Escaping Filter-based Adversarial Example Defense: A Reinforcement Learning Approach

Kaijian Dan
College of Computer Science
Chongqing University
Chongqing, China
kaijiandan@cqu.edu.cn

Xinyu Lei
Dept. of Computer Science
Michigan Technological University
Houghton, USA
xinyulei@mtu.edu

Yantao Li
College of Computer Science
Chongqing University
Chongqing, China
yantaoli@cqu.edu.cn

Huafeng Qin
Computer Science & Information Engineering
Chongqing Technology & Business University
Chongqing, China
qinhuafengfeng@163.com

Shaojiang Deng
College of Computer Science
Chongqing University
Chongqing, China
sj_deng@cqu.edu.cn

Gang Zhou
Dept. of Computer Science
William & Mary
Williamsburg, USA
gzhou@cs.wm.edu

***Abstract*—An adversarial example is a specially-crafted example with subtle and intentional perturbations that causes a machine learning model to make a false classification. A plethora of papers have proposed to use filters to effectively defend against adversarial example attacks. However, we demonstrate that the filter-based defenses may not be reliable in this paper. We develop AEDescaptor, a scheme to escape the filter-based defenses. AEDescaptor uses a specially-crafted policy gradient reinforcement learning algorithm to generate adversarial examples even if the filters are used to interrupt the backpropagation channel (that is used in traditional adversarial example attack algorithms). Furthermore, we design a customized algorithm to reduce the possible action space in policy gradient reinforcement learning to accelerate AEDescaptor training while still ensuring that AEDescaptor generates successful adversarial examples. The intensive experiments demonstrate that AEDescaptor-generated adversarial examples have good performance (in terms of success rate and transferability) to escape the filter-based defenses.**

***Index Terms*—Adversarial Examples, Image Classification, Reinforcement Learning, Filter**

I. Introduction

In the past decade, with the explosive growth of data, the traditional machine learning (ML) methods cannot make full use of big data and fail to provide ML models with superior performance. Deep learning, as a new research direction in the field of ML, has significantly improved performance over traditional ML. Therefore, it has been widely used in image classification [1], [2], semantic segmentation [3], [4], object detection [5], [6], and so on. Image classification is one of the most popular applications of deep learning. Some security-critical applications highly rely on image classification techniques. For example, autonomous driving vehicles employ images captured by cameras to classify road traffic signs (such as Stop signs and Speed Limit signs).

However, recent research has shown that deep neural network is easy to be fooled by adversarial examples (AEs) in image classification tasks. An adversarial example is a specially-crafted example with subtle and intentional perturbations that causes a machine learning model to make a false classification. Because the perturbations are very slight, the adversarial example is visually similar to the original image, so it is hard to be distinguished by human eyes. The above attack is called adversarial example attack in this paper. The adversarial example attack may lead to catastrophic damage to the victim. For instance, it has been demonstrated that an adversarial example attack could cause a classifier to interpret a subtly-modified physical Stop sign as a Speed Limit 45 sign [7]. Thus, in reality, such an attack may lead to serious traffic accidents for autonomous driving vehicles.

To defend the adversarial example attacks, researchers have proposed many methods. These methods can be roughly divided into two categories: (1) filter-based defense method [8], [9] and (2) AE-detectable networks [10], [11]. The filter-based defense method defends AE by using filters on model inputs without requiring any changes on the original ML model. On the contrary, the second method incorporates the AE-detection functionality into the ML model, so the original ML model needs to be redesigned and retrained. Compared with the second method, the filter-based defense method requires less computation and eliminates the requirements for modifying the original ML models. Thus, the filter-based defense method has been widely studied in a large amount of papers (e.g., [8], [9], [12]–[17]). Many of these papers have demonstrated the effectiveness of the filter-based method, making it a popular defense against AE attacks.

In this paper, we focus on the filter-based defense method. We consider two types of filter-based defense methods: (1) the series-filter-based defense method and (2) the parallel-

This work was supported in part by the National Natural Science Foundation of China under Grant 62072061, in part by the Fundamental Research Funds for the Central Universities under Grant 2021CDJQY-026, and in part by the National Science Foundation under Grant No. CNS-2153393.

Yantao Li is the corresponding author.

filter-based defense method. (1) The series-filter-based defense method simply feeds the input to multiple series-connected filters before forwarding it to a ML model. The intuition underlying this method is that the adversarial perturbed pixels are likely to be noisy pixels in an image. Hence, multiple series-connected filters with image denoising functionality would remove the perturbed pixels, and hence, the AE attack can be defended. (2) The parallel-filter-based defense method is first proposed in [9]. The key idea of this method is to compare the softmax output vector (at the last layer of the ML model) on the original image with the softmax output vector on the image after being filtered. If the two output vectors have a substantially longer distance, the image is likely to be an adversarial example. By calculating the distance between softmax output vector and selecting a specific threshold, this method can accurately detect adversarial examples. In this paper, we aim to answer the following question: *is it possible to develop an AE generation method to escape filter-based adversarial example defense?* Unfortunately, our study yields a positive answer "Yes" to the question. We propose **AE D**efense **escaptor** (AEDescaptor) to generate AEs to escape the filter-based defense methods. There are three technical challenges in AEDescaptor design.

First, it is challenging to design an AE generation algorithm in the presence of filters. The traditional gradient-based AE attacks use the backpropagation algorithm [18] [19] [20] to back-propagate gradients to the input space for generating AE. However, if a non-differentiable filter is pre-pended to the ML model, then the backpropagation channel is interrupted, making the traditional gradient-based AE generation algorithms infeasible. To address this challenge, we use the reinforcement learning (RL) method (with a properly designed reward function) to directly train AEDescaptor. Thus, the requirement for gradient propagation passing through a filter is eliminated.

Second, it is challenging to accelerate AEDescaptor training while still ensuring the proper functionality of AEDescaptor. An image usually contains many pixels and each pixel has many possible values, indicating that there is huge search space for AEDescaptor training. If the search space is too large, then the training time consumption is prohibitively long. To reduce the training time, we decrease the possible action space in each epoch of training. The action for one pixel is restricted to only two possible operations: flip or stay unchanged. In this way, the AEDescaptor training is significantly accelerated.

Third, it is challenging to keep the AEDescaptor-generated AEs to be as close as possible to the original examples so that they are hard to be noticeable by human eyes. To handle this challenge, we add a penalty term in the reward function to minimize the number of pixels modified while still ensuring that the AEDescaptor-generated AEs are misclassified. Our method ensures that the AEDescaptor-generated AEs are visually similar to their original images as much as possible.

In summary, this paper makes three main contributions.

