Towards a Reliable Hierarchical Android Malware Detection Through Image-based CNN

Jhonatan Geremias*, Eduardo K. Viegas†*, Altair O. Santin*, Alceu Britto*, Pedro Horchulhack*
*Pontificia Universidade Catolica do Parana (PUCPR) — Graduate Program in Computer Science (PPGIa), Brazil
{jhonatan, santin, alceu, pedro.horchulhack}@ppgia.pucpr.br
†Secure Systems Research Center — Technology Innovation Institute (TII), Abu Dhabi
eduardo@ssrc.tii.ae

2023 IEEE 20th Consumer Communications & Networking Conference (CCNC) | 978-1-6654-9734-3/23/$31.00 ©2023 IEEE | DOI: 10.1109/CCNC51644.2023.10060381

***Abstract*—The number of Android malicious applications keeps growing as time passes, even paving their way to official app markets. In recent years, a promising malware detection approach makes use of the compiled app source codes (*dex*), through convolutional neural networks (CNN) as an image classification task. Unfortunately, current proposals often rely on unrealistic datasets, focusing their detection on the malware families, while neglecting the detection of malware apps in the first place. In this paper, we propose a reliable and hierarchical Android malware detection through an image-based CNN scheme, implemented twofold. First, Android malware classification is performed in a hierarchically-structured local manner, initially identifying malware apps, then, their related family. Second, to ensure reliability and improve classification accuracy, only highly confident classified apps are reported, in a classification with reject option rationale. Experiments performed in a new dataset with over 26 thousand Android apps, divided into 29 malware families, compounding over 13 GB of app dex images, have shown that current image-based CNN for malware detection is unable to provide high detection accuracies. In contrast, our proposed model is able to reliably detect malware apps, improving the true-negative rates by up to 5.5%, and the average true-positive rate of the malware families of accepted apps by up to 12.7%, while rejecting only 10% of Android apps.**

***Index Terms*—Android Malware, CNN, Hierarchical Classification.**

I. Introduction

Android is currently the most popular mobile operating system, with estimations of 2.5 billion of active devices, accounting for over 70% of the smartphone market share [1]. Unfortunately, the number of unwanted Android applications, such as *malwares*, *adwares*, and *bloatwares*, is also increasing, currently affecting up to 24% of all Android users [2]. Surprisingly, 67% of all unwanted Android applications originated from official app markets [2], demonstrating that current Android *malware* detection approaches have been unable to secure users.

Over the last years, several techniques have been proposed for Android malware detection, through either *dynamic-based* or *static-based* approaches [3]. On one hand, *dynamic-based* schemes rely on executing the monitored Android app in a sandbox environment, while continuously evaluating the app behavior as time passes for malicious fingerprints [4]. As a result, it can realistically evaluate the app behavior, however, poses a great challenge concerning the generation of a proper app stimulus for the triggering of malicious activities, with reports of apps even being able to remain hidden as soon as they detect their execution in a sandbox environment [5]. On the other hand, *static-based* techniques evaluate the app characteristics in an offline manner, according to the contents of the *Android Application Pack* (apk) file, such as the requested app permissions (*manifest*), native compiled codes (*lib*), or even the Java compiled source files (*dex*) [6]. Such an approach does not require the execution of the analyzed Android app sample, significantly easing the detection process.

Several *static-based* techniques have been proposed for characterizing *malware* apps in recent years. A promising approach relies on the classification of the java compiled source files (*dex*) as an image classification task [3]. In such a case, the analyzed binary *dex* file is translated to an image format, typically by representing each *dex* byte as an image pixel, while the generated image is classified by a convolutional neural network (CNN) [7]. To achieve such a goal, a training dataset composed of a significant number of *malware* and *benign* app samples is used for the CNN training task. The built model can then be used in production to identify new malicious Android apps.

However, although such techniques can provide state-of-the-art accuracies in malware detection, proposed schemes are often unreliable for real-world settings [7]. In practice, authors mostly evaluate their model for the classification between several *malware* families, neglecting the identification of a *malware* app sample in the first place [3]. As a result, there is no guarantee that the reported accuracy rates will be achieved in real-world settings, when the built CNN model is used to identify *malware* app samples. In addition, as image-based malware detection is still in its beginnings, used evaluation datasets are often unrealistic, in their majority collected from a single source, thus, with generalization concerns.

