electronics

MDPI

Article

Deep Learning-Based Attack Detection and Classification in Android Devices

Alfonso Gómez and Antonio Muñoz *

Network, Information and Computer Security Lab (NICS), University of Malaga, 29071 Malaga, Spain; alfonso@lcc.uma.es

* Correspondence: anto@uma.es

Abstract: The increasing proliferation of Androidbased devices, which currently dominate the market with a staggering 72% global market share, has made them a prime target for attackers. Consequently, the detection of Android malware has emerged as a critical research area. Both academia and industry have explored various approaches to develop robust and efficient solutions for Android malware detection and classification, yet it remains an ongoing challenge. In this study, we present a supervised learning technique that demonstrates promising results in Android malware detection. The key to our approach lies in the creation of a comprehensive labeled dataset, comprising over 18,000 samples classified into five distinct categories: Adware, Banking, SMS, Riskware, and Benign applications. The effectiveness of our proposed model is validated using well-established datasets such as *CICMalDroid2020*, *CICMalDroid2017*, and *CICAndMal2017*. Comparing our results with state-of-the-art techniques in terms of precision, recall, efficiency, and other relevant factors, our approach outperforms other semi-supervised methods in specific parameters. However, we acknowledge that our model does not exhibit significant deviations when compared to alternative approaches concerning certain aspects. Overall, our research contributes to the ongoing efforts in the development of advanced techniques for Android malware detection and classification. We believe that our findings will inspire further investigations, leading to enhanced security measures and protection for Android devices in the face of evolving threats.

Keywords: android malware detection; deep learning; malware classification in mobile devices; machine learning

Citation: Gómez, A.; Muñoz, A. Deep Learning-Based Attack Detection and Classification in Android Devices. *Electronics* **2023**, *12*, 3253. https://doi.org/10.3390/electronics12153253

Academic Editors: Yiming Liu, Zhi Zhang and Yue Meng

Received: 8 June 2023
Revised: 21 July 2023
Accepted: 25 July 2023
Published: 28 July 2023

1. Introduction

Technological advancements in recent years have impacted the way our personal data are being shared and processed. The evolution of technology has brought forward new techniques to share, process, and store data. This has derived new models of data (including personal data) processing, but also introduced new threats and difficulties for the end user to understand and control the processing. The continuous online presence of end users has resulted in the increased processing of large amounts of personal data on a daily basis. Think of online shopping or using a mobile application to navigate to a specific location or contact friends and family. The whole data lifecycle has been augmented with many actors being involved and eventually end users not being able to fully understand and control who has access to their personal data, or for how long and for what purpose.

The growth in popularity of mobile devices is on the rise. This growth brings with it certain drawbacks. Figure 1 shows how, in 2021, the number of attacks on mobile users decreased from the trend of previous years and this is maintained with a slight growth in 2022. Although it may seem like a significant improvement in safety, this view is somewhat misleading. Malware detection systems [1–4] appear in the literature and have given good results in the past; however, today, they have serious limitations. In the current scenario, it is not enough to identify malicious software, but it is essential to specify what kind of

malware we find. This provides very useful information for the mitigation of possible damage.

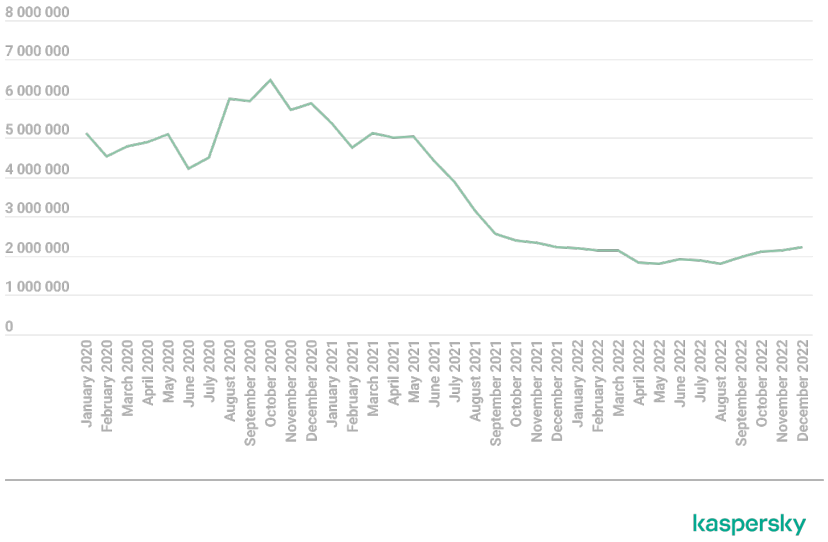


Figure 1. Attacks on mobile users during the period 2020–2022 (Karpersky Source).

It is considered that attacks are becoming increasingly sophisticated, both in terms of functionality and malware vectors with the capacity to produce greater damage. In fact, as far as *Adware* and *Riskware* are concerned, in 2021, there were again repeated incidents caused by malicious code injection in some well-known applications perpetrated through the use of advertising SDKs, such as the highly publicized case of *CamScanner* (https://www.kaspersky.com/blog/camscanner-malicious-android-app/28156, accessed on 24 July 2023), an application aiming to scan documents in which malicious code (the *AndroidOs.Necro.n dropper*) was included inside advertising libraries in the official *APKPure client*, in the same way as it was performed in a *WhatsApp* version. Although they have been in vogue, they have not been the only ones reported, but malware was also found in Google Play apps both with the introduction of Trojans and phishing attacks.

Regarding Trojans and specifically banking Trojans, it is relevant to mention that they have been acquiring new capabilities that make them very effective and difficult to detect on many occasions. Cases such as *Fakecalls* [5], *Sova* (https://www.threatmark.com/about-s-o-v-a-malware-family/) or *Vultur* [6] have had a strong impact in 2021.

Although the use of *SMS phishing* is not new and users are usually alert so as not to fall for this trick, the truth is that this method is evolving in such a way that it is once again a concern. Lately, there have been reports of *SMS malware* attacks that have resulted in the theft of passwords and banking credentials. The *FluBot* campaign has been adapting from text messages from courier companies to commercial brands such as Amazon. *FluBot* makes use of the order delivery notification that informs the user to click on a link to get tracking information, as the order is on its way, and from there the attack begins.

Despite the wide spectrum of different types of attacks on Android, in this paper, we focus on those based on *SMS*, *Banking*, *Adware* and *Riskware*. As a consequence of the recent attacks discussed, we focus on these attacks without losing sight of the fact that the studies carried out are applicable to other types of attacks in the imminent future. There are works that employ deep learning coupled with graph embedding to improve representation and malware detection [7,8].

In this paper, a framework with the main objective of detecting and classifying the use of malware on Android is proposed. As such, we rely on extracting data from the behavior of potential malware through a static observation process. The framework makes use of supervised neural models. Specifically, supervised learning techniques based on *machine learning (ML)* and *deep learning (DL)* are used. Such is the case of [9] that provides

significant improvements in IDS performance using machine learning techniques. We conclude that, for most cases, *DL* is more efficient for our purposes. Our ultimate goal is to resolve the ambiguity between threats during detection, primarily to achieve minimal error in predicting malware applications as benign applications. To do so, it is necessary to identify which ones are the most confusing to each other, in order to provide techniques to identify them unambiguously.

The major advantage of using *DL* and *ML* techniques for vulnerability detection is that it will be able to detect techniques that have not been seen before. Another advantage is the automation of the process, since it will not depend on external agents. Finally, they have the ability to react quickly and detect them quickly, which is a quality that is highly valued in a security system. On the other hand, this technology can also be used to create more complex threats. In turn, they can also take advantage of it to attack our model and find weaknesses in it, to generate threats that are not able to be detected.

Furthermore, our work encompasses several notable contributions, which are as follows:

- Data generation and release: We have derived and released three datasets, namely *CICMal-Droid2020, CICMal-Droid2017*, and *CIC-AndMal2017*, comprising a total of 18,188 Android samples. These datasets include up-to-date samples from the year 2021 and are categorized into five distinct classes: Benign, Adware, SMS malware, Banking, and Riskware.
- In-depth analysis of static characteristics: We conducted a comprehensive examination of the most relevant static characteristics associated with malware samples. This investigation delved into understanding the intrinsic properties and features of the malware instances.
- Exploration of DL and ML algorithms: We performed an extensive exploration of various deep learning (DL) and machine learning (ML) algorithms to determine the optimal choice for achieving the best performance across different problem domains. This evaluation aimed to identify the algorithms that yielded superior results in terms of accuracy and efficiency.
- Analysis of experimental results: We meticulously analyzed the outcomes of our experiments to identify the types of malware that posed the greatest challenges in terms of detection and classification. Our findings demonstrated that our model exhibits an impressive F1 score of 98.9% and a false positive rate of 0.99%. These results instill confidence in the robustness and effectiveness of our proposed approach.
- Development of an Android application: We developed an Android application that facilitates queries and investigations into the previously studied threats. This application serves as a practical tool for evaluating the performance of our trained algorithm in real-world mobile environments. By designing and implementing this proof of concept, we validated the applicability and viability of our model in realistic scenarios.

In summary, our contributions encompass the generation and release of comprehensive datasets, the in-depth analysis of static characteristics, the exploration of DL and ML algorithms, insightful analysis into experimental results, and the development of an Android application for the practical utilization and validation of our proposed model.

The rest of the paper is organized as follows. Section 2 provides an overview of the related work. Section 3 describes the methodology. The algorithm design is presented in Section 4. In Section 5, an analysis of the theoretical models proposed is included. Section 6 provides a comprehensive account of the vulnerability classifier, encompassing details on the dataset, model evaluation methodology, design, and outcomes. Section 7 conducts a comparative analysis of our research in relation to other analogous studies, elucidating the advancements it introduces. Section 8 summarizes the findings of our study and presents ongoing works that are currently in progress.

