JST: Smart Systems and Devices
Volume 31, Issue 1, May 2021, 009-016

Android Malware Classification Using Deep Learning CNN with Co-Occurrence Matrix Feature

Le Duc Thuan[1,2], Hoang Van Hiep[1*], Nguyen Kim Khanh[1]
[1] *Hanoi University of Science and Technology, Hanoi, Vietnam*
[2] *Academy of Cryptography Techniques, Hanoi, Vietnam*
Email: hiephv@soict.hust.edu.vn

Abstract

Recently, deep learning has been widely applying to speech and image recognition. Convolutional neural network (CNN) is one of the main categories to do image classifications with very high accuracy. In Android malware classification field, many works have been trying to convert Android malwares into "images" to make them well-matched with the CNN input to take advantage of the CNN model. The performance, however, is not significantly improved because simply converting malwares into images may lack several important features of the malwares. This paper proposes a method for improving the feature set of Android malware classification based on co-concurrence matrix (co-matrix). The co-matrix is established based on a list of raw features extracted from .apk files. The proposed feature can take the advantage of CNN while remaining important features of the Android malwares. Experimental results of CNN model conducted on a very popular Android malware dataset, Drebin, prove the feasibility of our proposed co-matrix feature.

Keywords: Android Malware classification, Drebin, Co-Matrix, CNN.

1. Introduction

As of April 2020, the market share of Android operating system (OS) accounts for 70.68% on mobile devices [1], which indicates the prevalence of Android OS over the others. According to AV-Test [2], in April 2020, there were more than 9 million malwares discovered on operating systems, in which 1.29 million malwares were found in Android-based platform. Although Android OS always offers security updates periodically (currently Android 10 is the newest), but with user habits that accept all applications' access requests, it is very likely that their mobile phone might be injected with malware and paved the way for hackers' actions. Without proper control of the app, the user is easily exposed to safety threats.

Many applications containing malwares can bypass the strict inspection of Google and make them available on the Google Play Store [3]. Moreover, in contrast to other platforms, Android allows users to install an app from unverified sources such as third-party websites or markets. This makes managing and testing applications more challenging. In addition, investigating an .apk file is more difficult than a normal execution file because the information is not only stored in one main file like Windows-based system (PE file). Malware classification on the Android platform is therefore still mandatory for the software industry and a hot topic for the research community as well.

ISSN: 2734-9373
https://doi.org/10.51316/jst.150.ssad.2021.31.1.2
Received: 17 November 2020; accepted: 10 March 2021

One of the most difficult issues when dealing with Android malware classification problems is to build up a good dataset, i.e., a dataset may have various malware families and a balance of the number of benign and malware files. Currently, there are two popular Android malware datasets opening for research groups, namely Drebin [4] and AMD [5]. These datasets include benign and malware-infected set for analysis and testing. They have been being used by many research groups from many prestigious universities. Benign files are collected from the app market and some other sources based on the assumption that they are verified as clean. The dataset is therefore a challenge for research teams working on Android malware classification.

Recently, the convolution neuron network (CNN) shows a great advantage in classification problem. CNN was originally applied for image classification (object identification, face classification, medical imaging, etc.). CNN was then applied to other classification problems such as speech recognition, handwriting recognition, malware classification, etc. By using a sequence of convolution, CNN can learn the relationship among pixels in an image, therefore it can help to increase the final classification rate. In the malware analysis field, to apply CNN, it is needed to find an effective way to convert from a malware file into an "image". In other words, we need to extract features from malware files and store them in a matrix-based format. Many related works simply convert an .apk file into a digital matrix (image) and treat it as input for CNN [7-8]. This approach has two drawbacks as follows: (i) CNN requires the size of all the input "images" exactly the same but the size of .apk files

are unequal, some paddings or cuttings are therefore required, this reshape may affect the final classification rate; and (ii) the "image" converted from an .apk file may lack several features of the original one such as API function list, permission request, and/or the relationship among API and permission, etc. In this paper, we propose to use a co-occurrence matrix (co-matrix) to represent the features, in which, one element of the matrix shows the concurrency of two different features. This idea comes with the assumption that malware may never call only one API (or request only one permission) but some APIs (or request several permissions). Therefore, the concurrence of API calling and/or permission requesting can be used as a feature to distinguish between a benign and a malware one. In addition, the co-matrix is a matrix so that it is perfectly fit as the input of the CNN. It is hoped that this proposed co-matrix feature can help improve the accuracy of classification.

