Are You Confident That You Have Successfully Generated Adversarial Examples?

Bo Wang, *Member, IEEE*, Mengnan Zhao, *Student Member, IEEE*, Wei Wang, *Member, IEEE*, Fei Wei, *Member, IEEE*, Zhan Qin, *Member, IEEE*, and Kui Ren, *Fellow, IEEE*

***Abstract*— Deep neural networks (DNNs) have seen extensive studies on image recognition and classification, image segmentation, and related topics. However, recent studies show that DNNs are vulnerable in defending adversarial examples. The classification network can be deceived by adding a small amount of perturbation to clean samples. There are challenges when researchers want to design a general approach to defend against a wide variety of adversarial examples. To solve this problem, we introduce a defensive method to prevent adversarial examples from generating. Instead of designing a stronger classifier, we built a more robust classification system that can be viewed as a structural black box. After adding a buffer to the classification system, attackers can be efficiently deceived. The real evaluation results of the generated adversarial examples are often contrary to what the attacker thinks. Additionally, we do not assume a specific attack method premise. This incognizance to underlying attacks demonstrates the generalizability of the buffer to potential adversarial attacks. Extensive experiments indicate that the defense method greatly improves the security performance of DNNs.**

***Index Terms*— Deep neural networks, adversarial examples, structural black box, buffer.**

I. Introduction

With the emergence of deep learning (DL) and the establishment of big data [1], DNN performance has significantly improved on image classification and related tasks [2]–[4]. However, until now, it still works as a black box. Adversarial examples are attracting attention from researchers, as they are helpful in understanding DL.

There are many attack methods available for generating adversarial examples. For instance, [5] notes that tiny perturbations for clean images can deceive classification networks. Goodfellow *et al.* [6] proposed the fast gradient sign method (FGSM), for which a one-step gradient update is performed based on the direction of the gradient sign at image pixels. To improve the FGSM attack performance, [7] uses the momentum iteration method to generate adversarial examples. In addition, BIM [8], JSMA [9], and C&W [10] are also widely used attack methods.

The studies of adversarial examples not only improve the network robustness but also create security concerns for researchers. Due to the potential hostile intentions of adversarial examples, it is necessary to establish defensive methods. Adversarial training [11] includes adversarial examples in the training process. Papernot *et al.* [12] proposed the network distillation method, which can interrupt the backward gradient. The simplest methods for distinguishing clean images and adversarial examples are training the classification network, but this seems to be an infinite game. Reference [13] introduced the auxiliary classification detector to extract *ReLU* outputs of the trained classifier (*e.g.* *VGG* [14]) as features to defend adversarial examples. However, all the above methods need to retrain the network and it is challenging to transfer the defensive ability to adversarial examples that are generated by different attacks.

Based on this, we focus our attention on inquiring about the possibility of preventing the generation of adversarial examples rather than designing universal defensive methods. Therefore, the exploration of the generalization performance is transferred from the measurement between adversarial examples to different attack methods. Compared with various adversarial examples, the existing attacks have greater similarities, such as gradient attacks and iterative attacks, which make it possible to propose a predefense method. Similar to the decorators in programming languages, we introduce the system packaging theory (SPT) to solve the retraining problem. Through packaging the classification system as a structural black box, the SPT can deceive attackers without changing the network parameters and structure.

Assume that attackers can access the original system (*OS*) and download an offline version, which we call the copied system (*CS*). As depicted in Fig. 1, the successfully generated adversarial examples for *CS* are transferred to the *OS* to evaluate the defensive performance of SPT. SPT aims to trigger different mechanisms toward the normal testing process and the attacking process to defend adversarial examples. The gradient attack method not only realizes the attack but also divulges the target to the attacked system. To deceive the

Manuscript received April 28, 2020; revised July 20, 2020; accepted August 13, 2020. Date of publication August 17, 2020; date of current version June 4, 2021. This work was supported by the National Natural Science Foundation of China under Grant U1936117, Grant U1736119, Grant 61972395, and Grant 61772111. This article was recommended by Associate Editor Y. Qin. *(Corresponding author: Wei Wang.)*

Bo Wang and Mengnan Zhao are with the School of Information and Communication Engineering, Dalian University of Technology, Dalian 116024, China (e-mail: bowang@dlut.edu.cn; zmnwelcome@mail.dlut.edu.cn).

Wei Wang is with the Center for Research on Intelligent Perception and Computing, Institute of Automation, Chinese Academy of Sciences, Beijing 100190, China (e-mail: wei.wong@ia.ac.cn).

Fei Wei is with the Department of Electrical Engineering, State University of New York (SUNY) at Buffalo, Buffalo, NY 14200 USA (e-mail: feiwei@buffalo.edu).

Zhan Qin and Kui Ren are with the College of Computer Science, Zhejiang University, Hangzhou 310000, China, and also with the Institute of Cyberspace Research (ICSR), Zhejiang University, Hangzhou 310000, China (e-mail: qinzhan@zju.edu.cn; kuiren@zju.edu.cn).

Color versions of one or more of the figures in this article are available online at https://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCSVT.2020.3017006

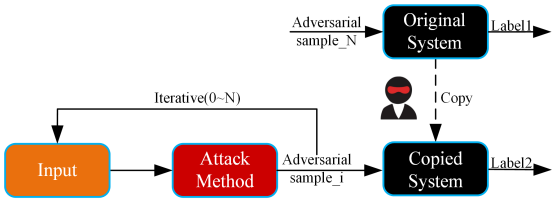


Fig. 1. The analysis of deceiving goals. The attacker can access the classification system and make a copy as an offline version, but with no knowledge of the system composition.

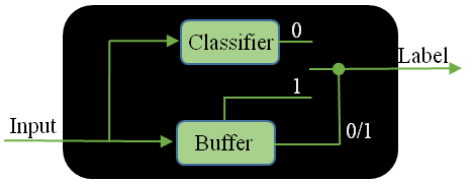


Fig. 2. The components of the structural black box. The buffer optionally controls the switch to output the label.

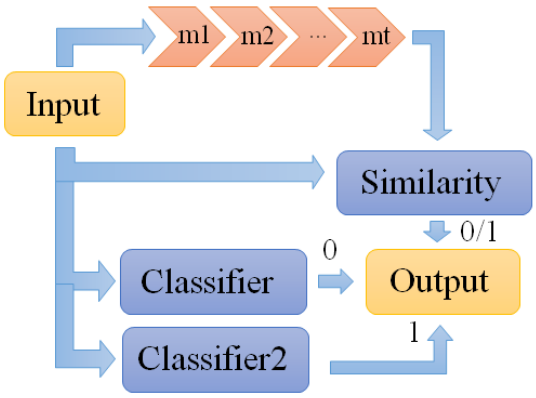


Fig. 3. Detailed description of the structural black box. The buffer consists of multiple buffer regions (*e.g.* $[m1, m2 \ldots, mt]$), a crafted $classifier_2$ and the similarity evaluation module.

attackers, for any test sample, *CS* offers the attacker labels, logits, and backward gradient values.

The SPT works in two scenarios. First, the testing process is interrupted when the structural black box senses that the attacker is generating adversarial examples. Second, the structural black box continues to back the error gradients when the test phase is assessed as an attacking process. Although attackers have confidence in generated adversarial examples for *CS* once accomplished, the *OS* can accurately classify the generated samples. As shown in Fig. 2, the buffer is applied to the structural black box for determining the working mode. The backward gradient for the buffer is designed to disrupt the attack direction. For the first scenario, we add a random digital generator (randomly output the label) to the buffer, and we experimentally choose an additional classifier to the buffer for the second scenario. The detailed composition of the structural black box is depicted in Fig. 3. To illustrate the importance of buffer regions, we designed a comparator that performs well in detecting adversarial examples but fails to prevent their generation.

