symmetry

MDPI

Article

AdvAndMal: Adversarial Training for Android Malware Detection and Family Classification

Chenyue Wang [1,*], Linlin Zhang [2,*], Kai Zhao [2], Xuhui Ding [2] and Xusheng Wang [2]

[1] School of Software, Xinjiang University, Urumqi 830046, China
[2] College of Information Science and Engineering (School of Cyber Science and Engineering), Xinjiang University, Urumqi 830046, China; zhawkk@xju.edu.cn (K.Z.); dingxh@stu.xju.edu.cn (X.D.); wangxs@stu.xju.edu.cn (X.W.)
* Correspondence: chy_wang@stu.xju.edu.cn (C.W.); zllnadasha@xju.edu.cn (L.Z.)

Abstract: In recent years, Android malware has continued to evolve against detection technologies, becoming more concealed and harmful, making it difficult for existing models to resist adversarial sample attacks. At the current stage, the detection result is no longer the only criterion for evaluating the pros and cons of the model with its algorithms, it is also vital to take the model's defensive ability against adversarial samples into consideration. In this study, we propose a general framework named AdvAndMal, which consists of a two-layer network for adversarial training to generate adversarial samples and improve the effectiveness of the classifiers in Android malware detection and family classification. The adversarial sample generation layer is composed of a conditional generative adversarial network called pix2pix, which can generate malware variants to extend the classifiers' training set, and the malware classification layer is trained by RGB image visualized from the sequence of system calls. To evaluate the adversarial training effect of the framework, we propose the robustness coefficient, a symmetric interval i = [−1, 1], and conduct controlled experiments on the dataset to measure the robustness of the overall framework for the adversarial training. Experimental results on 12 families with the largest number of samples in the Drebin dataset show that the accuracy of the overall framework is increased from 0.976 to 0.989, and its robustness coefficient is increased from 0.857 to 0.917, which proves the effectiveness of the adversarial training method.

Keywords: Android malware; detection and classification; adversarial training; system call sequence; visualization; robustness coefficient

Citation: Wang, C.; Zhang, L.; Zhao, K.; Ding, X.; Wang, X. AdvAndMal: Adversarial Training for Android Malware Detection and Family Classification. *Symmetry* **2021**, *13*, 1081. https://doi.org/10.3390/sym13061081

Academic Editor: Kuo-Hui Yeh

Received: 2 May 2021
Accepted: 7 June 2021
Published: 17 June 2021

1. Introduction

Mobile technology development and the diversity of smartphone application services have been shifting people's work and life scenes gradually from PC to mobile devices, and Android maintained its position as the leading mobile operating system(OS) worldwide, controlling the mobile OS market with a 71.93% share in January 2021 [1]. The openness and large market shares of the Android system platform make it popular among attackers, according to Skybox Security statistics, compared to the first half of 2019, vulnerabilities on mobile OSs increased by 50% within the same period of 2020, driven solely by the upsurge in Android flaws nearly 104% [2]. Based on the Kaspersky report, the number of new Android malware in 2020 grew by 2.1 million, breaking the established trend of continuous decline [3].

Deep learning has been gradually applied to the field of cyberspace security represented by malware detection and intrusion detection Since 2015 [4]. Compared with traditional machine learning methods, deep learning can fit more complex data distribution, and the outcomes have been significantly improved. In recent years, with the rapid escalation of network technology, emails, personal websites, cloud network platforms, etc, have all become the media for spreading Android malware, and the and concealment of the network have greatly increased the destructiveness of Android malware. Malware

developers use obfuscation methods to hide malicious behaviors of applications and generate malware variants that can bypass the security review mechanism, which can destroy the integrity and usability of the detection model, and lead to misjudgments and missed judgments in the detection results.

To explore the generation and attack methods of Android malware variants, researchers usually use black-box attack methods [5–9] to generate adversarial samples attack models based on features extracted from malware. However, it is unclear whether the adversarial samples generated by those (not all) techniques can be used as malware to implement attacks on models, because mutations based on malware characteristics may destroy the inherent behavioral attributes and functional structure of the malware, resulting in malicious payload crash or function abnormal. Therefore, in order to ensure the integrity and availability of the adversarial samples for the attack, the methods such as program code transplantation [10] and injection of system API calls [11] are used to construct new Android malware variants.

Researches limited to attack methods couldn't solve the problem that malware can evade neural network model detection through adversarial attacks. In response to this situation, there existed mainly five defense measures, consisting of defensive distillation [12], pre-processing [13], regularization [14], adversarial training [15,16], and rejection of adversarial samples [17]. Adversarial training is said the best solution to resist adversarial attacks from all of them, which utilizes the well-trained model to generate virtual adversarial samples, then adds them to the training set so as to retrain the model, and repeats this process until all possible virtual adversarial samples are obtained. So that the model is trained to the best state, and fundamentally enhance the robustness of the target classifier [18].

Given the increasing number of Android malware variants and the network threats it brings, enhancing the ability to recognize Android malicious variants and improving the robustness of the detection model are the key to improving the overall detection level of Android malware at the current stage. In this paper, We utilize adversarial training techniques and adversarial samples to improve the detection and classification effects of the classifier, and our contributions of this paper as follows:

- A feature database for system APIs. The database is a basis for the next feature sequence analysis and visualization, including a total of 50,998 methods in 5858 classes of Android API from level 1 to 30, and with a fixed color value setting for each system method.
- A visualization method for API call sequence. We choose the depth-first search (DFS) algorithm to substitute all internal function call methods in the application with a system-level API call sequence one by one. In order to generate adversarial samples more efficiently, we visualized the sequence feature by means of matching the fixed color value from the feature database, in the way to convert the system call sequence into an RGB image.
- A general framework named AdvAndMal for adversarial training. AdvAndMal is proposed to expand and balance the training set of the classifier, and increase the distribution of malicious samples in the model feature space, so that the boundary between benign and malicious applications or malware families in the feature space is adjusted, which increases the attack cost of real malware variants. The virtual malware variants are generated by pix2pix according to adversarial training, and we utilize real data set to train the independent model parameters for malware detection and family classification as two control groups, then with the same ways to train experimental groups on the mixed data set.
- A robustness evaluation method for the overall framework. we utilize the independent experiment results to construct the overall evaluation path architecture, and obtain the robustness expectation of the framework on the test set, then calculate the average value as an indicator to evaluate the robustness of the model.

The remainder of this paper is organized as follows. In Section 2, we introduce the related work about adversarial training on the algorithm and model structure. Section 3

thoroughly addresses the proposed framework and the adversarial training process on it. In Section 4, we give the experimental results with analysis. Finally, we summarize the work and propose the future research directions in Section 5.

2. Related Work

2.1. Android Malware Behaviour Detection

The structure of the Android application package (Apk) is complex because of the different file types and elements, and researches on Android malware classification (detection can be considered as a two-class classification) mainly focuses on feature representation and model optimization. The main categories are shown in Figure 1. Reasonable characterization methods can economize run time and calculation costs, and help improve the efficiency and accuracy of classification. Model optimization can make algorithms and network parameters more adapted to sample features, which can also achieve the purpose of improving effects.

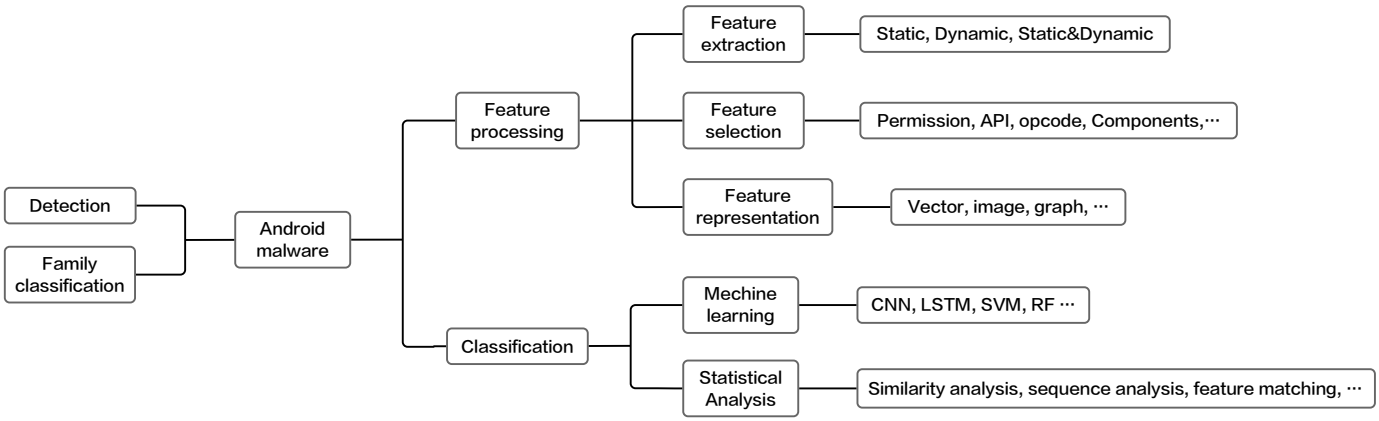

Figure 1. Android malware classification scheme.

At present, most studies focus on the application behavior characteristics, and API is one of the most frequently used features. The execution sequence of API calls [19,20] helps to fully understand the activity and thread information of the target malware application. Omid et al. [21] researched the similarity behaviors of different malware families based on sensitive APIs for classification. Tao et al. [22] analyzed the relevance of sensitive APIs protected by permissions and mined malware patterns from known malware samples to effectively identify malware.

The feature representation methods can transform the task of malware detection into image classification, and use advanced techniques and models in the image field to detect and analyze the malicious behaviors of the application. The characteristic image representation types of malware mainly include RGB color image [23], binary(black and white) image [24] and gray-scale image [25].

