computers

MDPI

Article

Deep Convolutional Generative Adversarial Networks in Image-Based Android Malware Detection

Francesco Mercaldo [1,*], Fabio Martinelli [2] and Antonella Santone [1]

[1] Department of Medicine and Health Sciences "Vincenzo Tiberio", University of Molise, 86100 Campobasso, Italy; antonella.santone@unimol.it
[2] Institute for Informatics and Telematics, National Research Council of Italy, 56124 Pisa, Italy; fabio.martinelli@iit.cnr.it
* Correspondence: francesco.mercaldo@unimol.it

Abstract: The recent advancements in generative adversarial networks have showcased their remarkable ability to create images that are indistinguishable from real ones. This has prompted both the academic and industrial communities to tackle the challenge of distinguishing fake images from genuine ones. We introduce a method to assess whether images generated by generative adversarial networks, using a dataset of real-world Android malware applications, can be distinguished from actual images. Our experiments involved two types of deep convolutional generative adversarial networks, and utilize images derived from both static analysis (which does not require running the application) and dynamic analysis (which does require running the application). After generating the images, we trained several supervised machine learning models to determine if these classifiers can differentiate between real and generated malicious applications. Our results indicate that, despite being visually indistinguishable to the human eye, the generated images were correctly identified by a classifier with an F-measure of approximately 0.8. While most generated images were accurately recognized as fake, some were not, leading them to be considered as images produced by real applications.

Keywords: malware; deep learning; GAN; Android; security

Citation: Mercaldo, F.; Martinelli, F.; Santone, A. Deep Convolutional Generative Adversarial Networks in Image-Based Android Malware Detection. *Computers* **2024**, *13*, 154. https://doi.org/10.3390/computers13060154

Academic Editor: Paolo Bellavista

Received: 23 April 2024
Revised: 11 June 2024
Accepted: 15 June 2024
Published: 19 June 2024

1. Introduction and Related Work

The threat posed by adversarial technologies primarily lies in their ability to bypass alarms and access control mechanisms that have been trained using available data [1–3]. Consequently, they pose a significant threat to defensive systems. However, it is worth noting that these very same adversarial technologies can also be used as a defensive tool. By leveraging generative adversarial networks (GANs) and related techniques, security experts can design anomaly-detection and authentication systems that are more resilient against such attacks. In this context, GANs serve as a valuable asset in fortifying cybersecurity measures.

GANs are a kind of convolutional neural network used in unsupervised machine learning. They comprise two main components working in opposition: a generator, which creates synthetic data, and a discriminator (or cost network), which distinguishes between real and generated data. In this adversarial setup, the generator strives to produce realistic data to fool the discriminator, while the discriminator aims to accurately identify which data are real and which are fake.

This adversarial interplay fosters the learning of a generator, which can create remarkably authentic data samples. Once the GAN is trained on a specific dataset, it becomes capable of tasks like future prediction or image generation with high fidelity. For this reason, GANs have broader applications across various domains, including style transfer, data augmentation, text generation, and video synthesis [4].

Recently, there has been a growing interest in the application of GANs in the field of cybersecurity. The rationale behind this interest is quite apparent: by generating deceptive data that closely resemble authentic information, GANs offer a way to exploit vulnerabilities in security systems. This process is known as "evasion", wherein the security system fails to detect the falsified data, leading to a successful breach. Biometric authentication systems, among others, can be particularly susceptible to such attacks.

Lately, with the advancement of deep learning and its ability to construct models proficient in image-related tasks [5,6], various approaches have emerged to detect malware using deep learning techniques. These methods leverage the ability of deep learning to achieve effective classification performance, especially when dealing with image-based data in the context of malware detection.

Starting from these considerations, in this paper, we propose a method aimed to understand whether GANs can represent a threat to image-based malware detection. In particular, we considered two deep convolutional generative adversarial networks (DCGAN) to generate a set of images starting from a dataset of images obtained from real-world Android malware. To the best of the authors' knowledge, the proposed method represents the first attempt to generate images related to Android malware. As a matter of fact, only the paper published by Nguyen et al. [7] proposes the evaluation of malware images generated by a GAN, but related to PC malware and not to mobile ones. Moreover, different from Nguyen et al.'s [7] paper, we considered two different kinds of images: the first one obtained from static analysis (where it is not required to run the application to obtain the image) and the second one generated with a dynamic analysis (where it is required to run the application to obtain the image). Two different ways to obtain the images were considered with the aim to generalize the obtained results: for the same reason, two different GANs were considered; in this way, the results are more generalizable and are not related to a specific set of images or to a specific GAN (and this aspect represents another difference with respect to the paper published by Nguyen et al. [7]).

Thus, with the aim of evaluating the quality of the fake images, we built several machine learning models aimed at discriminating between real and fake Android malware images.

Below, we itemize the main contributions of the paper:

- We propose to use two different DCGANs to generate malware images targeting the Android platform. To the best of the authors' knowledge, this paper represents the first attempt to generate images related to Android malware. As a matter of fact, only the paper published by Nguyen et al. [7] proposes the evaluation of malware images generated by a GAN, but related to PC malware and not to mobile ones;
- Two different variants of the DCGAN were considered;
- Two different sets of images were considered: the first one obtained through static analysis, while the second one generated with dynamic analysis;
- We evaluated the quality of the fake images, by building several machine learning models aimed at discriminating between real and fake images.

Considering their effectiveness in data generation, GANs are already adopted in cybersecurity for several purposes. In Table 1, we compare the state-of-the-art literature related to the adoption of GANs in cybersecurity.

As shown in Table 1, GANs are exploited for many purposes in cybersecurity, from malware detection to the development of anti-phishing tools.

Furthermore, GANs have been shown to be effective in generating realistic malware samples. GANs can be trained on a dataset of existing malware samples to learn the underlying distribution of these samples. Once trained, the GAN can be used to generate new malware samples that are indistinguishable from real malware samples.

Table 1. The state of the art of GAN adoption in cybersecurity.

Research	Method	Results
Zhu and Han (2019) [8]	Developed a GAN-based approach to enhance anomaly detection in network traffic data. Utilized a combination of a GAN and a Long Short-Term Memory (LSTM) network.	Achieved a significant improvement in detecting rare anomalies compared to traditional methods, with a 15% increase in detection accuracy.
Shirazi et al. (2023) [9]	Proposed a GAN-based framework for generating realistic phishing websites to improve the robustness of anti-phishing tools. The framework included a generator for website generation and a discriminator for classification.	The model effectively generated highly realistic phishing websites, helping to improve the detection rate of existing anti-phishing tools by 20%.
Wu et al. (2021) [10]	Introduced a GAN-based intrusion-detection system (IDS) that used a semi-supervised learning approach. The method combined a traditional IDS with a GAN to detect unknown attacks.	Enhanced the IDS performance, particularly in detecting zero-day attacks, with a 12% increase in the true positive rate and a reduction in false positives.
Chen and Liu (2022) [11]	Utilized GANs for malware detection by generating adversarial malware samples to train a more robust malware classifier. The approach involved adversarial training with a focus on evasion attacks.	The robustness of the malware detection system improved significantly, reducing the evasion success rate by 30% compared to conventional methods.
Chkirbene et al. (2021) [12]	Developed a GAN-based framework for data augmentation in cybersecurity datasets, addressing the issue of imbalanced data. The method generated synthetic data samples to augment the training dataset.	Improved the performance of machine learning models in cybersecurity tasks, with an average increase in accuracy of 10% on imbalanced datasets.
Rahman et al. (2024) [13]	Designed a GAN model to generate synthetic network traffic data to aid in training more effective intrusion-detection systems.	Successfully increased the accuracy of intrusion-detection systems by 18% by using the synthetic data for training.
Guo and Zhang (2021) [14]	Implemented a GAN-based approach to detect Advanced Persistent Threats (APTs) by simulating APT behaviors.	The method significantly improved the detection rate of APTs, achieving a 25% increase in detection accuracy compared to conventional methods.
Mustapha et al. (2023) [15]	Proposed a GAN-based architecture to enhance the detection of distributed denial-of-service (DDoS) attacks by generating realistic attack traffic for training purposes.	Enhanced the ability of DDoS-detection systems to identify attack patterns, leading to a 22% improvement in detection accuracy.
Kumar et al. (2024) [16]	Utilized GANs to create adversarial examples to test the robustness of cybersecurity systems, specifically focusing on intrusion-detection systems (IDSs).	The study showed that the generated adversarial examples were effective in identifying weaknesses in IDSs, leading to improved robustness and a 15% reduction in false positives.
Li et al. (2024) [17]	Developed a GAN-based system for automatic detection and classification of botnet attacks in IoT networks. The system used a deep convolutional GAN (DCGAN) for feature extraction and attack detection.	Achieved a high detection rate of 96% for botnet attacks, outperforming traditional detection systems by 20%.

One of the first papers to explore the use of GANs for malware generation was the one by Kurakin et al. [18]. In this paper, the authors showed that a GAN could be trained to generate realistic malware samples that were able to evade most antimalware software.

Another paper [19] investigated the use of GANs for malware generation. In this paper, the authors showed that a GAN could be used to generate adversarial malware samples that were able to fool deep learning-based anomaly-detection systems, similar to the method presented in [20].

In recent years, there has been a growing body of research on the use of GANs for malware generation. Some of the recent advances in this field include the following: the development of new GAN architectures that are more effective at generating realistic

malware samples; the development of new training techniques that allow GANs to be trained on larger and more complex datasets of malware samples; the development of new methods for evaluating the quality of generated malware samples.

As already cited in the Introduction Section, to the best of the authors' knowledge, the only paper close to our proposal is the one published by Nguyen et al. [7]. The main difference between our proposal and this work is due to the context; as a matter of fact in this paper, we focus on the Android platform (the most diffused one), while the authors in [7] considered PC malware.

Renjith et al. [21] presented a method designed to generate feature vectors for generating evasive Android malware and subsequently modifying the malware accordingly. Their proposal offers a twofold contribution: firstly, it can be employed to create datasets for validating detectors of GAN-based malware, and secondly, it can augment the training and testing datasets to enhance the robustness of malware classifiers.

The main difference between our proposal and the method shown in [21] is that authors in [21] considered a GAN to produce a modified version of a set of features extracted from the applications (for instance, from the manifest file related to permission), while we propose the adoption of a GAN to produce an image related to a malware application, i.e., we considered the entire application to generate the (fake) image.

Nagaraju et al. [22] put forth a method focused on generating counterfeit malware images using GANs and assessing the efficacy of diverse techniques for classifying these generated images. Their findings demonstrated that the ensuing multiclass classification problem presents challenges, but they achieved compelling results when limiting the problem to distinguishing between real and fake samples. The primary conclusion drawn from the paper is that, while the GAN-generated images may closely resemble authentic malware images, they do not attain the level of deep fake malware images from a deep learning perspective.