The contributions of this paper are summarized as follows:

- The proposed method, which packages the classification system to defend the attacking process, is new to the state-of-the-art.
- Different from detecting the generated adversarial examples, the SPT works on decreasing the number of attacks and disrupting the attack directions. The backward error gradient makes it possible for the structural black box to deceive attackers.
- To choose the appropriate models, we perform extensive experiments to evaluate the method performance on various datasets and models. The quantitative experiments prove the defensive efficiency of the proposed method.

II. Related Works

A. Attacking Methods

The problem of generating adversarial examples has recently been widely studied in computer vision. In addition to the white-box attack methods mentioned in Section I, Szegely *et al.* [5] indicated that the crafted adversarial example by attacking model *A* is usually transferable for model *B*. This transferability implies that adversarial examples can also be generated effectively without access to the underlying model.

The adversarial examples are transferable among classification networks that learn the same or similar features. Sarkar *et al.* [15] designed the *UPSET* and *ANGRI* networks to generate adversarial examples, which attack on multiple classifiers to obtain better generalization performance. Reference [16] describes universal perturbations that can simultaneously attack multiple images. One-pixel attacks [17] tamper one pixel of the sample to realize the attack. The researchers adopted the method of differential evolution to find the crafted pixel. Reference [18] proposed adding large perturbations to realize the attack. The researchers introduced two different models, the texture transfer model and the colorization model. It is novel that changing the color of a clean image will lead to misclassification.

In addition, Papernot *et al.* [19] successfully generated crafted adversarial input sequences for recurrent neural networks (RNNs). For the fields of semantic segmentation and object detection, Xie *et al.* [20] introduced dense adversary generation (DAG) to compute adversarial examples. Reference [21] proposed a targeted adversarial attack for black box audio systems.

B. Defending Methods

Adversarial examples pose a security threat to the field of computer vision, so it is necessary to introduce corresponding defense methods. Adversarial examples can be detected by the density estimation method and the Bayesian neural network uncertain method [22]. Papernot *et al.* [12] proposed network distillation for defense, which builds two identical networks. The probability produced by the first network is inherited as inputs of the second network.

Huang *et al.* [11] applied adversarial examples as training data such that the difficulty of generating effective adversarial examples increased. Distribution differences were introduced

by [23] to distinguish clean images and adversarial examples. Reference [24] adopted the reinforcement learning method to realize detection, as such multiple known images are used to predict future inputs and detect adversarial examples.

In addition, one can recover the adversarial example to its original label. Given inputs, Xu *et al.* [25] reduced pixel values and used a median filter to smooth them. The processed samples were then passed to the trained classifier to obtain the classification results. Based on PixelCNN, Song *et al.* [26] improved PixelDefend to restore an image by maximizing the distribution loss. Researchers first detected whether the images were processed and then rectified some points of the adversarial example to restore the original distribution.

However, using multiple defense methods simultaneously cannot significantly improve the defense performance [27]. Reference [28] proposed attribute-steered detection for face recognition models based on interpretability. The classification results from the attribute-steered classifier and the original classifier are compared with each other, and the image is considered an adversarial example if the results are not equal.

III. Proposed Methods

In this section, we clarify the deceiving goal, analyze several existing common attack methods, and describe our proposed method in detail.

A. The Deceiving Goals

As depicted in Fig. 1, there are two completely identical classification systems, *OS* and *CS*. We can deceive the attacker in two scenarios. In scenario one, the attacker has false confidence in a successful generation of adversarial examples (untargeted attack), $e.g.\ Label_1 = Label_0$ but $Label_2 \neq Label_0$. Here, $Label_0$ denotes the ground truth label. Similarly, for the targeted attack, the deceiving goals are $Label_1 \neq Targeted\_Class$ and $Label_2 = Targeted\_Class$. In the second scenario, the attacker mistakenly believes that the classification system does not successfully generate adversarial examples. To conclude, the label that attackers expect to generate is not consistent with the label obtained from the *OS* but the *CS* does.

B. The Buffer: Defending Method for Iterative Attacks

BaseThe attacker can access the OS and download its offline version, but they do not know the system composition. We did not consider the situation that attackers reset the offline system since attackers have not permissions to initialize the system. In this paper, we package the classification model as a structural black box. Namely, the attacker only allowed to obtain labels, logits, and gradient backward value. One may ask how to better protect a given image from generating adversarial examples? To answer such questions, we start by analyzing several existing attacks.

Basic Iterative Method (BIM) [8]:

The BIM crops the pixel values in each step and generates adversarial examples in multiple iterations.

$$x_n = \left\{x_{n-1} + \epsilon \cdot sign\left(\nabla_{x_{n-1}} J\left(x_{n-1}, y\right)\right)\right\}$$
$$Clip_{x,\xi}\left\{x_n\right\} = \min\left\{255, x + \xi, \max\left\{0, x - \xi, x_n\right\}\right\}$$

where x and y denote the clean image and the expected label, and J is the predefined loss function, such as the cross-entropy loss. x_n is the generated sample after the iterative attack. ξ and ϵ represent the maximum degree of perturbations and the single-step perturbations, respectively.

Momentum Iterative Fast Gradient Sign Method (MIFSGM): One-step attacks have difficulty generating anticipated adversarial examples. By adding momentum to the FGSM, adversarial examples generated by [7] with more iterations perform better in the black box context.

$$g_{n+1} = \mu \cdot g_n + \frac{\nabla_{x_n} J(x_n, y)}{\| \nabla_{x_n} J(x_n, y) \|}$$

$$x_{n+1} = x_n + \varepsilon \cdot sign(g_{n+1})$$

where μ is applied to balance the momentum g_n and the current gradient value.

Jacobian-based

Saliency Map Attack (JSMA): Papernot *et al.* [9] proposed the JSMA to accomplish targeted attacks. By searching one-pixel pairs in each step, any sample modifications increase the probability of the targeted class or decreases the sum probability of other classes. JSMA has high time requirements and memory consumption.

The large contrast between adversarial examples makes it difficult to propose a general method for distinguishing adversarial examples from clean images. However, it is possible to prevent the process from generating adversarial examples due to the similarity in different attacks, such as iterative attacks and gradient-based attacks. Imitating the construction principle of decorators, we package the original classification network into a structural black box. The buffer in the structural black box is built to disrupt the classification process once the operation is determined to be an adversarial generation process. As depicted in Fig. 3, the buffer consists of multiple buffer regions ($e.g.\ [m1, m2 \ldots, mt]$), another crafted selected classifier and a similarity evaluation module.

Attack pattern$_1$ The attacker iteratively attacks the system to generate adversarial examples. The attack goal is misclassification or targeted classification.

Similarity index We first consider processing a single input. The initialization of the buffer region is set to 0 matrices. First, the inputs are stored in $m1$. Next, samples saved in $m1$ are transferred into $m2$ and new inputs are stored in $m1$. When the number of inputs is larger than the length of the buffer regions, the last image in the buffer regions is iteratively removed from the buffer regions. Then, we calculate the difference between the image in $m1$ and the removed image P if P is not a 0 matrix. Both the L_1 norm and the mean squared error (MSE) value can be used to calculate this difference. For convenience, we adopt the peak signal-to-noise ratio (PSNR) method to measure the similarity in Fig. 3.

$$\mathrm{PSNR}(m1, P) = 10 \times \log_{10}\left(\frac{max\_pixel^2}{(m1 - P)^2}\right) \tag{1}$$

$max\_pixel = 1$ if the image is normalized; otherwise, $max\_pixel = 255$. The large PSNR value denotes the high similarity between the input images.