The language models help to analyze the behavior sequence of the application and mine the malicious behavior model. Jiang et al. [26] analyzed the specific semantic information of the malicious code, then constructed and transformed sequences of sensitive opcodes into semantically related vectors to achieve family classification. The behavior sequence can also be converted into the form of a dependence graph [27,28] or flow graph [29,30] for malware detection. For example, Xu et al. [30] extracted n-gram sequences from the feature flow diagram for model training, and selected the top-k sequence as the graph of the target family.

In order to enhance the robustness of the model algorithm, Android malware detection and classification models are developed towards integration and bagging. Sercan et al. [31] proposed a static Android malware family classification framework, which can use a large number of machine learning classification methods (support vector machine, decision tree, logistic regression, K-nearest neighbor, random forest, AdaBoost and multi-layer

Perceptron classifier) classify unknown families of malware. Calleja et al. [32] proposed an extension classifier named Revealdroid*, the enhanced version of RevealDroid, which employs a pool of C4.5 trees instead of relying on just one instance, and the final label assigned to sample results from majority voting.

2.2. Adversarial Attacks

2.2.1. Adversarial Algorithms

Szegedy et al. [15] first proposed the concept of adversarial samples. They discovered the weaknesses of neural network models in the image classification field and believed that the nonlinear expression ability and overfitting of deep neural networks were the possible reasons. They used the limited-memory BFGS (L-BFGS) algorithm, adding a small number of perturbations into the images to generate adversarial samples that humans cannot distinguish while the model would misclassify. Goodfellow et al. [33,34] believe that the linear behavior of deep neural networks in high-dimensional spaces is the root cause of adversarial samples. They proposed the Fast Gradient Sign Method (FGSM) algorithm and generative adversarial nets (GAN) based on the FGSM algorithm, which is composed of a generator and a discriminator. Through the game between the two models, the generated samples are getting closer to the distribution of real samples. Papernot et al. [35] proposed a method to generate adversarial samples based on the forward derivatives called Jacobian-based Saliency Map Attack(JSMA), which can reduce the search space of adversarial examples, and few modifications on original samples can be misclassified.

2.2.2. Adversarial Malware Samples

Biggio et al. [36] started early on hiding the adversarial samples by attacking the clustering algorithm, in a way that destroyed the training data distribution without changing the clustering results of other samples by the means of injecting the benign samples with poisoning behaviors. Calleja et al. [32] proposed a novel classification evasion attack model named IagoDroid, and the variants generated with evolutionary algorithms against any triage process where the family classification relies on static analysis. they tested the method on RevealDroid using 29 different malware families from the Drebin, and IagoDroid successfully forces misclassification of 28 of the 29 families.

Grosse et al. [8] generated adversarial malware samples against the RNN classification model based on the FGSM and JSMA algorithms, and the attack success rate was up to 63%. Hu et al. [6] proposed MalGAN based on GAN, utilized the generated adversarial samples labeled by the target black-box detector as the alternative detector's training samples, to learn the parameters of the target black-box detector and generate adversarial samples that can evade the target detector. Kawai et al. [7] improved the MalGAN and the data set by separating the detection model from MalGAN, and using different training data sets to create API lists separately for MalGAN and detector.

2.3. Adversarial Training

Adversarial training is an important way to enhance the robustness of neural networks, widely used in image processing [37–39] and computer vision [40,41], as well as other fields such as medical imaging [42,43], speech recognition [44,45] and cyber security [46,47]. Data augmentation by adversarial training techniques can effectively resist adversarial attacks on detection models, by adding few small disturbances (may cause misclassification) into training samples to robust the classifies.

Singh et al. [48] proposed a GAN training method, which embeds prior knowledge about the data set into the model algorithm, that evaluates training stability and adversarial samples' quality with different indicators. The malware detection results are more substantial than the benchmark. Chen et al. [49] used GAN for data augmentation for the categories with relatively few samples. They verified on two datasets of AMD and DREBIN, and the difference in F1-score accuracy between the two can reach 5∼20% compared with that before amplification. Sen et al. [50] used evolutionary computation technologies to

generate new variants that can successfully evade the detection system of mobile malware under static analysis, and provided the system with a better security solution to identify these variants.

3. Methodology

3.1. Feature Processing

Since the system method will not be modified by code obfuscation, and all functions of the Android application can be implemented by system API sequences. Thus, the system method becomes an ideal feature that can reflect the actual intent of the application. However, the API calls of Android applications include not only system calls (Android Platform APIs), but also internal methods (defined by the application developer) and external interfaces (third-party service interfaces). The differences in naming rules and obfuscation techniques of malicious code destroy the uniformity and uniqueness of feature representation, which is not conducive for batch analysis of applications and comparative analysis between applications.

Therefore, we utilize the system API sequences to express the behavior intention of the applications. Instead of directly filtering the internal methods and external interfaces out from the API call sequence, we replace them with the callback results (system API sequences) according to the traversal order of the DFS algorithm, which ensures the integrity of the behavioral logic, and finally visualize the system API sequences to an RGB image through color values matching from the constructed feature database. The entail process is given by the following Figure 2.

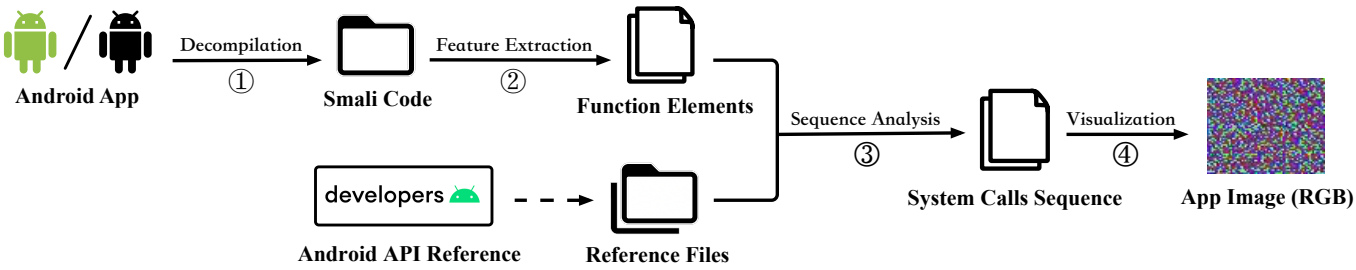

Figure 2. Android application feature processing.

3.1.1. Decompilation and Feature Extraction

The Android Apk is directly converted into the smali file package through the decompilation tool called Apktool, and then performs feature extraction of the API call sequences. Figure 3 shows how to traverse all the Smali files in turn, and find the object class, super class, interface and API calls in terms of the identifiers about ".class", ".super", ".implements" and ".invoke" in every *.smali file, in such a way as to extract the function elements from all internal methods of an application.

3.1.2. Sequence Analysis and Visualization

Sequence analysis is to replace all internal methods nested in the original API call sequences with the corresponding system API sequences in terms of DFS traversal order, and finally return the matched color value of the method from the system feature database shown as Figure 4.

In order to simplify the matching process, this paper only retains the method name that can reflect the functional intention when constructing the system feature database, and assigns a unique color value (three-digit hexadecimal number) to each remaining method after deduplication.

Due to the huge number of external interfaces, this paper does not establish a special feature database for third-party interfaces, just replaces them with a special color value targeted for all external interfaces. The whole process is shown in Figure 5.

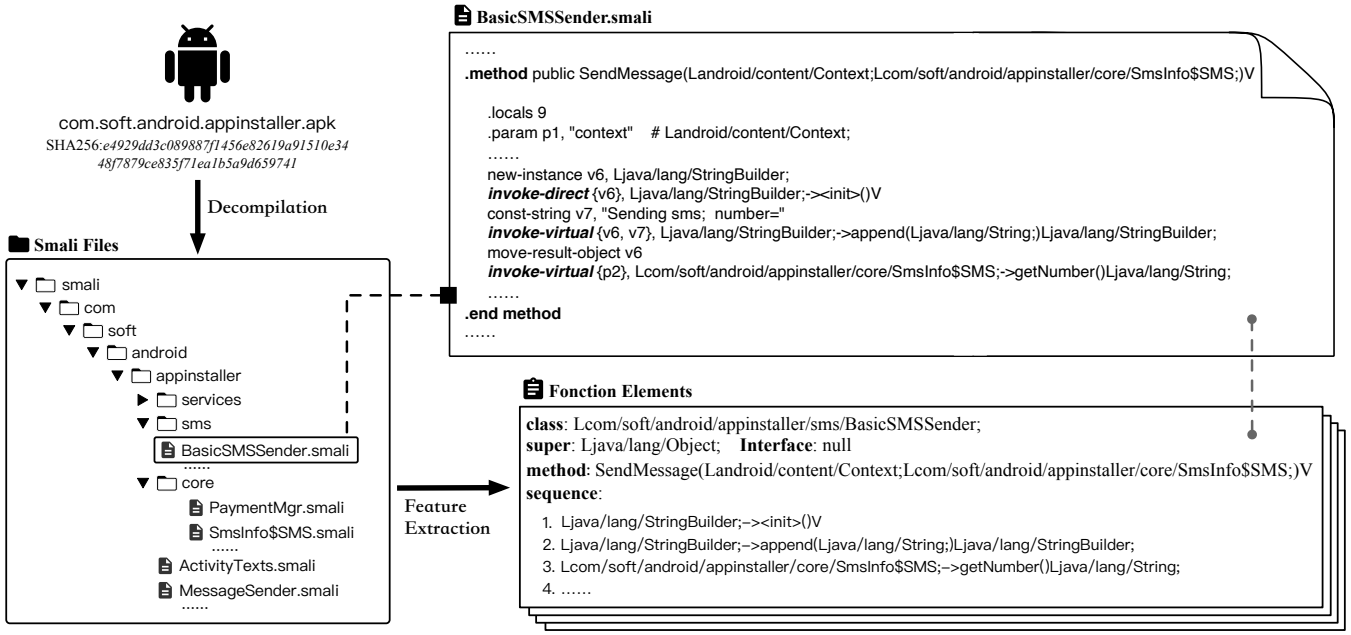

Figure 3. Decompilation and feature extraction.