2. Related Works

Due to the intricate nature of infrastructure and communication technologies, coupled with recent advancements, the task of detecting and classifying malware has become in-

creasingly challenging. In addition, the growth and expansion of malware for Android have prompted constant suggestions for defense mechanisms and new methods to combat malware by the research community. Among the most prominent techniques is the use of deep learning as a malware classifier based on its type. One of the main challenges in applying these techniques lies in finding the most suitable classification algorithm based on the detection purpose. Therefore, it is essential to consider aspects such as detection performance and accuracy. The identification and classification of diverse malware types offer a deeper understanding of their functionality, enabling the extraction of behavioral patterns and even the prediction of malware evolution. It is widely acknowledged that malware undergoes constant evolution [10], reaching unprecedented levels of sophistication and endowing new strains with stealth and dynamic capabilities. Consequently, detecting and comprehending the operational aspects of malware have become progressively arduous. Attackers employ anti-analysis techniques to launch sophisticated attacks, facilitating advanced malware concealment. This, combined with the dynamic evolutionary capacity of malware, allows it to assume semantically similar but structurally dissimilar forms, thereby causing novel damages such as connection blockages, system corruption, password theft, and more, posing threats to users, organizations, and systems. Recent history demonstrates how malware has become the primary tool for initiating large-scale attacks, resulting in severe damages and economic losses [11–16]. Traditionally, various techniques have been used for malware detection, including signature-based approaches (utilizing regular expressions, file names, etc.), behavior-based or heuristic methods [17–19], and static feature analysis [20]. However, these techniques have become ineffective against new malware variants that employ dynamic concealment techniques such as obfuscation, polymorphism, metamorphism, and packing. To tackle this problem, the research community has turned to machine learning techniques for malware detection and classification [21–23]. Some authors have categorized malware types based on features (extraction methods, feature types) and algorithms (signature-based, AI-based) [24]. Deep learning approaches have also been proposed to achieve higher precision in both tasks [25–27], capable of extracting valuable features. However, these approaches have also revealed new vulnerabilities in terms of deceiving AI-based malware analysis systems themselves [28–30], leading to erroneous decisions made by these systems. Nevertheless, research efforts are underway to develop defenses against such adversarial attacks [31]. Among the various techniques employed in AI-based malware detection, including machine learning, association mining, graph mining, concept analysis, and signature creation, the methods are typically classified as supervised or unsupervised based on the learning approach employed. In the supervised model, the classification algorithm learns to perform its task from a labeled input dataset. Numerous works have been conducted in this area [32–36], among others. In contrast, the unsupervised model does not rely on labeled data as input; instead, the algorithm itself generates different classes by extracting patterns from the input data without explicit guidance. Numerous studies have explored this approach as [35,37–39]. An intermediate alternative exists, utilizing both labeled and unlabeled data as input [40,41]. Now, let us review the different fronts to address. Firstly, the components of an Android application are grouped into what is known as an APK (Android package). There is an XML document known as AndroidManifest.xml that contains all the information required by the Android framework about the application. Among other information contained in this file are the permissions imposed and requested by the application, along with other sensitive information. On the other hand, different techniques exist for malware detection in Android (static vs. dynamic), although a sophisticated analysis is necessary. We highlight the sensor-based event system of mobile platforms, which allows malware to respond to nearby SMS, position changes, etc. Static analysis techniques involve examining specific parts of the application without actually executing them. Various static techniques exist, such as signature-based analysis, permission-based analysis, or component-based analysis. Regarding our proposal, component-based techniques are of particular interest, involving decompiling the entire app to inspect bytecode and significant components to identify pos-

sible vulnerabilities [42,43]. Although these techniques often yield good results, the main challenge is the lack of real execution paths and suitable execution conditions. This is compounded when code obfuscation and dynamic code loading techniques are employed [44]. On the other hand, dynamic analysis techniques involve executing the application in a virtual machine or on a physical device. During its execution, the application's behavior is examined and analyzed, allowing for deeper inspection. Dynamic analysis is more specific than static analysis since its primary objective is to achieve high code coverage to monitor any possible malicious behavior [45]. These techniques are not without disadvantages, with the main challenge being the amount of resources required to implement them adequately, limiting their application on resource-limited devices such as mobile phones. Additionally, advanced malware seeks to recognize emulators or other frameworks used for dynamic analysis to camouflage their behavior and evade detection [44]. Lastly, hybrid techniques exist that incorporate the most advantageous features of both static and dynamic analysis. However, these techniques do not reduce the amount of required resources and often consume system resources, resulting in excessive analysis times on Android devices. Although the Android Malware dataset has been previously used for malware classification [46,47], our proposed study focuses on predicting malware at the static level, specifically targeting *APKs* before their installation, in contrast to the dynamic approach employed in previous works [46,47]. Another distinguishing aspect of our research is the utilization of a fully labeled deep learning model, as opposed to the semi-labeled approach employed in both aforementioned papers. The considerable effort put into attaining a labeled dataset with a sufficient number of samples yields novel results, some of which are highly satisfactory, while others are relatively less favorable, as discussed in later sections of the manuscript.

3. Methodology

This section provides a description of the methodology employed to construct the threat type classifier algorithm. The methodology consists of several stages, as depicted in Figure 2, including static analysis, feature extraction, model training, and evaluation. The initial step of the methodology involves the examination of the feature set to be incorporated into the malware prediction and classification model. In this study, a proprietary dataset was curated, which encompassed the gathered characteristics necessary for training the algorithm. The subsequent stage focuses on algorithm implementation, utilizing the extracted features obtained during the design phase. A deep learning model is employed, utilizing the dataset obtained during the design phase for training purposes. The trained model serves as the foundation for the algorithm responsible for examining applications and classifying them based on potential threat types. To assess the effectiveness and validation of the model, a series of evaluation methods are employed. These methods analyze the performance of the threat type classifier algorithm and provide insights into its accuracy and reliability. For classification purposes, the applications are categorized as either benign or malware. Specifically, this study concentrates on four primary malware types: *Adware*, *Riskware*, *SMS*, and *Banking*. These types were chosen due to their significant impact on Android applications. *Adware* refers to applications that force the display of unwanted and potentially misleading advertisements. *Riskware* encompasses applications that are initially considered legitimate but can cause severe damage if they fall into the wrong hands. *SMS*, is utilized by certain threats for malicious purposes, while the *Banking* category of malware concentrates on unauthorized access to bank accounts.

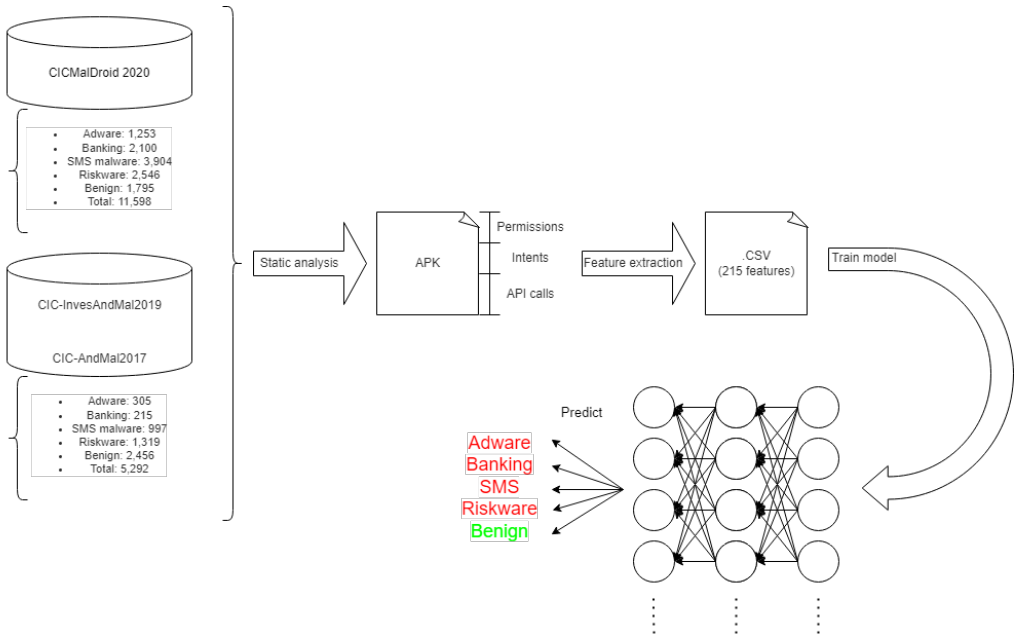


Figure 2. Methodology.

The methodology described above provides a systematic approach to constructing the threat type classifier algorithm, incorporating static analysis, feature extraction, model training, and evaluation. By following this methodology, we aim to develop an effective and reliable algorithm for classifying applications based on their potential threat types.

Data collection: The methodology begins with data collection, which varies slightly depending on the threat type being analyzed. For phishing, a repository based on a trusted source [48] was utilized to obtain data labeled according to [49]. The URLs were characterized based on different features, such as:

- *Address bar characteristics:* These refer to the properties of the URL, such as the size or absence of certain characters.
- *Abnormality-based characteristics:* This category is used to characterize uncommon aspects that are not typically found in legitimate web pages.
- *HTML and JavaScript-based functions:* These functions are utilized to obscure details and make phishing detection more challenging when accessing the website.
- *Domain-based characteristics:* In contrast to address bar characteristics, this category focuses more on the search engine optimization (SEO) aspects of the domain.

For SMS spam detection, a repository obtained from [50] was used. This database consists of a diverse collection of labeled text messages, indicating their classification as spam or non-spam. Since our input data consist of text messages, we need to characterize them using Tokenizer. This Keras tool allows us to characterize the messages based on the words present in them.

One of the key objectives of our model was to detect malware in APKs before installation, which is why we sought to base the labels in the training repository on the static analysis of the APKs. Static analysis involves examining various aspects of the application without executing it, thereby avoiding the activation of any malware. Since we could not find any existing databases with the desired characteristics, I decided to download a repository containing various malicious apps [47,51].

As previously mentioned, a thorough documentation work was been carried out, which included the review of relevant works [52–54] on the static analysis of malicious APKs, in order to identify up to a total of 210 features. A description of each of the features is beyond the scope of this paper; below, we describe the most relevant ones:

- *API calls:* These are pre-programmed function calls found in the APK's .dex files, which are used to save time by avoiding the need to create them.
- *Permissions:* Upon installation, permissions represent the initial requests made by an application. These play a crucial role in security as they determine the services to which the app gains access. Permissions are specified in the *AndroidManifest.xml* file of the APK.
- *Intents:* Intents are used to request an action from a component of another application. They are specified in the files.