Related works on Android malware classification can be divided into two approaches: based on conventional machine learning approach and based on deep learning one. For the former approach, *D. Arp et al.* has introduced a dataset named Drebin which is widely used by many other research groups later [4]. The dataset contains 5,560 malware files and 126,051 benign files. For classification, the author proposed to use SVM with feature sets including used permissions, suspicious APIs, network addresses which are extracted from API files. The classification rate is about 94%. *Rana M.S. et al.* evaluated Random Forest algorithm [6] with a part of Drebin dataset consisting of 5,560 malware files and 5,560 benign files (the number of malware and benign files are equal). The feature sets are almost the same as in [4] except API strings and URLs are added. The highest classification rate reached 97.24%. However, this result is only for binary classification, i.e., malware or benign.

For the deep learning approach, in [7], the authors proposed a dynamic routing-based Capsule network with feature sets are collected by simply converting malware binary files into color images. The performance of Capsule network is compared to the one of CNN on two malware datasets: Windows and Android dataset. The Android one is a part of Drebin dataset (4,000 malware files consist of 20 families and 6,000 benign files). The results indicated that the performance of Capsule network is almost the same as that of CNN in the case of Windows malware dataset (96.5% and 96.8% of accuracy). On the other hand, the performance of Capsule network is significantly higher than that of CNN in the case of Android dataset (classification rate is 99.3% with CapNets and 79.3% with CNN model respectively). Based on these results, can we conclude that CNN is not suitable for Android malware classification? It is hard to answer immediately but one point can be figured out is that simply converting Android .apk file to image file seems to take away the links between important features of Android malware, which leads to a worse result when using CNN.

Using the same idea of converting to image format, *T. H. Huang et al* [8] proposed a color-inspired CNN-based malware detection (R2-D2). The system converts the byte code of classes.dex file from Android archive to rgb color code and save it as a color image with a fixed size. The image is then input to CNN for automatic feature extraction and training. The evaluated dataset was manually built from Jan. 2017 to Aug. 2017 which contains 5,377 malware files and 6,249 benign files. Even though the authors showed good accuracy of classification (93%), evaluation on a self-build dataset may have some biases.

Follow a different approach, *Zhiwu Xu et al.* [9] did not convert .apk file into image but ".smali file" (running code on Dalvik Virtual Machine). The .smali file is then converted to digital matrix by using Control Flow Graph (CFG) and Data Flow Graph (DFG) algorithms. The digital matrix is treated as an