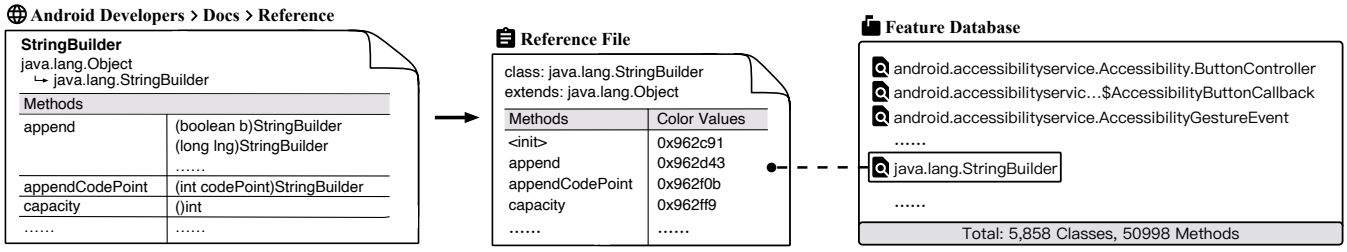

Figure 4. Feature database for system APIs.

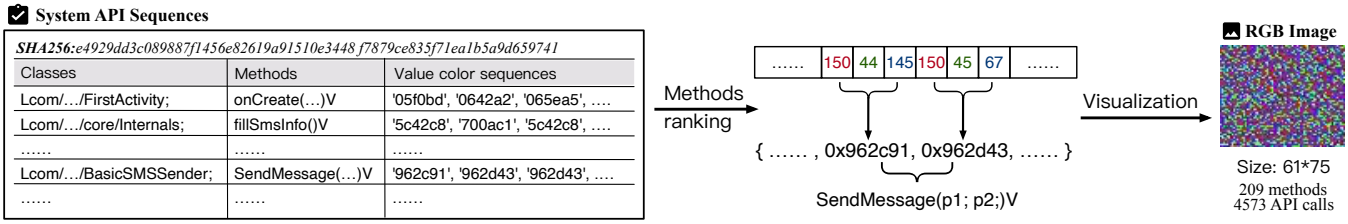

Figure 5. Sequence analysis and visualization.

3.2. Adversarial Training Framework

The structure of the Android malware adversarial training framework consists of a two-layer network. The first layer is pix2pix, generating an extended training set of adversarial samples with malicious attributes. The second layer is LeNet-5, charging Android malware detection and family classification. The two-layer network of the adversarial training framework is replaceable, and the effective adversarial example sets are screened based on the retraining results on the mixed data set. In this paper, we choose benign and malicious Android samples as the real x and condition y, and the training process is shown in Figure 6.

- STEP 1: Switch ①②③ off, train the classifier L on the original training set, and gain the validation results as the baseline;
- STEP 2: Switch ① off, ②③ on, train pix2pix;
- STEP 3: Switch ① on, ②③ off, generate and verify adversarial samples. retrain L with the mixed training set, and compare the new validation result with the baseline, the loss is passed to the generator for retraining;

- STEP 4: Repeat steps 2 and 3 until the test result of the verification set is higher than the baseline and tends to be stable, then test on the test set.

Adversarial training is used to find the mappings that are prone to deviation and misclassification in the feature space of the classifier with the help of virtual adversarial samples, and retrain the classifier with the training set added those adversarial samples to increase the probability distribution of malware in the sample space. The premise of real malware variants attacking the classifier through obfuscation technology is not to change the inherent malicious behavior, thus the model trained by the adversarial training method can increase the attack cost of adversarial samples and enhance the model robustness.

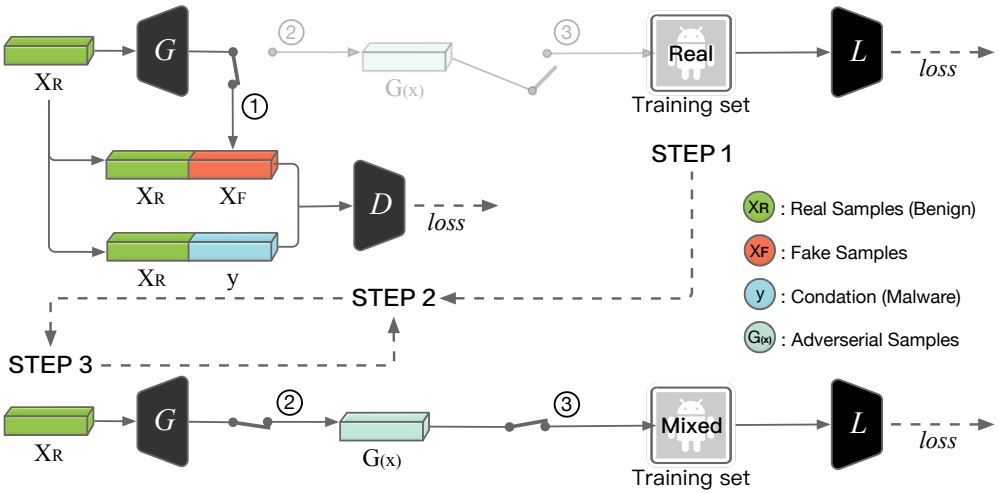

Figure 6. Adversarial training process.

3.2.1. Adversarial Sample Generation Layer

The original GAN [33] is an implicit generative model, and it unable to explicitly estimate the probability density with training samples randomly sampled from the data distribution of the collection. Due to the difficulty of sampling in high-dimensional space, GAN cannot perform point-to-point image conversion, so that the final sampling result is not comprehensive, and the target type output cannot be generated according to the input. In response to the above problems, the improved model conditional generation adversarial nets (CGAN) [51] added condition y into the input layer of generator and discriminator, and defined the learning mapping $G : \{z, y\} \rightarrow x$, changing the network from unsupervised to semi-supervised. CGAN can guide the generation process of the generator and specify the generation types of the adversarial samples. The objective function of CGAN is as Formula (1).

$$\begin{aligned} \min_G \max_D V(G, D) = & \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}(\mathbf{x})}[\log D(\mathbf{x}|\mathbf{y})] \\ & + \mathbb{E}_{\mathbf{z} \sim p_{\mathbf{z}}(\mathbf{z})}[\log(1 - D(G(\mathbf{z}|\mathbf{y})))] \end{aligned} \tag{1}$$

In the original CGAN, the generator not only has to fool the discriminator, but also generates real images as much as possible. The discriminator just needs to judge the authenticity of the images, and the problem of pattern collapse has not been obviously solved. The feature images in this paper are of different sizes, and the generated adversarial samples need to be more similar to condition y (malware). Therefore, we choose pix2pix [37] as the adversarial sample generation layer, a general image conversion framework which defines the learning mapping of CGAN is from the real sample x and random noise z to the condition y, as $G : \{x, z\} \rightarrow y$, so that the redefined objective function by pix2pix is as Formula (2):

$$\begin{aligned} \mathcal{L}_{cGAN}(G, D) = & \mathbb{E}_{x,y}[\log D(x, y)] \\ & + \mathbb{E}_{x,z}[\log(1 - D(x, G(x, z))]. \end{aligned} \tag{2}$$

On this basis, pix2pix improved the loss function by adding the L1 constraint to its objective function. Based on Formulas (2) and (3), the final loss function of pix2pix is calculated as Formula (4).

$$\mathcal{L}_{L1}(G) = \mathbb{E}_{x,y,z}[\|y - G(x,z)\|_1] \tag{3}$$

$$G^* = \arg\min_G \max_D \mathcal{L}_{cGAN}(G,D) + \lambda \mathcal{L}_{L1}(G). \tag{4}$$

The comparison of the original CGAN and pix2pix is shown in Figure 7. In the figure, pix2pix uses the $L1$ constraint to make the guiding effect of the condition y implicit, and the λ in Formula (4) used to adjust the similarity with condition y. In addition, pix2pix uses the dropout function to replace the noise z. If the λ is set to 0, the effect of pix2pix is the same as CGAN.

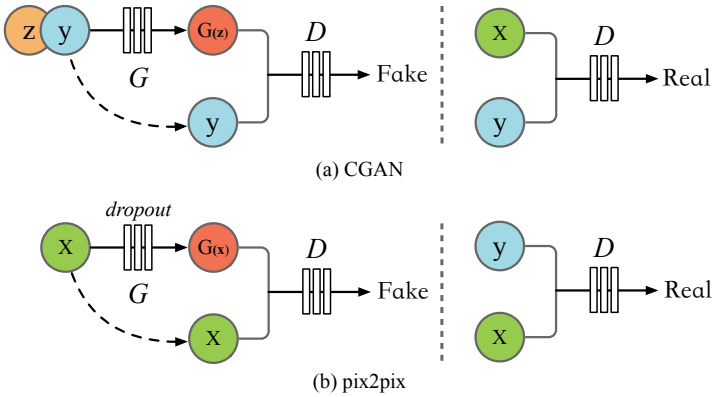

Figure 7. Adversarial training process.