In light of this, this paper proposes a new hierarchical and reliable image-based CNN model for classifying Android *malware*, implemented in two stages. First, Android *malware* classification is performed through an image-based CNN in a hierarchically-structured local classification setting. Thus, analyzed apps are first classified as *benign* or *malware* samples in a parent node, while the family of the malicious classified apps is identified in a child node by a specialized CNN model. Second, to improve the classification reliability, only highly

confident classifications as performed by the parent CNN node are passed to the child node, in a classification with a reject-option approach. The insight of such a proposal is that only highly confident *malware* samples must have their family identified, thus, maintaining the system's accuracy.

The main contributions of this paper are:

- A new Android malware dataset composed of over 29 thousand of *benign* and *malware* app samples, divided into 29 families.
- An evaluation of current image-based CNN approaches for Android *malware* classification, showing their unreliability to provide high accuracies when a more challenging dataset is used.
- A new hierarchical and reliable image-based CNN model for Android malware detection able to improve detection accuracies when compared to related works.

II. Preliminaries

Convolutional neural networks (CNN) have been successfully applied in several fields, typically for object detection, image classification [8], [9], and even intrusion detection [10]. However, in Android *malware* app classification it is still in its beginnings, wherein authors often apply it for classification of the Java compiled source file, namely *dex*. To achieve such a goal, the *dex* file of the analyzed Android app sample is represented as an image. In practice, the image pixels are often a direct representation of the *dex* file byte values [11]. Consequently, as the file size of each *dex* file may significantly vary, the size of the output image also varies greatly. Thus, image resizing is often applied before using it as input for a CNN model.

To implement such a process, authors rely in four sequential modules [12]. First, the *Data Acquisition* module extracts the *dex* file of the analyzed Android app *apk* file. Second, the *Image Builder* module depicts the *dex* file content as an image, often by representing each byte as an image pixel, while resizing the output image to a predefined dimension. Third, the built image is classified by a *Classification* module, that applies a previously trained CNN model. Finally, *malware*-classified samples are signaled by an *Alert* module.

In recent years, due to its promising results reported in several fields, many works have proposed highly accurate image-based CNN schemes for Android *malware* classification [3]. In general, proposed approaches classifies app samples according to their *malware* family, while overlooking the identification of *malware* samples in the first place [7]. Thus, even if they can provide highly accurate CNN-based schemes, they can only be used for classifying app samples already known to be *malware*, leaving the detection performance between *malware* and *benign* yet to be known [13].

The training procedure of CNN-based schemes requires huge amounts of diverse and realistic training data to be provided. Surprisingly, proposed approaches often make use of a single dataset, that is collected through a single data source, rendering the obtained results unrealistic. This is because, in real-world settings, the built detection scheme must be able to classify the analyzed app samples regardless of the app data source. Even if the proposed scheme takes into account the classification between *malware* and *benign* samples, due to the limitations of the underlying used dataset, the obtained results becomes unreliable.

III. Related Works

The detection of *malware* Android apps has been a widely explored topic in the literature over the last few years [3]. In general, proposed *static-based* schemes perform the classification as a pattern recognition task, typically through machine learning (ML) algorithms. For instance, J. Li *et al.* [14] proposes an ML-based model for identifying *malware* Android apps according to their requested permissions. Their model can provide high accuracies in a single dataset, but, relies on only the requested app permissions and overlooks the identification of *malware* families. Z. Ma *et al.* [15] applies ML algorithms according to the evaluated app source code control flow graph. Their proposed scheme can provide high accuracies for detecting in a two-class setting, thus, overlooking the identification of *malware* families. In contrast, S. Xue *et al.* [16] also uses the app source code control flow graph, but, for the identification of the *malware* family. Their proposed scheme, which relies on deep learning models, can provide high accuracy rates, however, neglects the identification of *malware* apps in the first place.