The authors in [19] proposed a GAN, named MalGAN, with the objective of generating adversarial malware applications. In this setup, a neural network-based detector was employed to fit the black-box detector, while a generator was trained to produce adversarial examples capable of deceiving the substitute detector.

Won et al. [23] introduced a malware-training framework called PlausMal-GAN, which leverages generative adversarial networks to generate malware data. PlausMal-GAN successfully generates high-quality and diverse malware images based on existing malware data. The discriminator, acting as a detector, learns a range of malware features from both real and generated malware images. Also, the authors in [23] considered PC malware, different from our paper, and a set of features obtained from the application and not an image obtained by converting all the code and the resources of the application under analysis.

Yuan and colleagues [24] designed and developed GAPGAN, a GAN specifically intended to generate adversarial padding bytes. In their attack framework, they converted the input discrete malware binaries into a continuous space and then input them into the generator of GAPGAN to generate adversarial payloads. By appending these payloads to the original binaries, they created an adversarial sample that maintains its functionality.

Different from all the papers we discussed [7,21], which are related to PC malware, the proposed paper is specifically focused on the Android platform.

The paper proceeds as follows: in the next section, preliminary background notions about GANs, image-based malware detection, and convolutional neural networks are provided; in Section 3 we outline the method we developed and implemented to determine if the DCGAN can generate images related to Android malware applications that are indistinguishable from real ones; the results of the experimental analysis are shown in Section 4; a discussion about the proposed method with conclusions and future works is presented in Section 5.

2. Background

In this section, introductory concepts are presented to ensure the comprehensiveness of the paper. The initial subsection covers details on the architecture of GANs. The subsequent subsection delves into concepts related to image-based malware detection, while the third subsection briefly outlines the architectures of convolutional neural networks.

2.1. GANs

In the fundamental architecture of a GAN, two networks play crucial roles: the generator model and the discriminator model [25]. The term "adversarial" in GANs is related to the concurrent training of these networks, which engage in a competition reminiscent of a zero-sum game, similar to chess. The generator's primary objective is to create new images that are so authentic they can outwit the discriminator. In the simplest GAN setup for image synthesis, the generator takes random noise as the input and produces a generated image as its output [4].

Conversely, the discriminator functions as a binary image classifier, tasked with distinguishing between real and fake images, thereby acting as the "adversary" to the creative efforts of the generator. This continual back-and-forth interplay between the generator and discriminator ultimately leads to the generator becoming more adept at producing realistic images, while the discriminator becomes more skilled at detecting real from generated images.

One of the main advantages of GANs is that they are able to generate high-quality synthetic data. Because the generator and discriminator work collaboratively, the generator can learn from the feedback from the discriminator and, thus, generate synthetic data that are very similar to the real thing. Additionally, GANs are generally quite fast and efficient compared to more traditional methods. Thanks to the parallelization possibilities offered, they use parallel neural networks for the calculation.

In a nutshell, a fundamental GAN architecture comprises three main components: the generator producing fake images, the real images from the training dataset, and the discriminator independently evaluating the authenticity of both real and fake images.

Unlike most deep learning models that optimize to minimize a single cost function (such as in image classification), GANs take a different approach. In GANs, the generator and discriminator each have their own cost functions and conflicting objectives. The generator aims to create fake images that resemble real ones to deceive the discriminator, while the discriminator's goal is to accurately distinguish between real and fake images. This adversarial relationship sets GANs apart from traditional optimization-based models.

As training progresses, both the generator and discriminator enhance their respective capabilities. The generator becomes increasingly adept at producing images that closely resemble the training data, while the discriminator becomes more proficient at accurately discerning between real and fake images. This continuous improvement process is a key characteristic of the GAN training dynamic.

The training of GANs revolves around achieving a delicate balance in the game. The objective is for the generator to generate data that bear a striking resemblance to the training data, while the discriminator reaches a point where it can no longer distinguish between fake and real images. This equilibrium signifies the successful training of the GAN model.

Figure 1 shows an example of a typical GAN architecture, with the generator and the discriminator model.

As shown in Figure 1, the training process involves both the generator and discriminator models working in tandem. Initially, the generator creates a batch of samples, which are then combined with real instances from the dataset and presented to the discriminator for classification as either genuine or counterfeit. The discriminator is subsequently updated to improve its ability to distinguish between real and fake samples in future iterations. Importantly, the updates of the generator are based on its success or failure in deceiving the discriminator with its generated samples.

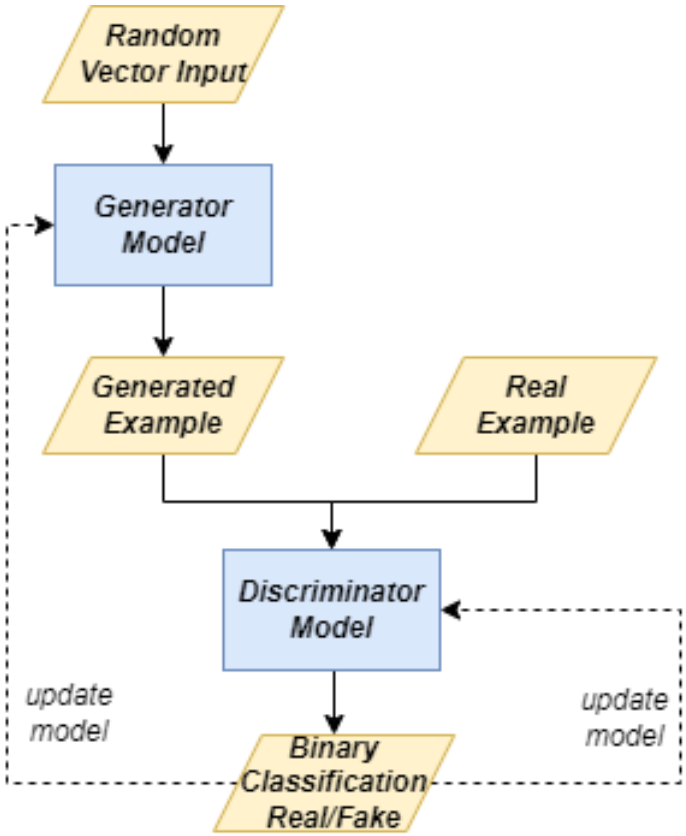

Figure 1. The GAN architecture.

During the training process of a GAN, the discriminator and the generator are trained simultaneously in a competitive manner. The generator generates fake samples (e.g., images), while the discriminator tries to distinguish between real samples (from the dataset) and fake samples (generated by the generator). As a matter of fact, the discriminator in a GAN learns to discriminate between real and fake samples by extracting relevant features from the input images and making classification decisions based on these features. Through iterative training and competition with the generator, the discriminator becomes increasingly effective at distinguishing between real and fake samples, ultimately contributing to the generation of high-quality synthetic data.

For a GAN model to be considered effective, it should demonstrate two key qualities:

- Good image quality, ensuring the generation of sharp and realistic images that closely resemble those in the training dataset. Blurry or distorted images should be minimized.
- Diversity in image generation, implying that the GAN should be capable of producing a wide variety of images that accurately capture the distribution of the training dataset. This diversity ensures that the model can create different instances of the same concept or object.

To evaluate GAN models, visual inspection of the generated images during training or using the generator model for inference can be employed. Additionally, there is a popular evaluation metric i.e., the Fréchet Inception Distance, which compares real and fake images, rather than evaluating the generated images in isolation.

Since the original GAN paper by Ian Goodfellow et al. [4,26] in 2014, numerous GAN variants have emerged. These variants often build upon each other to address specific training challenges or to introduce new architectures for improved GAN control or image quality.

There exist several GAN variants, for instance the deep convolutional generative adversarial network (DCGAN), i.e., the first GAN based on convolutional neural networks

(CNNs) [27], which currently represents one of the most adopted GANs. This is the reason why we resorted to the DCGAN for image generation.

The reason why we resort to GANs for image generation is due to their ability to produce high-quality, realistic images. Here are several reasons why GANs, and specifically DCGANs, are preferred for image generation:

- High-quality outputs: GANs, particularly when well-trained, can generate very high-quality images that are often indistinguishable from real images. This makes them suitable for applications requiring realistic image synthesis;
- Versatility: GANs can be used to generate a wide range of images across different domains, including natural images, medical images, and artwork. This versatility makes them applicable to various fields and tasks;
- Ability to capture data distribution: GANs are capable of learning complex data distributions, allowing them to generate diverse images that capture the variability of the training data;
- Unsupervised learning: GANs can learn to generate images in an unsupervised manner, meaning they do not require labeled data. This is particularly useful when labeled data are scarce or expensive to obtain.

DCGANs are a specific type of GAN that leverage deep CNNs to enhance the image-generation process. Here are the reasons why DCGANs are specifically chosen:

- Improved stability: DCGANs introduce architectural guidelines and practices that improve the stability of training GANs. These include the use of strided convolutions instead of pooling layers, batch normalization, and ReLU activations in the generator and LeakyReLU in the discriminator;
- Enhanced image quality: The use of convolutional layers allows DCGANs to capture spatial hierarchies in images more effectively than fully connected layers. This results in higher quality images with better local coherence and fine details;
- Scalability: DCGANs can be scaled to larger and deeper networks, enabling them to generate higher resolution images. This scalability is crucial for applications requiring detailed and high-resolution outputs;
- Reduced overfitting: The architectural choices in DCGANs, such as batch normalization, help reduce overfitting and improve generalization, leading to more diverse and realistic generated images;
- Effective use of the GPU: DCGANs are designed to take full advantage of GPU acceleration, making the training process more efficient and faster compared to traditional GAN architectures.

GANs could be used to perpetrate cyberattacks or to assist in the detection and analysis of malware. The following is a brief overview of GAN-based malware generation:

- Malware generation using GANs: Some research efforts aimed to use GANs to generate malware samples. GANs were employed to create synthetic malware that could evade traditional antivirus and intrusion-detection systems. These generated malware samples could be used for testing the robustness of security systems.
- Evasion techniques: GAN-based malware generation often focuses on finding ways to create malware that could evade detection. This could involve creating polymorphic or metamorphic malware, which can change their code structure while preserving malicious functionality.
- Data augmentation: GANs are used to augment datasets for training machine learning models used in malware detection. By generating additional malware samples, researchers have sought to improve the performance of machine learning classifiers.
- Anomaly detection: GANs are used for anomaly detection in the context of malware analysis. They could be trained on legitimate software samples, and any deviation from the learned distribution could indicate the presence of malware.
- Adversarial attacks: GANs are also explored for launching adversarial attacks against malware-detection models. Adversarial attacks involve generating malicious inputs

that, while functionally similar to benign inputs, could fool machine learning models into misclassifying them as non-malicious.

2.2. Image-Based Malware Detection

In a standard image-classification task, like identifying the depicted animal in a picture, deep learning models can accurately classify a sample because they have been trained to recognize specific patterns in the input image. These patterns represent distinctive features associated with the shape of the animal. Likewise, in the context of malware detection, the application under analysis is classified by searching for the distinct pattern indicative of a malware application.