In many tasks related to image conversion, the underlying features of the image (such as color, contour, etc.) are equally important. For example, in this paper, the color value and position of the pixel in the feature image of the Android application are closely related to the system API and its call sequence. Changing the key position and color of the pixel will destroy the behavioral intention expressed in the original image, and make the generated sample image inconsistent with the attributes of the condition.

Pix2pix replaces the original generator with U-Net [52], and utilizes the symmetry of the model to directly map the low-level information of the image to the high-level features, and conducts end-to-end training to preserve the entail information of the image to the greatest extent. Under normal circumstances, the discriminator makes judgments based on the entire image, while the pix2pix uses the PatchGAN to divide the feature image into N×N patches, and then judge the authenticity of each patch in turn. The patch result of the entire image calculates the average value as the output result of the discriminator, which is suitable for images of any size.

3.2.2. Malware Classification Layer

In this paper, the classifier is constructed according to the structure of LeNet-5. Compared with the fully-connected neural network, LeNet-5 [53] extracts image features through convolution calculation, parameter sharing, pooling, etc., and uses the fully-connected layer only for classification, which reduces time and computational overhead. In order to improve the accuracy of model prediction and speed up the convergence, it is necessary to standardize processing in the input layer. The specific network and training parameters are shown in Table 1.

When the convolution kernel size in the C1 layer is set to 8, the optimal effect is obtained on the data set. The network parameters of the classifiers are trained separately for malware detection and family classification, a total of four models in two categories were trained independently based on controlled experiments, so as to detect and classify Android malware.

Table 1. LeNet-5 model parameters.

Layer	Feature Map	Size	Kernal Size	Stride	Activation	Paras
Input (Image)	32×32	1	-	-	-	-
C1/Convolution	28×28	8	5×5	1	tanh	608
S2/Pooling	14×14	8	2×2	2	-	0
C3/Convolution	10×10	16	5×5	1	tanh	3216
S4/Pooling	5×5	16	2×2	2	-	0
C5/Convolution	1×1	120	5×5	1	tanh	48,120
F6/Full connection	84	-	-	-	relu	10,164
Output_Detection	2	-	-	-	softmax	170
Output_Classification	12	-	-	-	softmax	765
EPOCHS = 150	Batch_Size = 32		INIT_LR = 0.001	decay = INIT_LR/EPOCHS		

4. Experimental Results

4.1. Datasets

The data sets used in the experiments including 5831 benign and 5546 malicious Android applications. The benign applications are downloaded from AndroZoo [54], a big Android apps database that provides a comma-separated values (CSV) file about the information of all apps, and the format is shown in Table 2. AndroZoo supports accurate search based on the "*sha*256, *sha*1, *md*5, *pkg_name*" fields and filtered search with limited field ranges to download applications. We set "$vt\_detection = 0$" and "$markets = play.google.com$" to filter out pure applications and construct the benign data set.

Table 2. AndroZoo CSV file format.

Field	Explanation
sha256	Signature hash
sha1	Signature hash
md5	Signature hash
dex_date	The date attached to the dex file inside the zip
apk_size	The size of the apk
pkg_name	The name of the android Package
vercode	The version code
vt_detection	The number of AV from VirusTotal that detected the apk as a malware
vt_scan_date	Scan date of the VirusTotal scan reports
dex_size	The size of the classes.dex file
markets	Source (App store)

The malware samples come from the Drebin [55], which contains 5560 Android malware from 179 families. This paper successfully converted 5546 malicious samples to RGB images, and the statistics of family numbers are shown in Table 3.

Table 3. Malware family sample set statistics table.

No.	Family	Number	No.	Family	Number
1	FakeInstaller	925	8	Kmin	147
2	DroidKungFu	665	9	FakeDoc	131
3	Plankton	625	10	Adrd	91
4	Opfake	613	11	Geinimi	90
5	GinMaster	339	12	DroidDream	81
6	BaseBridge	327	-	others	1360
7	Iconosys	152	total	-	5546

The malware data set is divided in terms of family, and the ratio of the training set (including validation set) and test sets is 8:2, and the classification experiment is carried out on the data sets of No. 1 to No. 12.

4.2. Robustness Evaluation Method

The evaluation criteria of classifier robustness are not uniform. This paper proposes the "Independent experiment + Overall evaluation" method, which builds an overall evaluation standard based on the detection and classification results of the classifier in the independent experiment.

4.2.1. Independent Experiment Evaluation

Construct the confusion matrix according to the malware detection and family classification experiment of the classifiers, and the general confusion matrix framework is shown in Table 4. The row of C_i represents the predicted distribution of the actual category i, and the value corresponding to $C_{i,j}$ represents the number of samples in the test set of class i that were judged (i = j) or misjudged (i≠ j) as class j by the classifier. The model evaluation indicators and formulas of malware detection (n = 2) and family classification (n = the number of families to be classified, 12) are separately shown in Table 5.

Table 4. General confusion matrix.

Detection: n = 2 Classification: n = 12		**Predicted**			
		Class 1	**Class 2**	**.....**	**Class n**
Actual	**Class 1**	$C_{1,1}$	$C_{1,2}$	.....	$C_{1,n}$
	Class 2	$C_{2,1}$	$C_{2,2}$	.....	$C_{2,n}$
	.....	.....	.....	$C_{i,j}$	.....
	Class n	$C_{n,1}$	$C_{n,2}$	.....	$C_{n,n}$

Table 5. Evaluation indicators and formulas.

Domain	**Indicator**	**Formula**	
Detection & Classification	Accuracy	$\frac{\sum_{i=1}^{n} C_{i,i}}{\sum_{i,j=1}^{n} C_{i,j}}$	(5)
	Precision $_i$	$\frac{C_{i,i}}{\sum_{j=1}^{n} C_{j,i}}$	(6)
	Recall $_i$	$\frac{C_{i,i}}{\sum_{j=1}^{n} C_{i,j}}$	(7)
	F-score $_i$	$\frac{2 * \text{ Precision }_i * \text{ Recall }_i}{\text{Precision }_i + \text{ Recall }_i}$	(8)
Classification	Aver-Precision	$\frac{\sum_{i=1}^{n} \text{ Precision }_i}{n}$	(9)
	Aver-Recall	$\frac{\sum_{i=1}^{n} \text{ Recall }_i}{n}$	(10)
	Aver-Fscore	$\frac{\sum_{i=1}^{n} \text{ F-score }_i}{n}$	(11)

4.2.2. Overall Framework Evaluation

The overall evaluation method refines the classification results of the model framework and evaluates the model in combination with the actual needs of the Android malware detection field, as shown in Figure 8. Construct a "detection-classification" confusion matrix based on the overall evaluation framework.

Based on the test results of independent experiments and the overall evaluation path framework shown in Figure 8. The evaluation confusion matrix of "detection-classification", as is shown in Table 6 is constructed. From the perspective of model robustness, the more applications in path ① and ④, the more robust the classifiers are. Path ③ and ⑥ offset the interferences caused by detection and family classification, but they can play a warning role and contribute to the overall assessment. The applications in path ② and ⑤ can cause interference and confusion to the classifier, weakening the robustness of the model.

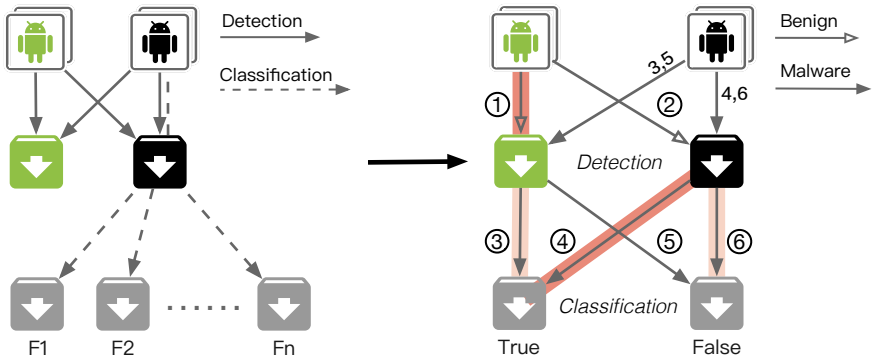

Figure 8. Robustness evaluation framework.

Table 6. Evaluation confusion matrix.

	T	F	Category	Explanation	Path
T	**TT**	**FT**	TT_B, TT_M	True Detection, True Classification	①, ④
			FT	False Detection , True Classification	③
F	**TF**	**FF**	TF	True Detection , False Classification	⑥
			FF_B, FF_M	False Detection , False Classification	②, ⑤

TT, FT, TF, FF respectively correspond to the samples detected or classified from different paths, and the accuracy of the overall adversarial training framework is calculated according to Formula (12).

$$Overall\text{-}Acc = \frac{TT \cup FT \cup TF}{TT + FT + TF + FF} \tag{12}$$

Acc_D and Acc_C represent detection accuracy and classification accuracy, and the joint probability density of the path is calculated by the accuracy of the detection and classification classifiers. The weighted summation represents the robustness expected value (Robustness Value) of the overall evaluation framework on the test set. The calculation process is shown as follows:

$$R_{TT} = Acc * TT_B + Acc * F - Acc * TT_M \tag{13}$$

$$R_{FT} = (1 - Acc) * F - Acc * FT \tag{14}$$

$$R_{TF} = Acc * (1 - F - Acc) * TF \tag{15}$$

$$R_{FF} = (1 - Acc) * FF_B + (1 - Acc) * (1 - F - Acc) * FF_M \tag{16}$$

$$\text{Robustness-Value } = R_{TT} + R_{FT} + R_{TF} - R_{FF} \tag{17}$$

Generally, for a model with good classification results, the more samples in the test set, the greater the Robustness Value. In order to facilitate the comparison between different sample sets and models, the mean value of robustness expectation is calculated as the robustness coefficient (RC value for short) of the model. The formula is as follows:

$$\text{Robustness-Coefficient } = \frac{\text{Robustness-Value}}{TT + FT + TF + FF} \tag{18}$$

The range of RC value is a symmetrical interval i = [−1, 1], and the value is positively correlated with model robustness. When the RC value is equal to 1, it means that the detection and classification accuracy of the model are both equal to 1. When the value is −1, the detection and classification accuracy of the model are both 0, and the same model may take different RC values on different test sets.