Judging the attack process When the value calculated by Eq. (1) is greater than the predefined threshold for the image similarity, we judge the inputs as similar samples. The testing process is interrupted when the network inputs the same samples multiple times. Once the buffer meets the truncation standard, the digital generator randomly outputs the image category. At this time, the output of the *CS* makes the attacker believe that the generated image has accomplished the attack goal. The probability of the event, for which the randomly generated label matches the ground truth label in one iteration, is $\frac{1}{N}$, where N denotes the total classification categories.

Deceiving mode To mislead the attacker, the testing process should not be interrupted, and the final classification result output by the *CS* needs to realize the attack goal. Therefore, it is not appropriate to apply the random digital generator to the buffer that will interrupt the testing process. Instead, we introduce another unrelated classification network $Classifier_2$ to achieve this deceive intention. As shown in Fig. 3, when the buffer meets the truncation standard, the *CS* outputs the label generated by $Classifier_2$ instead of *Classifier*. The backward error gradient disturbs the direction of the attack and makes it possible to deceive the attacker.

Next, we study the generation for the adversarial example in a continuous process to have a further understanding of the SPT. We denote the inputs as A, B, and $\cdots$ in alphabetical order. In the normal attack situation, the attacker generates the adversarial example for the next example when successfully generating the adversarial example for the current sample. The attack process is divided into different cases according to the number of iterations.

Case 1: The number of iterations (I) needed by A to generate adversarial examples is less than the length of buffer regions t. The adversarial example is successfully generated since the buffer is not triggered. To defend the new inputs from generated adversarial examples, we choose to clear the counting parameters when the PSNR value is less than the predefined threshold. Namely, the settings are reset when the following inputs are different from the current sample.

Case 2: The number of iterations (I) required by A to generate adversarial examples is greater than t. The buffer controls the propagation process once the attacker enters similar images enough times ($I \geq t{+}S$, $S \leq t$). Then, the output label of the *CS* is calculated by the $Classifier_2$ regardless of the targeted or untargeted attack. Similarly, we reset the settings when the following input is different from the current input. Another situation is that the buffer has been triggered, but the adversarial example is successfully generated. This is because few similar samples appear during the statistics.

Attack pattern$_2$ The attack process fixes the number of iterations to I. Similar to $Attack\ pattern_1$, the buffer controls the classification system when $I \geq t{+}S$. Then, the gradient values are passed back to the inputs through the $Classifier_2$.

Improving for discontinuous attack However, the above process is only useful when defending against continuous attacks such as the input samples with the order A_0, A_1, $\cdots$, A_n. Any discontinuous attack ($e.g.$ A_0, A_1, $\cdots$,B, $\cdots$, A_n) will cause the defense to fail. To solve this problem, instead of clearing the settings when the PSNR value is less than the predefined threshold, and we choose to set $s = 0$, $k = t$ when $k \geq 2t$. For convenience, we define the symbols in Table I.

TABLE I
DEFINITIONS OF SEVERAL SYMBOLS THAT ARE USED TO DESCRIBE THE BUFFER WORKFLOW

Notation	Description
k	Number of iterations.
s	Mounts of similarity step.
t	Length of the buffer regions.
S	Truncated threshold of the buffer.
T	Classification threshold of the image similarity.

The following section only considers that the number of attack iterations is larger than $t{+}S{+}p$. In this case, the system does not interrupt the generation process, and the gradient value is optionally put back to the input via *Classifier* or $Classifier_2$. If we set $s = 0$ and $k = 0$ when $k \geq 2t$, in each $2t$ iteration, the number of attacks for the *Classifier* is $t{+}S$. Therefore, the total number of attacks for the *Classifier* can be calculated by:

$$N \geq \left\lfloor \frac{I}{2 \times t} \right\rfloor \times (S + t) \approx \left(\frac{1}{2} + \frac{S}{t}\right) \times I$$

The rate of valid attacks is greater than $\frac{N}{I} \approx \frac{1}{2} + \frac{S}{t}$. This setting performs poorly to prevent the generation of adversarial examples for the existence of constants. At least half of the iterations are used to attack the *Classifier*. However, when we reset k to t if $k = 2t$, in each t iteration, the number of successful attacks for the *Classifier* is S.

$$N \geq \mathrm{t} + S + \left\lfloor \frac{I}{t} - 2 \right\rfloor \times S \tag{2}$$

The valid attack rate is expressed as:

$$\frac{N}{I} \geq \frac{(\mathrm{t} - S) + \lfloor \frac{I}{t} \rfloor \times S}{I} \approx \frac{(\mathrm{t} - S)}{I} + \frac{S}{t}$$

Namely, the rate of valid attack approximates $\frac{S}{t}$ when $\frac{(\mathrm{t}-S)}{I} \approx 0$. Therefore, this setting can better protect the system from iterative attacks.

The deceive rate (DR) is adopted as the evaluation index to evaluate the performance of the buffer:

$$DR = |1 - \frac{Acc(CS)}{Acc(OS)}| \tag{3}$$

where $Acc(\cdot)$ denotes the classification accuracy of the system. The detailed process is given in the algorithm 1.

Does this defense process have any effects on the Classifier? The buffer has little effect on the classification performance of the *Classifier* as long as we choose a reasonable length for buffer regions. In the extreme case, the length of the buffer regions is set to 1. That is, two consecutive similar inputs make the classification system output from $Classifier_2$. This situation greatly limits the practical application of the

buffer, such as applying the classification system into the scene of time continuity. Similarly, it is not available to set the length of the buffer regions to infinity since the buffer loses its ability to defend against attacks. Therefore, the value of t should be modified according to various application scenarios.

C. The Selection Principle for Classifiers

1) Defensive Performance Against an Untargeted Attack:

a) Misclassification: Once the system misclassifies the input, the attacker generates the adversarial example for the next sample. When the buffer is triggered and controls the system output from $Classifier_2$, the result needs to meet the demands of misclassification easily to stop the attacking process early.

b) Iterative Fixed: Regardless of whether the output of the classification system satisfies the misclassification, the current iterative process continues to update the samples. When $Classifier_2$ and *Classifier* perform transferability poorly, training on $Classifier_2$ may not cause the *Classifier* to misclassify.

2) Defensive Performance Against a Targeted Attack:

Limited Iterations: The training process is not interrupted until the classification system achieves a targeted attack or the number of iterations reaches the upper limit. The targeted attack is harder than the untargeted attack, so the targeted attack needs more iterations to successfully realize the goal, which means more opportunities to trigger the buffer. To deceive attackers, we choose the vulnerable network that more easily realizes the targeted attack as $Classifier_2$.

The classification accuracy of the *Classifier* for clean samples should be as high as possible. At the same time, the *Classifier* should have difficulty generating adversarial examples. For $Classifier_2$, the iterative attack for it should have little effect on the classification accuracy of the *Classifier*.

We adopt two different networks as $Classifier_2$, an untrained network, and a replacement label trained network. We use the ground truth label (GT) to train the *Classifier* and the $|nb\_class-1-GT|$ label to train $Classifier_2$, which we call replacement label training. nb_class denotes the number of categories in the dataset. For the same samples, both *Classifier* and $Classifier_2$ can classify the samples correctly, but the output labels are different.