Malware detection based on machine learning involves training models to automatically identify and classify malware based on patterns and features extracted from the application under analysis.

Figure 2 shows how machine learning-based malware detection works.

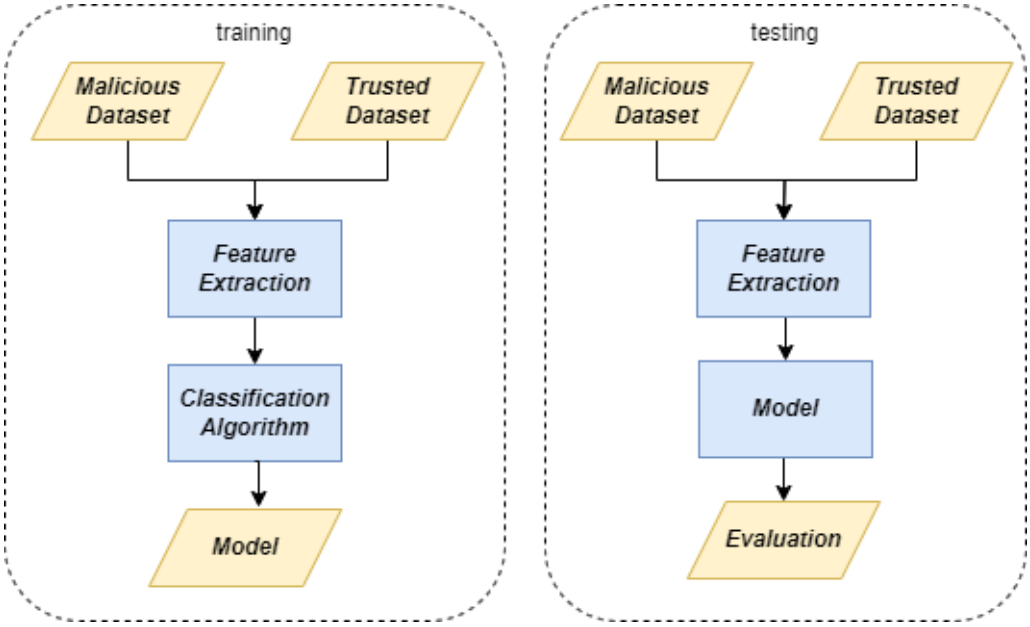

Figure 2. A diagram related to machine learning-based malware detection.

In the following, a simplified overview is given of how this works:

- Data collection: To train a machine learning model for malware detection, a dataset containing examples of both malicious and benign files is needed. This dataset should include a diverse range of malware samples and legitimate files.
- Feature extraction: Features are characteristics or attributes extracted from the files in the dataset. These features can include the file size, file type, API calls, code behavior, byte sequences, and many others. The goal is to capture distinguishing traits that differentiate malware from legitimate files.
- Model selection: In this phase, a machine learning algorithm or model that is well-suited for the problem to solve is considered. Common choices include decision trees, random forests, support vector machines, neural networks, and ensemble methods.
- Model training: The model is trained on the preprocessed dataset. During training, it learns to recognize patterns and relationships in the features that distinguish malware from non-malware.
- Evaluation: After training, the model is evaluated using a separate dataset (i.e., the so-called test set), which it has never seen before. Common evaluation metrics include precision, recall, and the F-measure.

The effectiveness of machine learning-based malware detection depends on the quality and diversity of the training data, the choice of features, the selection of the appropriate model, and ongoing maintenance and updates to keep pace with evolving malware threats. As the threat landscape evolves, machine learning models must adapt to stay effective in identifying new and sophisticated malware variants.

Indeed, both classical image-classification and malware-detection tasks share a common objective: to identify a discerning pattern that can categorize an input sample into one of the output classes.

Converting malware into an image is a crucial and comprehensive step in utilizing deep learning techniques developed for image-classification tasks in the context of malware-detection tasks [28]. The conversion method is widely adopted in the literature [28]: every file stored on the hard disk can be represented in byte code. Each byte can be cast to an eight-bit unsigned integer (ranging from 0 to 255), which can be viewed as a grayscale pixel, thereby representing each application as a grayscale image. Similarly, bytes can be grouped to form pixels in the RGB color model, creating a three-channel color image.

2.3. CNN

In the realm of image-classification tasks, CNNs and their various iterations are extensively embraced and widely used in the existing literature. Briefly, we introduce the main features of these models in this subsection, and we refer to the state-of-the-art literature [29] for further information.

CNNs stand out due to their unique utilization of convolutional layers, employing mathematical convolutional operators to extract features and gather relevant information from input images. This sets them apart from traditional machine learning models, where feature extraction requires manual preprocessing. In contrast, CNNs autonomously extract significant features from the input samples, showcasing their exceptional capacity to learn and identify meaningful patterns automatically from the data. In the literature, numerous intricate CNN architectures exist, for instance AlexNet, VGG, GoogLeNet, and ResNet, but they fundamentally rely on three essential operations: convolution, subsampling, and classification. During the convolution operation, each pixel is combined with its neighboring pixels by sliding a finite matrix (known as the kernel or filter) across the input image. An activation function, typically the Rectified Linear Unit, is then applied to introduce non-linearity to the results. Subsampling, also known as pooling, is a process that decreases the size of the two-dimensional matrix produced by the convolutional layers while preserving essential information. Moreover, subsampling improves the model's resilience and consistency in handling slight shape alterations and distortions found in the input image. Lastly, classification is accomplished through a sequence of dense layers comprising variable numbers of artificial neurons (perceptrons). This section of the model is trainable using the standard backpropagation algorithm.

Figure 3 shows a diagram related to a CNN architecture.

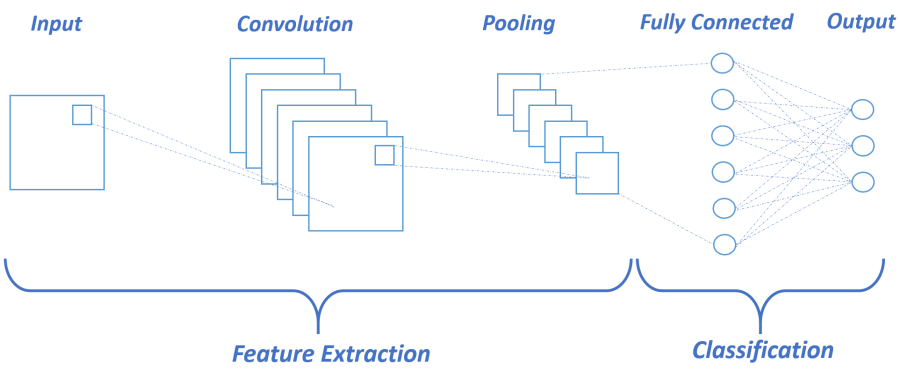

Figure 3. Diagram related to a CNN architecture.

From Figure 3, we note that the convolution operation combines each pixel with the neighboring ones, on the basis of the weights stored in a finite matrix (i.e., the kernel or filter), by sliding this graphic filter on the input image and applying an activation function (usually, the Rectified Linear Unit [30]). The subsampling (or pooling) is the operation that reduces the size of a two-dimensional matrix, usually generated by the convolutional layers, while preserving the most relevant information.

The original matrix is split into sub-matrices of a fixed size, and only one element is extracted from them with regard to the chosen strategy, usually the max pooling, which extracts the highest value. Thus, the CNN is able to correctly classify samples regardless of the relevant pattern spatial position in the input image. Finally, the classification is carried out by a series of dense layers, which are formed by a variable number of artificial neurons (i.e., perceptrons), each connected with all those of the next layer and forming a dense network of connections.

3. The Method

In this section, we introduce our devised method to accomplish two objectives: (i) generating images associated with Android malware applications and (ii) distinguishing these synthetic images from images obtained from real-world Android malware ones.

3.1. Image Generation

As we stated in the Introduction Section, two different datasets were considered: the first one was obtained using static analysis, while the second one was obtained through dynamic analysis. Static analysis and dynamic analysis are two different approaches used in software testing and code analysis. Static analysis is typically faster than dynamic analysis since it does not involve code execution, while dynamic analysis can be slower as it involves running the code, especially for large or complex software applications. Static analysis can be performed on the source code or compiled as a binary without executing the application, while dynamic analysis requires access to a running instance of the application. Moreover, with the aim of generating images associated with Android malware applications, two different variants of DCGAN are exploited.

In Figures 4 and 5, we, respectively, show how we generate the images belonging to the static and dynamic datasets.

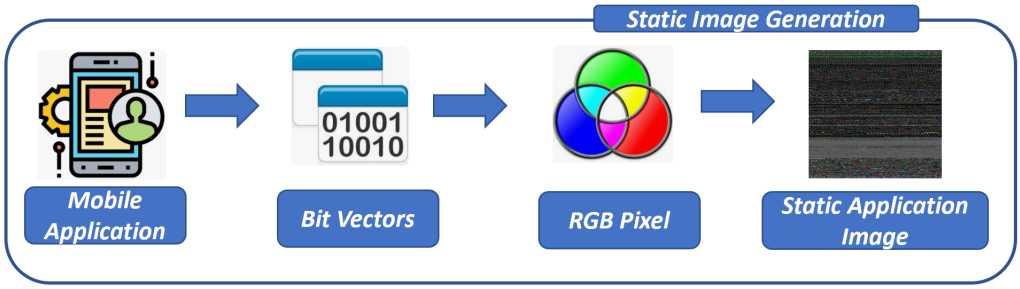

Figure 4. The static-image-generation step.

3.2. Static Analysis Generation

We first discuss how we generated the images from the applications with the static analysis, as shown by the static-image-generation workflow in Figure 4.

To generate an image from an application by exploiting static analysis, i.e., without running the application under analysis, we focused on extracting the byte values from a binary executable and, subsequently, creating the corresponding image.

To transform a binary into an image, we interpreted the sequence of bytes (Bit Vectors in Figure 4) that represents the binary as the bytes of a grayscale PNG image (Grayscale Pixel in Figure 4). For this conversion, we adopted a predetermined width of 256 and a variable length based on the size of the binary. To achieve this, we encoded any binary file into a lossless PNG format, as described in [31].

In essence, we considered the following steps:

- The binary file's individual bytes are converted into numerical values (ranging from 0 to 255), which will subsequently determine the pixel color (RGB Pixel in Figure 4);
- Each byte corresponds to an RGB pixel in the resultant PNG image (Application Image in Figure 4).