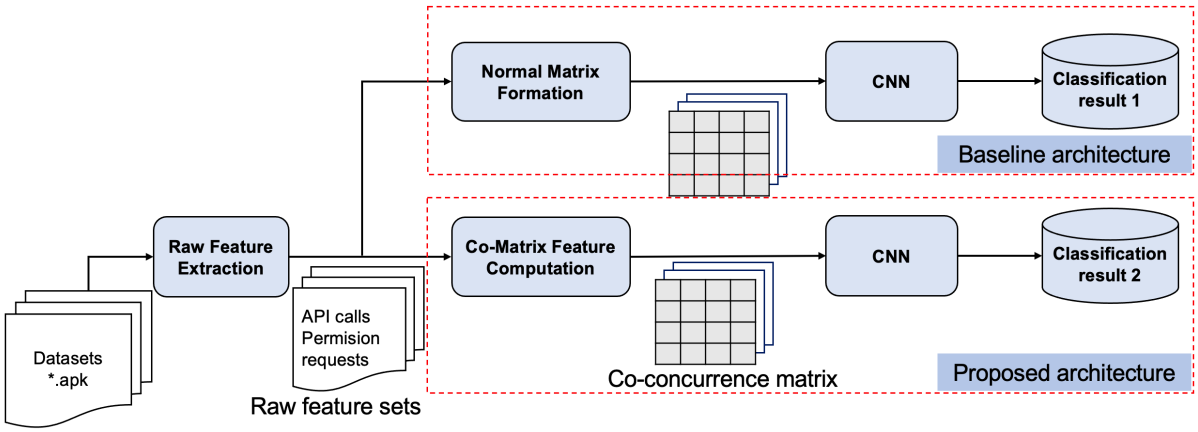

Fig. 1. System model for Android malware classification

input for the CNN model. The authors collect 24,485 benign files and 3,166 malware files for testing, and 50,501 benign files and 7,406 malware files for training. Those files were collected from several sources including Darvin, Drebin, VirusShare, ContaigioDump. The accuracy of binary classification is very precise, 99.5%. However, the proposed method is only applying for malware detection, not for malware classification.

In paper [10], *Chenglin Li et al.* have used two datasets including 5,560 malwares of Drebin with 5600 benigns, and 24553 malwares of AMD with 16,753 benigns. The authors proposed a Factorization algorithm, the feature set was the same as that used in Drebin work [4], the classification results with Drebin and AMD dataset were up to 99.46% and 99.05% respectively. Other machine learning algorithms such as SVM, NB-G, NB-B, NB-M, MLP. also produce very high classification rates on the Drebin dataset, especially MLP reached up to 99.73% rate. In these papers, both the AndroidManifest.xml and classes.dex files are used to convert into *.smali files, these *.smali files are then used to extract internal features. In the paper, the author also created a correlation between the features and sought to choose the appropriate algorithm for the dataset. However, it is necessary to classify many families of malware instead of just the two classes benign and malware.

In summary, the traditional machine learning approach requires complex feature extraction work in advance. The feature sets for Android malware normally are API calls, permission requests, network address, URL, etc. For the classification step, many algorithms could be applied such as SVM, Decision Tree, MLP, etc. The final classification result mainly depends on the quality of extracting feature sets. On the other hand, deep learning approach, the CNN, does not require feature extraction in advance but needs a large number of labeled samples. Moreover, CNN requires an effective way to convert Android .apk file into matrix-based format. Some works indicate that simply converting Android .apk file to image is not efficient.

This paper proposes to use co-matrix feature as input to CNN. We evaluate the proposed feature by evaluating the CNN in two scenarios: with and without using co-matrix. Experiments are conducted on the malware dataset provided by Drebin including 5,560 malware files of 179 families. To evaluate the performance, we use metrics like ACC, F1, FPR, PR, RC. It is found that when using the co-matrix, the accuracy average of CNN increased from 95.78% to 96.23%.

The remaining of the paper is structured as follows: in section 2, we show system modeling and implementation idea. Section 3 describes in detail the feature extraction process. Section 4 discusses the machine learning algorithms used to classify benign and malware families in this paper. In section 5, we describe experimental results. Section 6 draws some conclusions and points out future work.

2. Implementation Idea

Figure 1 shows the implementation idea of this paper. To prove the effectiveness of co-concurrence matrix feature, we set up two scenarios with and without using co-matrix feature computation module. The process is as follows:

- From .apk files, the raw feature extraction module extracts features including API call strings, and permission requests.
- For the baseline architecture, the raw features go to the Normal matrix formation module. The module converts the raw features, in string format, into a vector by using a dictionary of API calls and permission. Each element in the vector has one of two values: 1 or 0 depending on whether we can find the API or the permission in the current .apk file or not. The vector is then reshaped to a matrix which is latter treat as the input of CNN. For the proposed architecture, raw features go to the co-matrix feature computation module. The module forms a matrix based on the concurrence appearance of two APIs or permissions in the .API file.