After selecting these characteristics, we took advantage of Androguard to perform a static analysis of the application repository and create a database with the 210 chosen labels based on the study conducted.

Training and Analysis of Theoretical Models

The following steps were followed during the training and the analysis phases of the theoretical models:

- Data loading and separation: The labeled data were loaded into the system and randomly shuffled to prevent any biases and overfitting. Subsequently, the dataset was divided into training data (75%) and test data (25%) to effectively evaluate the accuracy of the trained model.
- Model training: Three models were chosen for study: decision tree, random forest, and SVM. For training these models, the tools provided by the *sklearn* library were utilized. On the other hand, training the neural network model was a more complex process that required the development of subroutines to replicate the learning process. Unlike the decision trees, random forest and SVM involve selecting and adjusting several parameters to optimize their accuracy and efficiency:
  - Optimization: Techniques such as gradient descent, specifically utilizing the negative log likelihood cost function, were employed to improve the weights of the neural network's neurons. The learning rate parameter, which determines the speed of learning in each iteration, played a crucial role in effectively training the neural network. In the optimization process, the learning rate was dynamically adjusted using exponential decay rates to enhance the efficiency of learning.
  - Iterations and nodes: Determining the optimal number of iterations for the model was essential to avoid underfitting or overfitting. Underfitting occurs when the model fails to capture the complexities of the data due to insufficient iterations, while overfitting arises from excessive training on the same dataset, causing the model to become overly specialized and perform poorly on new data. This issue is particularly significant when dealing with small databases, as the model might become excessively biased towards certain patterns present in the training set.
  - Batch size: The batch size parameter defines the number of examples from the dataset used to train the model in each iteration. It is particularly useful when memory is limited, as it allows for training with smaller batches of data instead of loading the entire dataset at once. Training with batch updates accelerates the training process by updating the weights of the model after each batch iteration.
  - Activation function: Activation functions are applied at the output of the weighted sum in a neural network, aiming to transmit the information derived by the combination of weights. The chosen activation functions for this study are presented in Figure 3, where each function serves a specific purpose in the network's architecture and learning process.

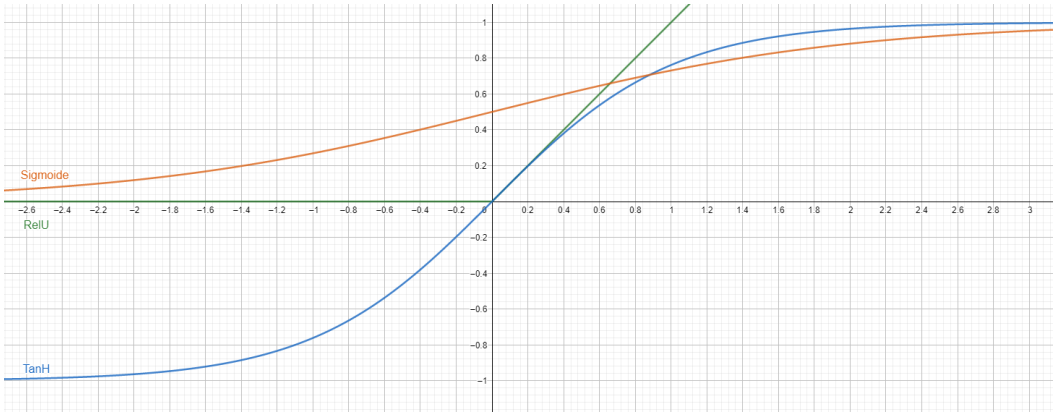


Figure 3. Activation functions.

By following this approach, we aimed to train and analyze the decision tree, random forest, and SVM models, optimizing their parameters and achieving high accuracy in classifying applications based on their potential threat types.

4. Deep Learning Algorithm Design and Model Description

In this section, we present the design of the deep learning algorithm developed as part of this scientific work. Building upon the previously described methodology, we outline the specific steps taken to construct the algorithm and introduce the chosen model for learning. The design and selection of an appropriate machine learning model are crucial for achieving accurate predictions and optimal performance in various applications. In this regard, we provide a detailed description and justification of the employed model, highlighting its suitability for the research objectives. The subsequent sections delve into the algorithm's implementation and evaluation, showcasing its effectiveness in addressing the research problem at hand.

4.1. Algorithm Design

The design of the threat classifier algorithm is based on the static analysis of Android applications, with a primary focus on achieving detection and classification before the installation of the application on the mobile device. Initially, we conducted a search for the most relevant features in the study of the static analysis of applications. Our search primarily relied on previous works that aimed to identify the most informative features in the threat detection phase.

We incorporated various features, including API calls, permissions, and intents, based on the approach proposed by Drebin [1], which employs support vector machines (SVMs) and evaluates detection on the mobile device. Grosse et al. [28] utilized Drebin's feature set in conjunction with neural networks. Additionally, we considered the control of frequently used and requested permissions, as described by Moonsamy et al. [52]. In total, we added 215 additional features to the permission set, encompassing API calls, permissions, and intents. The resulting feature set was organized into different categories:

- **API calls** refer to pre-coded functions used to optimize time by allowing their repeated utilization. These functions are stored in the *.dex* files of the Android Application Package (APK).
- **Permissions** come into effect immediately upon starting the installation of an application, as it is the first step in the process. Permissions play a critical role in system security, as they determine the services to which the application will be granted full access. The set of permissions in Android is defined in the *AndroidManifest.xml* file of the APK.
- **Intents** are used to request actions from the components of other applications. They facilitate the runtime linkage between the code of different applications, particularly

in launching activities. In this context, intents serve as the "glue" between activities hosted in files.

Once the relevant characteristics for application analysis were determined, we proceeded to design a method for feature extraction. To perform the static analysis, we utilized the Androguard tool (https://github.com/androguard/androguard (accessed on 24 July 2023)). The selection of this tool was motivated by its extensive documentation, open source nature, and the feedback obtained during the static analysis process for feature extraction.

4.2. Learning Model Description

The algorithms applied in this project are the **decision tree algorithm, random forest, and SVM**. Additionally, a multilayer perceptron neural network model has been proposed as the classifier. A **deep learning model** was utilized in this approach characterized by its composition of multiple layers. Deep learning models are achieved by iteratively improving the model through multiple iterations. Several parameters were carefully considered in the algorithm design to optimize the training time and achieve optimal results:

- **Iterations and nodes**: Determining the optimal number of iterations is crucial, as each iteration is computationally intensive. Additionally, the number of nodes in each layer influences the computational load and model accuracy. Overfitting and underfitting are two common problems encountered during model training. Underfitting occurs when insufficient iterations fail to reach the minimum cost of the function, while overfitting arises from excessive training with the same dataset.
- **Optimization of nodes**: This step aims to minimize the loss function during the backward prediction phase, enhancing the training process. Optimization algorithms rely on the learning rate, which determines the extent of weight adjustments during each iteration.
- **Learning rate**: The learning rate plays a crucial role in the speed of convergence and overall performance of the optimization algorithm. Selecting an appropriate learning rate is essential to achieve optimal training results and model efficiency.
- **Batch size**: The batch size refers to the number of examples from the dataset used to train the model in each iteration. Employing smaller batch sizes is beneficial when limited memory is available, as it allows loading and processing a single batch at a time. It can also lead to more efficient training by updating the weights after each batch iteration.
- **Activation function**: Activation functions play a vital role in transmitting information derived by the weighted combinations in the neurons. Several activation functions were considered, including Sigmoid [55], Hyperbolic Tangent [56], rectified linear unit (ReLU) [57], and parametric ReLU (PReLU) [58]. PReLU, a derivative of ReLU, was selected for its capability to prevent neuron death and demonstrate enhanced performance for the intermediate nodes.

A more detailed description together with all the code necessary for the reproduction of the experiments performed in this work can be found at the following url (https://github.com/alfonsogomezmartinez, accessed on 24 July 2023).

5. Analysis of Theoretical Models

In this section, we present the evaluation analysis of each of the theoretical models obtained after training with the selected databases as inputs. A subsection is dedicated to each type of attack chosen for our study (phishing, spam, and malware in APKs). Finally, the parameters used for training in each case are detailed.

5.1. Phishing

As mentioned earlier, this work is developed using the [48] database, which is labeled according to [49] for model training. It is important to note that, only under this threat, a comparison of the training results of the same models is provided but with different

databases. One database consists of the complete dataset, while the other is the same database but with a reduced number of input features from the initial 30 to 21. The reason for this is further explained in this section.

With a simple glance at the results shown in Tables 1–6, it is evident that all the trained models exhibit similar indices and curves. For this reason, we opted to select the PRelU model, which consistently provided more promising results in both tables. In Figure 4 the results of a comparative study among theoretical models for phishing with a complete database are depicted. The evolution of the ROC curve is presented on the left side of the figure, while the P-R curve is shown on the right side. In fact, the evolution graphs also indicated that this model had the fastest learning rate, and its ROC and P-R curves occupied the largest areas. In Figure 5 the results of the study for phishing with a reduced database are shown.

Table 1. Evaluation metrics of phishing models using the complete dataset.

	Precision	**Recall**	**Accuracy**	**F1**	**AUC ROC**	**AUC P-R**
Decision Tree	0.962	0.966	96.02%	0.964	0.96	0.973
Random Forest	0.968	0.980	97.18%	0.974	0.971	0.980
SVM	0.943	0.964	94.9%	0.954	0.947	0.964
Sigmoid	0.968	0.982	97.21%	0.975	0.971	0.981
Hyperbolic Tangent	0.967	0.982	97.18%	0.975	0.971	0.979
PRelU	0.971	0.981	97.32%	0.976	0.972	0.981

Table 2. Evaluation data of theoretical models for phishing with reduced database.

	Precision	**Recall**	**Accuracy**	**F1**	**AUC ROC**	**AUC P-R**
Decision Tree	0.9	0.923	90.12%	0.911	0.899	0.933
Random Forest	0.892	0.931	90.09%	0.912	0.897	0.931
SVM	0.868	0.944	89.07%	0.905	0.885	0.922
Sigmoid	0.91	0.914	90.27%	0.912	0.901	0.935
Hyperbolic Tangent	0.918	0.9	89.98%	0.908	0.9	0.936
PRelU	0.889	0.945	90.48%	0.916	0.9	0.932

Table 3. Evaluation metrics of theoretical models for spam SMS classification.