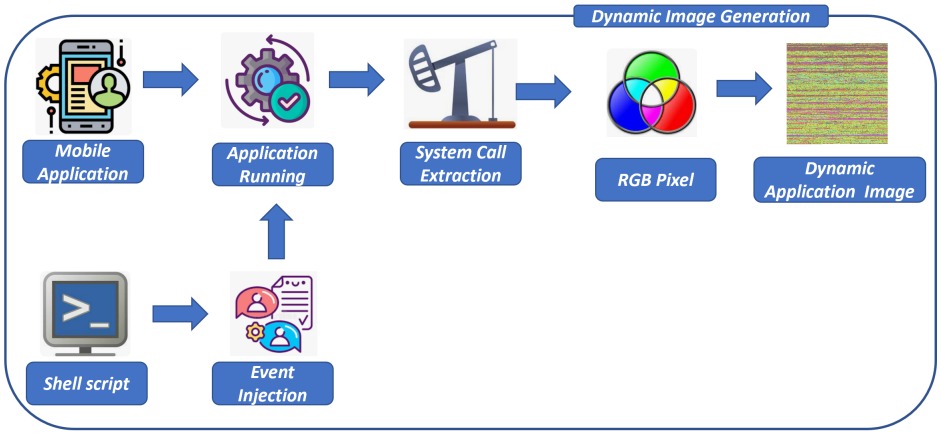

Figure 5. The dynamic-image-generation step.

3.3. Dynamic Analysis Generation

To generate the second dataset, we exploited dynamic analysis; in particular, we executed each application with the aim of extracting the system call traces, and from the obtained trace, we built the related image. Figure 5 depicts this process.

To generate images by exploiting dynamic analysis, we recorded and stored system call traces generated by Android running applications in a textual format. To achieve this, we considered the Android Package (APK) file, which represents the installation file of an Android application (referred to as the "Mobile Application" in Figure 5). Subsequently, we generated a series of 25 distinct operating system events at regular 10-second intervals (referred to as "Event Injection" in Figure 5). These events are then dispatched to the emulator to stimulate the behavior of the malicious payload within the application (referred to as "Mobile Application" in Figure 5). As a result, we obtained the corresponding sequence of system calls (referred to as "System Call Extraction" in Figure 5).

These 25 operating system events were selected based on prior research studies, including those by the authors in [32,33], which demonstrate that malicious actors employ this set of events to activate payloads within the Android environment. Specifically, we considered the operating system event used to trigger Android malware as exploited by the authors in [34].

The retrieval of system calls from the Android application under analysis was carried out using a script developed by the authors (referred to as the "Shell Script" in Figure 5). This script performs a sequence of actions as outlined below:

1. Initialization of the target Android device emulator.
2. Installation of the .apk file of the application under analysis on the Android emulator.
3. Waiting until the device reaches a stable state, typically when it is in an "epoll_wait" state and the application under analysis is awaiting user input or a system event.
4. Commencement of the retrieval of system call traces.
5. Sending one of the 25 selected operating system events to the application.
6. Dispatching the chosen operating system event to the application under analysis.
7. Capturing system calls generated by the application until a stable state is reached.
8. Selection of a new operating system event (i.e., the next one in the sequence) and repeating the above steps to capture system call traces for this new event.
9. Iterating through the previous step until all 25 operating system events have been used to stimulate the Android application.

10. Halting the capture of system calls and saving the acquired system call trace.
11. Terminating the process of the Android application under analysis.
12. Stopping the Android emulator.
13. Reverting the emulator's disk to a clean snapshot, restoring it to its state before the analyzed Android application was installed.

Moreover, to simulate user interaction with the Android operating system, we utilized the "monkey" tool from the Android Debug Bridge (ADB) version 1.0.32. This tool generates pseudo-random user events, including clicks, touches, and gestures. To collect the system call traces, we employed "strace", a tool available on Linux operating systems. Specifically, we used the command "strace-s PID" to hook into the running Android application process and intercept only the system calls generated by that specific application. Once we had obtained a log of system calls, we extracted each individual call one by one, adhering to the order provided by the log, and used this information to construct an image (referred to as "Image Generation" in Figure 5). Each system call corresponds to a specific RGB pixel, allowing us to create the images pixel by pixel.

In Figures 6 and 7, just as an example, we show the images obtained from both the static and dynamic analysis, obtained starting from the same Android application, i.e., GPS Fields Area Measure app https://play.google.com/store/apps/details?id=lt.noframe.fieldsareameasure&hl=en_US, (accessed on 17 June 2024), an app freely available on Google Play, the official Android market https://play.google.com/ (accessed on 17 June 2024), identified by the following package name: lt.noframe.fieldsareameasure. This app is typically exploited to measure an area, with the related details about the distance and perimeter.

Figure 6. An example of an image obtained with static analysis.

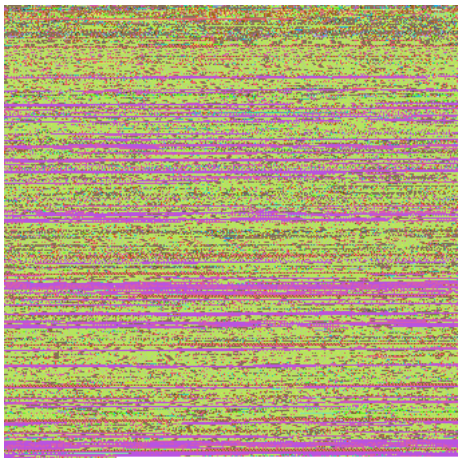

Figure 7. An example of an image obtained with dynamic analysis.

In both Figures 6 and 7, we consider a pixel of a different color to represent, related to the static analysis, a specific byte, while, related to the dynamic one, a specific system call.

3.4. DCGANs

Once having obtained the (static and dynamic) images from real-world Android malware applications, the subsequent step in the proposed method involves employing DCGANs to generate fake images associated with Android applications: this second step related to the proposed method is shown in Figure 8.

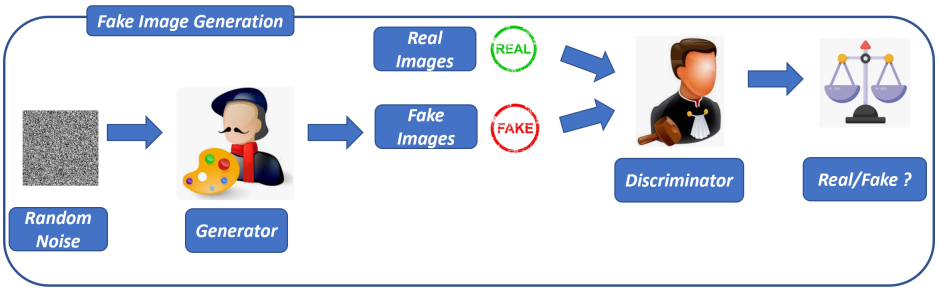

Figure 8. The fake-image-generation step.

In each GAN, there exists at least one generator (generator, as depicted in Figure 8) and one discriminator (discriminator, as shown in Figure 8). Through their adversarial interplay, the generator improves its ability to produce images that closely resemble the distribution of the training data, benefiting from the feedback provided by the discriminator.

In this paper, we developed and we experimented with two different DCGANs, i.e., GAN 1 and GAN 2: in the following subsections, we discuss and go into detail about both architectures with the aim to understand the differences between GAN 1 and GAN 2.

3.5. GAN 1

DCGAN introduced a GAN architecture that employs CNNs to define both the discriminator and generator.

DCGAN provides several architectural guidelines aimed at enhancing training stability [27]:

1. Substituting pooling layers with strided convolutions in the discriminator and fractionally strided convolutions in the generator;
2. Incorporating batch normalization (i.e., batchnorm) in the generator and also the discriminator;
3. Eliminating fully connected hidden layers in deeper architectures;

4. Using ReLU activation for all generator layers with the exception of the output, which employs tanh;
5. Employing LeakyReLU activation in all discriminator layers.

Strided convolutions, characterized by a stride of 2, are convolutional layers employed for downsampling within the discriminator. Conversely, fractionally strided convolutions, also known as Conv2DTranspose layers, employ a stride of 2 for upsampling in the generator.

In the domain of DCGANs, batch normalization (batchnorm) is employed in both the generator and the discriminator to improve the stability of GAN training. batchnorm operates by normalizing the input layer, ensuring it maintains a mean of zero and a variance of one. Typically, batchnorm is integrated after the hidden layer and before the activation layer.

In both the generator and discriminator of DCGANs, four frequently employed activation functions include sigmoid, tanh, ReLU, and LeakyReLU.

The sigmoid function compresses numbers to either 0 (indicating fake) or 1 (indicating real). Since the DCGAN discriminator performs binary classification, we employed the sigmoid activation function in its final layer.

Tanh (Hyperbolic Tangent) is an S-shaped function similar to sigmoid, but it is scaled and centered at 0, mapping the input values to the range of $[-1, 1]$. We applied tanh in the final layer of the generator. Consequently, our training images must be preprocessed to fall within the range of $[-1, 1]$ to match the input requirements of the generator.

The Rectified Linear Activation (ReLU) function produces a zero output for negative input values and preserves the input value for non-negative inputs. In the generator, ReLU activation is utilized for all layers, except the output layer, where tanh activation is employed.

LeakyReLU behaves similarly to ReLU, but introduces a slight slope (determined by a constant alpha) for negative input values. We set the slope (alpha) to 0.2, as shown in [27]. Within the discriminator, LeakyReLU activation is used in all layers, except for the final layer.

The generator and discriminator model training occurs concurrently.

The first step involves data preparation for training. In training a DCGAN, there is no necessity to split the dataset into training, validation, and test sets because we are not using the generator model for classification tasks. A set of images obtained from real-world Android malware was acquired using the procedure illustrated in Figure 8.

The generator expects input images in the format (60,000, 28, 28), representing 60,000 training grayscale images with dimensions of 28 × 28. The loaded data retain a shape of (60,000, 28, 28) as they are in grayscale format.

To ensure compatibility with the tanh activation function used in the generator's final layer, the input images are normalized to fall within the range of $[-1, 1]$.

The main objective of the generator is to generate lifelike images that can trick the discriminator into perceiving them as genuine.

The generator receives random noise as the input and produces an image that closely resembles the training images. To ensure compatibility with the grayscale images of dimensions 28 × 28 being generated, the model architecture must ensure that the output of the generator is shaped as 28 × 28 × 1.

To achieve this, the generator undergoes the following steps:

1. It transforms the 1D random noise (latent vector) into a 3D shape using the Reshape layer.
2. The generator consistently upsamples the noise using the Keras Conv2DTranspose layer (also known as fractionally strided convolution in the paper) to attain the desired output image size, which, in our case, is a grayscale image with dimensions of 28 × 28 × 1.

The generator incorporates several crucial layers as its fundamental building blocks:

1. Fully connected layers, also known as dense layers, are primarily utilized for reshaping and flattening the noise vector.
2. Conv2DTranspose is utilized to upscale the image in the generation process.
3. BatchNormalization: utilized to enhance training stability, positioned after the convolutional layer and before the activation function.

In the generator, ReLU activation is employed in all layers except the output layer, where tanh activation is utilized.

To construct the generator model, we introduced a dense layer to facilitate the reshaping of the input into a 3D format. It is essential to specify the input shape within this initial layer of the model architecture.