Algorithm 1: Convert string features to number

```
Input: ;
    dictionaryFeatures: dictionary of all APIs and permission;
    listFeaturesFile: list of API and permission string of a file;
Output: vectorOutput: feature vector;
length = length(dictionaryFeatures);
vectorOutput = new vector(length);
for i=1; i < length; i++ do
    aFeature = dictionaryFeatures[i];
    if aFeature exists in listFeatureFile then
        vectorOutput[i] = 1;
    else
        vectorOutput[i] = 0;
    end
end
```

Algorithm 2: co-concurence matrix computation algorithm

```
Input: ;
    vecFeature: feature vector;
Output: ;
    coMatrix: co-concurrence matrix;
length = length(vecFeature);
coMatrix = new Matrix(length, length);
for i=1; i < length; i++ do
    for j=1; j < length; j++ do
        if vecFeature[i] == 1 and vecFeature[j] == 1 then
            coMatrix[i][j] = 1;
        else
            coMatrix[i][j] = 0;
        end
    end
end
```

- Next, CNN module is applied to learn the features and to classify the apk files into benign or specific malware families.

3. Features Extraction

3.1. Raw Feature Extraction

An .apk file is essentially a compressed file with the following structure:

- META-INF/: this folder contains description information from java jar file
- Res/: this folder contains the source material
- Libs/: this folder contains libraries
- AndroidManifest.xml: contains configuration information about access rights and used services.
- Classes.dex: this file contains java bytecode of the .apk file
- Resources.asc: contains pre-complied resources such as string, colors, styles.

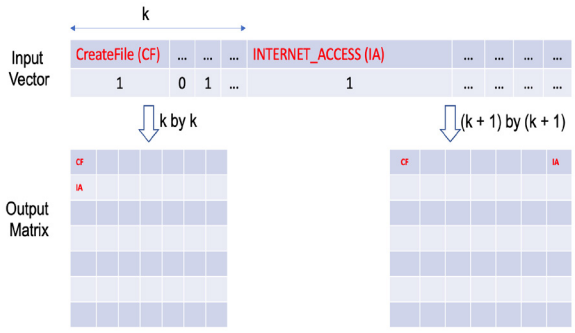

Fig. 2. Output matrix with different size

To extract features from .apk files, we can utilize many tools such as Apk tool, Dex2jar, Baksmali, Androguard, Jadx, Jd-gui, Androidpytool. In this paper, we use Androidpytool to extract features. All the features are static analysis ones extracted from two files: AndroidManifest.xml and classes.dex.

From the raw feature sets, we remove the outliers by keeping only topmost 200 popular APIs appearing in all .apk files. These features, in the form of strings, are input of the next module in the process chain as mentioned in Fig.1. Algorithm 1 illustrates the implementation to convert from string features into number vector.

3.2. Proposed Co-Matrix Feature Computation

After converting raw features in string form to a vector of numbers. The next step is to reshape this vector to a matrix which can be used as input of CNN latter. This step may have a huge impact on the final classification results. The reason is that the order of features might be changed a lot when we reshape the vector to different matrix size. Figure 2 illustrates an example of forming output matrix with different sizes. Due to the fact that a harmful malware tends to call an API together with another one or a permission request (e.g., the API CreateFile might be called together with INTERNET_ACCESS permission in a malware). CNN can learn the relationship between these two elements if they are located close to each other in the output matrix, i.e., in case of forming matrix as k by k. In contrast, CNN may lose the information if we form the output matrix in different sizes, i.e., (k+1) by (k+1), as shown in Fig.2. Hence, using CNN, the order of elements in the input vector also affects the final classification rate.

Our proposed co-matrix can solve the problem of input elements reordering because the co-matrix focuses on the concurrence that appears between two elements rather than a single element.

The co-matrix was first mentioned in 1957 when linguist *J.R Firth* [15] referred to the relationship between words in a sentence. A word is represented semantically by the words around it, so the placement of words will affect the meaning of the sentence. The co-matrix is described as follows:

	Roses	Are	Red	Sky	Is	blue
Roses	1	1	1	0	0	0
Are	1	1	1	0	0	0
Red	1	1	1	0	0	0
Sky	0	0	0	1	1	1
Is	0	0	0	1	1	1
blue	0	0	0	1	1	1

Here, the co-matrix is now connected in each word of the paragraph. We apply this idea to the Android malware features. The implementation of co-concurrence matrix computation is described in Algorithm 2.

Co-matrix is currently used in image classification problems as in [16-19]. Co-matrix is currently used in word recognition, image, or face classification problems.