D. The Comparator: Fail Deceive Case to Indicate the Importance of the Buffer Regions

In Fig. 4, we show how the comparator detects adversarial examples based on the standard set. For each category in the standard set, there is one standard image. We first choose images that can be classified correctly by the VGG-Face model. The model is implemented by TensorFlow using the pretrained model from [29]. Then, we combine manual selection and cutting with the VGG-Face model to generate the standard dataset. Only one image with the highest classification accuracy is stored for each image class. We adopt the $face\_recognition$ project to obtain the similarity. The similarity is calculated by comparing the difference between the inputs and the corresponding standard images. The standard images are searched based on the output labels of the

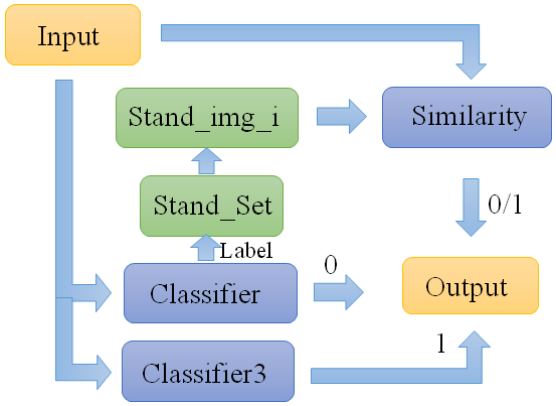


Fig. 4. Package the original classifier with the comparator. where *Classifier* and $Classifier_3$ represent the original network and another irrelevant network, respectively, both of which are trained using the same label and perform with high classification accuracy. $Label_i$ as the index to find the standard image from the stand_set.

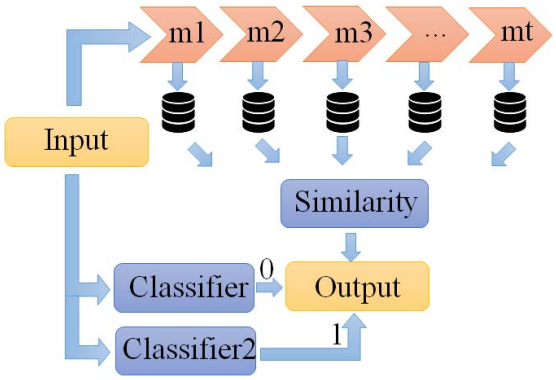


Fig. 5. To defense the discontinuous attack. Where *Classifier* and $Classifier_2$ respectively represent the original network and another irrelevant network, both of which are trained using the same label and perform the high classification accuracy.

VGG-Face model. A low degree of similarity means that the VGG-Face model misclassifies the input.

Imitating the construction of the buffer, we train another irrelevant network $Classifier_3$. Once the comparator controls the system, the system outputs labels from $Classifier_3$. What is the difference between the buffer and the comparator? The comparator directly calculates the difference between the inputs and the pre-extracted standard images. Therefore, there are no buffer regions for the comparator.

E. To Defense the Discontinuous Attack

To defense the generalized (are more practical) version of the discontinuous attack such as A_0, B_0, C_0, D_0, A_1, B_1, C_1, D_1, A_2, B_2, C_2, D_2, we propose a chain detection method by expanding the buffer region. As shown in Fig. 5, the chain detection method also includes multiple buffer regions. Each buffer region is composed of one chain. The detection steps are as follows:

1). For any input sample, check whether the current image exists in the buffer region. If not, create a new chain in buffer regions. Then, the current image stored in this new chain in order. If it already exists, it stored in the current buffer.

TABLE II

VULNERABILITY AND GENERALIZATION PERFORMANCE STUDY FOR BOTH TARGETED ATTACK AND UNTARGETED ATTACK. WHERE ORI DENOTES THE ORIGINAL CLASSIFICATION ACCURACY OF THE MODEL. FOR TARGETED ATTACKS, THE ATTACK GOAL IS 0 AND ORI = 0.1. WE SET THE ITERATIONS TO 10 FOR BOTH THE UNTARGETED ATTACK ($\epsilon = 0.0013$) AND THE TARGETED ATTACK ($\epsilon = 0.003$)

Original Classifier	Untargeted attack							Targeted attack					
	ORI	VGG	LeNet	ResNet	C4	GoogLeNet	C4_WO_BN	VGG	LeNet	ResNet	C4	GoogLeNet	C4_WO_BN
VGG	0.9376	**0.2575**	0.8718	0.7637	0.7565	0.8372	0.7000	**0.2835**	0.1268	0.1830	0.1822	0.1307	0.3375
LeNet	0.7417	0.7177	**0.0848**	0.7195	0.6753	0.7226	0.5121	0.1012	**0.9198**	0.0.0992	0.1134	0.1005	0.2069
ResNet	0.9548	0.7723	0.8937	**0.1578**	0.7487	0.7505	0.7058	0.1515	0.1199	**0.5967**	0.2010	0.1955	0.3620
C4	0.8812	0.7561	0.7252	0.7331	**0.0014**	0.7646	0.3621	0.1188	0.1485	0.1534	**1.0**	0.1246	0.5859
GoogLeNet	0.9531	0.8081	0.8953	0.6587	0.7194	**0.0903**	0.7250	0.1264	0.1144	0.2440	0.2133	**0.6859**	0.2896
C4_WO_BN	0.8461	0.7885	0.6165	0.7879	0.6312	0.8021	**0.0476**	0.1141	0.2239	0.1242	0.1780	0.1124	**0.9909**

2). Calculate the length of chains of all buffer regions. When the length of any chain is greater than the threshold, Classifier_2 outputs the result.

3). When the length of the chain of any buffer region is too long, or the buffer regions filled, reset the parameters.

F. Loss Function

Given a fixed classifier with parameters θ, a clean image x with true label y, and cross-entropy loss function *loss*, the untargeted attack is realized by solving:

$$\max_{\delta} f \cdot loss(x+\delta, y, \theta), \quad s.t. ||\delta||_p \leq \xi \quad ||f||_1 = 1 \tag{4}$$

where $||\cdot||_p$ denotes the norm function. x' ($x' = x + \delta$) is the generated adversarial example, which makes the fixed classification network produce the new label n ($n \neq y$). We modify the sign of the f to switch the untargeted or targeted attack. The targeted attacks also replace y with g, where g denotes the targeted label.

IV. EXPERIMENT

A. Datasets and Some Details

In the experiments, all images come from the open databases CIFAR10 [30] and VGG-FACE (VF) [31]. We implement the proposed method by using PyTorch® and TensorFlow®. All experiments are trained with NVIDIA GTX 1080Ti GPUs. Without the specialized statement, we set the length of the buffer regions t, the threshold of the image similarity T and the truncated threshold S to 10, 35 dB and 5, respectively.

B. Defense Performance of the Buffer

1) Vulnerability and Generalization Performance Study: We adopt the testing set in the CIFAR10 dataset to test the performance of the proposed method. Reference [15] used four kinds of networks to validate the attack transferability. where C_1 and C_2 are ReNnet [32] style networks, and C_3 and C_4 (without BN) are deep neural networks. The researchers simultaneously attack multiple trained networks to improve the transferability of the adversarial example, such as training on C_1, testing on C_2, C_3, C_4, or training on C_1, C_2, C_3, and testing on C_4. The experimental results in their paper show

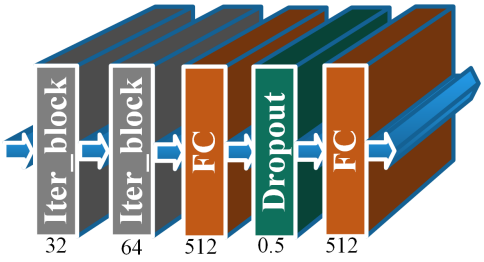


Fig. 6. The overall network proposed by [15] for C_4. Another fully connected layer is added to the end of the network to realize the different tasks.

TABLE III

DEFINITIONS OF SEVERAL SYMBOLS APPEARING IN THE FOLLOWING EXPERIMENTS

Notation	Discription
T-	Train *Classifier* and without train the Classifier_2.
Ts	Train *Classifier* and Classifier_2 with the same labels (GT).
Td	Train *Classifier* with GT and Classifier_2 with |nb_class-1-GT|.

the poor generalization performance for ResNet style networks when training on deep neural networks. Based on this, we also validate the attack transferability between several other networks, VGG [14], LeNet [33], ResNet, GoogLeNet [34], $C4\_wo\_BN$ and C_4 (with BN). The network structure of $C4\_wo\_BN$ is given in Fig. 6. For a fair comparison, we only modify the attacked model and leave the other settings unchanged. In this subsection, both targeted and untargeted attacks are executed through BIM. According to the discussion in Section III, we choose the network with high classification accuracy and the hardest to attack as the *Classifier* ($C1$). The network that is easy to attack and performs poor generalization performance to $C1$ is the Classifier_2 ($C2$).