4.3. Results and Analysis

4.3.1. Android Malware Feature Image

Take the three malware families—FakeInstaller(925), Adrd(91) and AccuTrack(10) as the representatives corresponding to the different intervals for family set sizes. The visualized feature images show obvious family characteristics, and the main behavioral characteristics are shown in Figure 9.

Figure 9. Visualization of malware family behavior. (**a**) Representation for the families, which samples' number belong to (100, $+\infty$). (**b**) Representation for the families, which samples' number belong to (10, 100). (**c**) Representation for the families, which samples' number belong to (1, 10).

The degree of behavioral similarity for samples within a family is very high, and the larger size of a family sample set, the more forms of malicious behavior that the set has. For example, the family set (**a**) with 925 samples and 8 main characteristics, and (**b**) with 91 samples and 4 main characteristics. Although (**c**) with a smaller sample set with just 10 samples, the characteristic from its feature images is more concentrated.

4.3.2. Adversarial Training

In order to generate adversarial samples with malicious attributes by using pix2pix, the condition y and the real sample x should be matched in color and size. The λ coefficient in Formula (3) is set 50, and the change of loss values during training process on the generator and the discriminator are shown in Figure 10.

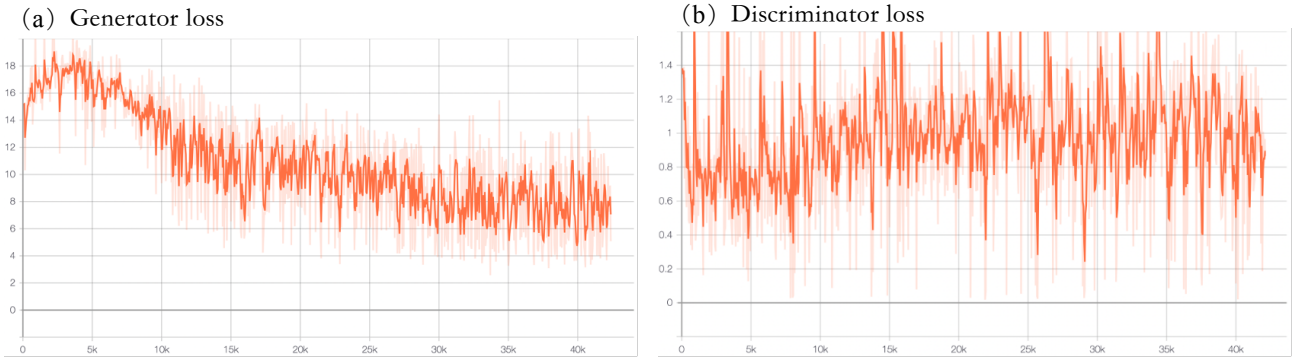

Figure 10. Adversarial training loss. (**a**) The loss function curve of generator model during training process. (**b**) The loss function curve of discriminator model during training process.

In the later stage of training, the discriminator loss fluctuates between 0.6 and 1, and the generator loss fluctuates around 8, which basically reaches a balanced state. Under the circumstances, the generated adversarial sample images are shown in Figure 11.

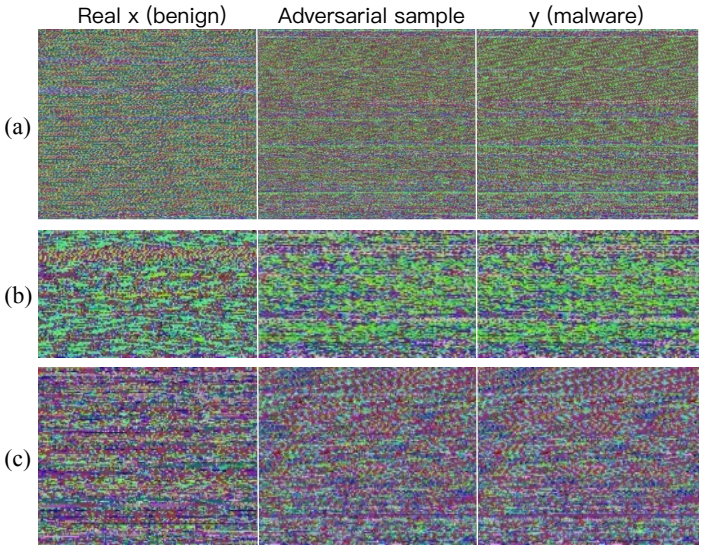

Figure 11. Visualization of Adversarial samples. (**a**–**c**) are three examples for input pairs (real sample x and condition y) and their corresponding output adversarial samples.

4.3.3. Malware Detection

Figure 12 shows the accuracy and loss function curves of the two-class classifier used for malware detection on the training and validation sample sets. The two curves in subgraphs respectively represent the training process of the control group (CG for short)

with the original real training sets, and the experimental group (EG for short) correspond to the mixed training set with generated adversarial samples added into it. It can be seen from (c) and (d) that the model trained with adversarial samples has a higher accuracy rate and a lower loss value, and the result comparison on the test set is shown in Figure 13.

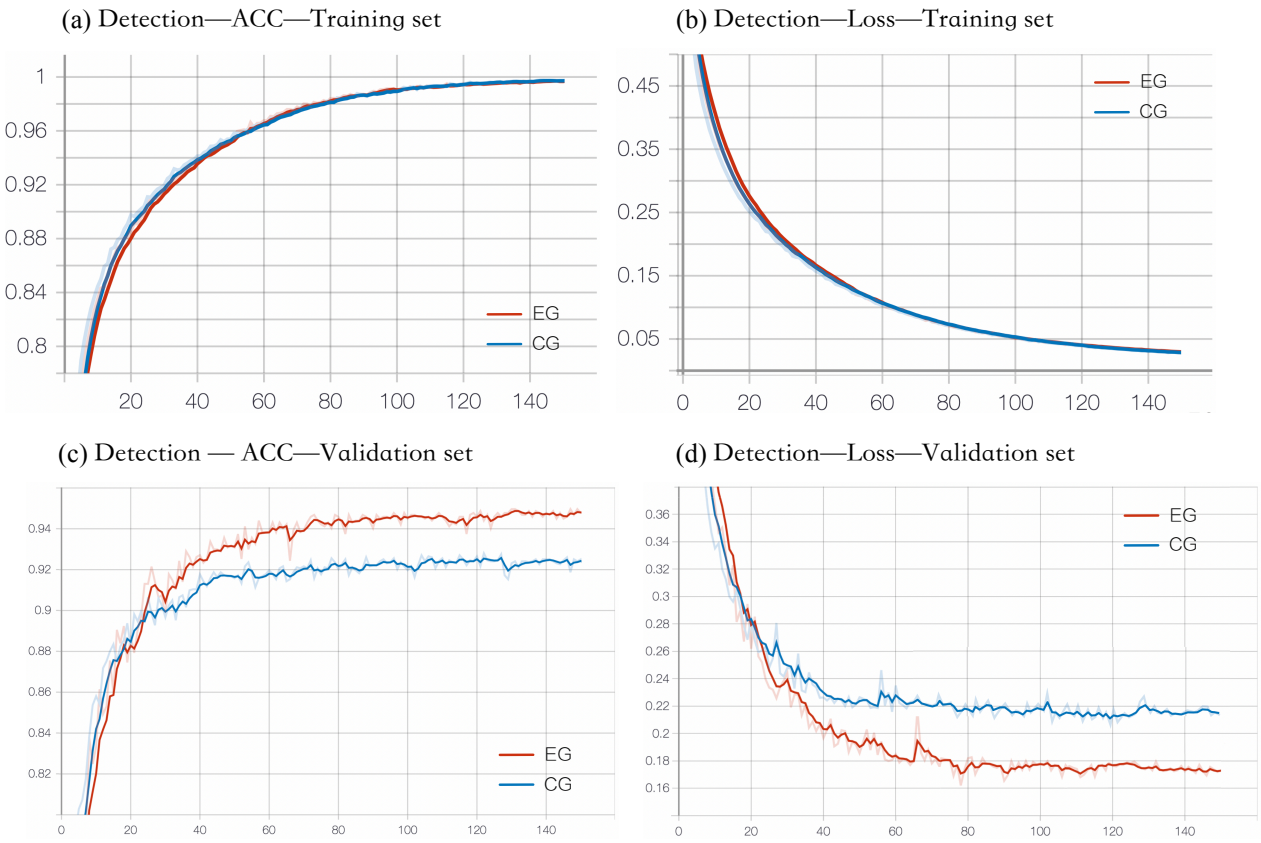

Figure 12. Detection model training. (**a**) The accuracy function curve of detection model on training sets. (**b**) The loss function curve of detection model on training sets. (**c**) The accuracy function curve of detection model on validation sets. (**d**) The loss function curve of detection model on validation sets.