4. Malware Classification Based on CNN Model

CNN model has shown state-of-the-art performance in many fields including image recognition, natural language processing, and malware classification [20, 21].

4.1. Input

The input is stated as a matrix of numbers, this input matrix can have the form of vector or *NxM* matrix. One or several matrices could be used as input.

4.2. Convolutional Layer

The convolution layer uses a sliding matrix, with the number of sliding matrix k is the number of matrices in the convolutional layer. If the input is of

*N*x*M* matrix, then there are *k* *N*x*M* matrices in the convolutional layer.

Each convolutional layer uses an activation function to produce the result for pooling. Without this function, a complete neural network will move in a linear direction from a node's input to its corresponding output.

ReLU function is used to transform the input to the maximum of either zero or input itself and it is defined by Equation (1).

$$\text{Relu}(x) = \begin{cases} 0 & when\, x < 0 \\ x & when\, x \geq 0 \end{cases} \quad (1)$$

4.3. Pooling Layer

Each matrix in the Convolutional layer will be processed to reduce the number of features. Pooling layer takes the representative for each matrix in Convolutional layer based on Formula (2):

$$Y_{kij} = \underset{(p,q)\in\eta_{ij}}{FUNC}\, x_{kpq} \quad (2)$$

where:

Y_{kij} : Output value *k* of mapping feature

x_{kpq} : Element *(p, q)* stored pooling η_{ij} .

FUNC is often used as *MAX* or *Average.*

4.4. Fully connected layers

When passing through all the convolutional layer and pooling layer, the features will be represented in vector form, namely flatten layer. The neuron will proceed through the hidden layer to the output layer. The neuron from flattening layer to output layer will be fully connected via hidden layer. For each connection from neuron *A* to neuron *B*, there will a weight number, and the final weighted matrix is the result of the training process.

The output layer is the labeling layer, which decides which label is assigned to which file. Softmax activation function is often used to compute the j^{th} output value according to Equation (3):

$$f_j(z) = \frac{e^{o_j}}{\sum_{k=1}^{n} e^{o_k}} \quad (3)$$

Figure 3 shows the CNN model used in this paper. An input data can go through many CNN models. We use two CNN models with each model being a feature group with different dimensions of the input matrix, after going through the convolutional and pooling layers, the features will be combined before putting into neural network to classify the output. In this experiment, we use three convolution and pooling layers to pre-process the input data, then one flatten layer of the matrix to form a one-dimensional vector and traverse the hidden layer of 1024 neurons and 180 output labels. Table 1 describes the use of the two-input CNN model when applying co-matrix with two API and Permission feature groups.

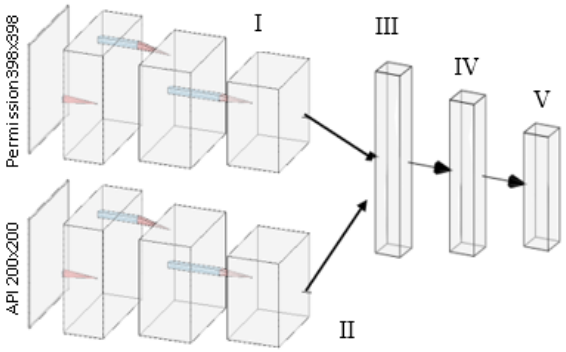

Fig. 3. CNN having multi convolutional networks

Table 1. Two CNN inputs

Input (I)	398x398	**Input(II)**	200x200
Conv_1(I)	398x398x32	Conv+1(II)	200x200x32
Pooling1(I)	199x199x32	Pooling_1(II)	100x100x32
Conv_2(I) + Pooling_2(I)	100x100x64	Conv_2(II) + Pooling_2(II)	50x50x64
Conv_3(I) + Pooling_3(I)	50x50x64	Conv_3(II) + Pooling_3(II)	25x25x64
Flatten (III)	50x50x64+25x25x64 = 200.000		
Hidden (IV)	1024		
Ouput (V)	180		

5. Experiments

5.1. Experimental Setup

This work uses Drebin dataset to evaluate the proposed scheme. The dataset includes 5,438 malware files with 179 families and 6,732 benign files including applications and games [22]. For feature extraction, there are many internal feature groups like permissions, APIs, services, urls, intents, etc. However, in this work, we only focus on getting permission and API features (including system calls and function calls in the program).