	Precision	**Recall**	**Accuracy**	**F1**	**AUC ROC**	**AUC P-R**
Decision Tree	0.832	0.862	95.76%	0.847	0.918	0.856
Random Forest	0.987	0.835	97.63%	0.905	0.917	0.923
SVM	1.0	0.613	94.76%	0.761	0.807	0.833
Sigmoid	0.994	0.915	98.78%	0.953	0.957	0.961
Hyperbolic Tangent	0.933	0.952	98.42%	0.942	0.971	0.946
PRelU	0.958	0.963	98.92%	0.96	0.978	0.963

Table 4. Evaluation metrics of theoretical models for malware classification.

	Precision	Recall	Accuracy	F1	AUC ROC	AUC P-R
Decision Tree	0.934	0.966	92.89%	0.949	0.906	0.962
Random Forest	0.910	0.972	91.47%	0.940	0.879	0.951
SVM	0.903	0.965	90.52%	0.934	0.867	0.947
Sigmoid	0.923	0.979	92.89%	0.950	0.897	0.958
Hyperbolic Tangent	0.934	0.973	93.36%	0.953	0.909	0.963
PRelU	0.929	0.986	93.84%	0.957	0.909	0.962

Table 5. Metrics to evaluate the theoretical models of malware classification.

	Precision	Recall	Accuracy	F1	AUC ROC	AUC P-R
Benign	0.934	0.877	94.31%	0.905	0.925	0.925
Adware	0.778	0.808	94.79%	0.792	0.888	0.805
Ransomware	0.905	0.826	97.15%	0.864	0.908	0.875
Scareware	0.714	0.8	93.84%	0.755	0.878	0.769
SMS	0.773	0.74	94.79%	0.756	0.856	0.77
Banking	0.942	0.999	98.58%	0.97	0.991	0.971

Table 6. Metric data of the implemented theoretical model.

	Precision	Recall	Accuracy	F1	AUC ROC	AUC P-R
Phishing	0.889	0.945	90.48%	0.916	0.9	0.932
Spam SMS	0.958	0.963	98.92%	0.96	0.978	0.963
APK's Malwares	0.929	0.986	93.84%	0.957	0.909	0.962

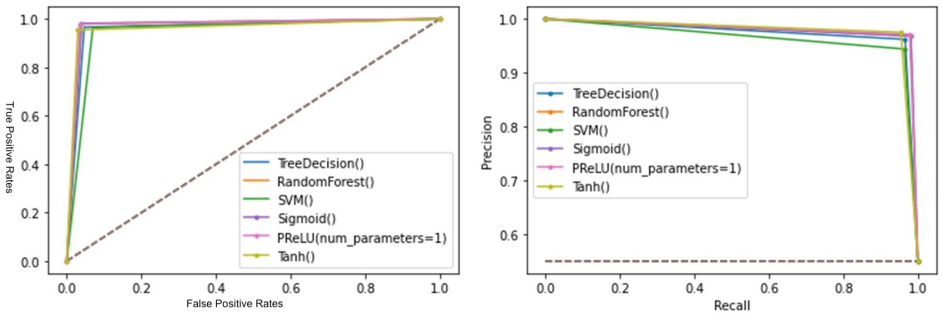


Figure 4. Comparative study of theoretical models for phishing with full database. ROC curve on the left and P-R curve on the right.

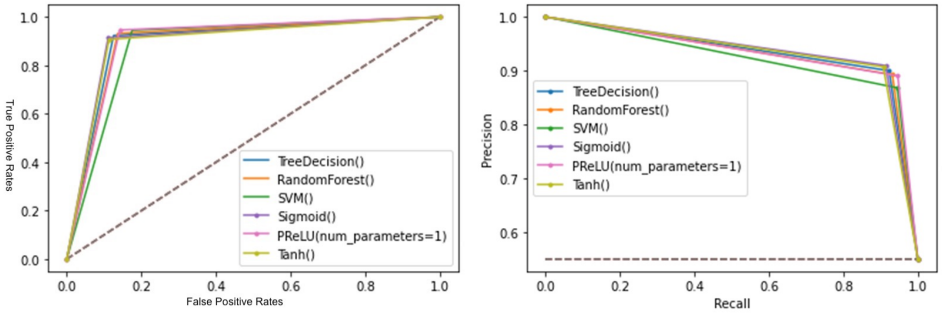


Figure 5. Comparative study of theoretical models for phishing with reduced database. ROC curve on the left and P-R curve on the right.

5.2. SMS Spam

Similar to the previous case, this work is based on the [50] database for training the models and achieving a predictive model capable of identifying SMS spam. The database was labeled using the Tokenizer tool. Please note that the translation provided may not be a perfect representation of the original text, but it captures the essence and meaning of the section.

In this case, we once again choose to use the PRelU model, as it provides better numerical data. The evolution graphs indicate that it also learns the fastest, and its ROC and P-R curves occupy the largest areas shown in Figure 6.

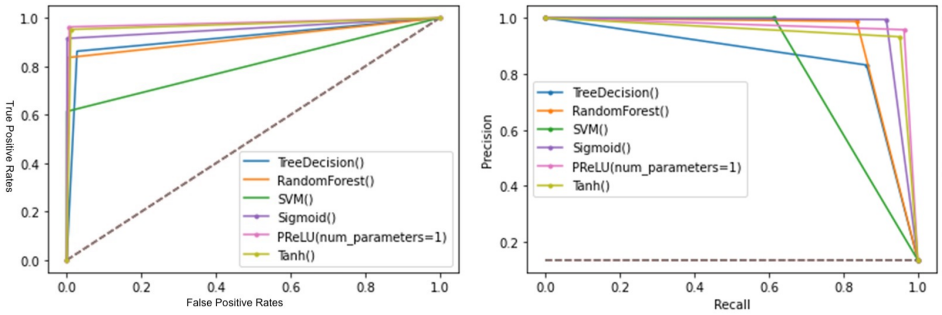


Figure 6. Comparative study of theoretical models for SMS spam classification. ROC curve on the left and P-R curve on the right.

For this study, we utilized a database developed entirely by the authors. To create this database, we started with a study of previous works on malware detection through static analysis [52–54], as well as the collection of malignant and benign APKs provided by [47,51]. Going into further detail, the APKs obtained from [51] were infected with Adware, SMS, Scareware, and Ransomware malware. The database from [47] provided us with applications infected with Adware and Banking malware. Due to the smaller size of the [51] dataset compared to [47], we reduced the number of samples to achieve a balanced dataset. The trained model aims to determine whether an application is malicious and, if so, classify it according to the type of threat it carries. Therefore, we consider the ability to classify applications as malignant or benign, as well as the type of threat, as essential parameters.

After this initial analysis shown in Figure 7, the results continue to confirm that the best option is to work with the PRelU model, as its data remain superior compared to the other alternatives. Taking this into account, we aim to examine the performance that the model provides in classifying different types of malware.

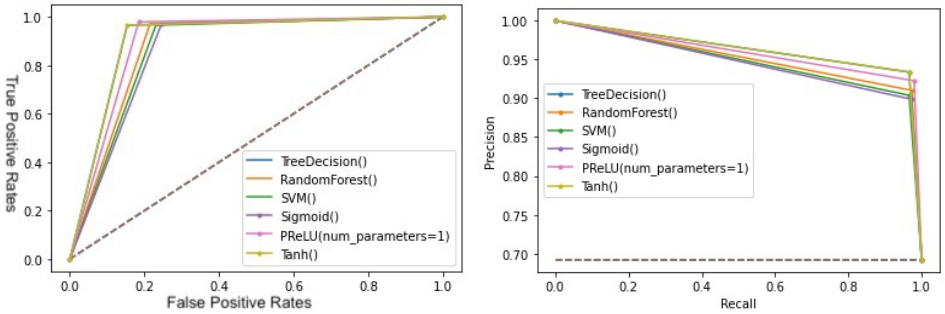


Figure 7. Comparative study of theoretical models for malware classification. ROC curve on the left and P-R curve on the right.

In this final study, we present the types of malware that our model can predict with greater effectiveness. We can conclude that threats such as Banking or Ransomware are detected with the highest accuracy. However, threats like SMS, Adware, and Scareware are identified with a lower efficacy rate. If we examine the confusion matrix, we can observe that Banking and Ransomware malware are rarely or almost never mistaken for other threats. On the other hand, SMS, Adware, and Scareware threats exhibit a higher rate of confusion than other types of threats. Another noteworthy observation from the matrix is that our algorithm struggles the most in detecting Adware malware. In summary, we outline the performance obtained for the selected models:

- Phishing: The PRelU model demonstrated promising results, showing fast learning and occupying a larger area in the ROC and P-R curves.
- Spam SMS: Once again, the PRelU model yielded better numerical data, demonstrated faster learning in the evolution graphs, and occupied a larger area in the ROC and P-R curves.
- Malware in APKs: The PRelU model continued to outperform other options, showing superior results in terms of classifying malignancy and different types of malware. These findings highlight the effectiveness of the PRelU model across all three types of threats studied.

As expected, the PRelU model was chosen for all three threats, despite initially showing lower effectiveness during training. However, it ultimately yielded the best results in terms of training performance, with curves that displayed superior outcomes. Nevertheless, due to the low ratios achieved for classifying malware types, this option was excluded from the proof-of-concept stage.

6. Malware Classifier

This section describes the fundamental steps for the elaboration of the final model of the malware classifier. For this purpose, different models and metrics were considered in order to characterize the model.

6.1. Dataset Preparation

For the dataset, a database initially consisting of a total of 18,188 *Android APK* samples was used. These *APKs* are distributed as shown in Table 7. Our dataset is divided into five distinct categories: *Adware, Banking malware, SMS malware, Riskware*, and Benign.

Table 7. Dataset samples by class.