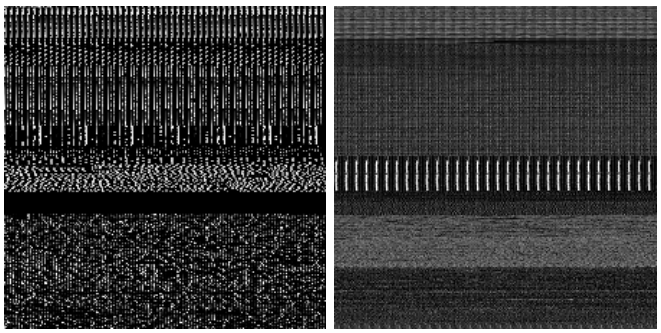
In recent years, due to their high reported accuracies, several works have resorted to image-based approaches for Android malware detection with CNN architectures [3]. For instance, J. Singh *et al.* [17] converts several *apk* files into a grayscale image and applies a CNN-based technique coped with a fusion method for classifying *malware* app families. Their proposed model was able to provide high accuracy rates, however, relies upon a single dataset and neglects the detection of *malware* apps in the first place. Another CNN-based model was proposed by D. Vasan *et al.* [7] which converts the *dex* file into a colored image format. The authors can improve accuracy through their proposed CNN architecture, however, overlooks the reliability of classification, and the hierarchy of Android apps. Similarly, T. H. Huang *et al.* [18] proposes a new CNN-based architecture for classifying *malware* apps according to the built image from the *dex* file. Their proposed model provided high accuracy rates for classifying malicious Android apps, but, overlooks the detection of the *malware* families.

IV. Problem Statement

CNN-based Android *malware* detection as an image classification task is still in its beginnings. In this section, we evaluate the performance of widely used schemes in the literature. More specifically, we first introduce our new dataset with real-world characteristics of Android *malware* samples, then, we evaluate the performance of image-based CNN schemes over it.

A. A Realistic Image Android Malware Dataset

Current datasets used in the literature for image-based classification of Android app samples cannot provide realistic



(a) *Benign* sample (b) *Malware* sample

Fig. 1: Sample images built from Android app *dex* files are available in our new IAMD dataset in a $224x224$ size.

TABLE I: Accuracy of the selected CNN architectures in a two-class IAMD dataset (*benign* vs *malware*).

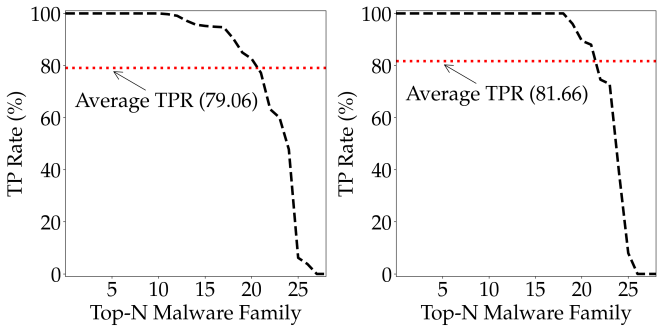
CNN Architecture	*TP (%)*	*TN (%)*	*F1*	*AUC*
Resnet50	92.95	92.77	0.929	0.97
InceptionV3	94.36	92.34	0.934	0.98

characteristics of real-world environments. Authors generally resort to a single dataset prone to the problem inherent in its data source. Ideally, used datasets must be made of a representative number of *malware* and *benign* samples that were collected from a variety of sources throughout time.

To address such a shortcoming, we present the *Image-based Android Malware Dataset* (IAMD). The built dataset comprises *benign* and *malware* samples from two publicly available datasets, namely *CICMalDroid* [19], and *MaliImages* [20]. The prior, *CICMalDroid*, is made of $17,341$ *benign* and *malware* samples, divided into 4 *malware* families. The former, *MaliImages*, is made of $9,458$ *malware* samples, divided into 25 families. As a result, our dataset, IAMD, is made of $26,799$ Android app samples, with $6,815$ *benign* samples and $19,984$ *malware* samples divided into 29 families. The collected Android *apk* files are represented as an image in our dataset, using the same methodology used in the literature (see Section II). Thus, the *dex* file is extracted from the *apk* app file and converted to a grayscale image file in PNG format. The images were generated through the *Python Image Library* (PIL) API $v.8.4.0$. The dimensions of the built images vary according to the *dex* file size, hence, all images were resized to a $224x224$ dimension before using them for the application of CNN architectures. The IAMD dataset is made of over 13.4 GB of images. Figure 1 shows sample images from the IAMD dataset.