We used top 398 permissions and top 200 API function calls that are used the most in all files. Therefore, each .apk file will have 598 raw features. We compute the co-matrix for each permission and API group, which generating 158,404 permission features and 40,000 API features. All features are stored in a .csv file as input for machine learning algorithms.

The data is divided into two sets:

Set 1: 598 features based on permissions and API.

Set 2: 198,404 features after using the co-matrix of 598 features in set 1.

These two sets were used as input to CNN model. For each set we divided into groups using 10-fold technique, i.e., dividing the data into 10 equal parts of samples having both benign and malware, with 80% for training, 10% for validation testing, and 10% for testing. We cross-tested 10 times and took the average of the classification results.

5.2 Experimental Results

Experimental results according to 10-fold and the average classification are shown in Table 2.

Table 2. Classification with CNN model

Set	CNN model (%)	
	Raw features	Co-matrix features
1	93.59	93.26
2	95.68	95.98
3	96.12	97.48
4	95.29	97.07
5	96.97	96.86
6	97.1	96.06
7	97.41	97.14
8	97.47	97.24
9	95.38	96.77
10	92.00	94.46
ACC	**95.78**	**96.23**

Table 3. Measurements used

MEASURE	DESCRIPTION
TP	The malware is true
TN	The benign is true
FP	The malware is false
FN	The benign is false
ACC	(TP+TN)/(TP+TN+FP+FN)
PR	TP/(TP+FP)
RC	TP/(TP+FN)
F_1-score	2*PR*RC/(PR+RC)
FPR	FP/(FP+TN)

Table 4. Measurements evaluate effectiveness (%)

MEASURE	CNN	CNN with co-matrix
PR	97.6	98
RC	91.9	92.63
F_1-score	94.66	95.25
FPR	1.56	1.3
ACC	95.78	96.23

It can be seen that using co-matrix has increased the average ACC by 0.58%, and the classification difference among 10-fold runs has also decreased from 5.5 (using raw feature set) to 3.98 (using co-matrix). It proved that the links between features did affect the classification results. When using co-matrix, both the quantity and quality of the feature sets are improved. With this method, we do not need to care about the trade-off between changing the matrix size and the classification performance. The input of co-matrix is a symmetric matrix *[n x n]*, after going through convolutional and pooling layer we will obtain correlated neurons between benign and malwares. The results will have better weight after training.

We used some added metrics to evaluate the effectiveness of proposed feature as shown in Table 3 and Table 4. It can be seen that the PR metric when using co-matrix feature increased by 0.3% compared with that of raw feature set. The F_1-score metric is also better, 0.58 when using co-matrix features. Overall, using co-matrix feature improved the ACC of the classification compared with using raw features set. However, the drawback of the proposed co-matrix feature is that the matrix size is quite large and thus requires high computation cost.

We also test our proposed co-matrix feature using another machine learning algorithm, Decision Tree (DT). The classification results are shown in Fig.4. As we can see, co-matrix is not so suitable for DT because the classification rate with co-matrix

feature was 0.1% lower than that of raw feature. This leads to a conclusion that co-matrix is good for CNN, since in CNN, we have convolutional and pooling layers that create the relationship among features. In contrast, DT uses branches, so the co-matrix feature makes the computation of branching more complicated.

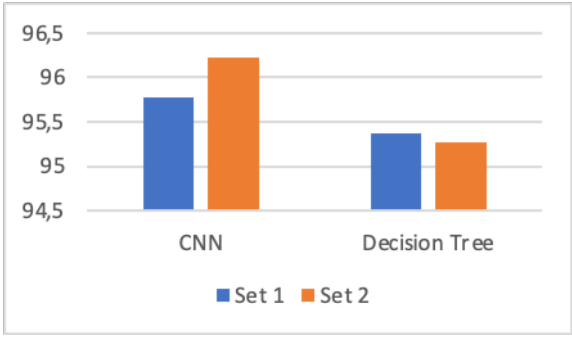

Fig.4. Classification results

6. Conclusion

In this study, we proposed to use co-concurrence matrix to represent Android malware features. The proposed co-concurrence matrix can be used as input of CNN model. Experimental results show the effectiveness of the proposed feature compared to the baseline using raw features.