Class	Benign	Adware	Riskware	SMS	Banking	Total
Samples	4251	2356	3865	4901	2815	18188

As described in Section 4.1, static analysis was employed to extract 215 input features from each application, resulting in a database consisting of 215 input labels, 5 potential

output states, and 18,188 samples. However, certain applications posed challenges during the static analysis, leading to their exclusion from the dataset. Consequently, the dataset was reduced to 16,950 samples.

6.2. Model Evaluation Method

In their work, Arp et al. [59] highlight several points that can affect experimental results and conclusions in the context of popular applications of deep learning in security. They propose estimating the experimental impact of specific pitfalls and provide recommendations to help identify and address these issues. This article focuses on the proposal of mobile malware detection, and three essential pitfalls are discussed.

Proper sampling bias control is crucial to ensure that the collected data sufficiently represent the actual distribution of the underlying security problem [60,61]. Understanding the real underlying distribution of the input space is essential, although certain biases may be inevitable in certain situations. However, a thorough understanding of these biases is necessary to limit their impact in practice. The sampling bias has a significant impact on security because data acquisition is a complex task that often relies on multiple different sources. In the case of data collection for Android malware detection discussed in this article, there is a limited number of reliable public sources [62,63], leading to the use of synthetic data for training deep learning models. This increases the possibility of introducing biases. As biases are often unavoidable, mitigation strategies such as generating pseudo-reliable synthetic data [64] and employing transfer learning [65,66] are recommended.

Introducing unrelated artifacts into the system can lead to the creation of spurious correlations, which are new classifications derived from the model's adaptation to these artifacts rather than focusing on solving the security problem. A consequence of these correlations is that they can go unnoticed during result interpretation, frequently leading to inflated estimations of a model's capabilities and erroneous evaluations of its practical constraints. Techniques exist to mitigate the potential impact of this pitfall. The use of explanation techniques for learning [67,68], while not infallible and presenting certain limitations [69,70], can reveal spurious correlations and provide a more accurate assessment of their impact on system capabilities.

In the case of evaluating a detection system, it is essential to select appropriate performance measures. Merely reporting a single performance value, such as precision, is insufficient. Since true positive and false positive decisions are unnoticed, the use of more advanced measures like ROC curves (which we employ) can conceal experimental results in certain application environments. Solely providing the ROC curve can be misleading, as the performance may appear excellent, while the actual precision and recall of the classifier's true ROC curve are low and thus impractical for many security applications.

To ensure the reliable experimental evaluation of mobile malware detection systems, researchers need to address several key pitfalls. The proper control of sampling bias, the mitigation of spurious correlations, and the careful selection of appropriate performance measures are crucial for making accurate assessments and drawing out the practical implications of these systems. By following the provided recommendations, researchers can improve the quality and reliability of their experimental evaluations in the field of deep learning-based security applications.

Once the possible problems have been analyzed, we move on to the description of the different metrics used for the analysis of the numerical results.

- **Confusion matrix**. This metric allows us to measure the number of true negatives (TNs), true positives (TPs), false negatives (FNs), and false positives (FPs) of our trained model with respect to the test data. This graph will be useful during the study, because it will show us the number of applications that are correctly predicted; and for the applications that are not correctly predicted, it shows us the threat with which they are confused.

- **Precision**: it is responsible for measuring the quality of the models. It is calculated with Equation (1).

$$\frac{TP}{TP + FP} \tag{1}$$

- **Recall**: it represents the amount of relevant data that the model has been able to obtain over the total relevant data. It is calculated with Equation (2).

$$\frac{TP}{TP + FN} \tag{2}$$

- **Fx**: this metric allows us to relate precision and recall, based on Equation (3).

$$(1 + x^2) \cdot \frac{precision \cdot recall}{(x^2 \cdot precision) + recall} \tag{3}$$

  In order to relate it in an equitable way, F1 was used.
- **Accuracy**: this is a metric that allows us to see the percentage of test data that the model correctly predicted out of the total test data. It is calculated by Equation (4).

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

In addition to these metrics, graphs are also used to see how parameters such as precision and function cost evolve with each iteration. Two other graphical resources that we are going to use are:

- **Precision–recall curve**: this curve plots the values of precision versus recall. It is used by analyzing the area under the curve, where the larger the area, the better the model. This plot becomes more important when we encounter models that have been trained with unbalanced datasets.
- **ROC curve**: this curve plots the values of FP versus TP. It is used by analyzing the area under the curve, where the larger the area, the better the model.

6.3. Design

For the design of the proposed solution, various possibilities were meticulously examined to achieve an optimal design for the derived dataset. A comparative analysis was conducted, where different proposals were trained using the same input dataset and evaluated using consistent evaluation methods. Through this rigorous procedure, it was determined that the decision tree, random forest, and SVM models and the convolutional networks, as discussed in Section 3, outperformed the other methods considered.

In the implementation of neural networks, extensive exploration was carried out to study the effects of different activation functions, hyperparameters, and configurations. To optimize node weights, the ADAM method [71] (adaptive moment estimation) was employed. This dynamic optimization approach facilitated the continuous improvement of the learning rate, thereby enhancing node weights with each iteration.

As mentioned in Section 3, the *softmax* activation function was chosen for the output layer. This decision was based on the function's efficacy in handling linearly separable data, which is relevant to the five-category classification task in this study.

The results of the analyzed models are presented in Table 8. Precision, recall, accuracy, F1 score, area under the ROC curve (AUC ROC), and area under the precision–recall curve (AUC P-R) were used as evaluation metrics. The table demonstrates that the deep learning algorithms consistently outperformed the deep learning algorithms. Comparing the decision tree, random forest, and support vector machine (SVM) classifier against three activation functions (*Sigmoid, PRelU, and TanH*) employed in deep learning, it is evident that the deep learning models yielded superior numerical results across all evaluation metrics. Furthermore, among the tested activation functions, *PRelU* demonstrated the

most favorable performance, showcasing the highest numerical results across all evaluation metrics. Consequently, *PRelU* was chosen as the activation function for the final model implementation.

Table 8. Results of the analyzed models.

	Precision	Recall	Accuracy	F1	AUC ROC	AUC P-R
Decision Tree Classifier	0.979	0.979	97.92%	0.979	0.97	0.991
Random Forest Classifier	0.919	0.972	97.63%	0.906	0.917	0.923
SVM Classifier	0.903	0.944	94.9%	0.954	0.947	0.964
Sigmoid	0.987	0.994	98.58%	0.991	0.978	0.993
PRelU	0.993	0.994	99.01%	0.993	0.986	0.996
TanH	0.99	0.99	98.47%	0.99	0.979	0.994

The results presented in Table 8 lead to two key conclusions. Firstly, deep learning algorithms consistently outperformed deep learning algorithms, as evidenced by their superior numerical results across all evaluation metrics. Secondly, the *PRelU* activation function consistently outperformed other activation functions, displaying the best numerical results across all evaluation metrics. Hence, it was selected as the activation function for the final model implementation.

6.4. Neural Network Training Parameters

This section defines the parameters used during training to optimize the models and provides details of the datasets used to facilitate experiment replication.

To ensure consistent parameter values in random processes applied to certain functions, a seed (random.seed(0)) and a random state of 32 were used. The number of hidden layers and intermediate nodes is the same for all models, with 2 layers and 128 nodes in each layer.

6.5. Results

A summary of the results obtained with the implemented model, which takes as input the dataset that was developed, can be found in Tables 9–11. The results obtained after training the model are included in Table 10. We note that the model is only able to classify whether the *APK* is malignant or benign. It is shown with a train/test split of 75/25. Table 11, on the other hand, is a compilation of the model data. It is able to categorize the malware into the five aggregated families, with a precision rate of 95.19%.

Table 9. Neural network training parameters.

	Iteractions	Lr	Batch
Phishing	200	0.0035 0.006 only for Sigmoid	150
Spam SMS	50	0.02	1500
APK Malware	75	0.02	100

Table 10. Results from the trained model without classification.

	Precision	Recall	Accuracy	F1	AUC ROC	AUC P-R
Benign vs. Malware	0.994	0.993	99.01%	0.989	0.986	0.996

Table 11. Results from the trained classification model.

	Precision	Recall	Accuracy	F1	AUC ROC	AUC P-R
Benign	0.982	0.972	98.89%	0.977	0.983	0.981
Adware	0.915	0.904	98.28%	0.91	0.948	0.914
Riskware	0.93	0.956	97.31%	0.943	0.967	0.948
SMS	0.977	0.974	98.56%	0.975	0.982	0.979
Banking	0.908	0.894	97.33%	0.900	0.94	0.908

Taking a look at the results shown in both Tables 10 and 11, the high performance of our model in the prediction of threats of the SMS malware class is reflected. On the other hand, it is also clear that it is more expensive to predict the threats of the banking class.

We realize that another feature of our model is that it infrequently confuses an application of the malware class with a benign one. This is clear from the data in the first column of Figure 8. This fact is a justification of why such encouraging results were achieved in distinguishing between one of the possible threats and a benign one. We emphasize that our main priority was to achieve improvements in this classification. The fact that we consider it essential that the minimum of malicious applications could be mistaken as a benign one and be classified as benign in a wrong way, this implies a vulnerability in the system.