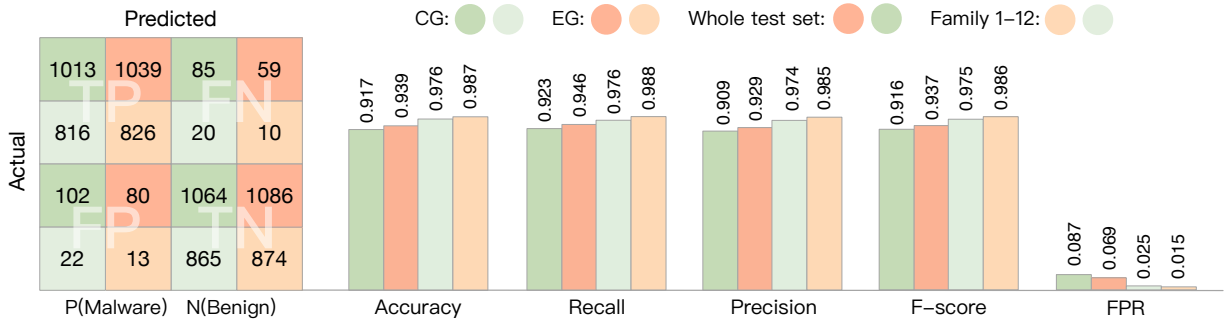

Figure 13. Detection result comparison.

For malware detection, it is necessary to increase the recall and accuracy values and reduce the rate of false positives rate (FPR). In order to evaluate the effect of the adversarial training on framework, 887 benign samples selected randomly as well as 836 malicious samples from families No.1 to No.12 formed a test subset according to the same ratio as the original test set. It can be found from Figure 13 that the indicators of test results on the subset are significantly better than those on the whole test set, from which can derive that data augmentation is beneficial to model training.

4.3.4. Family Classification

Compared with the two-class classification model of malware detection, the adversarial training process of family classification is more complicated. As shown in the EG(a) in Figure 14, the adversarial sample blurs the boundaries of the family, resulting in the classification result not rising but falling compared to the CG.

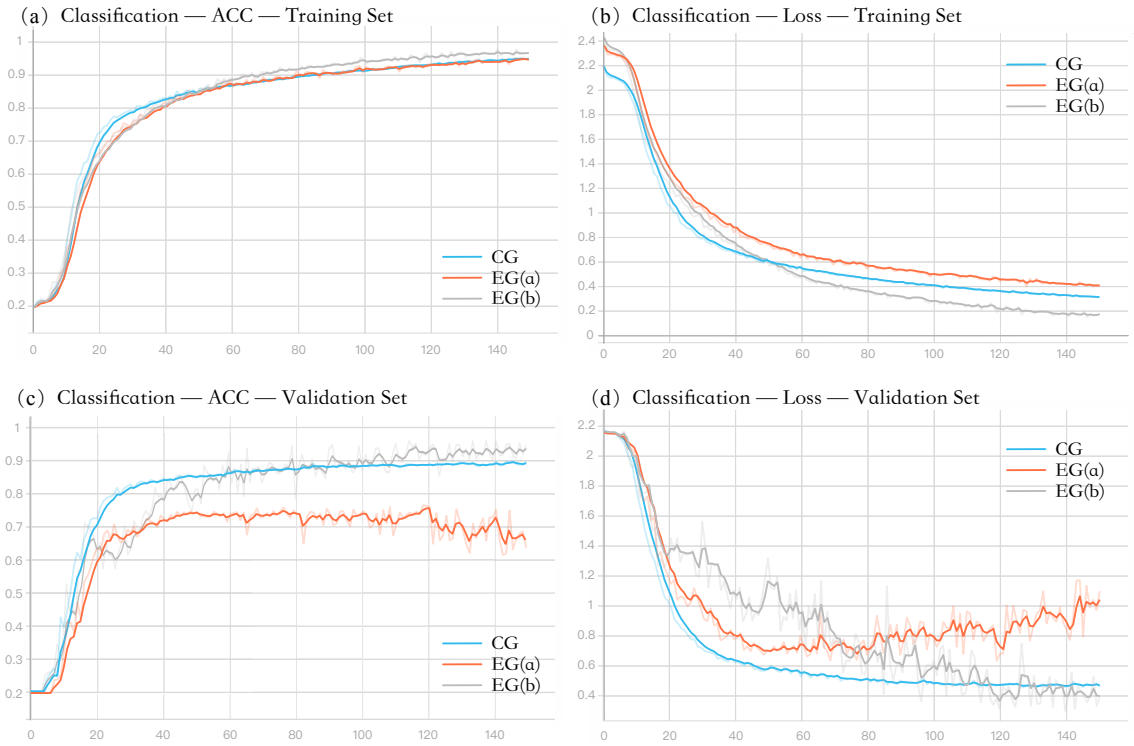

Figure 14. Classification model training. (**a**) The accuracy function curve of classification model on training sets. (**b**) The loss function curve of classification model on training sets. (**c**) The accuracy function curve of classification model on validation sets. (**d**) The loss function curve of classification model on validation sets.

The results of the EG(b) on the validation set are slightly improved compared to the CG, and it is found that the adversarial training has the effect of preventing overfitting. The test results of classifiers with adversarial samples added to the training set are generally 0.5% higher than those on the validation set, while the test results of models without adversarial training are on the contrary. The confusion matrices are shown in Figure 15.

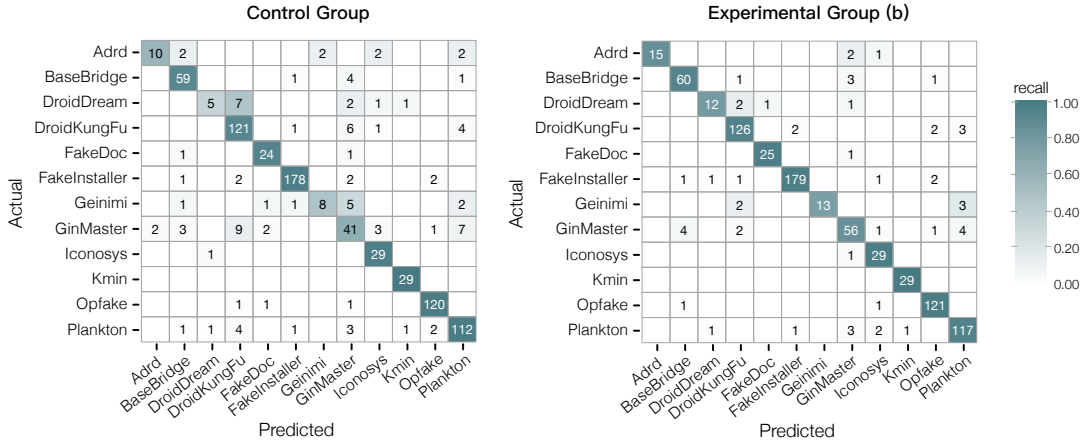

Figure 15. Confusion matrix of family classification for adversarial training.

As can be seen from the confusion matrix of the CG that the classifier cannot effectively learn the features of family Adrd, DroidDream, Geinimi, and GinMaster because of the small sample sizes, thus leading to false positives and false negatives. The results from EG(b) show that data augmentation through adversarial training enables the model to divide the boundaries of the family more accurately, so as to improve the classification effect. The top 12 families' classification results are shown in Table 7.

Table 7. Evaluation confusion matrix.

Family	Precision		Recall		F-Score	
	CG	EG(b)	CG	EG(b)	CG	EG(b)
Adrd	0.833	1	0.556	0.833	0.667	0.909
BaseBridge	0.868	0.909	0.908	0.923	0.868	0.909
DroidDream	0.714	0.857	0.313	0.75	0.435	0.8
DroidKungFu	0.84	0.94	0.91	0.947	0.874	0.944
FakeDoc	0.857	0.962	0.923	0.962	0.889	0.962
FakeInstaller	0.978	0.984	0.962	0.968	0.97	0.975
Geinimi	0.8	1	0.444	0.722	0.571	0.839
GinMaster	0.63	0.836	0.603	0.824	0.617	0.83
Iconosys	0.806	0.829	0.967	0.967	0.879	0.892
Kmin	0.967	0.983	1	1	0.935	0.967
Opfake	0.96	0.953	0.976	0.984	0.968	0.968
Plankton	0.875	0.921	0.896	0.936	0.885	0.929
Average	0.841	0.929	0.788	0.901	0.801	0.912
Accuary	**CG:** 0.88			**EG(b):** 0.935		

Compared with the similar work of Chen et al. [50], whose Experiment 1 used the same dataset as this paper, and the detailed comparison is shown in Table 8. By comparing the F-score values of the two experiments, it can be seen that the overall effect of the method in this paper is better.

Table 8. Adversarial training evaluation result.

Experiment	Chen et al.	This Paper
Training Model	CNN, GAN	CNN, CGAN
Dataset	10 families, Drebin	12 families, Drebin
Data Augmentation	Adrd, DroidDream, Geinimi	Families with small sizes and poor results
Evaluation Indicators	F-score, *p*-values	Accuracy, Recall, Precision, F-score
Results (F-score)	Adrd: 60.1% → 84.3% DroidDream: 73.1% → 84.2% Geinimi: 50.2% → 77%	Adrd: 66.7% → 90.9% DroidDream: 43.5% → 80% Geinimi: 57.1% → 83.9%

4.3.5. Robustness Evaluation

Construct a "Detection-Classification" confusion matrix based on the test set of family No. 1–12, which is used to evaluate the robustness of the adversarial training framework. The numerical data in the confusion matrix can be disassembled into two sets of actual paths, as shown in Figure 16, and the evaluation parameters and evaluation results of the overall framework are shown in Table 9.

Table 9. Adversarial training evaluation result.