This paper focuses only on the feature set improvement of Android malware but not the modification of CNN model. In the future, we will improve the feature sets by adding more features in static analysis and dynamic analysis [23-25], hybrid analysis [26-28]. We also plan to embed the co-matrix since it is now quite spard.

References

[1] Mobile Operating System Market Share Worldwide. Available: https://gs.statcounter.com/os-market-share/mobile/worldwide

[2] Statistics malware: available at https://www.av-test.org/en/statistics/malware/

[3] Bernard Meyer, These camera apps with billions of downloads might be stealing your data and infecting you with malware. Available: https://cybernews.com/security/popular-camera-apps-steal-data-infect-malware

[4] D. Arp, M. Spreitzenbarth, M. Hübner, H. Gascon, and K. Rieck, Drebin: Effective and Explainable Detection of Android Malware in Your Pocket, Proceedings 2014 Network and Distributed System Security Symposium, 2014 https://doi.org/10.14722/ndss.2014.23247

[5] F. Wei, Y. Li, S. Roy, X. Ou, and W. Zhou, Deep Ground Truth Analysis of Current Android Malware, Detection of Intrusions and Malware, and Vulnerability Assessment, vol. 10327, pp. 252–276, 2017. https://doi.org/10.1007/978-3-319-60876-1_12

[6] Md. S. Rana, S. S. M. M. Rahman, and A. H. Sung, Evaluation of Tree Based Machine Learning Classifiers for Android Malware Detection, Computational Collective Intelligence, vol. 11056, pp. 377–385, 2018, https://doi.org/10.1007/978-3-319-98446-9_35

[7] S. Wang, G. Zhou, J. Lu, and F. Zhang, A Novel Malware Detection and Classification Method Based on Capsule Network, Lecture Notes in Computer Science, vol. 11632, pp. 573–584, 2019, https://doi.org/10.1007/978-3-030-24274-9_52

[8] T. H. Huang and H. Kao, R2-D2: ColoR-inspired Convolutional NeuRal Network (CNN)-based AndroiD Malware Detections, 2018 IEEE International Conference on Big Data (Big Data), Seattle, WA, USA, 2018, pp. 2633-2642, https://doi.org/10.1109/BigData.2018.8622324

[9] Z. Xu, K. Ren, S. Qin, and F. Craciun, CDGDroid: Android Malware Detection Based on Deep Learning Using CFG and DFG, in Formal Methods and Software Engineering, 2018, vol. 11232, pp. 177–193, https://doi.org/10.1007/978-3-030-02450-5_11

[10] C. Li, K. Mills, D. Niu, R. Zhu, H. Zhang and H. Kinawi, Android Malware Detection Based on Factorization Machine, in IEEE Access, vol. 7, pp. 184008-184019, 2019, https://doi.org/10.1109/ACCESS.2019.2958927

[11] R. Nix and J. Zhang, Classification of Android apps and malware using deep neural networks, 2017 International Joint Conference on Neural Networks (IJCNN), Anchorage, AK, 2017, pp. 1871-1878, https://doi.org/10.1109/IJCNN.2017.7966078

[12] Y. Ding, W. Zhao, Z. Wang and L. Wang, Automaticlly Learning Featurs Of Android Apps Using CNN, 2018 International Conference on Machine Learning and Cybernetics (ICMLC), Chengdu, 2018, pp. 331-336, https://doi.org/10.1109/ICMLC.2018.8526935

[13] Y. Jin, T. Liu, A. He, Y. Qu and J. Chi, Android Malware Detector Exploiting Convolutional Neural Network and Adaptive Classifier Selection, 2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC), Tokyo, 2018, pp. 833-834, https://doi.org/10.1109/COMPSAC.2018.00143