We show the experimental results in Table II. It seems that deeper networks increase the difficulty of attacks. For example, *VGG*, *LeNet*, and the attack success rates decrease in the order. Additionally, networks with residual connections are more vulnerable. Compared with *LeNet*, *ResNet* possesses higher neural network complexity but has poor antiattack performance. In addition, the C_4 network without BN is more

TABLE IV

THE INFLUENCE OF $C2$ TO THE DEFENSIVE PERFORMANCE. WHERE $C4_{C1}$ INDICATES THAT THE NETWORK $C4$ IS TRAINED WITH THE SAME LABELS AS $C1$. ADV DENOTES THE SITUATION THAT DIRECTLY ATTACKING $C1$ WITHOUT USING THE BUFFER. THE TARGETED ATTACK GOAL IS 0

Original Classifier	Untargeted attack $C2$=C_4				Untargeted attack $C2$=LeNet			Targeted attack $C2$=C_4				Targeted attack $C2$=LeNet		
	Adv	Ts	T-	Td	Ts	T-	Td	Adv	Ts	T-	Td	Ts	T-	Td
VGG ($C1$)	0.2567	0.427	0.4702	**0.5011**	0.4477	0.4703	**0.4835**	0.2171	0.1887	**0.1756**	0.1787	0.1797	**0.1760**	0.1773
LeNet	0.7188	0.7049	0.7248	0.7318	0.4272	0.7250	0.7389	0.1012	0.1054	0.1013	0.1024	0.1859	0.1019	0.1000
ResNet	0.7934	0.7576	0.8562	0.8657	0.8273	0.8543	0.8602	0.1231	0.1291	0.1150	0.1156	0.1186	0.1158	0.1159
$C4_{C1}$	0.7589	0.1598	0.8045	0.8195	0.7563	0.8046	0.8095	0.1116	0.4211	0.1089	0.1049	0.1137	0.1054	0.1035
GoogleNet	0.8224	0.7731	0.8713	0.8785	0.8478	0.8711	0.8734	0.1115	0.1242	0.1070	0.1069	0.1115	0.1076	0.1072

TABLE V

ANALYSIS TO UNDERSTAND HOW THE BUFFER IS USED TO DECEIVE ATTACKERS. THE EXPERIMENTS ARE REALIZED BY USING BIM UNTARGETED ATTACKS. ADV DENOTES THE SITUATION OF DIRECTLY ATTACKING $C1$ WITHOUT USING THE BUFFER

Attack	Adv	C_2 =C_4			C_2 =LeNet		
		Ts	T-	Td	Ts	T-	Td
CS	-	0.3713	0.190	0.2017	0.4290	0.2282	0.2198
OS	0.2567	0.4543	0.4752	0.5029	0.4481	0.4698	0.4846
DR	-	0.1827	0.5818	**0.5989**	0.0426	0.5143	**0.5464**

TABLE VI

GENERALIZATION PERFORMANCE STUDY FOR DIFFERENT ATTACK METHODS. THE EXPERIMENTS ARE REALIZED BY TARGETED ATTACK

Attack	System	C_2 =C_4				C_2 =LeNet		
		Adv	Ts	T-	Td	Ts	T-	Td
JSMA	*CS*	-	0.85	0.96	0.81	0.93	0.31	0.87
	OS	0.93	0.74	0.59	**0.51**	0.63	0.59	**0.56**
	DR	-	0.15	0.63	0.59	0.46	0.48	0.55
MIFGSM	*CS*	-	0.69	0.68	0.69	0.48	0.10	0.50
	OS	0.33	0.24	**0.23**	**0.23**	0.24	**0.23**	**0.23**
	DR	-	1.88	1.96	2.00	1.00	0.58	1.17

TABLE VII

IMPACT OF DIFFERENT PARAMETER SETTINGS ON DEFENSE PERFORMANCE. WHERE ITER AND ε DENOTE THE TOTAL NUMBER OF ATTACKS AND THE SINGLE-STEP PERTURBATION FOR THE GIVEN SAMPLE

Attack	Iter	t	ε	System	Targeted attack			
					C_1	Ts	T-	Td
$MIFGSM_1$	40	10	3e-3	*CS*	-	0.6746	0.6746	0.6746
				OS	0.6746	0.6746	0.6746	0.6746
				DR	-	0.0000	0.0000	0.0000
$MIFGSM_2$	40	10	3e-4	*CS*	-	0.2970	0.4357	0.3500
				OS	0.1938	0.1768	0.1754	**0.1668**
				DR	-	0.6799	1.4840	1.0983
$MIFGSM_3$	100	10	3e-4	*CS*	-	0.6922	0.6873	0.6910
				OS	0.3117	0.2418	0.2300	**0.2256**
				DR	-	1.8627	1.9883	2.0629
$MIFGSM_4$	100	30	3e-4	*CS*	-	0.8583	0.8813	0.8643
				OS	0.3126	0.2097	0.1963	**0.1953**
				DR	-	3.0930	3.4900	3.4255

robust than with one. The adversarial example generated by the C_4 network without BN has better transferability performance. *LeNet* shows the worst generalization performance, and C_4 is more vulnerable to attack. Therefore, on the grounds of the experimental results, we choose the *VGG* network as $C1$, both C_4 and *LeNet* as $C2$.

2) The Influence of $C2$ on Defensive Performance: To conveniently express the experiments, Table III defines several symbols appearing in the following sections. As mentioned above, we hold the experimental settings unchanged, and four kinds of experiments are performed to detect the defensive performance of the buffer.

The experimental results are shown in Table IV. Ts, T- and Td decreased the successful attack rate since the attack for C_2 cannot transfer the ability to C_1 very well. Compared with Ts, T- and Td can better prevent the generation of adversarial examples. Additionally, Td performs better than T- against untargeted attacks and the opposite situation in targeted attacks. In the Ts training mode, C_2 and C_1 are trained with the same label, which means that the two classifiers share the same optimization direction. However, in T-, C_2 is an untrained network. Therefore, when attacking C_2 (targeted and untargeted attack), the system easily realizes the attack goal. For untargeted attacks, the probability for which the randomly generated label matches the ground truth label in one iteration is $\frac{1}{N}$, where N is the total classification category. In the Td training mode, we use replacement label training. For untargeted attacks, the probability that the output from C_2 is inconsistent with the output from C_1 in Td is higher than that from T-. For targeted attacks, C_2 and C_1 are optimized in completely different directions. However, the trained network has difficulty realizing the targeted attack compared to the untrained network.