- We make the first step to demonstrate that the filter-based adversarial example defense can be escaped by using RL to generate AEs. The proposed approach can undermine the validity of a type of popular AE attack defense.
- We design a customized algorithm to reduce the possible action space in policy-gradient RL to speed up AEDescaptor training while still ensuring AEDescaptor to generate successful AEs.
- Our intensive experimental results show that AEDescaptor-generated AEs have a 100% attack success rate on different filter parameters for both series-filter-based defense and parallel-filter-based defense. Besides, the generated adversarial examples have good cross-model transferability.

TABLE I
NOTATIONS

Notation	Meanings
$\mathbf{a}$	Action
$\mathbf{s}$	State
r	Reward
$\mathbf{X^0}$	Original example
$\overline{\mathbf{X}}$	Adversarial example
$\mathbf{Y^t}$	The softmax output vector at last layer of $\mathbf{X^t}$
$\mathbf{\Gamma}$	The policy matrix
$\mathbf{P}$	The probability matrix
C	The confidence of $\mathbf{X}$ is classified as the target label
D_{max}	The disagreement of softmax output vector at last layer
θ	The policy network's parameters
n	The number of classes
y_i^t	The probability that $\mathbf{X^t}$ is classified as class label i
x_{ij}^t	The pixel value of the i-th row and j-th column in $\mathbf{X^t}$
a_{ij}^t	The element of the i-th row and the j-th column in $\mathbf{a}^t$
p_{ij}^t	The element of the i-th row and the j-th column in $\mathbf{P}^t$
γ_{ij}^t	The element of the i-th row and the j-th column in $\mathbf{\Gamma}^t$
$\pi_{\theta^t}(\mathbf{s}^t, \mathbf{a}^t)$	The probability of taking action a_t in the state s_t

Fig. 1. How the series-filter-based defense works

II. Escaping the Series-filter-based Defense

In this section, we first have the problem description. Then, we introduce our approach to escape the series-filter-based defense in detail. The frequently used notations are summarized in Table I.

A. Problem Description

Figure 1 shows how the series-filter-based defense method works. It simply feeds the input to multiple series-connected filters before forwarding it to the ML model. The intuition underlying this method is that the adversarial perturbed pixels are likely to be noisy pixels in an image. The filters (e.g., median filter, bit depth filter) are usually capable of image denoising. Hence, once the noise is removed from an image, the image can be classified correctly again. Our goal is to design

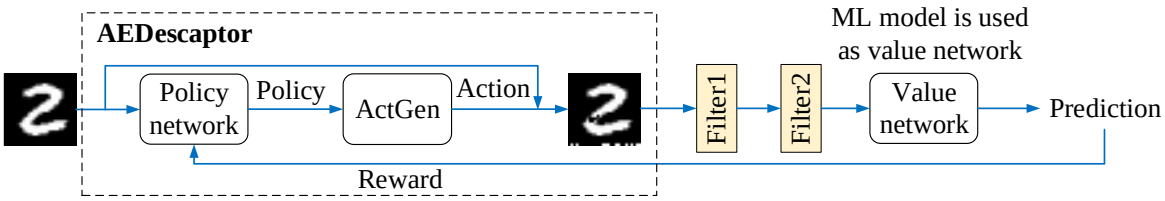


Fig. 2. Use AEDescaptor to escape the series-filter-based defense

AEDescaptor to generate specially-crafted adversarial examples. It is expected that the generated adversarial examples, after being filtered, can still lead to model misclassification. Given an input, the output of the ML model is assumed to be available. However, the knowledge of ML model details (e.g., network depth, network parameters) is not required. Thus, the AEDescaptor-based attack is a black-box attack. In this section, we consider the defense that uses two series-connected filters for preprocessing the input images. Note that AEDescaptor can be extended to escape the defense where more filters are series-connected.

B. Policy Gradient RL

Policy gradient, as a policy-based RL algorithm, can directly output the probability of each action and update policy parameters by continuously calculating the gradient of policy expected reward, and finally converges to the optimal policy [21]. Specifically, the policy gradient uses reward r to directly enhance or weaken the possibility of choosing action $\mathbf{a}$. Good action will increase the probability of being selected next time, whereas bad action will weaken the probability of being selected next time. Based on the reward r, the policy network's parameters θ is optimized by the formula $\mathcal{L}(\theta) = -\sum \log \pi_\theta(\mathbf{s}, \mathbf{a})\mathrm{r}$, where $\mathcal{L}(\cdot)$ is the loss function and $\pi_\theta(\mathbf{s}, \mathbf{a})$ is the probability of taking action $\mathbf{a}$ in the state $\mathbf{s}$.

C. AEDescaptor Design

Figure 2 shows how AEDescaptor works. AEDescaptor adopts the policy gradient algorithm to generate adversarial examples with the goal of escaping the series-filter-based defense. Compared with the value-based RL method, the policy gradient algorithm is effective in high-dimensional space or continuous action space, has better convergence, and can learn random policy.

In AEDescaptor, we use an $m \times n$ matrix $\mathbf{X}^0$ to represent the original image. Each element of the matrix denotes a pixel in the image. Each pixel is a real number in $[0, 1]$. Let x_{ij}^0 represent the pixel in the i-th row and j-th column of $\mathbf{X^0}$. Let θ represent the policy network's parameters. Let γ_{ij}^t represent the element in the i-th row and j-th column of $\mathbf{\Gamma}^t$. Let p_{ij}^t represent the pixel in the i-th row and j-th column of $\mathbf{P}^t$. Giving an input $\mathbf{X}^0$, a policy network outputs an $m \times n$ policy matrix $\mathbf{\Gamma}^t$ at the t-th training epoch. Each entry in $\mathbf{\Gamma}^t$ represents a probability. According to $\mathbf{\Gamma}^t$, the action generation (ActGen) algorithm can output an $m \times n$ action matrix $\mathbf{a}^t$ and an $m \times n$ probability matrix $\mathbf{P}^t$ at the t-th training epoch. It holds that $a_{ij}^t = 1$ with γ_{ij}^t probability and $a_{ij}^t = 0$ with $1 - \gamma_{ij}^t$ probability. If $a_{ij}^t = 1$, then $p_{ij}^t = \gamma_{ij}^t$; otherwise, $p_{ij}^t = 1 - \gamma_{ij}^t$. Then, $\mathbf{X}^t$ is computed by using

$$|\mathbf{a}^t - \mathbf{X}^0| = \mathbf{X}^t. \tag{1}$$

Next, $\mathbf{X}^t$ is forwarded to the filters and the value network to get a softmax output vector $\mathbf{Y}^t = \{y_1^t, y_2^t, ..., y_n^t\}$, where y_i^t represents the probability that $\mathbf{X}^t$ is classified as class label i and n is the number of classes. This vector $\mathbf{Y}^t$ can be used to compute a reward r^t, which can be applied to train the policy network. To facilitate description, several terminologies are defined as follows.