B. Android Malware Detection as an Image Classification Task

The evaluation aims at answering two main research questions: (***RQ1***) *How do traditional CNN architectures perform classifying malware and benign Android app samples?* (***RQ2***) *How do traditional CNN architectures perform classifying malware families?*



(a) InceptionV3 (b) Resnet50

Fig. 2: Top N TP rate of selected CNN architectures for classifying the *malware* family in a 29-class IAMD dataset.

Two widely used CNN architectures were evaluated, namely Restnet50, and InceptionV3. The evaluated architecture were executed for $1,000$ epochs, and their learning rate was set empirically according to the resulting loss and a momentum weight of 0.9. A random undersampling without replacement is used in the training procedure to balance the occurrence between the classes. For training purposes, the IAMD dataset was split into train, validation, and test datasets, comprising 40%, 30%, and 30% of the original dataset, respectively. The CNNs were implemented through $keras$ API $v.2.4.0$, and $tensorflow$ API $v.2.4.1$. The classification performance was measured according to their True-Positive (TP), True-Negative (TN), F1-Score, and AUC metrics. The TP denotes the ratio of *malware* instances correctly classified as *malware*, while the TN denotes the ratio of *benign* instances correctly classified as *benign*. The F1 score was computed as the harmonic mean of precision and recall values while considering *malware* as positive samples and *benign* as negative samples.

The first experiment aims at answering $RQ1$ and evaluates the classification performance of the selected CNN architectures while classifying Android app samples as either *benign* or *malware*, in a two-class setting. To achieve such a goal, the selected architectures are trained with the training dataset, randomly built with 40% of IAMD, regardless of the underlying *malware* family, and also using *benign* samples.

Table I shows the classification performance of the selected CNN architectures while classifying IAMD samples as either *malware* or *benign*. Surprisingly, the evaluated approaches were unable to provide significantly high classification accuracies. For instance, the InceptionV3 CNN, which provided the highest AUC of 0.98, presented a TP rate of only 94.36%, and a TN rate of only 92.37%. Thus, the experiment shows that traditional CNN-based techniques cannot reliably identify malicious Android app samples.

The second experiment aims at answering $RQ2$ and evaluates the classification performance of the selected CNN architectures when used for the identification of the *malware* family, as commonly made in the literature. To achieve such a goal, the *benign* samples are removed from the IAMD dataset, and the CNN architectures classify its input between one of the 29 *malware* families. Thus, it assumes that the CNN input

is always a *malware* sample.

Figure 2 shows the *malware* family identification accuracy as ordered from the most accurate to the least accurate. It is possible to note that the selected architectures could also not provide high accuracy for identifying the *malware* family. For instance, on average, the selected architectures presented a TP rate of only 79% and 81%. Thus, image-based CNN classification of Android *malware* families can also not provide reliability.

C. Discussion

Detection of *malware* Android apps through image-based CNN is still in its beginnings. In this section, we have introduced a new dataset, namely IAMD, with over 26 thousand Android apps. Experiments with widely used detection approaches have shown that current techniques cannot provide high accuracies in their detection. Thus, when used for the detection in a two-class setting, considering *malware* and *benign* samples, or the identification of the *malware* family, selected approaches cannot reach the desired level of reliability, significantly degrading their accuracy. As a result, current image-based CNN techniques for *malware* Android app detection are unreliable for production deployment.

V. A Hierarchical and Reliable Android Malware Detection Model

To address the challenges above, we present a new reliable hierarchical Android *malware* detection through image-based CNN. The insight of the proposal is that *malware* classification accuracy can be improved through classification with reject option rationale. In contrast, the *malware* family can be identified in a hierarchical classification setting. The proposed model is shown in Figure 3 and is composed by two main steps, namely *Android App Classification* and *Reliable Malware Family Classification*.