Subsequently, the BatchNormalization and ReLU layers were integrated into the generator model. Next, the preceding layer was reshaped from 1D to 3D, followed by two upsampling operations using Conv2DTranspose layers with a stride of 2. This sequential process facilitated the transition from a 7 × 7 size to 14 × 14 and, ultimately, to 28 × 28, achieving the desired image dimensions.

Following each Conv2DTranspose layer, a BatchNormalization layer was added, succeeded by a ReLU layer.

Finally, a Conv2D layer with a tanh activation function was employed in the generator model. The generator model encompasses a total of 2,343,681 parameters, with 2,318,209 being trainable and the remaining 25,472 being non-trainable parameters.

Moving on, let us delve into the design of the discriminator model.

The discriminator functions as a binary classifier tasked with determining whether an image is real or fake. Its main objective is to precisely classify the given images.

However, there are a few distinctions between a discriminator and a typical classifier:

1. We utilized the LeakyReLU activation function in the discriminator.
2. The discriminator deals with two categories of input images: real images sourced from the training dataset labeled as 1 and fake images generated by the generator labeled as 0.

It is noteworthy that the discriminator network is typically smaller or simpler compared to the generator. This is because the discriminator has a relatively simpler task than the generator. In fact, if the discriminator becomes too powerful, it may impede the progress and improvement of the generator.

In formulating the discriminator model, we will once more define a function. The discriminator takes as the input either real images from the training dataset or fake images generated by the generator. These images have dimensions of 28 × 28 × 1, and we pass the arguments (width, height, and depth) according to the function.

In constructing the discriminator model, we incorporated the Conv2D, BatchNormalization, and LeakyReLU layers twice for downsampling. Following this, we introduced the Flatten layer and applied dropout. Finally, in the last layer, we employed the sigmoid activation function to yield a single value for binary classification.

The discriminator model encompasses 213,633 parameters, comprising 213,249 trainable parameters and 384 non-trainable parameters.

Within the framework of the considered DCGAN, we adopted the modified minimax loss, involving the utilization of the binary cross-entropy (BCE) loss function, as illustrated in [27].

It is necessary to calculate two distinct losses: one for the discriminator and another for the generator.

Regarding the discriminator loss, since the discriminator receives two sets of images (real and fake), we computed the loss for each group independently and then merged them to derive the overall discriminator loss.

$$TotalDloss = loss\_from\_real\_images + loss\_from\_fake\_images$$

Concerning the generator loss, our approach diverges from training G to minimize log(1 − D(G(z))), aiming to improve the probability of the discriminator D correctly classify-

ing fake images as fake. Instead, we concentrated on training the generator G to maximize $\log D(G(z))$, representing the probability that D incorrectly classifies the fake images as real. This encapsulates the modified minimax loss strategy we employed. The objective is to enhance the probability of the discriminator D accurately classifying fake images as fake by training the generator G to maximize this probability.

3.6. GAN 2

The second GAN (i.e., GAN 2) we considered for image generation is a variant of GAN 1, slightly modified to train color images.

With respect to GAN 1, we introduced the following modifications in GAN 2:

- Generator: we modified the approach for increasing the model architecture's resolution to produce a color image.
- Discriminator: we adapted the input image dimensions from 28 × 28 × 1 to 64 × 64 × 3.

In the following, we discuss the adjustments required for upsampling to achieve the desired color image size of 64 × 64 × 3:

- We updated CHANNELS = 3 for color images instead of 1, which is for grayscale images;
- A stride of 2 halves the width and height, so we can work backward to figure out the initial image size dimension: in GAN 1, we upsampled as $7 -> 14 -> 28$. Now, we are working with a training image size of 64x64, so we upsampled a few times as $8 -> 16 -> 32 -> 64$. This means we added one more set of Conv2DTranspose $->$ BatchNormalization $->$ ReLU.
- Another change made to the generator is to update the kernel size from 5 to 4 to avoid reducing checkerboard artifacts in the generated images. This is because the kernel size of 5 is not divisible by a stride of 2. So, the solution was to use a kernel size of 4 instead of 5.

The main change in the discriminator architecture is the image input shape: we are using the shape of [64, 64, 3] instead of [28, 28, 1]. We also added one more set of Conv2D $->$ BatchNormalization $->$ LeakyReLU to balance out the increased architecture complexity in the generator as mentioned above. Everything else remained the same.

3.7. The Detection

Once having generated the images with GAN 1 and GAN 2, the last step of the proposed method, shown in Figure 9, is devoted to building several supervised machine learning models aimed at discriminating between real and fake images related to Android malware applications.

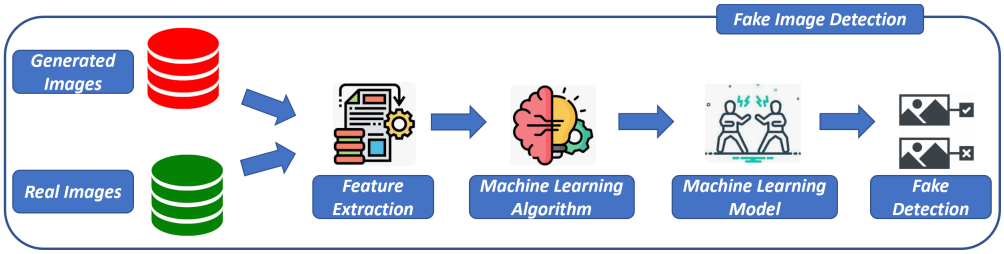

Figure 9. The fake-image-detection step.

As shown in the third step of the proposed method in Figure 9, to build a model aimed at discerning between generated and real images, we need two datasets: the first one was composed by images obtained from Android malware (Real Images in Figure 9) and the second one composed by images generated from GAN 1 and GAN 2, shown in Figure 8 (Generated Images in Figure 9). The real images are the same as those we exploited in step 2 of the proposed method (i.e., fake image generation in Figure 8). From the two sets of images (i.e., the static and the dynamic one), we extracted a set of numeric features

using the Simple Color Histogram Filter. This filter is designed to compute the histogram representing the pixel frequencies of each image. By applying this filter, we extracted 64 numeric features from each image.

Once we had extracted the feature sets from both the generated and real images, we employed these features as the inputs to a supervised machine learning algorithm. The objective was to develop a model capable of discerning whether an image is associated with a fake or real application (i.e., Fake Detection in Figure 9).

Obviously, if the classifiers exhibits optimal performance, the generated images will be significantly different from the original ones; otherwise, the machine learning models will not be able to distinguish the original images from the generated ones.

4. Experimental Analysis

In this section, we present and analyze the results of the experimental analysis.

For this reason, after having appropriately generated a series of images through GAN 1 and GAN 2, we trained several classifiers, in order to understand if they are able to distinguish between malware and trusted applications. As the DCGAN generates a sequence of images during each epoch, we assessed the classifiers' ability to differentiate between real and fake images. This evaluation aimed to determine whether, with increasing epochs, as the generated images presumably become more similar to real ones, the classifier's performance in distinguishing between real and fake application images declines if it fails to accurately identify real images from fake ones.

4.1. Experimental Settings

With the aim to generate images related to Android malware, we exploited a dataset composed by 1000 real-world Android malicious applications (and, thus, we generated 1000 images related to real-world applications), among 71 malware families with the aim to cover the current landscape of Android malware [35]. The dataset offers a representation of the current state of Android malware, is openly shared with the broader community, and is one of the largest publicly available Android malware datasets.

Both GAN 1 and GAN 2 were trained for 50 epochs, and each epoch required approximately 25 s in the experimental analysis (where we exploited the NVIDIA T4 Tensor Core GPU). We set GAN 1 and GAN 2 to generate, for each epoch, 1000 fake images.

Considering that we obtained two different sets of images from the same set of real-world Android malware applications (i.e., the first one obtained from static analysis and the second obtained from dynamic analysis), we define a total of four different experiments:

- GAN 1 static dataset: this experiment is related to the evaluation of images generated with GAN 1 trained with the static dataset;
- GAN 2 static dataset: this experiment is related to the evaluation of images generated with GAN 2 trained with the static dataset;
- GAN 1 dynamic dataset: this experiment is related to the evaluation of images generated with GAN 1 trained with the dynamic dataset;
- GAN 2 dynamic dataset: this experiment is related to the evaluation of images generated with GAN 1 trained with the dynamic dataset.

To evaluate the effectiveness of the classifiers, the following metrics were considered: precision, recall, and F-measure.

Four different widespread supervised machine learning classifiers were exploited with the aim of enforcing conclusion validity: J48 [36], SVM [37], random forest [38], and Bayes [39].

For each algorithm, we built a model for each epoch, for a total of 4 models × 50 epochs = 200 different models. Each model was built with the images obtained from real-world applications and with the image generated for a certain epoch.

We repeated this process for each experiment; this is the reason why, considering the four different experiments, a total of 200 × 4 = 800 different models were evaluated,

by taking into account GAN 1 and GAN 2, trained with the two different sets of images, i.e., the static and the dynamic one.

In the following, we explain how we built and evaluated the machine learning models aimed to discriminate between real and fake images related to Android malware applications.

Related to the model learning, we considered T as a set of labels $\{(M, l)\}$, where each M is the label that is associated with an $l \in \{ real, fake\}$.

For the M model, we built a numeric vector of features $F \in R_y$, where y represents the feature number exploited in the learning phase ($y = 64$; as a matter of fact, this is the number of numeric features obtained, from each image, by applying the Simple Color Histogram Filter).

In detail, with respect to the training phase, the k-fold cross-validation was exploited. We explain this process: the instances of the dataset were split in a random way into a set denoted as k.

Let us consider D as the dataset; in order to test the effectiveness of both models we propose, the procedure explained below was considered.

1. Generation of a set for training, i.e., $T \subset D$;
2. Generation of an evaluation set $T' = D \div T$;
3. Execution of the model training T;
4. Application of the model previously generated to each element of the T' set.

To mitigate overfitting, we employed cross-validation, ensuring that all samples in the dataset were assessed during the testing phase. In this process, the full dataset was divided into k equal-sized parts, and in each iteration, one part served as the validation set, while the rest was utilized as the training set. This approach allows for a more comprehensive evaluation of the dataset while preventing overfitting. The k-fold cross-validation procedure comprised dividing the training dataset into k-folds. In each iteration, $k - 1$ folds were used for training the model, while the remaining k-th fold acted as the test set. This process was repeated, giving each fold an opportunity to serve as the holdout test set. A total of k models were trained and evaluated, and the model's performance was computed as the mean of these runs. This approach provides a more realistic estimation of model performance on small training datasets compared to a single training/test split. In this paper, we considered a value of k = 10 for model training and testing.

This procedure was considered for all the algorithms involved in the experiment (i.e., J48, SVM, random forest, and Bayes) for each epoch.

4.2. The Results

Table 2 shows the experimental analysis results obtained with the procedure previously explained.

In Table 2, we present the results of the experimental analysis, limiting the presentation to data from three epochs due to space constraints: the first one (i.e., 0 in the column Epoch), the middle one (i.e., 25 in the column Epoch), and the final one (i.e., 49 in the column Epoch), with the aim to understand the general trend.