- **State.** Suppose that there are n training epochs. The state can be represented by $\{\mathbf{X}^0, \cdots, \mathbf{X}^n\}$. Let the $\mathbf{s}^t$ represent the t-th state $\mathbf{X}^t$;
- **Action.** Action is defined as an $m \times n$ matrix $\mathbf{a}^t$ (named action matrix). It holds that $x_{ij}^t = |a_{ij}^t - x_{ij}^0|$. Or equivalently, $x_{ij}^t = x_{ij}^0$ if $a_{ij}^t = 0$ and $x_{ij}^t = 1 - x_{ij}^0$ if $a_{ij}^t = 1$.
- **Reward.** Reward is used to measure the benefits of taking an action. Let r^t represent the reward of taking action $\mathbf{a}^t$ in state $\mathbf{s}^t$.

Reward Design. The reward is designed as a weighted sum of the two factors f_1 and f_2. The first factor measures the ability to escape the series-filter-based defense. It is described as $f_1 = y_{target}^t - y_{true}^t$, where y_{target}^t is the probability that $\mathbf{X}^t$ is classified as target class label, and y_{true}^t is the probability that $\mathbf{X}^t$ is classified as true class label. The larger f_1, the larger the reward. The second factor measures the degree of indistinguishability to human eyes. It is described as $f_2 = \|\mathbf{a}^t\|_0 = \sum_{i,j} a_{ij}^t$. The larger f_2, the smaller the reward. Therefore, the reward r^t is defined as

$$\mathrm{r}^t = y_{target}^t - y_{true}^t - \beta\|\mathbf{a}^t\|_0, \tag{2}$$

where β is a hyperparameter to balance attack performance and the number of perturbations. Note that using this reward implies this is an L_0 attack since it uses the L_0 norm to measure the degree of modification.

D. AEDescaptor Training

Next, we introduce how the policy network is trained according to the reward. The training procedure is shown in

Algorithm 1 AEDescaptor Training to Escape the Series-filter-based Defense

Input: Input image $\mathbf{X}^0$, Policy network's parameters θ;
Output: Adversarial example $\overline{\mathbf{X}}$;

1: Randomly initialize the policy network's parameters θ^0;
2: Normalize the pixel value of $\mathbf{X}^0$ to be between 0 and 1
3: **for** epoch $t = 1, 2, ...$ **do**
4: Feed $\mathbf{X}^0$ into the policy network;
5: The policy network outputs an $m \times n$ policy matrix $\mathbf{\Gamma}^t$;
6: According to $\mathbf{\Gamma}^t$, the ActGen algorithm outputs an $m \times n$ action matrix $\mathbf{a}^t$ and an $m \times n$ probability matrix $\mathbf{P}^t$;
7: Modify the pixels of the image $\mathbf{X}^0$ according to action $\mathbf{a}^t$ matrix by using Equation (1);
8: Input the generated example $\mathbf{X}^t$ into the value network to get a softmax output vector $\mathbf{Y}^t = \{y_1^t, y_2^t, ..., y_n^t\}$ at last layer;
9: Calculate reward r^t according to Equation (2);
10: Update the policy network's parameters θ: $\theta^{t+1} \leftarrow \theta^t + \alpha \nabla_{\theta^t} \log \pi_{\theta^t}(\mathbf{s}^t, \mathbf{a}^t)\mathrm{r}^t$;
11: **end for**
12: **return** $\mathbf{X}^t$;

Algorithm II-D. First, in lines 1-2, AEDescaptor initializes policy network parameter θ^0 and inputs image $\mathbf{X^0}$. At the t-th training epoch, the input image $\mathbf{X}^0$ is fed into policy network to generate policy matrix $\mathbf{\Gamma}^t$(lines 4-5). According to $\mathbf{\Gamma}^t$, the ActGen algorithm outputs action matrix $\mathbf{a}^t$ and probability matrix $\mathbf{P}^t$(line 6). Next, AEDescaptor modifies the pixels of the image $\mathbf{X}^0$ according to action $\mathbf{a}^t$ by using Equation (1) (line 7). Then, $\mathbf{X}^t$ is forwarded to the value network to get a softmax output vector $\mathbf{Y}^t$ at last layer, and $\mathbf{Y}^t$ is used to calculate reward r^t according to Equation (2) (lines 8-9). Last, in line 10, the policy network's parameters θ^t is updated by the reward r^t as

$$\theta^{t+1} \leftarrow \theta^t + \alpha \nabla_{\theta^t} \log \pi_{\theta^t}(\mathbf{s}^t, \mathbf{a}^t)\mathrm{r}^t, \tag{3}$$

where $\pi_{\theta^t}(\mathbf{s}^t, \mathbf{a}^t) = \frac{1}{m \times n} \sum_i \sum_j p_{ij}^t$ and α is the learning rate.

After multiple epochs of training, AEDescaptor can finally output an optimized action to generate an adversarial example $\overline{\mathbf{X}}$ visually close to the input image $\mathbf{X}^0$ as much as possible. Meanwhile, the adversarial example can escape the series-filter-based defense.

We have two remarks about Algorithm II-D. First, AEDescaptor introduces ActGen to make the action matrix be a binary matrix. This can help to significantly reduce the search space during training. Second, Equation (3) is used to update the policy network's parameters, such that the generated action can gradually increase the expected reward.

III. Escaping the Parallel-Filter-based Defense

In this section, we first have the problem description. Then, we introduce our approach to escape the parallel-filter-based defense in detail.

