Mobility grants in Ministry of Higher Education in Cameroon, and WebWeWant F.A.S.T project. Devoted to volunteering in reviewing and conference involvement, he has been (senior) member of EAI, ACM, UWB and ISOC SIG. His interests include cyber security and artificial intelligence.

Jean Louis Kedieng Ebongue Fendji received the B.S. and M.S. degrees in Computer Science from the University of Ngaoundéré, Cameroon, in 2010 and the Ph.D. degree from the University of Bremen, Germany, in 2015. From 2011 to 2013, he was a Research Assistant with the BMBF Project between the University of Bremen and the University of Ngaoundéré. Since 2015, he has been a Senior Lecturer with the Computer Engineering Department, University Technology Institute, at the University of Ngaoundéré. He is the author of more than 20 publications and book's chapters. He has been working for more than 10 years on Network modeling and optimization. His current research interests include Machine Learning, Deep Learning, NLP, Speech recognition, Optimization, AI in Agriculture and Education.

Marcellin Atemkeng received the Ph.D. with specialization in big data and statistical signal processing from Rhodes University, South Africa, in 2016. He was a Signal Analyst with VASTech, a telecommunication industry in Stellenbosch, South Africa. From 2017 to 2018, he was a Postdoctoral research fellow working with the Square Kilometre Array related projects. Since 2019, he is a Senior Lecturer in the Department of Mathematics at Rhodes University and the coordinator of Rhodes AI Research Group (RAIRG). He has published several articles in peer-reviewed journals and was the 2019 recipient of the Kambule Doctoral Award for recognising and encouraging excellence in research and writing by doctoral candidates at African universities in any area of computational and statistical sciences. His research interests include big data, statistical signal processing, artificial intelligence, deep learning and radio interferometric techniques and technologies.

Didier Bassolé is an Assistant Professor and Researcher in Computer Science from University Joseph KI-ZERBO, Ouagadougou, Burkina Faso. He received Advanced Studies in Computer Science degree in 2009 and Computer Engineering degree in 2008 from Higher School of Informatics at Polytechnic University of Bobo Dioulasso. His PhD degree in 2018 from University Joseph KI-ZERBO is focused on impacts of fault injection attacks on software security components by model checking. He has published several articles in peer-reviewed journals and International conferences. His research interests are in the area of Security of embedded systems, security in android apps, malware/intrusion detection and classification using machine learning techniques and deep learning approaches, IoT security, and artificial intelligence.