From the results shown in Table 2, we can observe that the performances of the precision, recall, and F-measure were quite similar for all the epochs (as a matter of fact, all the values were greater than 0.8): there were no substantial differences between the performances obtained with the models trained with images obtained with the 25 epoch exams or the 50-th.

This behavior is symptomatic of the fact that the images generated during the various epochs were no longer similar to the real ones as the epochs increased. Furthermore, the values of the precision, recall, and F-measure obtained in any case make it understood that a part of the fake applications was not correctly recognized by the various classifiers, even when they were generated in the initial epochs.

Table 2. Experimental analysis results for the 0, 25, and 49 epochs with GAN 1 static.

Epoch	Algorithm	Precision	Recall	F-Measure
0	J48	0.877	0.876	0.876
	SVM	0.870	0.869	0.869
	Random Forest	0.880	0.880	0.880
	Bayes	0.839	0.838	0.838
25	J48	0.892	0.892	0.892
	SVM	0.880	0.879	0.879
	Random Forest	0.892	0.892	0.892
	Bayes	0.832	0.832	0.832
49	J48	0.835	0.833	0.833
	SVM	0.836	0.835	0.835
	Random Forest	0.837	0.837	0.837
	Bayes	0.816	0.816	0.816

Table 3 shows the results obtained with the images generated with GAN 2 trained with the static dataset.

Table 3. Experimental analysis results for the 0, 25, and 49 epochs with GAN 2 static.

Epoch	Algorithm	Precision	Recall	F-Measure
0	J48	0.891	0.892	0.891
	SVM	0.883	0.884	0.883
	Random Forest	0.890	0.890	0.890
	Bayes	0.847	0.842	0.844
25	J48	0.877	0.878	0.887
	SVM	0.888	0.887	0.902
	Random Forest	0.901	0.904	0.845
	Bayes	0.844	0.847	0.845
49	J48	0.825	0.826	0.825
	SVM	0.844	0.847	0.845
	Random Forest	0.841	0.847	0.843
	Bayes	0.827	0.831	0.828

We can note that similar results were obtained with the images generated from GAN 1 and GAN 2 with the static dataset: as a matter of fact, the F-measure is approximately equal to 0.8 in both cases.

Table 4 shows the results obtained with GAN 1 trained with the dynamic dataset, i.e., the one obtained starting from the system call traces.

Table 4. Experimental analysis results for the 0, 25, and 49 epochs with GAN 1 dynamic.

Epoch	Algorithm	Precision	Recall	F-Measure
0	J48	0.871	0.882	0.876
	SVM	0.873	0.874	0.873
	Random Forest	0.880	0.880	0.880
	Bayes	0.837	0.832	0.834
25	J48	0.877	0.878	0.877
	SVM	0.878	0.877	0.877
	Random Forest	0.885	0.854	0.869
	Bayes	0.831	0.826	0.828
49	J48	0.825	0.826	0.825
	SVM	0.834	0.835	0.834
	Random Forest	0.832	0.835	0.833
	Bayes	0.809	0.811	0.809

As shown from Table 4, also in this case, the F-measure reached a similar value to the ones obtained with the images generated from GAN 1. This is symptomatic of the obtained results be able to be considered generalizable: as a matter of fact, they do not depend on the typical GAN architecture (in fact, we considered two different DGCANs, with different architectures and different input image dimensions and colors, i.e., one grayscale and another one RGB). Moreover, the results are also not dependent on the trained images we considered: as a matter of fact, we took into account two different sets of images: the first one obtained from static analysis and the second one generated from the system call trace, i.e., obtained by running the applications and, thus, by a dynamic analysis process.

Table 5 shows the results obtained with GAN 1 trained with the dynamic dataset.

Table 5. Experimental analysis results for the 0, 25, and 49 epochs with GAN 2 dynamic.

Epoch	Algorithm	Precision	Recall	F-Measure
0	J48	0.904	0.901	0.902
	SVM	0.901	0.901	0.901
	Random Forest	0.903	0.904	0.903
	Bayes	0.868	0.877	0.872
25	J48	0.877	0.878	0.877
	SVM	0.888	0.887	0.887
	Random Forest	0.898	0.886	0.891
	Bayes	0.854	0.859	0.856
49	J48	0.825	0.826	0.825
	SVM	0.864	0.855	0.859
	Random Forest	0.874	0.876	0.874
	Bayes	0.844	0.841	0.842

Also from the results shown in Table 5, we can confirm a similar F-measure trend, so confirming that the classifiers were able to discriminate between real and fake images with the F-measure approximately equal to 0.8.

To better understand the trend of the classifiers during several of the epochs, in Figure 10, we show the plot of the F-measure trend for the 50 epochs for the J48 model.

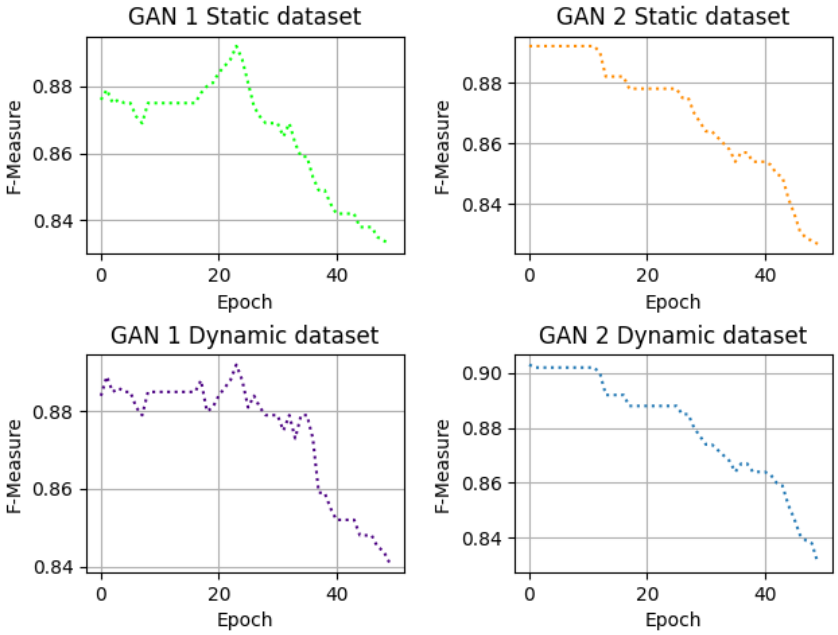

Figure 10. The F-measure trend, obtained with the J48 model, for the 50 epochs.

From Figure 10, we can observe that, for all the classifiers built with J48 (i.e., GAN 1 static dataset, GAN 2 static dataset, GAN 1 dynamic dataset, and GAN 2 dynamic dataset), similar performances were obtained: in particular, we note that also the F-measure trends during the epochs are really similar between the four plots. As a matter of fact, it appears to be very similar for GAN 2 (both for the static and dynamic dataset) and for GAN 1 (also in this case, the static and dynamic dataset).

A similar consideration can be made for the results obtained from the other classifiers; as a matter of fact, we can observe that there was no presence of a particular trend; the F-measure, for all the models involved in the experiment, was stable to a value greater than 0.8, and this happened regardless of the epoch. From the machine learning model point of view, as the epochs increase, images that are more similar to the real ones are not generated, and this aspect is noted by the F-measure values, which were extremely similar in all epochs (both the initial and the final ones). This behavior was found in all four models used, so we can conclude that this is a general trend and not specific to a single classification algorithm. For this reason, it is possible to conclude that, due to the current state of the art of GANs, they do not currently represent a threat, as a classifier is able to discriminate between real images and fake images with quite good performances. On the other hand, however, we highlight that a small percentage of images managed to evade the controls and, for this reason, elude detection, and this aspect in the future could actually pose a threat in the context of image-based malware detection.

With the aim to confirm the obtained results, in Figure 11, we represent the ROC curves obtained with the J48 model, for the 50th epoch.

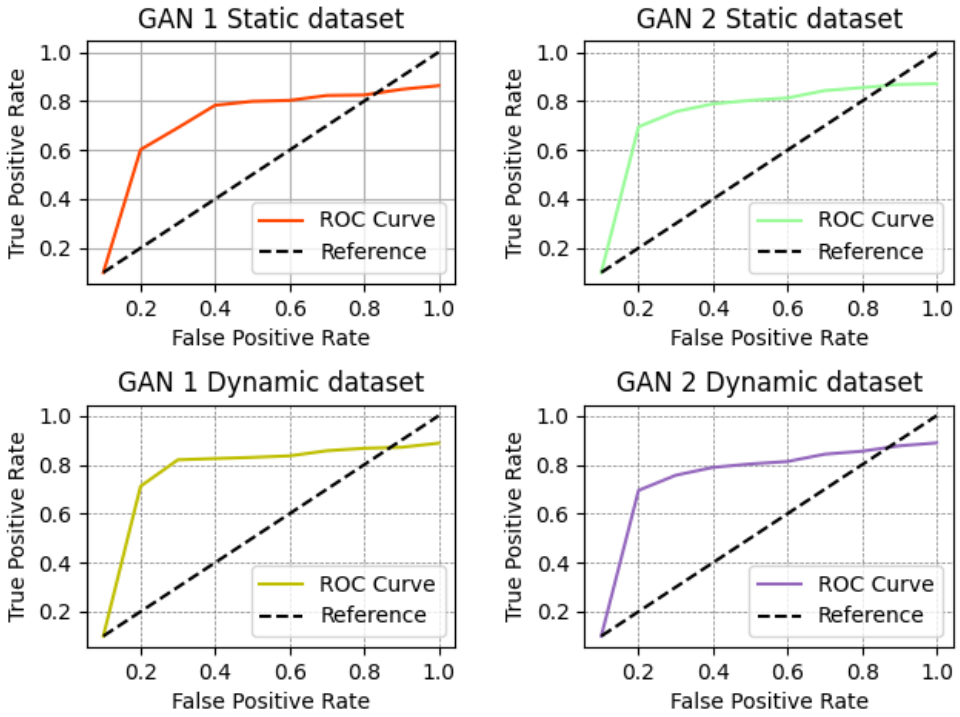

Figure 11. The ROC curves' trend, obtained with the J48 model, for the 50 epochs.

The ROC curve is a graphical representation used in statistics and machine learning to evaluate the performance of a binary classification model. It helps assess the model's ability to distinguish between two classes, typically the positive class. In an ROC curve, the diagonal line represents the performance of a random classifier with no predictive power. The closer the ROC curve is to the top-left corner of the plot, the better the model's performance is. As shown from the ROC curves in Figure 11, all the ROC curves reach approximately a value greater than 0.8, confirming the ability of the classifier to discriminate

between fake and original samples, even if there are several samples able to elude the classifiers. This analysis confirms the obtained results.