A. Problem Description

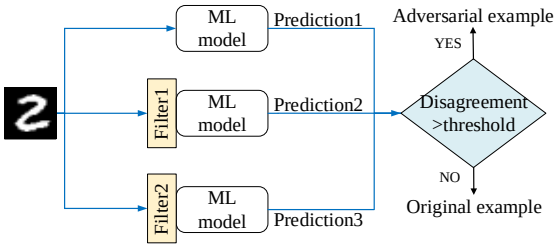


Fig. 3. How the parallel-filter-based defense works

Figure 3 reveals the working principle of the parallel-filter-based defense method. It works by detecting the adversarial examples. The intuition of this solution is that the model's softmax output vectors on the original example and the example after being filtered should be similar. In other words, if the original example and the example being filtered produce substantially different softmax output vectors, this example is likely to be adversarial. This solution compares the model's softmax output vector on the input image with its softmax output vectors on the image being filtered by different filters, and then calculates the disagreement between every two vectors. Let $\mathbf{Y}_1$ represent the softmax output vector of the model for the original image, $\mathbf{Y}_2$ represent the softmax output vector of the model for the image being filtered by Filter1, and $\mathbf{Y}_3$ represent the softmax output vector of the model for the image being filtered by Filter2. The disagreement can be measured by L_1 norm:

$$Dis(\mathbf{Y}_1, \mathbf{Y}_2) = \|\mathbf{Y}_1 - \mathbf{Y}_2\|_1. \tag{4}$$

We define D_{max} as

$$D_{max} = \max\{Dis(\mathbf{Y}_1, \mathbf{Y}_2), Dis(\mathbf{Y}_2, \mathbf{Y}_3), Dis(\mathbf{Y}_1, \mathbf{Y}_3)\}. \tag{5}$$

If D_{max} is greater than a specific threshold, this image is likely to be an adversarial example. Our goal is to design AEDescaptor to generate adversarial examples that can escape the parallel-filter-based defense. Given an input, the softmax output vector of the ML model is available. The knowledge of ML model details is not required.

B. AEDescaptor Design

As is shown in Figure 4, AEDescaptor adopts policy gradient algorithm to generate specific adversarial examples which can escape the parallel-filter-based defense. Given an input $\mathbf{X}^0$, the policy network outputs an $m \times n$ policy matrix $\mathbf{\Gamma}^t$ at the t-th training epoch. Each entry in $\mathbf{\Gamma}^t$ represents a probability. According to $\mathbf{\Gamma}^t$, the action generation (ActGen) algorithm can output an $m \times n$ action matrix $\mathbf{a}^t$ to generate adversarial example $\mathbf{X}^t$. Then, $\mathbf{X}^t$ is forwarded to the value network to get a softmax output vector $\mathbf{Y}_1^t$. $\mathbf{X}^t$ is also forwarded to filter1 and then is forwarded to the value network to get a softmax output vector $\mathbf{Y}_2^t$. Meanwhile, $\mathbf{X}^t$

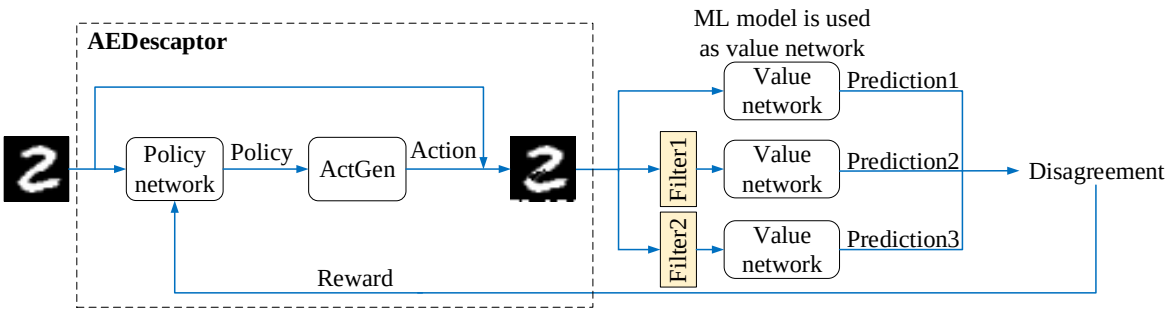


Fig. 4. Use AEDescaptor to escape the parallel-filter-based defense

is forwarded to filter2 and then is forwarded to the value network to get a softmax output vector $\mathbf{Y}_3^t$. These softmax output vectors can be used to compute a reward r^t, which can be applied to train the policy network.

Reward Design. The reward is designed as a weighted sum of the two factors f_3 and f_4. The first factor is described as $f_3 = -\sum_{i=1}^{3} \|\mathbf{Y}_i^t - \mathbf{Y}_{target}\|_1$, where $\mathbf{Y}_{target}$ is the target softmax output vector. The second factor measures the number of pixels modified by AEDescaptor. It is described as $f_4 = \|\mathbf{a}^t\|_0$. Therefore, the reward r^t is defined as

$$\mathrm{r}^t = -\sum_{i=1}^{3} \|\mathbf{Y}_i^t - \mathbf{Y}_{target}\|_1 - \delta\|\mathbf{a}^t\|_0, \tag{6}$$

where δ is a hyperparameter to balance attack performance and the number of perturbations. If all of $\mathbf{Y}_i^t$ are closer to the $\mathbf{Y}_{target}$, then the reward is larger. The larger f_3 also leads to a smaller D_{max}. The smaller D_{max} can help to escape the parallel-filter-based defense.

C. AEDescaptor Training

Next, we introduce how the policy network is trained according to the reward. The training procedure is shown in Algorithm III-C. Different from Algorithm 1, in lines 8-9, $\mathbf{X}^t$ is forwarded to three value networks to get three softmax output vectors $\mathbf{Y}^1$,$\mathbf{Y}^2$ and $\mathbf{Y}^3$, and then reward r^t can be calculated according to Equation (6).

After multiple epochs of training, AEDescaptor can finally output an optimized action to generate an adversarial example $\overline{\mathbf{X}}$ which can make the model get the wrong prediction. Meanwhile, the D_{max} is still less than the selected threshold, so as to escape the parallel-filter-based defense.

IV. Experiments

In this section, we evaluate the performance of AEDescaptor against the series-filter-based defense and the parallel-filter-based defense.

A. Setup

Dataset. MNIST dataset [22] is used in experiments. It consists of 60,000 training images and 10,000 test images,

Algorithm 2 AEDescaptor Training to Escape the Parallel-filter-based Defense

Input: Input image $\mathbf{X}^0$, Target softmax output vector $\mathbf{Y}_{target}$, Policy network's parameters θ;
Output: Adversarial example $\overline{\mathbf{X}}$;

1: Randomly initialize the policy network's parameters θ^0;
2: Normalize the pixel value of $\mathbf{X}^0$ to be between 0 and 1
3: **for** epoch $t = 1, 2, ...$ **do**
4: Feed $\mathbf{X}^0$ into the policy network;
5: The policy network outputs an $m \times n$ policy matrix $\mathbf{\Gamma}^t$;
6: According to $\mathbf{\Gamma}^t$, the ActGen algorithm outputs an $m \times n$ action matrix $\mathbf{a}^t$ and an $m \times n$ probability matrix $\mathbf{P}^t$;
7: Modify the pixels of the image $\mathbf{X}^0$ according to action $\mathbf{a}^t$ matrix by using Equation (1);
8: Input the generated example $\mathbf{X^t}$ into three value networks to get three softmax output vectors $\mathbf{Y}^1$,$\mathbf{Y}^2$, and $\mathbf{Y}^3$;
9: Calculate reward r^t according to Equation (6);
10: Update the policy network's parameters θ: $\theta^{t+1} \leftarrow \theta^t + \alpha \nabla_{\theta^t} \log \pi_{\theta^t}(\mathbf{s}^t, \mathbf{a}^t)\mathrm{r}^t$;
11: **end for**
12: **return** $\mathbf{X}^t$;