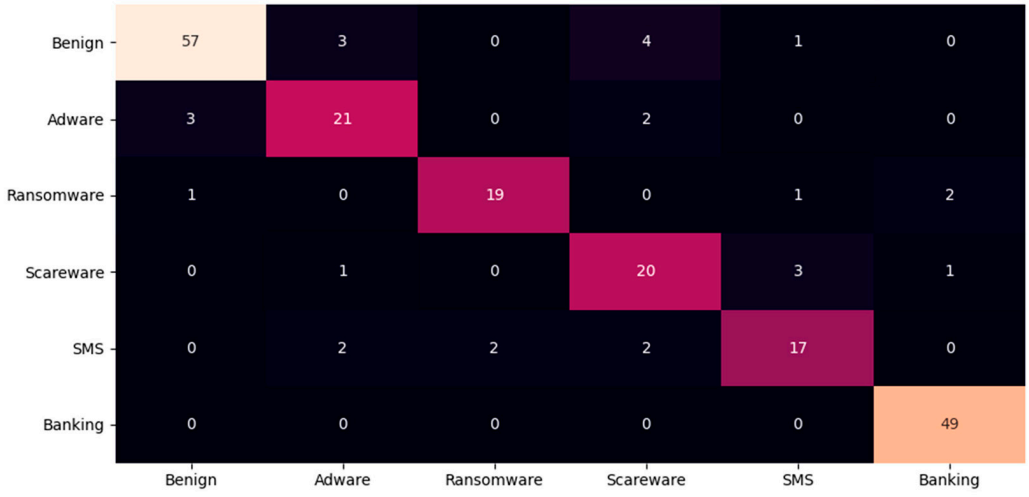
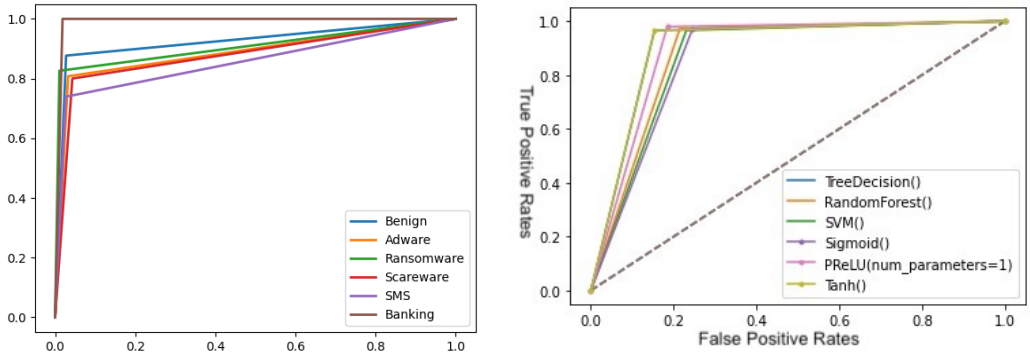


Figure 8. Confusion matrix: Derived from the outcomes of theoretical models, the success rates of malware type classification were computed. Reality on the vertical axis vs. prediction on the horizontal axis.

Figure 9 shows plots that support the results discussed above. In them, it can be seen that there is a great closeness of the joint to the upper corners. The interpretation of these plots is that the closer they are to their corresponding corner, the closer they are to the idealization of the model. Another result that confirms this fact is that the benign predictions are the ones that offer less false positives.



(**a**) (**b**)

Figure 9. (**a**) ROC curve: results of the theoretical models according to malware type classification success rates; (**b**) P-R curve: results of the theoretical models according to malware type classification success rates.

6.6. Malware Classifier Tool: A Proof of Concept

This section presents a proof of concept of the application of the proposed model in an *Android* app, which is named the Malware Classifier application source code and can be found at url (https://github.com/alfonsogomezmartinez, accessed on 24 July 2023). It has been implemented as an Android application that uses the malware classifier model proposed using deep learning techniques. Malware Classifier tool allows the user of the Android device to analyze applications prior to the start of its installation, which is why we classify it as static. It performs the analysis of the applications and identifies whether it as malware. If so, it will classify it among the possible categories considered in this study.

The APK file is analyzed and identified as malware: in such a case, the type of Malware (*Riskware, Adware, SMS* and *Banking*) is classified. A menu is shown to choose the *APK* file to classify.

A common use case is the analysis of a particular *APK* file. The first step is that the classifier prompts the user to grant permissions to access the files in the user's space. Once the user grants access to the app to access the files, the list of possible files stored for analysis is displayed. The user selects the *APK* file to analyze by clicking analyze. From this point on, the *APK* file is packaged in a *JSON (JavaScript Object Notation* and sent to the trained model that is loaded in the cloud. As shown in Figure 10, the model is loaded in *Flask* (https://flask.palletsprojects.com/en/2.1.x/#, accessed on 24 July 2023) (a local *Python* server) and this is located inside an *Ngrok* (accessed on 24 July 2023) (https://ngrok.com/ that provides a bridge to the Internet. Retrofit is also used to consume the API that is responsible for the analysis of *APK* files and process the response. Once the *JSON* is loaded in the cloud, we proceed to perform a static analysis that allows us to obtain the set of features. These will be used to perform the classification using the trained model. This will provide the classification result that will be sent to the classifier, which displays the result in its *GUI*, as shown in Figure 10.

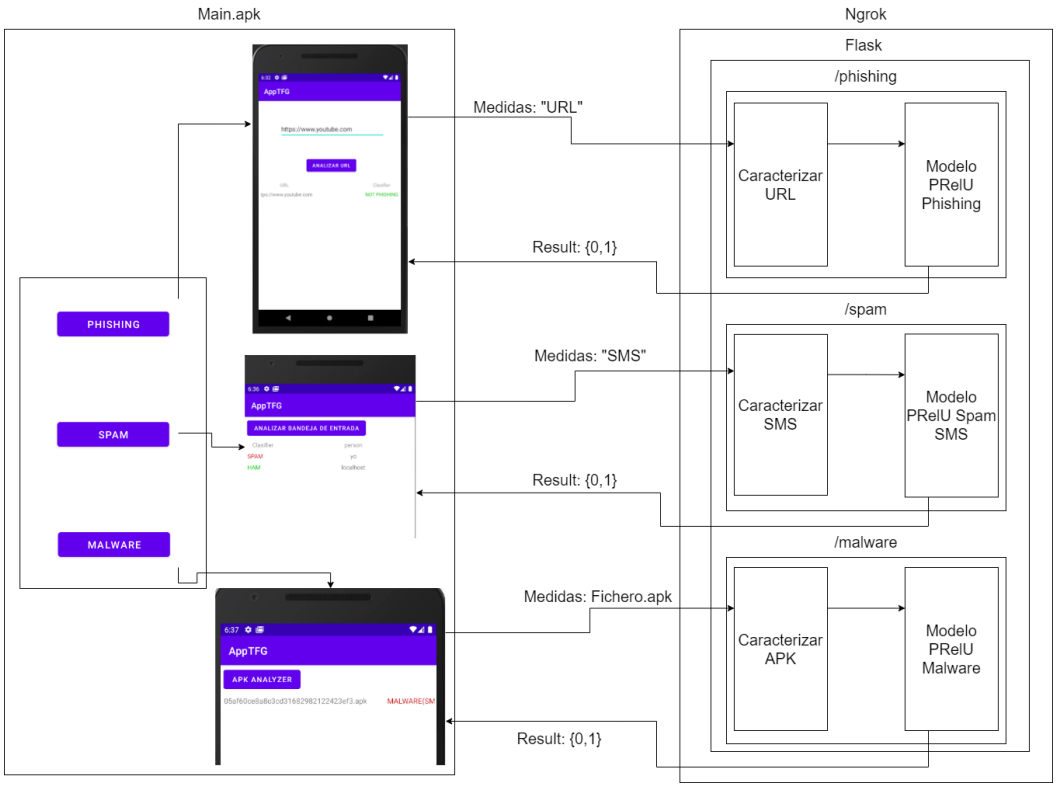


Figure 10. Malware Classifier tool workflow.

7. Discussion

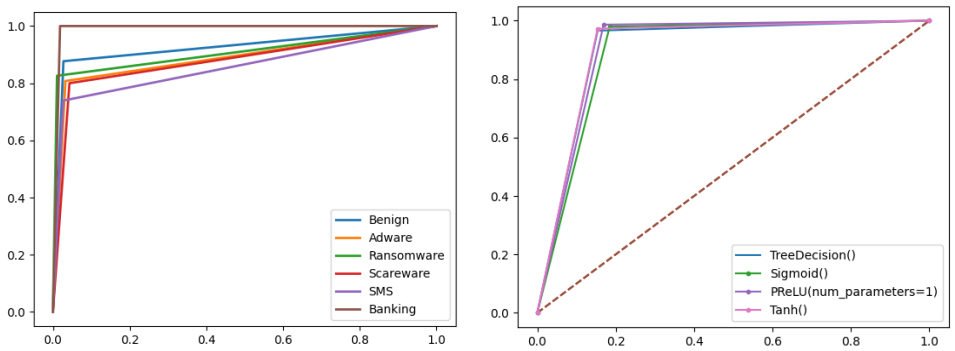
This section includes a discussion of the results obtained. A comparison is made with those obtained in other studies with similar objectives, but with studies using different models or parameters.

In Sandeep's study [72], a strategy similar to the one proposed in our article is presented. Sandeep et al. proposed a neural network-based system capable of performing the static classification of Android applications, i.e., classifying them prior to installation. In comparison to their work, our proposed approach achieves a precision rate of 95.19%, whereas Sandeep's system achieves a precision rate of 94.65%. This higher precision rate is particularly relevant considering that our proposal utilizes 215 input features, while Sandeep's system relies on 331 input features. One possible explanation for this difference is that our study of static features is more diverse, incorporating factors such as permissions, API calls, and intensities. In contrast, their analysis is solely based on permissions. This narrow focus on permissions may limit the versatility of their system, which could explain the slightly lower precision rate they achieved.

Other studies have used some of the datasets used to train our model. For example, Lashkari [73] in 2018 used *CICAndMal2017* to implement a deep learning model capable of classifying an *Android APK*, with a success rate of 85%. Their model is based on a decision classifier tree, which, as demonstrated in this study (Table 8), gives worse results than our proposal. It is also relevant that it makes use of a feature labeling valued at 80, offering a lower rate than our system.

Finally, the studies developed by Mahdavifar [46,47] in 2020 are discussed in depth. Despite the different strategies in decision making, already described in Section 2, both proposals present similar results. However, there are some differences. Comparing the results provided by our model (ROC curve Figure 11) against the results obtained by them, we observe that our model obtains better threat prediction results in general. However, for the cases of Adware and banking, the prediction is slightly lower, and better results are obtained for the rest of the threats. It can be considered that the proposal presented in this work is a significant improvement over their proposal. Better results are obtained for the

identification of benign and non-benign applications, and this was the main objective of our work.



(**a**) (**b**)

Figure 11. (**a**) ROC curve: results of the theoretical models according to malware type classification success rates; (**b**) P-R curve: results of the theoretical models according to malware type classification success rates.