Next, we experimentally analyze how the buffer is applied to deceive attackers. Three kinds of classification accuracy are adopted to prove the defensive performance: the accuracy detected by *OS* after directly attacking $C1$, the accuracy detected by CS after attacking CS and the accuracy detected by *OS* after attacking CS. The experimental results are shown in Table V. Similar to the results in Table IV, Td can better prevent the generation of adversarial examples for untargeted attacks. Additionally, since network C_2 is more vulnerable to

TABLE VIII

THE COMPARISON OF THE DEFENSIVE ABILITY BETWEEN THE EXISTING METHODS AND OURS. WE REALIZE THE TARGETED ATTACK USING BIM WITH ITERATION = 40 AND $\epsilon = 0.0013$. AC/ATTACK DENOTES THE CLASSIFICATION ACCURACY AFTER THE ADVERSARIAL ATTACK. MAC/ATTACK IS THE CLASSIFICATION ACCURACY AFTER THE ADVERSARIAL ATTACK AND THE POSTPROCESSING

Defensive	Post-Process	Detection Type	Retrain	Deceive Rate	Attack Rate(AR)	AC/Attack	MAC/Attack	Accuracy
ORI	-	-	-	0	0.7064	0.2234	**-**	**0.9376**
	MedianSmoothing [35]	Modify	**N**				0.7190	0.8162
	JPEGFilter [36]	Modify	**N**				**0.7945**	0.8674
Adver-Train [37]	-	Prevent	Y	0	0.5735	**0.3410**	-	0.8845
	MedianSmoothing [35]	M+P	Y				0.4794	0.7867
	JPEGFilter [36]	M+P	Y				0.4372	0.8711
Ours	-	Prevent	**N**	| 1-1/AR|=**0.93**	**0.5187**	0.3295	-	**0.9376**

attack, the attack rate in CS is higher than the attack rate in OS. In essence, the buffer protects against adversarial attacks by reducing the number and disrupting the direction of the attack on $C1$. According to the results, C_4 performs slightly better than *LeNet*.

3) Generalization Performance on Different Attack Methods: All experiments mentioned above adopt the BIM as the attack method. Next, we study the generalization performance of the buffer for other attacks. JSMA [9] is proposed to realize the targeted attack, and the attack process is transferred until the network outputs the targeted goal. Similar to the above experiments, we also choose *VGG* as $C1$, *LeNet* and C_4 as $C2$.

The experimental results in Table VI are consistent with the situation in Table IV, such that C_4 performs better than *LeNet*. This happens because C_4 is more vulnerable to attack. For example, in Table IV, when we use the BIM and Ts methods to implement the targeted attack, the successful attack rate of C_4 (0.4211) is much higher than that of *LeNet* (0.1859). We also adopt MIFGSM to detect the defense performance of the buffer. The number of attack iterations and single-step perturbations are set to 100 and 3e-4. It is obvious that the proposed method is still useful for other iterative attacks.

4) Impact of Different Hyperparameter Settings on Defensive Performance: In this subsection, we test the influence of hyperparameters in MIFSGM on the defensive performance of the buffer. All experiments are realized with targeted attacks and fixed iterations. By setting different hyperparameters, we set up several groups of experiments. The experimental results are shown in Table VII.

MIFGSM_1 has lost its defense performance since $\forall_{m1,P} PSNR(m1, p) < 35$. MIFGSM_1 and MIFGSM_2 indicate that the buffer has good defensive performance against iterative attacks when the single-step perturbation is small enough $\exists_{m1,P} PSNR(m1, p) > 35$. Additionally, it is difficult to generate targeted adversarial examples for small perturbations. MIFGSM_2 and MIFGSM_3 show that the defense method has good generalization performance for increasing the number of attacks. MIFGSM_4 performs better *DR* than MIFGSM_3 due to the decreased overall number of attacks for $C1$, which can be calculated by the formula (2). A t that is too large ($e.g.\ t = 100$) will decrease the success rate since the buffer loses its ability.

TABLE IX

THE ADVERSARIAL EXAMPLE DETECTION RATE OF THE COMPARATOR FOR SEVERAL ATTACK METHODS (VF DATABASE)

Detection	Patch [38]	C&W$_0$ [10]	C&W$_1$	C&W$_2$	FGSM [6]	BIM [8]
AMI [28]	0.97	0.91	0.99	0.97	0.91	0.90
Comparator	**1.0**	**0.99**	**1.0**	**1.0**	**1.0**	**1.0**

5) Comparative Experiment: To show the effectiveness of our method, we also compared it with the existing defense methods. We classify the defensive methods according to their detection mode. The detection methods MedianSmoothing [35] and JPEGFilter [36] distinguish the adversarial examples by modifying the image pixels. Adversarial training [37] is a prevention method that increases the difficulty of successfully generating adversarial examples. We also combine detection methods and adversarial training. The experimental results are given in Table VIII. It is worth noting that only our defense method has the possibility of deceiving attackers. This method also has little effect on the classification accuracy for clean images and successfully decreases the attack rate.

C. Defense of the Comparator

We use the same experimental settings following [28] and adopt the VGG-FACE (VF) dataset [31] to test the performance of the comparator. We randomly select 100 images as the testing set (1 image for each class, a total of 100 categories). Among them, all images can be classified correctly by the VGG-Face model. Six attacks are utilized to generate adversarial examples, which means that we obtain 600 adversarial examples. 0 and 1 in the similarity module represent the benign samples and the adversarial examples, respectively. We show the detection rate of the comparator (without $Classifier_3$) for adversarial examples in Table IX. The generated adversarial examples cause the classifier (C_1) to output the wrong label. However, the standard image searched according to the wrong label has little relevance to the adversarial

TABLE X

TO DEFEND AGAINST DISCONTINUOUS ATTACKS SUCH AS A_0, B_0, C_0, D_0, A_1, B_1, C_1, AND D_1 BY CHAIN DETECTION METHOD. WE REALIZE THE TARGETED ATTACK USING BIM WITH ITERATION = 40 AND $\epsilon = 0.0013$. ADV DENOTES THE GENERATED ADVERSARIAL EXAMPLES. CUT MEANS TO CONSTRAIN THE IMAGE PIXELS AFTER EACH GENERATION

Targeted Attack	ORI		ADV(CUT))		ADV		CHAIN		CHAIN(CUT PSNR=25dB)		CHAIN(CUT PSNR=30dB)	
	VGG	C_4	VGG	C_4	VGG	C_4	VGG	C_4	VGG	C_4	VGG	C_4
Accuracy	0.1030	0.1037	0.7064	0.1513	0.7945	0.1688	**0.2272**	**1.0000**	0.2599	1.0000	0.7922	0.3540

TABLE XI

TO DEFEND AGAINST DISCONTINUOUS ATTACKS SUCH AS A_0, B_0, C_0, D_0, A_1, B_1, C_1, AND D_1. WE REALIZE THE TARGETED ATTACK USING BIM WITH ITERATION = 40 AND $\epsilon = 0.0013$. ADV DENOTES THE GENERATED ADVERSARIAL EXAMPLES. CUT MEANS TO CONSTRAIN THE IMAGE PIXELS AFTER EACH GENERATION

Targeted Attack	ORI		ADV(CUT))		ADV		$f_n \& f_{n-1}$(CUT)		$f_n + f_{n-1}$(CUT)	
	VGG	C_4	VGG	C_4	VGG	C_4	VGG	C_4	VGG	C_4
Accuracy	0.1030	0.1037	0.7064	0.1513	0.7945	0.1688	**0.3602**	**0.9840**	0.7819	0.7051

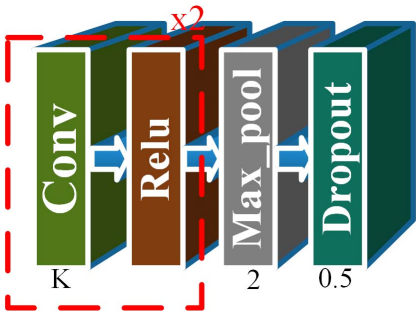


Fig. 7. The specific components of the Iter_block are shown in Fig. 6. where K denotes the depth of the feature map. For C_4 (with BN), the BN operation is added between the Conv and ReLU layers.

example. Therefore, almost all adversarial examples can be detected.

However, for the comparator, regardless of the targeted or untargeted attacks, $DR = 0$. The attacker needs to further attack the *Classifier*$_3$ after successfully causing the *Classifier* to misclassify. Therefore, regardless of the targeted and untargeted attacks, the comparator triggers the same mechanism in *OS* and *CS*. Namely, the adversarial examples that can successfully deceive the *CS* can also deceive the *OS*. Therefore, the buffer regions are vital to trick the attacker, which can trigger different modes for CS and OS.