Experiment	Accuracy			Robustness Value	R-C
	Detection	Classification	Framework		
CG	0.976	0.88	0.976	1477	0.857
EG(b)	0.987	0.935	0.989	1580.835	0.917

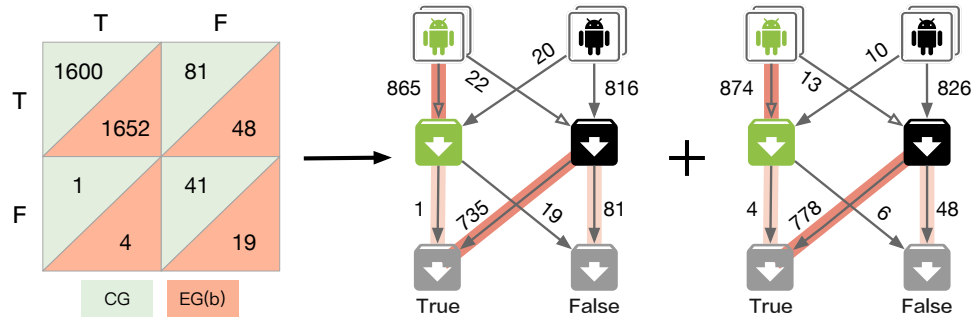

Figure 16. Robustness evaluation framework.

As is clear from the table, the robust performance of the Android malware variant detection framework has been improved after the adversarial training. The correct results of comprehensive detection and classification have also increased the accuracy of the overall framework.

From the classification path of the overall framework, it can be seen that the classifiers have significantly improved the detection effects on Android malware after the adversarial training. Although the accuracy of family classification is lower than the value of the detection model, the classification model based on behavior learning of the malware family can capture the features not learned by the detection model and help the overall framework to identify more malware.

5. Conclusions

This paper utilizes adversarial training methods to build a general model framework named AdvAndMal for the detection and family classification of Android malware by the image features of the system API sequence, and improves the accuracy of the classifier through retraining. The proposed robustness evaluation method verifies the detection and classification effects of the classifier before and after the adversarial training, and the experimental results confirm the feasibility and effectiveness of the method. There are still some problems and improvements in the work:

- The quality of the generation of adversarial samples is unstable. The adversarial samples generated by CGAN cannot be used to visually confirm whether they have malicious attributes or not, and without measuring the statistically significant differences between the different results. The generated datasets used to improve the model effect may contain useless or problematic samples, which will affect the learning effect of the model. In this stage, adversarial samples that can improve the classification effect are just screened through model training and verification in batches. In the future, we will combine the behavioral characteristics of Android malware to generate adversarial samples effectively.
- The classifier used in this article has a small number of layers and network parameters, which is easy to train, but the learning ability of the model will be limited. Moreover, the data set required for CGAN training has a one-to-one correspondence between real samples and conditions. If the paired images are too different, it will affect the effectiveness of the generated samples. Therefore, our model need to be further improved to apply into many scenarios.

Author Contributions: Conceptualization, C.W., L.Z., and K.Z.; methodology, C.W., and L.Z.; validation and visualization, X.D., X.W. and C.W.; writing-original draft preparation, C.W.; resources, L.Z., and K.Z.; supervision, L.Z., and K.Z. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported in part by the Natural Science Foundation of Xinjiang Uygur Autonomous Region under Grant 2019D01C041, Grant 2019D01C062, Grant 2019D01C205, and Grant 2020D01C028; in part by the Graduate Research Innovation Project of Xinjiang Uygur Autonomous Region under Grant XJ2020G065; in part by the National Natural Science Foundation of China under

Grant 12061071 and Grant 61867006; in part by the Higher Education of Xinjiang Uygur Autonomous Region under Grant XJEDU2017M005, Grant XJEDU2020Y003, and Grant XJEDU2019Y006; in part by the Funds for Creative Research Groups of Higher Education of Xinjiang Uygur Autonomous Region under Grant XJEDU2017T002; in part by Tianshan Innovation Team Plan Project of Xinjiang Uygur Autonomous Region under Grant 202101642; in part by Xinjiang Uygur Autonomous Region Major Science and Technology Special Project under Grant 2020A02001-1; and in part by the Major Science and Technology Project of Sichuan Science and Technology Plan Grant 2020YFQ0018.

Data Availability Statement: The data presented in this study are available in [54,55].

Acknowledgments: The authors would like to thank editors and referees for their precious remarks and comments.

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

References

1. Market Share of Mobile Operating Systems Worldwide 2012–2021. Available online: https://www.statista.com/statistics/272698/global-market-share-held-by-mobile-operating-systems-since-2009/ (accessed on 8 February 2021).
2. 2020 Vulnerability and Threat Trends (Mid-Year Update). Available online: https://lp.skyboxsecurity.com/WICD-2020-07-WW-VT-Trends_Asset.html (accessed on 22 July 2020).
3. Mobile Malware Evolution 2020. Available online: https://securelist.com/mobile-malware-evolution-2020/101029/ (accessed on 1 March 2021).
4. Zhang, Y.; Dong, Y.; Liu, C.; Lei, K.; Sun, H. Situation, Trends and Prospects of Deep Learning Applied to Cyberspace Security. *Comput. Res. Dev.* **2018**, *55*, 1117–1142.
5. Rosenberg, I.; Shabtai, A.; Rokach, L.; Elovici, Y. Generic Black-Box End-to-End Attack Against State of the Art API Call Based Malware Classifiers. Available online: https://arxiv.org/abs/1707.05970v4 (accessed on 15 February 2018)
6. Hu, W.; Tan, Y. Generating Adversarial Malware Examples for Black-Box Attacks Based on GAN. Available online: https://arxiv.org/abs/1702.05983 (accessed on 20 May 2017).
7. 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; pp. 40–45.
8. Grosse, K.; Papernot, N.; Manoharan, P.; Backes, M.; McDaniel, P. Adversarial Examples for Malware Detection. In Proceedings of the 22nd European Symposium on Research in Computer Security, Oslo, Norway, 11–15 September 2017; pp. 62–79.
9. Yuan, J.; Zhou, S.; Lin, L.; Wang, F.; Cui, J. Black-Box Adversarial Attacks Against Deep Learning Based Malware Binaries Detection with GAN. In Proceedings of the 24th European Conference on Artificial Intelligence, Santiago de Compostela, Spain, 29 August–8 September 2020; pp. 2536–2542.
10. Yang, W.; Kong, D.; Xie, T.; Gunter, C.A. Malware Detection in Adversarial Settings: Exploiting Feature Evolutions and Confusions in Android Apps. In Proceedings of the 33rd Annual Computer Security Applications Conference, Orlando, FL, USA, 4–8 December 2017; pp. 288–302.
11. Cara, F.; Scalas, M.; Giacinto, G.; Maiorca, D. On the Feasibility of Adversarial Sample Creation Using the Android System API. *Information* **2020**, *11*, 433. [CrossRef]
12. Papernot, N.; McDaniel, P.; Wu, X.; Jha, S.; Swami, A. Distillation as a defense to adversarial perturbations against deep neural networks. In Proceedings of the 2016 IEEE Symposium on Security and Privacy, San Jose, CA, USA, 22–26 May 2016; pp. 582–597.
13. Wang, Q.; Guo, W.; Zhang, K.; Ororbia, A.G.; Xing, X.; Liu, X.; Giles, C.L. Adversary resistant deep neural networks with an application to malware detection. In Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, New York, NY, USA, 13 August 2017; pp. 1145–1153.
14. Lyu, C.; Huang, K.; Liang, H.N. A unified gradient regularization family for adversarial examples. In Proceedings of the 2015 IEEE International Conference on Data Mining (ICDM), Atlantic City, NJ, USA, 14–17 November 2015; pp. 301–309.
15. Szegedy, C.; Zaremba, W.; Sutskever, I.; Bruna, J.; Erhan, D.; Goodfellow, I.; Fergus, R. Intriguing Properties of Neural Networks. Available online: https://arxiv.org/abs/1312.6199 (accessed on 21 December 2013).
16. Kwon, H.; Lee, J. Diversity Adversarial Training against Adversarial Attack on Deep Neural Networks. *Symmetry* **2021**, *13*, 428. [CrossRef]
17. Hosseini, H.; Chen, Y.; Kannan, S.; Zhang, B.; Poovendran, R. Blocking Transferability of Adversarial Examples in Black-Box Learning Systems. Available online: https://arxiv.org/abs/1703.04318 (accessed on 13 March 2017).
18. Li, D.; Li, Q. Adversarial Deep Ensemble: Evasion Attacks and Defenses for Malware Detection. *IEEE Trans. Inf. Forensics Secur.* **2020**, *15*, 3886–3900. [CrossRef]
19. Onwuzurike, L.; Mariconti, E.; Andriotis, P.; Cristofaro, E.D.; Ross, G.; Stringhini, G. MaMaDroid: Detecting Android Malware by Building Markov Chains of Behavioral Models (Extended Version). *ACM Trans. Priv. Secur.* **2019**, *22*, 1–33. [CrossRef]