In the following, with the aim of showing that, by increasing the number of samples considered to train the GAN and the epochs, we confirmed the obtained results, and we present the results we obtained by using 10,000 samples, where 5000 were real-world malware and the remaining 5000 were generated by GAN 1 with the static analysis. In this experiment, 100 epochs were considered (also, the number of epochs was increased to understand what happens when the number of epochs and samples is increased). The idea behind this experiment was to show that, by increasing the number of epochs and the samples, as expected, the generated images showed better quality, and for this reason, it was more difficult for the classifier to discriminate between real and fake images (although, we have to note that the performance of the classifiers was slightly lower than in the previously evaluated cases).

As shown from the results in Table 6, by using a larger number of applications and increasing the number of epochs, we obtained fake images that were decidedly more similar to the original. The classifiers showed slightly lower performance compared to the previous cases, where 1000 images were used to train the GANs (and not 5000, as in this last experiment), but despite this aspect, although having lower performance, the classifiers were able to distinguish a fair percentage of fake images. We computed the Fréchet Inception Distance related to this experiment, considering 5000 real and 5000 fake images. The score provides a summary of the similarity between the two groups based on statistical analysis of computer vision features extracted from the raw images using the Inception V3 model for image classification. A lower score suggests greater similarity between the two image groups, with a perfect score of 0.0 signifying that the two groups are identical in terms of their image statistics. We obtained a Fréchet Inception Distance value equal to 14.327; this value indicates that the generated images are very similar to real images and are of high quality. The generator was capable of producing images that were hard to distinguish from real ones.

Table 6. Experimental analysis results for the 99 epochs with GAN 1 static.

Epoch	Algorithm	Precision	Recall	F-Measure
99	J48	0.723	0.721	0.721
	SVM	0.701	0.715	0.707
	Random Forest	0.702	0.703	0.702
	Bayes	0.697	0.699	0.697

To summarize, we observed that all classifiers built with J48 (i.e., GAN 1 static dataset, GAN 2 static dataset, GAN 1 dynamic dataset, and GAN 2 dynamic dataset) exhibited similar performance.

The F-measure trends during the epochs are remarkably similar across all four plots shown in Figure 10.

This similarity is consistent for both GAN 2 (static and dynamic datasets) and GAN 1 (static and dynamic datasets).

From the perspective of the machine learning model, increasing epochs did not generate images more similar to real ones and the F-measure values remained extremely consistent across all epochs, from initial to final.

This consistent behavior across all four models indicates a general trend, not specific to a single classification algorithm. Therefore, it can be concluded that, given the current state of GAN technology, GAN-generated images do not currently pose a significant threat and classifiers are able to discriminate between real and fake images with good performance. However, it is important to note that a small percentage of images managed to evade detection, and this aspect could potentially pose a future threat in the context of image-based malware detection.

5. Conclusions and Future Work

Considering that GANs continue to advance, there is a growing concern about their potential misuse for malicious purposes, such as creating undetectable malware variants.

Below are some practical scenarios where our proposed method for distinguishing between real and fake malware-based images could be valuable:

Enhancing cybersecurity measures: By accurately identifying AI-generated malware, cybersecurity experts can develop more effective defense mechanisms to protect against emerging threats. This could involve refining intrusion-detection systems, antivirus software, and other security tools to better recognize and respond to novel attack vectors.

Forensic analysis: In the event of a cyberattack, forensic malware analysts may encounter malware samples that have been artificially generated to evade traditional detection methods. Our approach provides forensic analysts with an additional tool to ascertain the authenticity of suspicious files, aiding in the attribution of attacks and the identification of threat actors.

Training and evaluation: The proposed method for detecting malware images generated by a GAN can contribute to the development and evaluation of machine learning models for malware detection. As a matter of fact, by generating realistic synthetic samples, we can assess the robustness and generalization capabilities of classifiers, thereby improving their performance in real-world scenarios.

Policy and regulation: As AI technologies continue to evolve, policymakers and regulatory bodies may need to consider the potential risks associated with AI-generated content, including malware. Our research sheds light on these risks and underscores the importance of proactive measures to mitigate them, such as establishing guidelines for the responsible use of AI in cybersecurity.

Moreover, from the real-world malware analysis point of view, the proposed method can provide some insights:

Representation for analysis: While malware is executable code, its analysis often involves multiple facets, including static and dynamic analysis of code behavior, metadata extraction, and more. Our decision to represent malware as images stems from the recognition that visual representations can offer complementary information for analysis. For instance, visual representations can capture structural patterns, byte-level distributions, and other features that may not be immediately apparent in the raw executable code. By leveraging techniques from computer vision and deep learning, we aim to extract meaningful insights from these visual representations to aid in malware detection and analysis.

Adversarial attacks and evasion techniques: It is essential to recognize that the evasion techniques employed by malware authors continue to evolve. As attackers increasingly leverage AI and machine learning to create sophisticated malware variants, there is a growing need for robust detection methods capable of identifying both traditional and AI-generated threats. In this context, the proposed manuscript contributes to this broader goal by exploring the potential of GAN-generated images as a means of representing and analyzing malware, thereby enhancing our understanding of the evolving threat landscape.

Exploration of novel detection methods: By framing the problem of malware detection in the context of image analysis, we aim to stimulate new approaches and perspectives within the malware analysis research community. While traditional methods focus primarily on code-level analysis, the proposed manuscript encourages researchers to explore alternative modalities and representations for detecting and mitigating malware threats. Through experimentation and empirical evaluation, we demonstrated the feasibility of using machine learning models trained on image data to discriminate between real and AI-generated malware images, thus opening up new avenues for research and innovation in the field.

The proposed method can be useful for real-world malware analysis, for instance in a scenario where the capability to differentiate between a real malware-based image and a fake image is needed:

Let us imagine, for instance, a malware analyst tasked with identifying and mitigating a sophisticated malware campaign targeting mobile devices. The attackers have employed advanced techniques, including the use of AI-generated malware variants designed to evade traditional detection methods. As part of their investigation, the analyst comes across a suspicious application that exhibits unusual behavior and raises suspicions of being malicious.

In such a scenario, the proposed method for distinguishing between real and fake malware-based images could be of interest. By subjecting the suspicious application to the proposed classifier, trained on a dataset of both real-world malware samples and AI-generated malware images, the analyst can quickly assess the likelihood of the application being part of the ongoing attack campaign.

If the classifier identifies the application as fake, with high confidence, the analyst can confidently conclude that it is an AI-generated malware variant and take appropriate action to quarantine and analyze it further. On the other hand, if the application is classified as real, the analyst can proceed with traditional forensic analysis techniques to investigate its origins and potential impact on the targeted devices.

This scenario highlights the practical utility of our research in empowering cybersecurity professionals to combat emerging threats posed by AI-generated malware. By providing a reliable means of differentiating between real and fake malware-based images, our method enhances the effectiveness of malware detection and response efforts, ultimately contributing to the overall security of digital ecosystems.

Thus, considering the ability of GANs to generate images that are indistinguishable from the human eye, the need arises to understand if they can pose a threat to image-recognition-based systems, including malware detection that analyzes suitably converted applications through deep learning in images. We proposed a method aimed to evaluate whether the images (related to Android malware applications) generated by two different DCGANs can be discriminated by real ones. We obtained two different sets of images, respectively obtained by employing static and dynamic analysis. Once we had generated the images, we resorted to four different supervised machine learning algorithms to understand whether it is possible to build a model aimed at discriminating between real images (obtained from Android malware) and fake images. The experimental analysis showed that all the supervised models we considered were able to discriminate real images from fake ones with an F-measure greater than 0.80, therefore, on the one hand, demonstrating that most of the fake images were recognized, but with the awareness that several images related to Android malware managed to evade the fake-detection models.

From the limitations point of view, we are aware of concept drift [40], i.e., the phenomenon where the statistical properties or characteristics of malware change over time. This can make it challenging for machine learning models and classifiers to accurately identify and categorize malware instances. As a matter of fact, concept drift occurs because malware authors constantly evolve their techniques to evade detection. They may employ new tactics, modify existing malware code, or create entirely new variants to bypass traditional security measures. As a result, the features and patterns that were once indicative of malware may become less relevant or even obsolete. To address concept drift in malware classification, security researchers and data scientists need to regularly update their models and feature sets. This involves monitoring the evolving threat landscape, collecting new malware samples, and adapting the classification algorithms to better capture the changing characteristics of malware. Continuous monitoring and adaptation are crucial to maintaining the effectiveness of malware-detection systems in the face of evolving threats. To mitigate this aspect, we considered an additional dataset, i.e., the one obtained with dynamic analysis, with the aim to understand the effect of concept drift.

In future work, we plan to consider other kinds of images obtained, for instance from the malware system call trace, and we will evaluate other GANs, for instance the conditional generative adversarial network and the cycle-consistent generative adversarial network, to compare the obtained results with the one shown by the two DGCANs ex-

ploited in this paper. Moreover, we will explore whether the Autoencoder or OneClassSVM can be helpful to obtain better performances than the supervised machine learning models we exploited. We will also explore the possibility of utilizing a GAN to generate executable code, meaning code that is intended to be run directly as a program once it is extracted from the generated image. This involves creating a GAN model that not only produces data resembling executable files, but also ensures that the generated code is functional and capable of executing specific tasks when run. In this way, we will effectively understand if GANs are able to automatically generate executable code.

Author Contributions: Conceptualization, F.M. (Francesco Mercaldo), F.M. (Fabio Martinelli) and A.S.; methodology, F.M. (Francesco Mercaldo), and A.S.; software, F.M. (Francesco Mercaldo), F.M. (Fabio Martinelli) and A.S.; validation, F.M. (Francesco Mercaldo), F.M. (Fabio Martinelli) and A.S.; formal analysis, F.M. (Francesco Mercaldo), F.M. (Fabio Martinelli) and A.S.; investigation, F.M. (Francesco Mercaldo), F.M. (Fabio Martinelli) and A.S.; resources, F.M. (Francesco Mercaldo), F.M. (Fabio Martinelli) and A.S.; data curation, F.M. (Francesco Mercaldo), F.M. (Fabio Martinelli) and A.S.; writing—original draft preparation, F.M. (Francesco Mercaldo), F.M. (Fabio Martinelli) and A.S.; writing—review and editing, F.M. (Francesco Mercaldo), F.M. (Fabio Martinelli) and A.S.; supervision, F.M. (Francesco Mercaldo). All authors have read and agreed to the published version of the manuscript.

Funding: This work has been partially supported by EU DUCA, EU CyberSecPro, SYNAPSE, PTR 22-24 P2.01 (Cybersecurity), and SERICS (PE00000014) under the MUR National Recovery and Resilience Plan funded by the EU-NextGenerationEU projects, by MUR-REASONING: foRmal mEthods for computAtional analySis for diagnOsis and progNosis in imagING-PRIN, e-DAI (Digital ecosystem for integrated analysis of heterogeneous health data related to high-impact diseases: innovative model of care and research), Health Operational Plan, FSC 2014–2020, PRIN-MUR-Ministry of Health, the National Plan for NRRP Complementary Investments D^3 4 Health: Digital Driven Diagnostics, prognostics and therapeutics for sustainable Health care, Progetto MolisCTe, Ministero delle Imprese e del Made in Italy, Italy, CUP: D33B22000060001, and FORESEEN: FORmal mEthodS for attack dEtEction in autonomous driviNg systems CUP N.P2022WYAEW.