8. Conclusions and Future Works

The widespread use of Android devices in today's society makes them the target of many attackers. Many cybercriminals exploit the use of *Adware, Banking, SMS* or *Riskware* malware for their purposes. For this reason, it is necessary to use some kind of malware detection mechanism to eliminate, reduce, or mitigate the possible damage to the targeted system. This paper presents a classification model that makes use of deep learning techniques. The model is able to identify whether an *Android APK* is malware. Moreover, the model is able to classify an *Android APK* into five different categories. The model has been trained using a dataset of 16,950 *Android APK* samples. These have been statically analyzed, labeled with 215 entries, and classified into five categories. Additionally, an application based on the proposed model has been developed as a proof of concept. We believe that our model can be a very useful tool as an additional element, as it provides versatility to the current analysis methods. The proposed model achieves a method capable of updating itself to the changes that the threats under study integrate to go undetected during the detection processes. This update is performed autonomously during training.

During the implementation of the present work, we found several works in progress with different data sources (*Androware, Virus Total* and other data sources). For this reason, we propose to integrate these data sources in future experiments. This was performed in order to train our model with a larger volume of input data and perform studies on the results obtained with a larger sample. Also, in our work, we used third-party data sources, although we are working on the idea of building our own data source. As mentioned throughout the paper, this work was based on static analysis, prior to the installation of the software to be analyzed. We also propose as future work that the computational complexity of the training procedures of the classifiers studied herein be further researched. Finally, we propose to explore the possibility of conducting a study to perform a dynamic analysis to analyze applications that have passed our static analysis.

Author Contributions: Methodology, A.M.; Software, A.G.; Validation, A.G.; Formal analysis, A.M.; Investigation, A.G. and A.M.; Writing—original draft, A.M.; Writing—review & editing, A.M. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

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

References

1. Arp, D.; Spreitzenbarth, M.; Hubner, M.; Gascon, H.; Rieck, K.; Siemens, C.E.R.T. Drebin: Effective and Explainable Detection of Android Malware in Your Pocket. In Proceedings of the 21st Annual Network and Distributed System Security Symposium, NDSS, San Diego, CA, USA, 23–26 February 2014; Volume 14, pp. 23–26.
2. Liu, K.; Xu, S.; Xu, G.; Zhang, M.; Sun, D.; Liu, H. A review of android malware detection approaches based on machine learning. *IEEE Access* **2020**, *8*, 124579–124607. [CrossRef]
3. Qiu, J.; Zhang, J.; Luo, W.; Pan, L.; Nepal, S.; Xiang, Y. A survey of android malware detection with deep neural models. *ACM Comput. Surv. (CSUR)* **2020**, *53*, 1–36. [CrossRef]
4. Zhang, M.; Duan, Y.; Yin, H.; Zhao, Z. Semantics-aware android malware classification using weighted contextual api dependency graphs. In Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, Scottsdale, AZ, USA, 3–7 November 2014; pp. 1105–1116.
5. Dhalaria, M.G.; Otra, E. Risk Detection of Android Applications Using Static Permissions. In *Advances in Data Computing, Communication and Security*; Springer: Singapore, 2022; pp. 591–600.
6. Lakshmanan, R. New Android Malware Uses VNC to Spy and Steal Passwords from Victims. 2021. Available online: https://thehackernews.com/2021/07/new-android-malware-uses-vnc-to-spy-and.html (accessed on 10 May 2022).
7. Gao, H.; Xiao, J.; Yin, Y.; Liu, T.; Shi, J. A mutually supervised graph attention network for few-shot segmentation: The perspective of fully utilizing limited samples. *IEEE Trans. Neural Netw. Learn. Syst.* **2022**, 1–13. [CrossRef] [PubMed]
8. Jiang, H.; Turki, T.; Wang, J.T. DLGraph: Malware detection using deep learning and graph embedding. In Proceedings of the 2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA), Orlando, FL, USA, 17–20 December 2018; IEEE: Manhattan, NY, USA, 2018; pp. 1029–1033.
9. Ahmad, Z.; Shahid, K.A.; Wai, S.C.; Abdullah, J.; Ahmad, F. Network intrusion detection system: A systematic study of machine learning and deep learning approaches. *Trans. Emerg. Telecommun. Technol.* **2021**, *32*, e4150. [CrossRef]
10. Malware Statistics and Facts for 2022. 2022. Available online: https://www.comparitech.com/antivirus/malware-statistics-facts/ (accessed on 8 January 2022).
11. Eder, J.; Shekhovtsov, V.A. Data quality for federated medical data lakes. *Int. J. Web Inf. Syst.* **2021**, *17*, 407–426. [CrossRef]
12. Gao, H.; Qiu, B.; Barroso, R.J.D.; Hussain, W.; Xu, Y.; Wang, X. Tsmae: A novel anomaly detection approach for internet of things time series data using memory-augmented autoencoder. *IEEE Trans. Netw. Sci. Eng.* **2022**, 1–11. [CrossRef]
13. Jakobsson, M.; Ramzan, Z. *Crimeware: Understanding New Attacks and Defenses*; Addison-Wesley Professional: Boston, MA, USA, 2008.
14. Kimani, K.; Oduol, V.; Langat, K. Cyber security challenges for IoT-based smart grid networks. *Int. J. Crit. Infrastruct. Prot.* **2019**, *25*, 36–49. [CrossRef]
15. Tariq, N. Impact of cyberattacks on financial institutions. *J. Internet Bank. Commer.* **2018**, *23*, 1–11.
16. Wong, W.; Stamp, M. Hunting for metamorphic engines. *J. Comput. Virol.* **2006**, *2*, 211–229. [CrossRef]
17. Bazrafshan, Z.; Hashemi, H.; Fard, S.M.H.; Hamzeh, A. A survey on heuristic malware detection techniques. In Proceedings of the 5th Conference on Information and Knowledge Technology, Shiraz, Iran, 28–30 May 2013; IEEE: Manhattan, NY, USA, 2013; pp. 113–120.
18. Christodorescu, M.; Jha, S. Static analysis of executables to detect malicious patterns. In Proceedings of the 12th USENIX Security Symposium (USENIX Security 03), Washington, DC, USA, 4–8 August 2003.
19. Schultz, M.G.; Eskin, E.; Zadok, F.; Stolfo, S.J. Data mining methods for detection of new malicious executables. In Proceedings of the 2001 IEEE Symposium on Security and Privacy, S&P, Oakland, CA, USA, 14–16 May 2001; IEEE: Manhattan, NY, USA, 2011; pp. 38–49.
20. Shabtai, A.; Moskovitch, R.; Elovici, Y.; Glezer, C. Detection of malicious code by applying machine learning classifiers on static features: A state-of-the-art survey. *Inf. Secur. Tech. Rep.* **2009**, *14*, 16–29. [CrossRef]
21. Dang, Q.V. Improving the performance of the intrusion detection systems by the machine learning explainability. *Int. J. Web Inf. Syst.* **2021**, *17*, 537–555. [CrossRef]
22. Saxe, J.; Berlin, K. Deep neural network based malware detection using two dimensional binary program features. In Proceedings of the 2015 10th International Conference on Malicious and Unwanted Software (MALWARE), Fajardo, PR, USA, 20–22 October 2015; IEEE: Manhattan, NY, USA, 2015; pp. 11–20.
23. Ucci, D.; Aniello, L.; Baldoni, R. Survey of machine learning techniques for malware analysis. *Comput. Secur.* **2019**, *81*, 123–147. [CrossRef]
24. Abusitta, A.; Li, M.Q.; Fung, B.C. Malware classification and composition analysis: A survey of recent developments. *J. Inf. Secur. Appl.* **2021**, *59*, 102828. [CrossRef]
25. Dahl, G.E.; Stokes, J.W.; Deng, L.; Yu, D. Large-scale malware classification using random projections and neural networks. In Proceedings of the 2013 IEEE International Conference on Acoustics, Speech and Signal Processing, Vancouver, BC, Cananda, 26–31 May 2013; IEEE: Manhattan, NY, USA, 2013; pp. 3422–3426.
26. Huang, W.; Stokes, J.W. MtNet: A multi-task neural network for dynamic malware classification. In Proceedings of the International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment, San Sebastián, Spain, 7–8 July 2016; Springer: Cham, Switzerland, 2016; pp. 399–418.