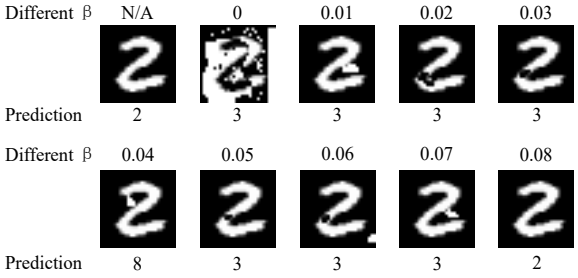


Fig. 5. AEDescaptor-generated adversarial examples on different β values

TABLE II
POLICY NETWORK ARCHITECTURE

Layer	Configuration
Convolution+ReLU+BatchNorm	3×3 × 32
Convolution+ReLU+BatchNorm	3×3 × 64
Convolution+ReLU+BatchNorm	3×3 × 128
Convolution+ReLU+BatchNorm	3×3 × 128
Convolution+ReLU+BatchNorm	3×3 × 64
Convolution+ReLU+BatchNorm	3×3 × 32
Convolution+ReLU+BatchNorm	3×3 × 16
Convolution+BatchNorm	3×3 × 2
Softmax	2

and each image is a handwritten grey-scale digital image with 28×28 pixels.

Policy Network Architecture. Table II shows the architecture of the policy network. In the policy network, the input size is $1 \times 28 \times 28$, and the output size is $2 \times 28 \times 28$. The policy network consists of convolutional layers and batch normalization layers. The kernel size in convolutional layers is 3.

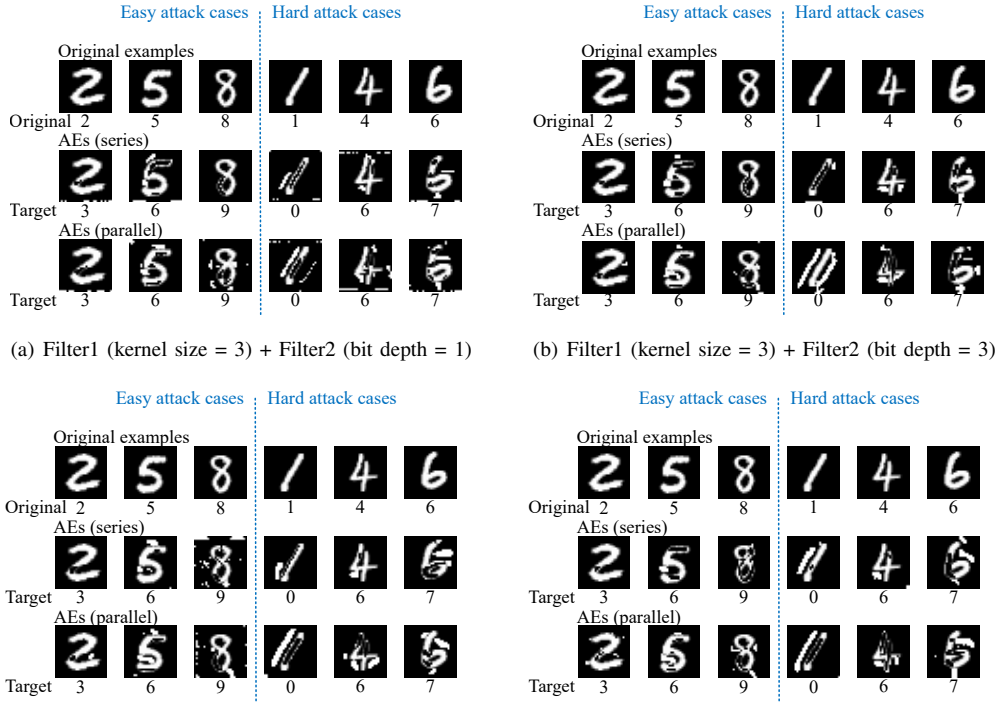
Value Network Architecture. The adopted value network is the conventional neural network (CNN) used in the Cleverhans library [23]. This network is named CleNet in this paper. It has 4 layers and is optimized by using Adam with a learning rate of 0.0001. The epoch is 500 and the batch size is 64.

Filters. We use two kinds of filters with different parameters for experiments, including median filter and bit depth filter. The median filter is used to calculate the median of several neighboring pixels of a certain pixel. The calculated median is used to replace the original pixel. A sliding square window is usually used to identify the neighboring pixels. The length of the square windows is called kernel size. The bit depth filter is used to compresses the image to a specific bit depth. For example, each pixel of an 8-bit depth gray-scale image in MNIST dataset has 2^8 = 256 possible values, where 0 is totally black, 255 is totally white, and other values represent varying degrees of gray shading.

Metrics. We consider two groups of test cases. The first group is easy attack cases (2→3, 5→6, 8→9). 2→3 means the original label is 2, and the target label (that AEDescaptor aims to mislead model to classify) is 3. The second group is hard attack cases (1→0, 4→6, 6→7). The performance of AEDescaptor against both the series-filter-based defense and the parallel-filter-based defense are measured. For the series-filter-based defense, we use the confidence C (i.e., the probability of an input image is classified as the target label in an adversarial example attack) to represent the AEDescaptor's performance. For the parallel-filter-based defense, we use D_{max} (Equation (5)) to represent the AEDescaptor performance. In case of a successful AE attack, the smaller D_{max}, the higher AEDescaptor's performance.

B. AEDescaptor Performance on different β

First, we test the performance of hyperparameter β in the generation of adversarial examples. The kernel size of the median filter is set to 3. The bit depth of the bit depth filter is set to 1. In Figure 5, adversarial ex-



(a) Filter1 (kernel size = 3) + Filter2 (bit depth = 1)

(b) Filter1 (kernel size = 3) + Filter2 (bit depth = 3)

(c) Filter1 (kernel size = 5) + Filter2 (bit depth = 1)

(d) Filter1 (kernel size = 5) + Filter2 (bit depth = 3)

Fig. 6. AEDeacaptor-generated adversarial examples on four different filter parameter settings

amples generated by AEDescaptor are shown at $\beta = \{0, 0.01, 0.02, 0.03, 0.04, 0.05, 0.06, 0.07, 0.08\}$. It should be noted that the first example with N/A is an original example labeled 2 rather than a generated adversarial example.