The proposal considers an Android *malware* image-based CNN classification scheme, first aiming at the detection of *malware* samples, and their family if a given *app* is classified as malicious. The classification procedure starts with an Android *apk* file as input for classification. The corresponding *apk dex* file is extracted and represented in an image format. Several techniques can be used to fulfill such a task, such as those used in our IAMD dataset (see Section IV-A). The built image is classified as either *benign* or *malware* by a *CNN* module (*Hierarchical Structure*, Fig. 3), which outputs a corresponding classification confidence value. Our scheme uses the confidence value in a classification with a reject option approach to improve detection accuracy. The insight of such a proposal is that only highly confident classifications, thus more likely to be correctly classified, are accepted by our model, maintaining the system's reliability. Our model discards rejected classifications. In contrast, accepted *malware*-classified apps are classified by a child CNN model (*Hierarchical Structure*, Fig. 3), for the proper identification of the *malware* family.

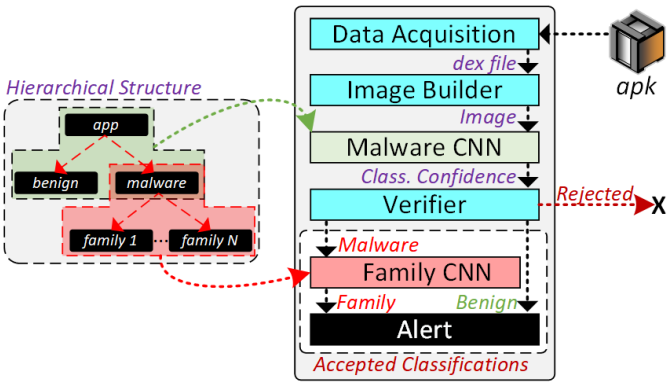


Fig. 3: Proposed reliable hierarchical Android malware detection through image-based CNN.

The next subsections further describe our proposed classification scheme and verification process.

A. Android App Classification

In general, image-based Android *malware* proposals aim for the classification of *malware* samples among a variety of families. Therefore, the input of the proposed schemes is assumed to be *malware*. However, in a real-world setting, Android apps must first be identified as *malware* or *benign*. Only then, if a given *malware* is signaled, can its family be identified.

In light of this, our proposed model performs the classification of Android apps considering a hierarchically-structured local classification manner. The rationale of such a scheme is that Android apps can be classified in a parent node as either *benign* or *malware* and only then, the *malware* family can be identified in a child node. To achieve such a goal, our proposed model relies on two CNNs, that perform the classification in a hierarchical manner (*Hierarchical Structure*, Fig. 3). On one hand, the scheme input is classified as *benign* or *malware* by a two-class CNN model (*Malware CNN*, Fig. 3). On the other hand, *malware*-classified apps, with high confidence in their classification, are also classified according to their family (*Family CNN*, Fig. 3).

As a result, our proposed scheme can properly identify *malware* and its related family. This is because the hierarchical classification scheme first identifies *malware* apps, before using them to identify the *malware* family. However, to provide such identification, our proposed model must ensure classification reliability.

B. Reliable Malware Family Classification

Identifying Android *malware* apps is a challenging task, wherein detection schemes are subject to a high rate of false positives (see Table I). Therefore, proposed schemes must ensure that only correct classifications are used to trigger *malware* alerts, otherwise, the user may ignore additional alerts. To provide such a level of reliability, our model copes with the aforementioned hierarchical classification with a verifier module. The module goal is to ensure that only highly

confident classifications are used to alert users, thus, improving the system classification accuracy. Therefore, classification is performed through the reject option approach, wherein the low confident classification is not used to trigger alerts.