27. Kolosnjaji, B.; Zarras, A.; Webster, G.; Eckert, C. Deep learning for classification of malware system call sequences. In *Australasian Joint Conference on Artificial Intelligence*; Springer: Cham, Switzerland, 2016; pp. 137–149.
28. Grosse, K.; Papernot, N.; Manoharan, P.; Backes, M.; McDaniel, P. Adversarial examples for malware detection. In *European Symposium on Research in Computer Security*; Springer: Cham, Switzerland, 2017; pp. 62–79.
29. Suciu, O.; Coull, S.E.; Johns, J. Exploring adversarial examples in malware detection. In Proceedings of the 2019 IEEE Security and Privacy Workshops (SPW), Francisco, CA, USA, 19–23 May 2019; IEEE: Manhattan, NY, USA, 2019; pp. 8–14.
30. Wang, Q.; Guo, W.; Zhang, K.; Ororbia, A.G.; Xing, X.; Liu, X.; Giles, C.L. Adversary resistant deep neural networks with an application to malware detection. In Proceedings of the 23rd ACM Sigkdd International Conference on Knowledge Discovery and Data Mining, Halifax, NS, USA, 13–17 August 2017; pp. 1145–1153.
31. Chen, L.; Ye, Y.; Bourlai, T. Adversarial machine learning in malware detection: Arms race between evasion attack and defense. In Proceedings of the 2017 European intelligence and Security Informatics Conference (EISIC), Athens, Greece, 11–13 September 2017; IEEE: Manhattan, NY, USA, 2017; pp. 99–106.
32. Jang, J.; Brumley, D.; Venkataraman, S. Bitshred: Feature hashing malware for scalable triage and semantic analysis. In Proceedings of the 18th ACM Conference on Computer and Communications Security, New York, NY, USA, 17–21 October 2011; pp. 309–320.
33. Mishra, P.; Varadharajan, V.; Tupakula, U.; Pilli, E.S. A detailed investigation and analysis of using machine learning techniques for intrusion detection. *IEEE Commun. Surv. Tutor.* **2018**, *21*, 686–728. [CrossRef]
34. Santos, R.; Souza, D.; Santo, W.; Ribeiro, A.; Moreno, E. Machine learning algorithms to detect DDoS attacks in SDN. *Concurr. Comput. Pract. Exp.* **2020**, *32*, e5402. [CrossRef]
35. Upchurch, J.; Zhou, X. Variant: A malware similarity testing framework. In Proceedings of the 2015 10th International Conference on Malicious and Unwanted Software (MALWARE), Fajardo, PR, USA, 20–22 October 2015; IEEE: Manhattan, NY, USA, 2015; pp. 31–39.
36. Ahmadi, M.; Ulyanov, D.; Semenov, S.; Trofimov, M.; Giacinto, G. Novel feature extraction, selection and fusion for effective malware family classification. In Proceedings of the Sixth ACM Conference on Data and Application Security and Privacy, New York, NY, USA, 9–11 March 2016; pp. 183–194.
37. Alzaylaee, M.K.; Yerima, S.Y.; Sezer, S. DL-Droid: Deep learning based android malware detection using real devices. *Comput. Secur.* **2020**, *89*, 101663. [CrossRef]
38. Elsayed, M.S.; Le-Khac, N.A.; Dev, S.; Jurcut, A.D. Ddosnet: A deep-learning model for detecting network attacks. In Proceedings of the 2020 IEEE 21st International Symposium on "A World of Wireless, Mobile and Multimedia Networks" (WoWMoM), Cork, Ireland, 31 August–3 September 2020; IEEE: Manhattan, NY, USA, 2020; pp. 391–396.
39. Polino, M.; Scorti, A.; Maggi, F.; Zanero, S. Jackdaw: Towards automatic reverse engineering of large datasets of binaries. In Proceedings of the International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment, Milan, Italy, 9–10 July 2015; Springer, Cham, Switzerland, 2015; pp. 121–143.
40. Farajzadeh-Zanjani, M.; Hallaji, E.; Razavi-Far, R.; Saif, M.; Parvania, M. Adversarial semi-supervised learning for diagnosing faults and attacks in power grids. *IEEE Trans. Smart Grid* **2021**, *12*, 3468–3478. [CrossRef]
41. Tamersoy, A.; Roundy, K.; Chau, D.H. Guilt by association: Large scale malware detection by mining file-relation graphs. In Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, New York, NY, USA, 24–27 August 2014; pp. 1524–1533.
42. Hsieh, W.C.; Wu, C.C.; Kao, Y.W. A study of android malware detection technology evolution. In Proceedings of the 2015 International Carnahan Conference on Security Technology (ICCST), Taipei, Taiwan, 21–24 September 2015; IEEE: Manhattan, NY, USA, 2015; pp. 135–140.
43. Muttoo, S.K.; Badhani, S. Android malware detection: State of the art. *Int. J. Inf. Technol.* **2017**, *9*, 111–117. [CrossRef]
44. Wang, X.; Yang, Y.; Zeng, Y. Accurate mobile malware detection and classification in the cloud. *SpringerPlus* **2015**, *4*, 583. [CrossRef]
45. Richter, L. Common weaknesses of android malware analysis frameworks. In *IT Security Conference*; University of Erlangen-Nuremberg during Summer Term: Erlangen, Germany, 2015; pp. 1–10.
46. Mahdavifar, S.; Kadir, A.F.A.; Fatemi, R.; Alhadidi, D.; Ghorbani, A.A. Dynamic Android Malware Category Classification using Semi-Supervised Deep Learning. In Proceedings of the (DASC/PiCom/CBDCom/CyberSciTech), Calgary, AB, Canada, 17–22 August 2020; pp. 515–522.
47. Mahdavifar, S.; Alhadidi, D.; Ghorbani, A.A. Effective and Efficient Hybrid Android Malware Classification Using Pseudo-Label Stacked Auto-Encoder. *J. Netw. Syst. Manag.* **2022**, *30*, 22. [CrossRef]
48. Dua, D.; Graff, C. *UCI Machine Learning Repository*; University of California, Irvine, School of Information and Computer Sciences: Irvine, CA, USA, 2017. Available online: http://archive.ics.uci.edu/ml (accessed on 24 July 2023).
49. Mohammad, R.M.; Thabtah, F.; McCluskey, L. Intelligent rule-based phishing websites classification. *Iet Inf. Secur.* **2014**, *8*, 153–160. [CrossRef]
50. Rho Lall. SMS Spam Collection. 2018. Available online: https://www.kaggle.com/assumewisely/sms-spam-collection (accessed on 24 July 2023).
51. Taheri, L.; Kadir, A.F.A.; Lashkari, A.H. Extensible android malware detection and family classification using network-flows and API-calls. In Proceedings of the 2019 International Carnahan Conference on Security Technology (ICCST), Chennai, India, 1–3 October 2019; IEEE: Manhattan, NY, USA, 2019; pp. 1–8.

52. Moonsamy, V.; Rong, J.; Liu, S. Mining permission patterns for contrasting clean and malicious android applications. *Future Gener. Comput. Syst.* **2014**, *36*, 122–132. [CrossRef]
53. Sharma, A.; Dash, S.K. Mining api calls and permissions for android malware detection. In *Cryptology and Network Security, Proceedings of the 13th International Conference, CANS 2014, Heraklion, Crete, Greece, 22–24 October 2014*; Proceedings 13; Springer International Publishing: Cham, Switzerland, 2014; pp. 191–205.
54. Yerima, S.Y.; Sezer, S. Droidfusion: A novel multilevel classifier fusion approach for android malware detection. *IEEE Trans. Cybern.* **2018**, *49*, 453–466. [CrossRef] [PubMed]
55. Han, J.; Moraga, C. The influence of the sigmoid function parameters on the speed of backpropagation learning. In *International Workshop on Artificial Neural Networks*; Springer: Berlin/Heidelberg, Germany, 1995; pp. 195–201.
56. Anastassiou, G.A. Univariate hyperbolic tangent neural network approximation. *Math. Comput. Model.* **2011**, *53*, 1111–1132. [CrossRef]
57. Bracewell, R.N.; Bracewell, R.N. *The Fourier Transform and Its Applications*; McGraw-Hill: New York, NY, USA, 1986; Volume 31999, p. 1986.
58. He, K.; Zhang, X.; Ren, S.; Sun, J. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In Proceedings of the IEEE International Conference on Computer Vision, Santiago, Chile, 7–13 December 2015; pp. 1026–1034.
59. Arp, D.; Quiring, E.; Pendlebury, F.; Warnecke, A.; Pierazzi, F.; Wressnegger, C.; Rieck, K. Dos and do nots of machine learning in computer security. In Proceedings of the 31st USENIX Security Symposium (USENIX Security 22), Boston, MA, USA, 10–12 August 2022; pp. 3971–3988.
60. Chio, C.; Freeman, D. *Machine Learning and Security: Protecting Systems with Data and Algorithms*; O'Reilly Media, Inc.: Newton, MA, USA, 2018.
61. Cortes, C.; Mohri, M.; Riley, M.; Rostamizadeh, A. Sample selection bias correction theory. In Proceedings of the Conference on Algorithmic Learning Theory (ALT), Budapest, Hungary, 13–16 October 2008.
62. Allix, K.; Bissyé, T.F.; Klein, J.; Traon, Y.L. Androzoo: Collecting millions of android apps for the research community. In Proceedings of the Conference on Mining Software Repositories (MSR), Austin, TX, USA, 14–15 May 2016.
63. Wei, F.; Li, Y.; Roy, S.; Ou, X.; Zhou, W. Deep ground truth analysis of current android malware. In Proceedings of the Conference on Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA), Bonn, Germany, 6–7 July 2017.
64. Wong, S.C.; Gatt, A.; Stamatescu, V.; McDonnell, M.D. Understanding data augmentation for classification: When to warp? In Proceedings of the Conference on Digital Image Computing: Techniques and Applications (DICTA), Gold Coast, Australia, 30 November–2 December 2016.
65. Zhu, Y.; Xi, D.; Song, B.; Zhuang, F.; Chen, S.; Gu, X.; He, Q. Modeling users' behavior sequences with hierarchical explainable network for cross-domain fraud detection. In Proceedings of the International World Wide Web Conference (WWW), Taipei, Taiwan, 20–24 April 2020.
66. Zhuang, F.; Qi, Z.; Duan, K.; Xi, D.; Zhu, Y.; Zhu, H.; Xiong, H.; He, Q. A comprehensive survey on transfer learning. *Proc. IEEE* **2021**, *1091*, 43–76. [CrossRef]
67. Lapuschkin, S.; Wäldchen, S.; Binder, A.; Montavon, G.; Samek, W.; Müller, K.-R. Unmasking Clever Hans predictors and assessing what machines really learn. *Nat. Commun.* **2019**, *10*, 1096. [CrossRef]
68. Warnecke, A.; Arp, D.; Wressnegger, C.; Rieck, K. Evaluating explanation methods for deep learning in security. In Proceedings of the IEEE European Symposium on Security and Privacy (EuroS&P), Genoa, Italy, 7–11 September 2020.
69. Hooker, S.; Erhan, D.; Kindermans, P.J.; Kim, B. A benchmark for interpretability methods in deep neural networks. *arXiv* **2019**, arXiv:1806.10758.
70. Tomsett, R.; Harborne, D.; Chakraborty, S.; Gurram, P.; Preece, A. Sanity checks for saliency metrics. In Proceedings of the AAAI Conference on Artificial Intelligence (AAAI), New York, NY, USA, 7–12 February 2020.
71. Kingma, D.P.; Ba, J. Adam: A method for stochastic optimization. *arXiv* **2014**, arXiv:1412.6980.
72. Sandeep, H.R. Static Analysis of Android Malware Detection using Deep Learning. In Proceedings of the 2019 International Conference on Intelligent Computing and Control Systems (ICCS), Madurai, India, 15–17 May 2019; pp. 841–845.
73. Lashkari, A.H.; Kadir, A.F.A.; Taheri, L.; Ghorbani, A.A. Toward Developing a Systematic Approach to Generate Benchmark Android Malware Datasets and Classification. In Proceedings of the 2018 International Carnahan Conference on Security Technology (ICCST), Montreal, QC, Canada, 22–25 October 2018; pp. 1–7.