D. Defense the Discontinuous Attack

To defense the discontinuous attack, we adopt the chain detection method. Namely, similar images are stored in the same buffer region in a chain, and different images are stored in different buffer regions.

In the experiment, we realize the discontinuous attack in the way of alterate input (A_0, B_0, C_0, D_0, A_1, B_1, C_1, D_1, A_2, B_2, C_2, D_2). The max length of the buffer regions is 10, and the length of the chain is 20. The system outputs results from $Classifier_2$ when the length of any chain is greater than 5. We set the PSNR value to 30 dB for the discontinuous attack without CUT and to 25 dB for the discontinuous attack with CUT, where CUT means to constrain the image pixels after each generation. It can be seen from the experiment results shown in Table X, the chain detection method successfully deceive the generation of adversarial examples. Namely, low attack rate for $Classifirer_1$ and high attack rate for $Classifier_2$. Meanwhile, this method little affects the classification accuracy (decrease less than 0.01) for the clean images. The method without CUT performs better than with CUT since the CUT expands the difference between similar inputs.

E. Related Experiment

The core of our paper is to introduce how to deceive attackers. Namely, sensing an attack before the generated samples deceive the model. The defensive ability of the buffer is limited to certain conditions. For instance, the buffer just detects the discontinuous attack (e.g., A_0, B_0, C_0, D_0, A_1, B_1, C_1, D_1) when t%Epoch=0 (e.g., Epoch=4). Here, is a similar method that can defend the targeted attack and without a need to consider the mentioned questions:

1) Train the original classifier $Net(\cdot)$ and another classifier $Net2(\cdot)$ that performs transferability poorly to the original classifier.

2) Obtain the current flag f_n by deterimining whether both classifiers produce the same label.

3) Combine the flags f_n and f_{n-1} to determine the current output. For example, $label = (f_n \& f_{n-1}) \cdot Net(x) + (1 - f_n \& f_{n-1}) \cdot Net2(x)$.

Even for a discontinuous attack, this method is still useful. We realize the discontinuous attack in the way of alterate input (A_0, B_0, C_0, D_0, A_1, B_1, C_1, D_1, A_2, B_2, C_2, D_2) and show the experiment results in Table XI. Analyzed by the experiment results, this method increased the difficulty of generating adversarial examples. However, this method only useful for deceiving the targeted attack.

Algorithm 1 The Buffer Workflow

Require: Construct global variables: k = 0; s = 0; x
Constant:T≥ 0; S≥0; t≥0;
Ensure: Targeted attack (L) for structural black box.
Construct the buffer regions:
$M \leftarrow [m_1, m_2, \ldots m_t]$
Get the outputs:
$Label_1 \leftarrow Classifier(x)$; $Label_2 \leftarrow Classifier_2(x)$
if $++k < t$ **then**
Buffer regions shift right:
1. $N \leftarrow M$
2. $M[0] \leftarrow x$
3. $M(1:) \leftarrow N(0 : t-1)$
return $Label_1$
else $\{++k < 2t\}$
P $\leftarrow N[-1]$
Repeat the buffer regions shift right 1,2,3.
D = PSNR (M[0], P)
if $D \geq T$ **then**
s ← s+1
if $s > S$ **then**
return $Label_2$
end if
else
return $Label_1$
end if
else $\{k \geq 2t\}$
$s \leftarrow 0$; $k \leftarrow t$;
return $Label_2$
end if

F. Analysis of Storage Requirements

Two main places in this paper take up storage, the buffer regions, and the classifier that is unrelated to the original classifier. The size of the storage area occupied by the buffer regions depends on the length of the buffer regions and the image size. In this paper, we set the value of t to 10 100 and choose the CIFAR10 as the testing set. Hence, the structural black box requires an extra 100×P pixels storage. P denotes the number of image pixels. To store these images, we need approximately 0.4 MB of memory. The memory occupied by $Classifier_2$ depends on the number of parameters of $Classifier_2$ (e.g., an extra 8.28 MB for storing C_4). LeNet is smaller than C_4. Therefore, the memory occupied by $Classifier_2$ can be ignored.

V. Conclusion

In this paper, we propose a defensive method that can deceive an attacker in the attacking process. With the analysis based on existing attacks, we construct the structural black box by simulating the decorator in programming languages. The buffer can fool the attacker by backing the error gradients of the $Classifier_2$ such that it can defend both targeted and untargeted attacks. Meanwhile, the method has little effect on the normal classification process. However, this method is less effective against fewer step attacks. The t should not be set too large to increase the sensitivity to fewer steps of attack. The S should not be set too small to increase the tolerance to continuous similar inputs. By extensive experimental studies, we choose the appropriate models to improve the defensive performance of the network. The experimental results indicate that the proposed method performs well in defending against iterative attacks.

References

[1] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2009, pp. 248–255.

[2] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 770–778.

[3] L. Gaborini, P. Bestagini, S. Milani, M. Tagliasacchi, and S. Tubaro, "Multi-clue image tampering localization," in *Proc. IEEE Int. Workshop Inf. Forensics Secur. (WIFS)*, Dec. 2014, pp. 125–130.

[4] K. Dale, K. Sunkavalli, M. K. Johnson, D. Vlasic, W. Matusik, and H. Pfister, "Video face replacement," in *ACM Trans. Graph. (TOG)*, vol. 30, no. 6., 2011, p. 130.

[5] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," in *Proc. Int. Conf. Learn. Represent.*, 2014, pp. 1–10. [Online]. Available: http://arxiv.org/abs/1312.6199

[6] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," 2014, *arXiv:1412.6572*. [Online]. Available: https://arxiv.org/abs/1412.6572

[7] Y. Dong *et al.*, "Boosting adversarial attacks with momentum," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 9185–9193.

[8] A. Kurakin, I. J. Goodfellow, and S. Bengio, "Adversarial examples in the physical world," in *Proc. ICLR Workshop*, 2017, pp. 1–14.

[9] N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami, "The limitations of deep learning in adversarial settings," in *Proc. IEEE Eur. Symp. Secur. Privacy (EuroS&P)*, Mar. 2016, pp. 372–387.

[10] N. Carlini and D. Wagner, "Towards evaluating the robustness of neural networks," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2017, pp. 39–57.

[11] B.-C. Huang, S.-W. Chou, J. M. Hong, and C.-C. Yen, "Global transonic solutions of planetary atmospheres in a hydrodynamic region–hydrodynamic escape problem due to gravity and heat," *SIAM J. Math. Anal.*, vol. 48, no. 6, pp. 4268–4310, Jan. 2016.

[12] N. Papernot, P. McDaniel, X. Wu, S. Jha, and A. Swami, "Distillation as a defense to adversarial perturbations against deep neural networks," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2016, pp. 582–597.

[13] J. H. Metzen, T. Genewein, V. Fischer, and B. Bischoff, "On detecting adversarial perturbations," in *Proc. 5th Int. Conf. Learn. Represent. (ICLR)*, 2017, pp. 1–12.

[14] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2014, *arXiv:1409.1556*. [Online]. Available: http://arxiv.org/abs/1409.1556

[15] S. Sarkar, A. Bansal, U. Mahbub, and R. Chellappa, "UPSET and ANGRI: Breaking high performance image classifiers," 2017, *arXiv:1707.01159*. [Online]. Available: http://arxiv.org/abs/1707.01159

[16] S.-M. Moosavi-Dezfooli, A. Fawzi, O. Fawzi, and P. Frossard, "Universal adversarial perturbations," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 1765–1773.

[17] J. Su, D. V. Vargas, and K. Sakurai, "One pixel attack for fooling deep neural networks," *IEEE Trans. Evol. Comput.*, vol. 23, no. 5, pp. 828–841, Oct. 2019.