As shown in Figure 3, the classification procedure starts with a to-be-classified Android app *apk* file. The *Data Acquisition* module extracts the *dex* file for further analysis. The obtained *dex* file is input by an *Image Builder* module, which compounds a correlated image. Several approaches can be used to fulfill such a task, such as the technique used to build our IAMD dataset (see Section IV-A), which represents each *dex* file byte as an image pixel. Then, the built image is classified by the *Malware CNN*, as either *malware* or *benign*, and outputs a corresponding classification confidence value. The *Verifier* module assesses the classification confidence value to ensure that the desired level of classification confidence threshold is met. Our model rejects low confident classifications. In contrast, highly confident classifications are used to alert users, a situation wherein highly confident *malware*-classified apps are also used as input by a second CNN, following our hierarchical structure (*Family CNN*, Fig. 3). Finally, accepted classifications are sent to the *Alert* module, reporting the found *malware* family or attesting *benign* apps.

VI. Evaluation

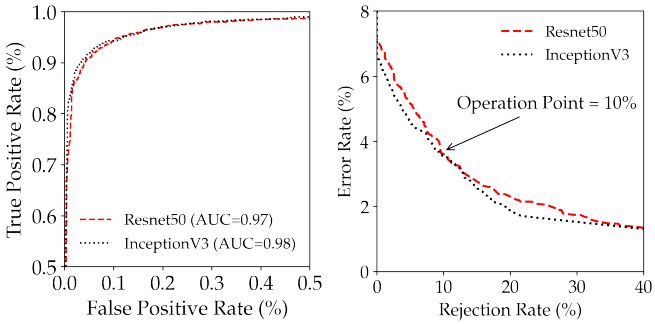
The evaluation aims at answering two main research questions: (***RQ3***) *How does the proposed classification assessment approach improves the malware classification model?* (***RQ4***) *How does the proposed model perform at classifying accepted malware families?*

A. Model Building

The proposed model (Fig. 3) was implemented and evaluated on top of our built dataset, namely IAMD (see Section IV-A). Similarly, the two used CNN architectures were also evaluated in our proposal, namely Resnet50, and InceptionV3. The evaluated architecture was executed for $1,000$ epochs, and its learning rate was set empirically according to the resulting loss and a momentum weight of 0.9. A random undersampling without replacement is used in the training procedure to balance the occurrence between the classes. For training purposes, the IAMD dataset was split into train, validation, and test datasets, comprising 40%, 30%, and 30% of the original dataset, respectively.

B. A Reliable Hierarchical Classification

The first experiment aims at answering $RQ3$. It evaluates the classification performance of our proposed model while making use of the *Verifier* module for the assessment of the performed classifications in a two-class setting (*Malware CNN*, Fig. 3). Figure 4a shows the ROC curve of the selected CNN architectures. Then, making use of the two selected CNNs, we evaluate the proposed *Verifier* module for the assessment of the performed classifications. The classification thresholds were defined through the Class-Related-Threshold (CRT) approach,



(a) ROC Curve without *Verifier* module (b) Error $vs.$ reject with *Verifier* module

Fig. 4: Proposed scheme accuracy performance and error-reject tradeof for the classification of *benign* and *malware* samples (*Malware CNN*, Fig. 3).

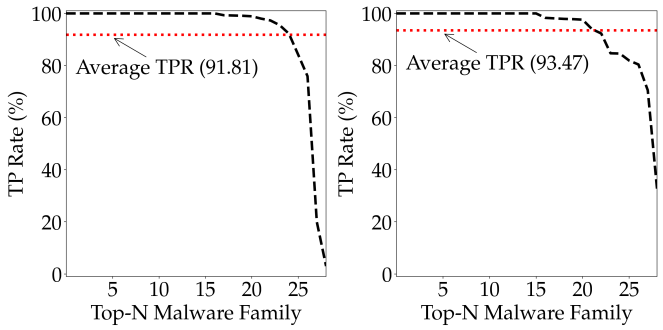
TABLE II: Proposed scheme accuracy performance for the classification of *benign* and *malware* samples (*Malware CNN*), according to the rejection rate (Fig. 4b).

CNN	*TP (%)*	*TN (%)*	*F1*	*Rejection (%)*
Resnet 50	92.95	92.77	0.929	0.00
	96.25	98.27	0.972	5.00
	96.54	**98.30**	**0.970**	**10.00**
	97.00	98.33	0.977	15.00
Inception V3	94.26	92.34	0.934	0.00
	93.08	95.60	0.943	5.00
	95.59	**96.59**	**0.959**	**10.00**
	97.51	97.12	0.975	15.00

which searches for the optimum acceptance threshold for each class.