20. Sun, Y.S.; Chen, C.C.; Hsiao, S.W.; Chen, M.C. ANTSdroid: Automatic Malware Family Behaviour Generation and Analysis for Android Apps. In Proceedings of the 23rd Information Security and Privacy, Wollongong, NSW, Australia, 11–13 July 2018; pp. 796–804.
21. Mirzaei, O.; Suarez-Tangil, G.; de Fuentes, J.M.; Tapiador, J.; Stringhini, G. AndrEnsemble: Leveraging API Ensembles to Characterize Android Malware Families. In Proceedings of the AsiaCCS'19, Auckland, New Zealand, 9–12 July 2019; pp. 304–314.
22. Tao, G.; Zheng, Z.; Guo, Z.; Lyu, M.R. MalPat: Mining Patterns of Malicious and Benign Android Apps via Permission-Related APIs. *IEEE Trans. Reliab.* **2018**, *67*, 355–369. [CrossRef]
23. Zhang, Y.; Yang, Y.; Wang, X. A Novel Android Malware Detection Approach Based on Convolutional Neural Network. In Proceedings of the 2nd International Conference on Cryptography, Security and Privacy, Guiyang, China, 16–18 March 2018; pp. 144–149.
24. Hojjatinia, S.; Hamzenejadi, S.; Mohseni, H. Android Botnet Detection using Convolutional Neural Networks. In Proceedings of the 28th Iranian Conference on Electrical Engineering (ICEE2020), Tabriz, Iran, 4–6 August 2019; pp. 674–679.
25. Jung, J.; Choi, J.; Cho, S.J.; Han, S.; Park, M.; Hwang, Y. Android malware detection using convolutional neural networks and data section images. In Proceedings of the RACS '18, Honolulu, HI, USA, 9–12 October 2018; pp. 149–153.
26. Jiang, J.; Li, S.; Yu, M.; Li, G.; Liu, C.; Chen, K.; Liu, H.; Huang, W. Android Malware Family Classification Based on Sensitive Opcode Sequence. In Proceedings of the 2019 IEEE Symposium on Computers and Communications, Barcelona, Spain, 29 June–3 July 2019; pp. 1–7.
27. Ikram, M.; Beaume, P.; Kaafar, M.A. DaDiDroid: An Obfuscation Resilient Tool for Detecting Android Malware via Weighted Directed Call Graph Modelling. In Proceedings of the 16th International Joint Conference on e-Business and Telecommunications—SECRYPT, Prague, Czech Republic, 26–28 July 2019; pp. 211–219.
28. Zhao, B. Mapping System Level Behaviors with Android APIs via System Call Dependence Graphs. Available online: https://arxiv.org/pdf/1906.10238v1.pdf (accessed on 24 June 2019).
29. Xu, Z.; Ren, K.; Qin, S.; Craciun, F. CDGDroid: Android Malware Detection Based on Deep Learning Using CFG and DFG. In Proceedings of the 20th International Conference on Formal Engineering Methods, Gold Coast, QLD, Australia, 12–16 November 2018; pp. 177–193.
30. Xu, Z.; Ren, K.; Song, F. Android Malware Family Classification and Characterization Using CFG and DFG. In Proceedings of the 2019 International Symposium on Theoretical Aspects of Software Engineering (TASE), Guilin, China, 29–31 July 2019; pp. 49–56.
31. Türker, S.; Can, A.B. AndMFC: Android Malware Family Classification Framework. In Proceedings of the 2019 IEEE 30th International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC Workshops), Istanbul, Turkey, 8 September 2019; pp. 1–6.
32. Calleja, A.; Martín, A.; Menéndez, H.D.; Tapiador, J.; Clark, D. Picking on the family: Disrupting android malware triage by forcing misclassification. *Expert Syst. Appl.* **2018**, *95*, 113–126. [CrossRef]
33. Goodfellow, I.; Pouget-Abadie, J.; Mirza, M.; Xu, B.; Warde-Farley, D.; Ozair, S.; Courville, A.; Bengio, Y. Generative adversarial nets. In Proceedings of the Annual Conference on Neural Information Processing Systems, Montreal, QC, Canada, 8–13 December 2014; pp. 2672–2680.
34. Goodfellow, I.J.; Shlens, J.; Szegedy, C. Explaining and Harnessing Adversarial Examples. Available online: https://arxiv.org/pdf/1412.6572 (accessed on 12 December 2014).
35. Papernot, N.; McDaniel, P.; Jha, S.; Fredrikson, M.; Celik, Z.B.; Swami, A. The Limitations of Deep Learning in Adversarial Settings. In Proceedings of the 2016 IEEE European Symposium on Security and Privacy, Saarbrucken, Germany, 21–24 March 2016; pp. 372–387.
36. Biggio, B.; Rieck, K.; Ariu, D.; Wressnegger, C.; Corona, I.; Giacinto, G.; Roli, F. Poisoning Behavioral Malware Clustering. In Proceedings of the 2014 ACM Workshop on Artificial Intelligent and Security, Scottsdale, AZ, USA, 7 November 2014; pp. 27–36.
37. Isola, P.; Zhu, J.Y.; Zhou, T.; Efros, A.A. Image-to-Image Translation with Conditional Adversarial Networks. In Proceedings of the 2017 IEEE Conference on Computer Vision and Pattern Recognition, Honolulu, HI, USA, 21–26 July 2017; pp. 5967–5976.
38. Ledig, C.; Theis, L.; Huszar, F.; Caballero, J.; Cunningham, A.; Acosta, A.; Aitken, A.; Tejani, A.; Totz, J.; Wang, Z.; et al. Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network. In Proceedings of the 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Honolulu, HI, USA, 21–26 July 2017; pp. 1063–6919.
39. Wang, C.; Xu, C.; Wang, C.; Tao, D. Perceptual Adversarial Networks for Image-to-Image Transformation. *IEEE Trans. Image Process.* **2018**, *27*, 4066–4079. [CrossRef] [PubMed]
40. Vondrick, C.; Pirsiavash, H.; Torralba, A. Generating Videos with Scene Dynamics. In Proceedings of the 2016 Neural Information Processing Systems (NIPS 2016), Barcelona, Spain, 5–10 December 2016; pp. 1–11.
41. Tulyakov, S.; Liu, M.Y.; Yang, X.; Kautz, J. MoCoGAN: Decomposing Motion and Content for Video Generation. Available online: https://arxiv.org/abs/1707.04993 (accessed on 14 December 2017).
42. Xie, X.; Chen, J.; Li, Y.; Shen, L.; Ma, K.; Zheng, Y. MI2GAN: Generative Adversarial Network for Medical Image Domain Adaptation. In Proceedings of the Medical Image Computing and Computer Assisted Intervention (MICCAI 2020), Lima, Peru, 4–8 October 2020; pp. 1–10.
43. Chang, Q.; Qu, H.; Zhang, Y.; Sabuncu, M.; Chen, C.; Zhang, T.; Metaxas, D.N. Synthetic Learning: Learn From Distributed Asynchronized Discriminator GAN Without Sharing Medical Image Data. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Seattle, WA, USA, 14–19 June 2020; pp. 13856–13866.

44. Kwon, H.; Yoon, H.; Park, K.W. POSTER: Detecting Audio Adversarial Example through Audio Modification. In Proceedings of the 26th ACM Conference on Computer and Communications Security (CCS'19), London, UK, 11–15 November 2019; pp. 2521–2523.
45. Kwon, H.; Yoon, H.; Park, K.W. Acoustic-Decoy: Detection of Adversarial Examples through Audio Modification on Speech Recognition System. *Neurocomputing* **2020**, *417*, 357–370. [CrossRef]
46. Dey, S.; Kumar, A.; Sawarkar, M.; Singh, P.K.; Nandi, S. EvadePDF: Towards Evading Machine Learning Based PDF Malware Classifiers. In Proceedings of the 2019 International Conference on Security and Privacy(ISEA-ISAP 2019), Jaipur, India, 9–11 January 2019; pp. 140–150.
47. Rosenberg, I.; Shabtai, A.; Elovici, Y.; Rokach, L. Defense Methods Against Adversarial Examples for Recurrent Neural Networks. Available online: https://arxiv.org/pdf/1901.09963.pdf (accessed on 20 November 2019).
48. Singh, A.; Dutta, D.; Saha, A. MIGAN: Malware Image Synthesis Using GANs. In Proceedings of the 33rd AAAI Conference on Artificial Intelligence, Hilton Hawaiian Village, Honolulu, HI, USA, 27 January–1 February 2019; pp. 10033–10034.
49. Chen, L.; Hou, S.; Ye, Y.; Xu, S. DroidEye: Fortifying Security of Learning-Based Classifier Against Adversarial Android Malware Attacks. In Proceedings of the 2018 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM), Barcelona, Spain, 28–31 August 2018; pp. 782–789.
50. Chen, Y.M.; Yang, C.H.; Chen, G.C. Using Generative Adversarial Networks for Data Augmentation in Android Malware Detection. In Proceedings of the 2021 IEEE Conference on Dependable and Secure Computing (DSC), Aizuwakamatsu, Fukushima, Japan, 30 January–2 February 2021; pp. 1–8.
51. Mirza, M.; Osindero, S. Conditional Generative Adversarial Nets. Available online: https://arxiv.org/abs/1411.1784 (accessed on 6 November 2014).
52. Ronneberger, O.; Fischer, P.; Brox, T. U-net: Convolutional networks for biomedical image segmentation. In Proceedings of the 18th International Conference on Medical Image Computing and Computer Assisted Interventions, Munich, Germany, 5–9 October 2015; pp. 234–241.
53. LeCun, Y.; Bottou, L.; Bengio, Y.; Haffner, P. Gradient-based learning applied to document recognition. *Proc. IEEE* **1998**, *86*, 2278–2324. [CrossRef]
54. Allix, K.; Bissyandé, T.F.; Klein, J.; Le Traon, Y. AndroZoo: Collecting Millions of Android Apps for the Research Community. In Proceedings of the 13th International Conference on Mining Software Repositories (MSR), Austin, TX, USA, 14–15 May 2016; pp. 468–471.
55. Arp, D.; Spreitzenbarth, M.; Hubner, M.; Gascon, H.; Rieck, K.; Siemens, C.E.R.T. DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket. In Proceedings of the 2014 Network and Distributed System Security (NDSS), San Diego, CA, USA, 23–26 February 2014; pp. 1–12.