[14] A. Abderrahmane, G. Adnane, C. Yacine and G. Khireddine, Android Malware Detection Based on System Calls Analysis and CNN Classification, 2019 IEEE Wireless Communications and Networking Conference Workshop (WCNCW), Marrakech, Morocco, 2019, pp. 1-6, https://doi.org/10.1109/WCNCW.2019.8902627

[15] Wikipedia, John Rupert Firth. Available: https://en.wikipedia.org/wiki/John_Rupert_Firth

[16] T. Watanabe, S. Ito, and K. Yokoi, Co-occurrence Histograms of Oriented Gradients for Pedestrian Detection, in Advances in Image and Video Technology, 2009, vol. 5414, pp. 37–47, https://doi.org/10.1007/978-3-540-92957-4_4

[17] W. Gomez, W. C. A. Pereira and A. F. C. Infantosi, Analysis of Co-Occurrence Texture Statistics as a Function of Gray-Level Quantization for Classifying Breast Ultrasound, in IEEE Transactions on Medical Imaging, vol. 31, no. 10, pp. 1889-1899, Oct. 2012, https://doi.org/10.1109/TMI.2012.2206398

[18] B. Pathak and D. Barooah, Textture analysis based on the gray-level Co-occurrence matrix considering possible orientations, International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering, vol. 2, no. 9.

[19] A. Eleyan and H. Demirel, Co-occurrence based statistical approach for face recognition, 2009 24th International Symposium on Computer and Information Sciences, Guzelyurt, 2009, pp. 611-615, https://doi.org/10.1109/ISCIS.2009.5291895

[20] L.Đ. Thuan, P.V. Huong, L.T.H. Van, HQ. Cuong, H.V. Hiep and N.K. Khanh, Improvement of feature set based on Apriori algorithm in Android malware classification using machine learning method, Nghiên cứu khoa học và công nghệ quân sự, no. August, pp. 32–41, 2018, ISSN 1859 – 1043.

[21] L. D. Thuan, P. Van Huong, H. Van Hiep and N. Kim Khanh, Improvement of feature set based on Apriori algorithm in Android malware classification using machine learning method, 2020 RIVF International Conference on Computing and Communication Technologies (RIVF), Ho Chi Minh City, Vietnam, 2020, pp. 1-7, https://doi.org/10.1109/RIVF48685.2020.9140779

[22] https://archive.org/details/2018-02-random-apk-collection.

[23] C.-W. Yeh, W.-T. Yeh, S.-H. Hung, and C.-T. Lin, Flattened data in convolutional neural networks: Using malware detection as case study, in Proc. Int. Conf. Res. Adapt. Convergent Syst., 2016, pp. 130–135, https://doi.org/10.1145/2987386.2987406

[24] Mohammed K. Alzaylaee, Suleiman Y. Yerima, Sakir Sezer, DL-Droid: Deep learning based android malware detection using real devices, Computers & Security, Volume 89, 2020, 101663, ISSN 0167-4048, https://doi.org/10.1016/j.cose.2019.101663.

[25] P. Feng, J. Ma, C. Sun, X. Xu and Y. Ma, A Novel Dynamic Android Malware Detection System With Ensemble Learning, in IEEE Access, vol. 6, pp. 30996-31011, 2018, https://doi.org/10.1145/2987386.2987406

[26] Z. Yuan, Y. Lu, Z. Wang, and Y. Xue, 'Droid-sec: Deep learning in Android malware detection, in Proc. ACM Conf. SIGCOMM, 2014, pp. 371–372, https://doi.org/10.1145/2740070.2631434.

[27] Z. Yuan, Y. Lu and Y. Xue, Droiddetector: android malware characterization and detection using deep learning, in Tsinghua Science and Technology, vol. 21, no. 1, pp. 114-123, Feb. 2016, https://doi.org/10.1109/TST.2016.7399288

[28] L. Xu, D. Zhang, N. Jayasena, and J. Cavazos, HADM: Hybrid analysis for detection of malware, in Proc. SAI Intell. Syst. Conf. Springer, 2016, pp. 702–724. https://doi.org/10.1007/978-3-319-56991-8_51