Figure 4b shows the error vs reject tradeof of the two selected CNN architectures for the classification in a two-class setting (*Malware CNN*, Fig. 3). It is possible to note a relationship between the error rate and the rejection rate, thus, a higher rejection of the evaluated Android apps can decrease the average error rate of our proposal. Table II shows the accuracy performance of our model coped with the *Verifier* module according to a variety of rejection rate. It is important to note that the operation point must be chosen according to the user's discretion. For instance, rejecting only 10.0% of instances, our proposed model can improve the TP rate by 3.6% and 1.33% for the Resnet50 and InceptionV3, respectively. As a result, the proposed scheme that evaluates the classification confidence values through the *Verifier* module can improve the system's reliability in *malware* detection.

The second experiment aims at answering $RQ4$ and evaluates our proposed model for the classification of the *malware* families accepted by our *Verifier* module. To achieve such a goal, we evaluate the classification accuracy for identifying the 29 families in the IAMD dataset, using only the accepted apps according to the used rejection operation point (10% *Rejection*, Table II). Figure 5 shows the *malware* family identification accuracy of our proposal (*Family CNN*, Fig 3) as ordered from the most accurate to the least accurate from our



(a) InceptionV3 (b) Resnet50

Fig. 5: Top N TP rate of selected CNN architectures with *Verifier* module (*Operation Point*, Fig. 4b) for classifying the *malware* family of accepted *malware* classified apps.

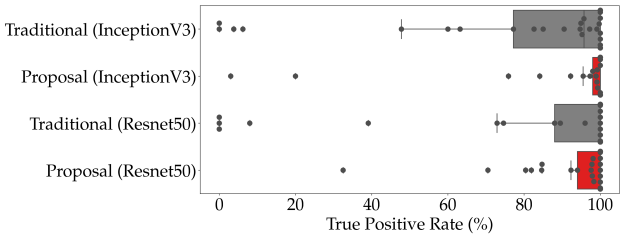


Fig. 6: Comparison of the TP rate distribution for identifying the *malware* families of our proposal versus their counterparts without the classification assessment.

proposal. In such a case, our proposed scheme can improve the classification of the accepted *malware* families significantly. More specifically, by rejecting only 10% of evaluated apps, our proposed model improves the average TP rate of *malware* family identification by 12.75% and 11.81% for the Resnet50 and InceptionV3 respectively (Fig. 2 vs 5).

We further investigate the impact of our proposed classification assessment approach in Figure 6. In such a case, our proposed model, when rejecting only 10% of Android apps, classifies 75% (22 out of the 29) of *malware* families with at least 98% and 94% of TP rate. In comparison, the traditional approach identifies the same number of families with at least 77% and 88% for the InceptionV3 and Resnet50, respectively.

VII. Conclusion

Android *malware* detection as an image classification task through CNNs is still in its beginnings. In this paper, we have proposed a new reliable and hierarchical Android *malware* detection through image-based CNN for the detection of malicious apps as well as their family. Our proposed scheme can improve the detection of *malware* apps while only rejecting a small subset of app samples. In addition, it can improve the average detection rate of the accepted *malware* families compared to traditional techniques.

Acknowledgment

This work was partially sponsored by Brazilian National Council for Scientific and Technological Development (CNPq), grant nº 304990/2021-3.

References

[1] T. inMobi, "Understanding android users worldwide," 2021. [Online]. Available: https://www.inmobi.com/blog/2021/08/09/understanding-android-users-worldwide

[2] P. Kotzias, J. Caballero, and L. Bilge, "How did that get in my phone? unwanted app distribution on android devices," in *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE, May 2021, p. 53–69.

[3] J. Qiu, J. Zhang, W. Luo, L. Pan, S. Nepal, and Y. Xiang, "A survey of android malware detection with deep neural models," *ACM Computing Surveys (CSUR)*, vol. 53, no. 6, pp. 1–36, Feb. 2021.