We observe that, compared with the original example, the generated adversarial example with $\beta = 0$ has large number of perturbations. As the hyperparameter β becomes larger, the generated adversarial example is closer to the original example, which still makes the model get wrong predictions. When β is greater than 0.08, the perturbation added to the original example disappears and the adversarial attack fails.

In conclusion, the hyperparameter β has a significant effect on balancing attack performance and the number of perturbations.

C. AEDescaptor Performance on Different Filter Parameters

We test the performance of AEDescaptor under different filter parameter settings. The kernel size of the median filter is set to 3 and 5. The bit depth of the bit depth filter is set to 1 and 3. Therefore, there are four types of combinations of filter parameter settings.

The experimental results are shown in Figure 6. The first row exhibits the original images. The second row shows the adversarial examples generated by AEDescaptor in series-filter-based defense case. The third row shows the adversarial examples generated by AEDescaptor in parallel-filter-based defense case. The number below each adversarial example represents the target label.

We have two observations. First, AEDescaptor can successfully generate adversarial examples to escape both the series-filter-based defense and the parallel-filter-based defense. The attack success rate is 100%. If only a machine-based automatically filter-based defense system is deployed, AEDescaptor can circumvent such defense system. Second, there is at least one attack case (i.e., $2 \rightarrow 3$) that the adversarial examples are hard to be noticed by humans eyes. In other five attack cases, the adversarial examples have relatively large modifications (over the original examples). They may be detected by humans eyes. Note that manual check by eyes is usually impractical in real-world applications (especially for large datasets). Even if there are one-by-one manual adversarial example checks, AEDescaptor may still escape the filter-based defense when launching some easy attack cases. In a nutshell, AEDescaptor significantly undermines the validity of filter-based defenses.

D. Transferability

It has been noted that some adversarial examples generated for model A may also be misclassified by another model B (that has the same function as model A). Such a property is referred to as cross-model transferability, which can be used to perform black-box attacks. In this experiment, we use AEDescaptor trained over CleNet and test its transferability on other three popular classifiers (LeNet [24], AlexNet [25], and VGGNet [26]).

- **LeNet.** The LeNet network has 7 layers. Its architecture can be found in [24]. It is trained by using stochastic gradient descent (SGD) with a learning rate of 0.001. The epoch is 100 and the batch size is 64.
- **AlexNet.** The AlexNet network has 8 layers. Its architecture can be found in [25]. It is trained by using SGD with a learning rate of 0.001. The epoch is 100 and the batch size is 64.
- **VGGNet.** The VGGNet network has 16 layers. Its architecture can be found in [26]. It is trained by using Adam with a learning rate of 0.001. The epoch is 100 and the batch size is 64.

Tables III-VIII show the transferability performance of AEDescaptor-generated adversarial examples on different classifiers. **S** means the series-filter-based defense case. If the confidence of the target label is the largest among all labels, then the attack succeeds. **P** means the parallel-filter-based defense case. If the disagreement D_{max} is less than a specific threshold 0.3076 (as used in [9]), then the attack succeeds. N/A indicates that the prediction label of the adversarial example is inconsistent with the target label, so the attack fails. We find that in the series-filter-based defense case, most AEDescaptor-generated AEs succeed, and they have high confidence C. In the parallel-filter-based defense case, most AEDescaptor-generated AEs can mislead the model, and the corresponding D_{max} is very small.

Table IX shows that AEDescaptor-generated adversarial examples trained using CleNet have a 100% attack success rate on itself. The attack success rates on other three classifiers are 16.67%-66.67%. All successful attacks have high averarge confidence C (i.e., $\geq$ 98.71%) and relatively small average D_{max} (≤ 0.178). It can be observed that the deeper the classifier, the lower the attack success rate (i.e., the lower cross-model transferability capability). In summary, AEDescaptor has a good cross-model transferability capability, especially on non-deep classifiers.

V. Related Work

In this section, we review the state-of-the-art in adversarial example attacks, adversarial example defenses, and reinforcement learning, respectively.

Adversarial Example Attacks. The concept of adversarial example is first proposed by Szegedy et al. [27]. According to different threat models, adversarial example attack algorithms can be divided into white-box attacks [18], [28], [29] and black-box attacks [30], [31]. White-box attackers can obtain network model structure, parameters, defense mechanism, and sometimes even training set. Black-box attackers can only obtain the corresponding output according to the neural network input, without the knowledge of the structure and parameters of the network model. In white-box attack cases, Goodfellow et al. [18] apply linear features to nonlinear models and propose the fast gradient sign method (FGSM). Carlini and Wagner [28] propose three optimized C&W attack algorithms of L_0, L_2 and L_∞, which can achieve an almost 100% attack success rate. Kurakin et al. [29] propose basic iterative method (BIM), which is an extension of FGSM that divides single step into small steps to iteration. Chen et al. [30] propose a

TABLE III
ATTACK CASE ($2 \rightarrow 3$)

		Adversarial example	Succeed?
CleNet	**S**	C=99.99+%	✔
	P	D_{max}=0	✔
LeNet	**S**	C=99.99+%	✔
	P	D_{max}=0	✔
AlexNet	**S**	C=99.99+%	✔
	P	D_{max}=0	✔
VGGNet	**S**	C=0.04%	✗
	P	N/A	✗

TABLE IV
ATTACK CASE ($5 \rightarrow 6$)

		Adversarial example	Succeed?
VNet	**S**	C=99.99+%	✔
	P	D_{max}=0	✔
LeNet	**S**	C=2.12%	✗
	P	D_{max}=0.0005	✔
AlexNet	**S**	C=94.49%	✔
	P	D_{max}=0.2718	✔
VGGNet	**S**	C=0%	✗
	P	D_{max}=0	✔

TABLE V
ATTACK CASE ($8 \rightarrow 9$)

		Adversarial example	Succeed?
CleNet	**S**	C=99.99+%	✔
	P	D_{max}=0	✔
LeNet	**S**	C=99.94%	✔
	P	D_{max}=0.0963	✔
AlexNet	**S**	C=99.99+%	✔
	P	N/A	✗
VGGNet	**S**	C=0.67%	✗
	P	N/A	✗

TABLE VI
ATTACK CASE ($1 \rightarrow 0$)

		Adversarial example	Succeed?
CleNet	**S**	C=99.99+%	✔
	P	D_{max}=0	✔
LeNet	**S**	C=99.99+%	✔
	P	D_{max}=0	✔
AlexNet	**S**	C=99.99+%	✔
	P	D_{max}=0	✔
VGGNet	**S**	C=0.04%	✗
	P	N/A	✗