Data Availability Statement: The Android Malware application exploited are freely available for research purposes from the following http://www.malgenomeproject.org/.

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

References

1. Tasneem, S.; Gupta, K.D.; Roy, A.; Dasgupta, D. Generative Adversarial Networks (GAN) for Cyber Security: Challenges and Opportunities. In Proceedings of the 2022 IEEE Symposium Series on Computational Intelligence, Singapore, 4–7 December 2022; pp. 4–7.
2. Chhetri, S.R.; Lopez, A.B.; Wan, J.; Al Faruque, M.A. Gan-sec: Generative adversarial network modeling for the security analysis of cyber-physical production systems. In Proceedings of the 2019 Design, Automation & Test in Europe Conference & Exhibition (DATE), Florence, Italy, 25–29 March 2019; IEEE: Piscataway, NJ, USA, 2019; pp. 770–775.
3. Dumagpi, J.K.; Jeong, Y.J. Evaluating gan-based image augmentation for threat detection in large-scale xray security images. *Appl. Sci.* **2020**, *11*, 36. [CrossRef]
4. Goodfellow, I.; Pouget-Abadie, J.; Mirza, M.; Xu, B.; Warde-Farley, D.; Ozair, S.; Courville, A.; Bengio, Y. Generative adversarial networks. *Commun. ACM* **2020**, *63*, 139–144. [CrossRef]
5. He, H.; Yang, H.; Mercaldo, F.; Santone, A.; Huang, P. Isolation Forest-Voting Fusion-Multioutput: A stroke risk classification method based on the multidimensional output of abnormal sample detection. *Comput. Methods Programs Biomed.* **2024**, *253*, 108255. [CrossRef]
6. Huang, P.; Li, C.; He, P.; Xiao, H.; Ping, Y.; Feng, P.; Tian, S.; Chen, H.; Mercaldo, F.; Santone, A.; et al. MamlFormer: Priori-experience Guiding Transformer Network via Manifold Adversarial Multi-modal Learning for Laryngeal Histopathological Grading. *Inf. Fusion* **2024**, *108*, 102333. [CrossRef]
7. Nguyen, H.; Di Troia, F.; Ishigaki, G.; Stamp, M. Generative adversarial networks and image-based malware classification. *J. Comput. Virol. Hacking Tech.* **2023**, *19*, 579–595. [CrossRef]
8. Zhu, X.; Han, J. Improving anomaly detection in network traffic with GANs. *IEEE Trans. Netw. Serv. Manag.* **2019**, *16*, 1234–1245.
9. Shirazi, H.; Muramudalige, S.R.; Ray, I.; Jayasumana, A.P.; Wang, H. Adversarial autoencoder data synthesis for enhancing machine learning-based phishing detection algorithms. *IEEE Trans. Serv. Comput.* **2023**, *16*, 2411–2422. [CrossRef]

10. Wu, Y.; Nie, L.; Wang, S.; Ning, Z.; Li, S. Intelligent intrusion detection for internet of things security: A deep convolutional generative adversarial network-enabled approach. *IEEE Internet Things J.* **2021**, *10*, 3094–3106. [CrossRef]
11. Chen, H.; Liu, W. Adversarial Malware Detection with Generative Adversarial Networks. *IEEE Access* **2022**, *10*, 38472–38483.
12. Chkirbene, Z.; Abdallah, H.B.; Hassine, K.; Hamila, R.; Erbad, A. Data augmentation for intrusion detection and classification in cloud networks. In *2021 International Wireless Communications and Mobile Computing (IWCMC)*; IEEE: Piscataway, NJ, USA, 2021; pp. 831–836.
13. Rahman, S.; Pal, S.; Mittal, S.; Chawla, T.; Karmakar, C. SYN-GAN: A robust intrusion detection system using GAN-based synthetic data for IoT security. *Internet Things* **2024**, *26*, 101212. [CrossRef]
14. Guo, J.; Zhang, W. Application of GANs in detecting Advanced Persistent Threats. *Cybersecurity* **2021**, *4*, 45–58.
15. Mustapha, A.; Khatoun, R.; Zeadally, S.; Chbib, F.; Fadlallah, A.; Fahs, W.; El A.A. Detecting DDoS attacks using adversarial neural network. *Comput. Secur.* **2023**, *127*, 103117. [CrossRef]
16. Kumar, V.; Kumar, K.; Singh, M. Generating practical adversarial examples against learning-based network intrusion detection systems. *Ann. Telecommun.* **2024**, 1–18. [CrossRef]
17. Li, S.; Cao, Y.; Liu, S.; Lai, Y.; Zhu, Y.; Ahmad, N. HDA-IDS: A Hybrid DoS Attacks Intrusion Detection System for IoT by using semi-supervised CL-GAN. *Expert Syst. Appl.* **2024**, *238*, 122198. [CrossRef]
18. Kurakin, A.; Goodfellow, I.; Bengio, S. Adversarial machine learning at scale. *arXiv* **2016**, arXiv:1611.01236.
19. Hu, W.; Tan, Y. Generating adversarial malware examples for black-box attacks based on GAN. In Proceedings of the International Conference on Data Mining and Big Data, Beijing, China, 21–24 November 2022; Springer: Berlin/Heidelberg, Germany, 2022; pp. 409–423.
20. Kawai, M.; Ota, K.; Dong, M. Improved malgan: Avoiding malware detector by leaning cleanware features. In Proceedings of the 2019 International Conference on Artificial Intelligence in Information and Communication (ICAIIC), Okinawa, Japan, 11–13 February 2019; IEEE: Piscataway, NJ, USA, 2019; pp. 40–45.
21. Renjith, G.; Laudanna, S.; Aji, S.; Visaggio, C.A.; Vinod, P. GANG-MAM: GAN based enGine for Modifying Android Malware. *SoftwareX* **2022**, *18*, 100977.
22. Nagaraju, R.; Stamp, M. Auxiliary-classifier GAN for malware analysis. In *Artificial Intelligence for Cybersecurity*; Springer: Berlin/Heidelberg, Germany, 2022; pp. 27–68.
23. Won, D.O.; Jang, Y.N.; Lee, S.W. PlausMal-GAN: Plausible malware training based on generative adversarial networks for analogous zero-day malware detection. *IEEE Trans. Emerg. Top. Comput.* **2022**, *11*, 82–94. [CrossRef]
24. Yuan, J.; Zhou, S.; Lin, L.; Wang, F.; Cui, J. Black-box adversarial attacks against deep learning based malware binaries detection with GAN. In *ECAI 2020*; IOS Press: Amsterdam, The Netherlands, 2020; pp. 2536–2542.
25. Goodfellow, I.; Bengio, Y.; Courville, A. *Deep Learning*; MIT Press: Cambridge, MA, USA, 2016.
26. Odena, A.; Buckman, J.; Olsson, C.; Brown, T.; Olah, C.; Raffel, C.; Goodfellow, I. Is generator conditioning causally related to GAN performance? In Proceedings of the International Conference on Machine Learning, PMLR, Stockholm, Sweden, 10–15 July 2018; pp. 3849–3858.
27. Radford, A.; Metz, L.; Chintala, S. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv* **2015**, arXiv:1511.06434.
28. He, K.; Kim, D.S. Malware detection with malware images using deep learning techniques. In Proceedings of the 2019 18th IEEE International Conference on Trust, Security and Privacy in Computing and Communications/13th IEEE International Conference on Big Data Science and Engineering (TrustCom/BigDataSE), Rotorua, New Zealand, 5–8 August 2019; IEEE: Piscataway, NJ, USA, 2019; pp. 95–102.
29. LeCun, Y.; Bottou, L.; Bengio, Y.; Haffner, P. Gradient-based learning applied to document recognition. *Proc. IEEE* **1998**, *86*, 2278–2324. [CrossRef]
30. Ramachandran, P.; Zoph, B.; Le, Q.V. Searching for Activation Functions. *arXiv* **2017**, arXiv:1710.05941.
31. Mercaldo, F.; Santone, A. Deep learning for image-based mobile malware detection. *J. Comput. Virol. Hacking Tech.* **2020**, *16*, 157–171. [CrossRef]
32. Zhou, Y.; Jiang, X. Dissecting android malware: Characterization and evolution. In Proceedings of the 2012 IEEE Symposium on Security and Privacy (SP), San Francisco, CA, USA, 20–23 May 2012; IEEE: Piscataway, NJ, USA, 2012; pp. 95–109.
33. Zhou, Y.; Jiang, X. Android Malware. In *SpringerBriefs in Computer Science*; Springer: Berlin/Heidelberg, Germany, 2013.
34. Medvet, E.; Mercaldo, F. Exploring the usage of Topic Modeling for Android Malware Static Analysis. In Proceedings of the 2016 11th International Conference on Availability, Reliability and Security (ARES), Salzburg, Austria, 31 August–2 September 2016; IEEE: Piscataway, NJ, USA, 2016; pp. 609–617.
35. Li, Y.; Jang, J.; Hu, X.; Ou, X. Android malware clustering through malicious payload mining. In Proceedings of the Research in Attacks, Intrusions, and Defenses: 20th International Symposium, RAID 2017, Atlanta, GA, USA, 18–20 September 2017; Proceedings; Springer: Berlin/Heidelberg, Germany, 2017; pp. 192–214.
36. Bhargava, N.; Sharma, G.; Bhargava, R.; Mathuria, M. Decision tree analysis on j48 algorithm for data mining. *Proc. Int. J. Adv. Res. Comput. Sci. Softw. Eng.* **2013**, *3*.
37. Xue, H.; Yang, Q.; Chen, S. SVM: Support vector machines. In *The Top Ten Algorithms in Data Mining*; Chapman and Hall/CRC: Boca Raton, FL, USA, 2009; pp. 51–74.

38. Liu, Y.; Wang, Y.; Zhang, J. New machine learning algorithm: Random forest. In Proceedings of the Information Computing and Applications: Third International Conference, ICICA 2012, Chengde, China, 14–16 September 2012; Proceedings 3; Springer: Berlin/Heidelberg, Germany, 2012; pp. 246–252.
39. Rish, I. An empirical study of the naive Bayes classifier. In Proceedings of the IJCAI 2001 Workshop on Empirical Methods in Artificial Intelligence, Seattle, WA, USA, 4 August 2001; Volume 3, pp. 41–46.
40. Jordaney, R.; Sharad, K.; Dash, S.K.; Wang, Z.; Papini, D.; Nouretdinov, I.; Cavallaro, L. Transcend: Detecting concept drift in malware classification models. In Proceedings of the 26th USENIX Security Symposium (USENIX Security 17), Berkeley, CA, USA, 16–18 August 2017; pp. 625–642.