[4] M. Spreitzenbarth, F. Freiling, F. Echtler, T. Schreck, and J. Hoffmann, "Mobile-sandbox," in *Proceedings of the 28th Annual ACM Symposium on Applied Computing*. ACM Press, 2013, pp. 1808–1815.

[5] T. Vidas and N. Christin, "Evading android runtime analysis via sandbox detection," in *Proceedings of the 9th ACM symposium on Information, computer and communications security*. ACM, Jun. 2014, pp. 447–458.

[6] R. Taheri, M. Ghahramani, R. Javidan, M. Shojafar, Z. Pooranian, and M. Conti, "Similarity-based android malware detection using hamming distance of static binary features," *Future Generation Computer Systems*, vol. 105, pp. 230–247, Apr. 2020.

[7] D. Vasan, M. Alazab, S. Wassan, H. Naeem, B. Safaei, and Q. Zheng, "IMCFN: Image-based malware classification using fine-tuned convolutional neural network architecture," *Computer Networks*, vol. 171, p. 107138, Apr. 2020.

[8] J. Geremias, E. K. Viegas, A. S. Britto, and A. O. Santin, "A motion-based approach for real-time detection of pornographic content in videos," in *Proceedings of the 37th ACM/SIGAPP Symposium on Applied Computing*. ACM, Apr. 2022.

[9] F. Ramos, E. Viegas, A. Santin, P. Horchulhack, R. R. dos Santos, and A. Espindola, "A machine learning model for detection of docker-based APP overbooking on kubernetes," in *ICC 2021 - IEEE International Conference on Communications*. IEEE, Jun. 2021.

[10] B. B. Bulle, A. O. Santin, E. K. Viegas, and R. R. dos Santos, "A host-based intrusion detection model based on OS diversity for SCADA," in *IECON 2020 The 46th Annual Conference of the IEEE Industrial Electronics Society*. IEEE, Oct. 2020.

[11] J. Geremias, E. K. Viegas, A. O. Santin, A. Britto, and P. Horchulhack, "Towards multi-view android malware detection through image-based deep learning," in *2022 International Wireless Communications and Mobile Computing (IWCMC)*. IEEE, May 2022.

[12] R. R. dos Santos, E. K. Viegas, A. O. Santin, and V. V. Cogo, "Reinforcement learning for intrusion detection: More model longness and fewer updates," *IEEE Transactions on Network and Service Management*, pp. 1–17, 2022.

[13] P. Horchulhack, E. K. Viegas, and A. O. Santin, "Toward feasible machine learning model updates in network-based intrusion detection," *Computer Networks*, vol. 202, p. 108618, Jan. 2022.

[14] J. Li, L. Sun, Q. Yan, Z. Li, W. Srisa-an, and H. Ye, "Significant permission identification for machine-learning-based android malware detection," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 7, pp. 3216–3225, Jul. 2018.

[15] Z. Ma, H. Ge, Y. Liu, M. Zhao, and J. Ma, "A combination method for android malware detection based on control flow graphs and machine learning algorithms," *IEEE Access*, vol. 7, pp. 21 235–21 245, 2019.

[16] S. Xue, L. Zhang, A. Li, X.-Y. Li, C. Ruan, and W. Huang, "AppDNA: App behavior profiling via graph-based deep learning," in *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*. IEEE, Apr. 2018, pp. 1475–1483.

[17] J. Singh, D. Thakur, T. Gera, B. Shah, T. Abuhmed, and F. Ali, "Classification and analysis of android malware images using feature fusion technique," *IEEE Access*, vol. 9, pp. 90 102–90 117, 2021.

[18] T. H.-D. 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, Dec. 2018, pp. 2633–2642.

[19] S. Mahdavifar, A. F. A. Kadir, R. Fatemi, D. Alhadidi, and A. A. Ghorbani, "Dynamic android malware category classification using semi-supervised deep learning," in *2020 IEEE Intl Conf on Dep., Aut. and Secure Computing (DASC)*. IEEE, Aug. 2020, pp. 515–522.

[20] 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*. ACM Press, 2011, pp. 1–7.