TABLE VII
ATTACK CASE ($4 \rightarrow 6$)

		Adversarial example	Succeed?
VNet	**S**	C=99.99+%	✔
	P	D_{max}=0	✔
LeNet	**S**	C=2.12%	✗
	P	D_{max}=0.0005	✔
AlexNet	**S**	C=94.49%	✔
	P	D_{max}=0.2718	✔
VGGNet	**S**	C=0%	✗
	P	D_{max}=0	✔

TABLE VIII
ATTACK CASE ($6 \rightarrow 7$)

		Adversarial example	Succeed?
CleNet	**S**	C=99.99+%	✔
	P	D_{max}=0	✔
LeNet	**S**	C=99.94%	✔
	P	D_{max}=0.0963	✔
AlexNet	**S**	C=99.99+%	✔
	P	N/A	✗
VGGNet	**S**	C=0.67%	✗
	P	N/A	✗

zero-order optimization method based on confidence. In black-box attack cases, Su et al. [31] propose a single-pixel attack based on a differential evolution algorithm, which iteratively modifies single pixel to generates a sub image, and retains the sub image with the best attack effect according to the selection criteria to realize the adversarial attack.

Different from these works, this paper focuses on developing approaches to launch attacks in the presence of filters prepended to ML model.

Adversarial Example Defenses. In recent years, researchers have proposed many defense methods against adversarial examples. Xu et al. [9] propose a defense method based on filters to defense adversarial examples, and shows that their method also has acceptable resistance to C&W attack [28]. Xie et al. [32] develop a new convolution network architecture, which includes a denoising module for feature image denoising that can significantly improve the robustness of the model. Mustafa et al. [33] propose an enhanced defense method based on super-resolution [34] and wavelet denoising [35]. Sankaranarayanan et al. [36] present a novel approach to regularize deep neural networks by perturbing intermediate layer activations in an efficient manner. Akhtar et al. [37] present the first dedicated framework to effectively defend the networks against such perturbations, which learns a Perturbation Rectifying Network (PRN) as 'pre-input' layers to a targeted model, such that the targeted model needs no modification. Raff et al. [38] explore the idea of stochastically combining a large number of individually weak defenses into a single barrage of randomized transformations to build a strong defense against adversarial attacks.

In this paper, we aim to generate adversarial example to escape the filter-based defenses.

Reinforcement Learning. In recent years, RL, as a branch of machine learning, has been widely used in computer vision [39], [40], parameter optimization [41], [42], robot control [43], [44], etc. In 1989, Watkins et al. [45] proposed Q-learning algorithm which is one of the most important algorithms of reinforcement learning. Mnih et al. [46] combine convolutional neural networks with Q-learning algorithm, and propose deep q-network (DQN) model that is used to deal with control tasks based on visual perception. The algorithms described above are all value-based algorithms that need to find the value function, and then select the action according to the value function. Others are policy-based algorithms. Sutton et al. [21] propose the policy gradient algorithm which can directly approximate policy and optimize policy, and finally, get the optimal policy. Silver et al. [47] propose an effective deterministic policy gradient algorithm, which has

TABLE IX
THE ATTACK SUCCESS RATES OF ADVERSARIAL EXAMPLES (TRAINED OVER CLENET) ON FOUR CLASSIFIERS. AVG. C MEANS THE AVERAGE CONFIDENCE IN CASE OF SUCCESSFUL ATTACKS. AVG. D_{max} MEANS THE AVERAGE D_{max} IN CASE OF SUCCESSFUL ATTACKS

	S		**P**	
	Success rate	**Avg.** C	**Success rate**	**Avg.** D_{max}
CleNet	100%	99.99%	100%	0
LeNet	66.67%	98.71%	66.67%	0.0047
AlexNet	66.67%	99.65%	66.67%	0.1780
VGGNet	16.67%	100%	20.83%	0.0546

better performance in high-dimensional action space.

VI. Conclusion

In this paper, we have developed AEDescaptor to generate adversarial examples to successfully escape both the series-filter-based defense and the parallel-filter-based defense. The key innovation of this paper is to use specially-crafted policy gradient RL to generate adversarial examples even if filters are used to interrupt the backpropagation channel (that is used in traditional adversarial generation algorithms). Our intensive experimental results show that AEDescaptor-generated adversarial examples have good performance (in terms of success rate and transferability) to escape filter-based defenses. Our study has paved the way for subsequent research on RL-based adversarial example generation.

References

[1] Z. Xie, Z. Zhang, X. Zhu, G. Huang, and S. Lin, "Spatially adaptive inference with stochastic feature sampling and interpolation," in *Proc. ECCV*, 2020, pp. 531-548.

[2] T. Yang, S. Zhu, C. Chen, S. Yan, M. Zhang, and A. Willis, "Mutualnet: Adaptive convnet via mutual learning from network width and resolution," in *Proc. ECCV*, 2020, pp. 299–315.

[3] X. Li, A. You, Z. Zhu, H. Zhao, M. Yang, K. Yang, S. Tan, and Y. Tong, "Semantic flow for fast and accurate scene parsing," in *Proc. ECCV*, 2020, pp. 775–793.

[4] G. Sun, W. Wang, J. Dai, L. Van Gool, "Mining cross-image semantics for weakly supervised semantic segmentation," in *Proc. ECCV*, 2020, pp. 347–365.

[5] X. Zhao, Y. Pang, L. Zhang, H. Lu, and L. Zhang, "Suppress and balance: A simple gated network for salient object detection," in *Proc. ECCV*, 2020, pp. 35–51.

[6] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko, "End-to-end object detection with transformers," in *Proc. ECCV*, 2020, pp. 213–229.

[7] K. Eykholt, I. Evtimov, E. Fernandes, B. Li, A. Rahmati, C. Xiao, A. Prakash, T. Kohno, and D. Song, "Robust physical-world attacks on deep learning models," in *Proc. CVPR*, 2018, pp. 1625-1634.

[8] B. Liang, H. Li, M. Su, X. Li, W. Shi, and X. Wang, "Detecting adversarial image examples in deep neural networks with adaptive noise reduction," *IEEE Trans . Dependable Secure Comput.*, vol. 18, no. 1, pp. 72–85, 2021.

[9] W. Xu, D. Evans, and Y. Qi, "Feature squeezing: Detecting adversarial examples in deep neural networks," in *NDSS*, 2018, pp. 1-15.

[10] X. Li, F. Li, "Adversarial examples detection in deep networks with convolutional filter statistics," in *Proc. ICCV*, 2017, pp. 5764–5772.