[18] A. Bhattad, M. J. Chong, K. Liang, B. Li, and D. A. Forsyth, "Unrestricted adversarial examples via semantic manipulation," in *Proc. ICLR*, 2020, pp. 1–18.

[19] N. Papernot, P. McDaniel, A. Swami, and R. Harang, "Crafting adversarial input sequences for recurrent neural networks," in *Proc. MILCOM IEEE Mil. Commun. Conf.*, Nov. 2016, pp. 49–54.

[20] C. Xie, J. Wang, Z. Zhang, Y. Zhou, L. Xie, and A. Yuille, "Adversarial examples for semantic segmentation and object detection," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Oct. 2017, pp. 1369–1378.

[21] R. Taori, A. Kamsetty, B. Chu, and N. Vemuri, "Targeted adversarial examples for black box audio systems," in *Proc. IEEE Secur. Privacy Workshops (SPW)*, May 2019, pp. 15–20.

[22] R. Feinman, R. R. Curtin, S. Shintre, and A. B. Gardner, "Detecting adversarial samples from artifacts," in *Proc. Int. Conf. Mach. Learn.*, 2017, pp. 1–9.

[23] K. Grosse, P. Manoharan, N. Papernot, M. Backes, and P. McDaniel, "On the (statistical) detection of adversarial examples," 2017, *arXiv:1702.06280*. [Online]. Available: http://arxiv.org/abs/1702.06280

[24] Y.-C. Lin, M.-Y. Liu, M. Sun, and J.-B. Huang, "Detecting adversarial attacks on neural network policies with visual foresight," 2017, *arXiv:1710.00814*. [Online]. Available: http://arxiv.org/abs/1710.00814

[25] W. Xu, D. Evans, and Y. Qi, "Feature squeezing: Detecting adversarial examples in deep neural networks," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2018, pp. 1–16. [Online]. Available: http://wp.internetsociety.org/ndss/wp-content/uploads/sites/25/2018/02/ndss2018_03A-4_Xu_paper.pdf

[26] Y. Song, T. Kim, S. Nowozin, S. Ermon, and N. Kushman, "Pixeldefend: Leveraging generative models to understand and defend against adversarial examples," in *Proc. 6th Int. Conf. Learn. Represent. (ICLR)*, May 2018, pp. 1–20.

[27] W. He, J. Wei, X. Chen, N. Carlini, and D. Song, "Adversarial example defense: Ensembles of weak defenses are not strong," in *Proc. 11th USENIX Workshop Offensive Technol. (WOOT)*, 2017, p. 15.

[28] G. Tao, S. Ma, Y. Liu, and X. Zhang, "Attacks meet interpretability: Attribute-steered detection of adversarial samples," in *Proc. Adv. Neural Inf. Process. Syst.*, 2018, pp. 7717–7728.

[29] A. Vedaldi and K. Lenc, "MatConvNet: Convolutional neural networks for MATLAB," in *Proc. 23rd ACM Int. Conf. Multimedia MM*, 2015, pp. 689–692.

[30] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," M.S. thesis, Dept. Comput. Sci., Univ. Toronto, Toronto, ON, Canada, 2009.

[31] Q. Cao, L. Shen, W. Xie, O. M. Parkhi, and A. Zisserman, "VGGFace2: A dataset for recognising faces across pose and age," in *Proc. 13th IEEE Int. Conf. Autom. Face Gesture Recognit. (FG)*, May 2018, pp. 67–74.

[32] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi, "Inception-v4, inception-resnet and the impact of residual connections on learning," in *Proc. 31st AAAI Conf. Artif. Intell.*, 2017, pp. 1–7.

[33] R. Al-Jawfi, "Handwriting arabic character recognition lenet using neural network," *Int. Arab J. Inf. Technol.*, vol. 6, no. 3, pp. 304–309, 2009.

[34] C. Szegedy *et al.*, "Going deeper with convolutions," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2015, pp. 1–9.

[35] W. Xu, D. Evans, and Y. Qi, "Feature squeezing: Detecting adversarial examples in deep neural networks," 2017, *arXiv:1704.01155*. [Online]. Available: http://arxiv.org/abs/1704.01155

[36] G. K. Dziugaite, Z. Ghahramani, and D. M. Roy, "A study of the effect of JPG compression on adversarial images," 2016, *arXiv:1608.00853*. [Online]. Available: http://arxiv.org/abs/1608.00853

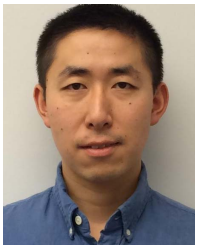
[37] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, "Towards deep learning models resistant to adversarial attacks," 2017, *arXiv:1706.06083*. [Online]. Available: http://arxiv.org/abs/1706.06083

[38] T. B. Brown, D. Mané, A. Roy, M. Abadi, and J. Gilmer, "Adversarial patch," 2017, *arXiv:1712.09665*. [Online]. Available: http://arxiv.org/abs/1712.09665

Bo Wang (Member, IEEE) received the B.S. degree in electronic and information engineering, and the M.S. and Ph.D. degrees in signal and information processing from the Dalian University of Technology, Dalian, China, in 2003, 2005, and 2010, respectively. From 2010 to 2012, he was a Post-Doctoral Research Associate with the Faculty of Management and Economics, Dalian University of Technology, where he is currently an Associate Professor with the School of Information and Communication Engineering. His current research interests focus on the areas of multimedia processing and security, such as digital image processing and forensics.

Mengnan Zhao (Student Member, IEEE) received the B.S. degree in electronic and information engineering from the Tianjin University of Technology, China, in 2018. He is currently pursuing the master's degree with the School of Information and Communication Engineering, Dalian University of Technology. His research interest includes adversarial examples and forensics.

Wei Wang (Member, IEEE) received the B.E. degree in computer science and technology from North China Electric Power University in 2007. Since 2012, he has been with the National Laboratory of Pattern Recognition, Institute of Automation, Chinese Academy of Sciences, where he is currently an Assistant Professor. His research interests include pattern recognition, image processing, and digital image forensics, including watermarking, steganalysis, and tampering detection.



Fei Wei (Member, IEEE) received the B.S. degree in electrical engineering from the Harbin University of Science and Technology, Harbin, China, in 2012, and the M.S. degree in electrical engineering from Harbin Engineering University, Harbin, in 2015. He is currently pursuing the Ph.D. degree in electrical engineering with the State University of New York (SUNY) at Buffalo, Buffalo, NY, USA, where he has been a Research Assistant with the Department of Electrical Engineering, since 2015. His research interests are varied and include the cross fields of network information theory, coding theory, machine learning, and data mining.

Zhan Qin (Member, IEEE) received the Ph.D. degree from the Computer Science and Engineering Department, State University of New York at Buffalo, in 2017. He was an Assistant Professor with the Department of Electrical and Computer Engineering, University of Texas at San Antonio. He is currently a ZJU100 Young Professor with both the College of Computer Science and Technology and the Institute of Cyberspace Research (ICSR), Zhejiang University, China. His current research interests include data security and privacy, secure computation outsourcing, artificial intelligence security, and cyberphysical security in the context of the Internet of Things. His works explore and develop novel security-sensitive algorithms and protocols for computation and communication in the general context of cloud and Internet devices.

Kui Ren (Fellow, IEEE) received the Ph.D. degree from the Worcester Polytechnic Institute, Worcester, MA, USA. He is currently a Professor of computer science and technology and the Director of the Institute of Cyberspace Research, Zhejiang University, Hangzhou, China. His current research interests span cloud and outsourcing security, wireless and wearable system security, and artificial intelligence security. He was a recipient of the NSF CAREER Award in 2011 and the IEEE CISTC Technical Recognition Award in 2017. He is also a Distinguished Scientist of the ACM.