[11] J.H. Metzen, T. Genewein, V. Fischer, B. Bischoff, "On detecting adversarial perturbations," in *Proc. ICLR*, 2017, pp. 1-12.

[12] W. He, J. Wei, X. Chen, N. Carlini, D. Song, "Adversarial example defense: Ensembles of weak defenses are not strong," in *USENIX WOOT*, 2017, pp. 1-15.

[13] M. Osadchy, J. Hernandez-Castro, S. Gibson, O. Dunkelman, D. Perez-Cabo, "No Bot Expects the DeepCAPTCHA! Introducing Immutable Adversarial Examples, With Applications to CAPTCHA Generation," *IEEE Trans. Inf. Foren. Sec.*, vol. 12, no. 11, pp. 2640–2653, 2019.

[14] E. Quiring, D. Klein, D. Arp, M. Johns, K. Rieck, "Adversarial preprocessing: Understanding and preventing image-scaling attacks in machine learning," in *USENIX Security*, 2020, pp. 1363–1380.

[15] Y. Song, T. Kim, S., Ermon, S. Nowozin, N. Kushman, "Pixeldefend: Leveraging generative models to understand and defend against adversarial examples," in *Proc. ICLR*, 2018, pp. 1-20.

[16] B. Wang, Y. Yao, S. Shan, H. Li, B. Viswanath, H. Zheng, B.Y. Zhao, "Neural cleanse: Identifying and mitigating backdoor attacks in neural networks," in *Proc. S&P*, 2019, pp. 707–723.

[17] Q. Xiao, Y. Chen, C. Shen, Y. Chen, K. Li, "Seeing is not believing: Camouflage attacks on image scaling algorithms," in *USENIX Security*, 2019, pp. 443–460.

[18] I. J. Goodfellow, J. Shlens, C. Szegedy, "Explaining and harnessing adversarial examples," in *Proc. ICLR*, 2015, pp. 1-11.

[19] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, A. Vladu, "Towards deep learning models resistant to adversarial attacks," in *Proc. ICLR*, 2018, pp. 1-23.

[20] Y. Dong, F. Liao, T. Pang, H. Su, J. Zhu, X. Hu, J. Li, "Boosting adversarial attacks with momentum," in *Proc. CVPR*, 2018, pp. 9185–9193.

[21] R. S. Sutton, D. McAllester, S. Singh, Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," *Proc. NeurIPS*, 1999, pp. 1057-1063.

[22] Y. LeCun, C. Cortes, C. J.C. Burges, "The mnist database of handwritten digits," http://yann.lecun.com/exdb/mnist/, 1998.

[23] N. Papernot, I. Goodfellow, R. Sheatsley, R. Feinman, P. Mcdaniel, "CleverHans v1.0.0: an adversarial machine learning library," Technical Report, 2016.

[24] Y. Lecun, L. Bottou, "Gradient-based learning applied to document recognition," in *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[25] A. Krizhevsky, I. Sutskever, G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Proc. NeurIPS*, 2012, pp. 1-9.

[26] K. Simonyan, A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *Proc. ICLR*, 2015.

[27] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R.Fergus, "Intriguing properties of neural networks," in *Proc. ICLR*, 2014, pp. 1-10.

[28] N. Carlini, D. Wagner, "Towards evaluating the robustness of neural networks," in *Proc. S&P*, 2017, pp. 39–57.

[29] A. Kurakin, I. Goodfellow, S. Bengio, "Adversarial machine learning at scale," *Proc. ICLR*, 2017, pp. 1-17.

[30] P. Y. Chen, H. Zhang, Y. Sharma, J. Yi, C. J. Hsieh, "Zoo: Zeroth order optimization based black-box attacks to deep neural networks without training substitute models," in *Proc. AISec*, 2017. pp. 15–26.

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

[32] C. Xie, Y. Wu, L. Maaten, A. L. Yuille, K. He, "Feature denoising for improving adversarial robustness," in *Proc. CVPR*, 2019, pp. 501–509.

[33] A. Mustafa, S. H. Khan, M. Hayat, J. Shen, L. Shao, "Image super-resolution as a defense against adversarial attacks," *IEEE Trans. Image Process.*, vol. 29, pp. 1711–1724, 2019.

[34] B. Lim, S. Son, H. Kim, S. Nah, K.M. Lee, "Enhanced deep residual networks for single image super-resolution," in *Proc. CVPR Workshops*, 2017.

[35] S. G. Chang, B. Yu, M. Vetterli, "Adaptive wavelet thresholding for image denoising and compression," *IEEE Trans. Image Process.*, 2000, pp. 1532–1546.

[36] S. Sankaranarayanan, A. Jain, R. Chellappa, S.N. Lim, "Regularizing deep networks using efficient layerwise adversarial training." in *Proc. AAAI*, 2018, pp. 4008-4015.

[37] N. Akhtar, J. Liu, A. Mian, "Defense against universal adversarial perturbations." in *Proc. CVPR*, 2018, pp. 3389-3398.

[38] E. Raff, J. Sylvester, S. Forsyth, M. McLean, "Barrage of random transforms for adversarially robust defense." in *Proc. CVPR*, 2019, pp. 6528-6537.

[39] J. C. aicedo, S. Lazebnik, "Active object localization with deep reinforcement learning," in *Proc. ICCV*, 2015, pp. 2488–2496.

[40] J. Oh, X. Guo, H. Lee, R. L. Lewis, S. Singh, "Action-conditional video prediction using deep networks in Atari games," *Proc. NeurIPS*, 2015, pp. 2863–2871.

[41] M. Andrychowicz, M. Denil, S. Gomez, M.W. Hoffman, D. Pfau, T. Schaul, B. Shillingford, N. De Freitas, "Learning to learn by gradient descent by gradient descent," *Proc. NeurIPS*, 2016, pp. 1-9.

[42] S. Hansen, "Using deep q-learning to control optimization hyperparameters," *arXiv preprint arXiv:1602.04062*, 2016.

[43] C. Inn, S. Levine, P. Abbeel, "Guided cost learning: Deep inverse optimal control via policy optimization," in *Proc. ICML*, 2016. pp. 49–58.

[44] S. Levine, C. Finn, T. Darrell, P. Abbeel, "End-to-end training of deep visuomotor policies," *J. Mach. Learn. Res.*, 2016, pp. 1334–1373.

[45] C. J.C.H. Watkins, "Learning from delayed rewards," PhD Thesis, 1989.

[46] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, M. Riedmiller, "Playing atari with deep reinforcement learning," *Proc. NeurIPS*, 2013.

[47] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, M. Riedmiller, "Deterministic policy gradient algorithms," in *Proc. ICML*, 2014, pp. 387–395.