A SUPPLEMENTAL MATERIALS

A.1 Datasets

We use the whole dataset of Fashion-MNIST [34] and CIFAR-10 [15], but select images of 20 object classes from ImageNet [5] as a subset for our experiments. All pixels are projected into [0, 1].

A.2 Model Structures

Here we report the model structures in our experiments. For Fashion-MNIST and CIFAR-10, VGG11[23] structure is applied to the target classifier, detector and rectifier in X-Ensemble, while VGG16[23] is for ImageNet experiments. The black-box classifiers for Fashion-MNIST , CIFAR-10 and ImageNet are contructed by CWnet[4], Wide-ResNet[22] and ResNet152[13] respectively.

Since it is very computational resource consuming to train a VGG16 and an ResNet152 model for ImageNet, we adopt the pre-trained VGG16 as the target classifier and the pre-trainded ResNet152 as the black-box classifier. When using these models for ImageNet, we select the output logits of those 20 chosen classes out of the 1000 categaries and then compute the probabilities for them with $softmax$ function. In this way, the pre-trained VGG16 and ResNet152 both have more than 95% accuracy on the ImageNet subset.

A.3 Setting of Attackers

The codes of all attack methods in our experiments are implemented by AdverTorch [1] v0.2.

The ϵ in FGSM and PGD is to constrain the L_∞ perturbation of adversarial examples; the α in PGD and the learning rate (lr) in CW control the step size in each iteration when searching the adversarial perturbation; the iteration (I) limits how many times that an iterative attack can compute. $\epsilon = 0.031$, $\alpha = 0.0078$ and $lr = 0.01$ for all the three datasets. I is set to 100 for Fashion-MNIST and CIFAR-10, 50 for ImageNet.Other parameters use their default values in AdverTorch.

OnePixel attack is only evaluated in unvaccinated training. We set the pixel numbers to 3, which means the attack can only modify 3 pixel values in an image. And its iteration is set to 30, with 100 candidates in each iteration.

A.4 Interpretation Methods

VG, GBP and IG methods are implemented on our own and use LRP code from [20]. The integrated step in IG is set to 50. Note that LRP from [20] cannot be applied on ResNet152 directly. So we remove the LRP detector only for ImageNet.

A.5 Details of Rectifier

We use Alg. 1 to compute rectified images on adversarial examples and the hyperparameter α is set 0.6, 0.9, 0.5 for Fashion-MNIST, CIFAR-10 and ImageNet respectively.And we find that a rectifier tuned with rectified images of DDN-T mixed with clean images has a better performance.

A.6 Baselines

In this section, we briefly introduce the sources of four baselines used in our work.

• **PD.** The code is from [24]. When purifying images, ϵ is set to 0.125 for Fashion-MNIST and 0.063 for both CIFAR-10 and ImageNet.

Algorithm 1 Rectified Image For Tuning Rectifier

Variables: $\{D_1, ..., D_j\}$ are the sub-detectors that predict an input image x as an adversarial one, $\{R_1, ..., R_j\}$ are the interpreting methods corresponding to $\{D_1, ..., D_j\}$ respectively, $\alpha \in (0, 1)$ is a threshold parameter, $rand()$ returns a random value in $[0, 1]$, and σ is the variance of pixel values in x.

for $k = 1$ to j **do**
  $E_k \leftarrow Entropy(D_k(x))$
end for
$R \leftarrow R_i$ where $i = argmin(E_1, ..., E_j)$
$g \leftarrow R(x)$
$thres \leftarrow \alpha * (\max(g) - \min(g)) + \min(g)$
for ixel (i, j) in x **do**
  if $g_{i,j} > thres$ **and** $rand() > 0.5$ **then**
    $x_{i,j} \leftarrow x_{i,j} + Normal(0, \sigma)$
  end if
end for
return x

The pretrained model of PixelCNN for CIFAR-10 is from $openai$[2], and we train PixelCNN on Fashion-MNIST and ImageNet with code from $openai$ [2].

• **TWS.** The code is from [14]. We set the parameters n_radius = 0.01, targeted_lr = 0.0005, t_radius = 0.5, u_radius = 0.5 and untargeted_lr = 1.

• **MHL.** The code is from [16]. The magnitude of noise starts from 0.05 to 0.3 with an interval of 0.05 to compute its AUC.

• **TVM.** The code is from [10]. We set TVM_WEIGHT = 0.03, PIXEL_DROP_RATE = 0.5, TVM_METHOD = 'tvl2'.

• **Adversarial Training** Adversarial training is computational expensive. To speed up the training process while achieving a good accuracy, we use adversarial data to fine tune the original classifier with 15 epochs.

A.7 White-box Attacker for X-Ensemble

In this section, we show the details of the end-to-end white-box evaluation in our experiments.

Carlini *et al.* [3] proposed a variant of CW attack to combine a classifier and a neural network detector into a new classifier G with L+1 classes, whose $(L+1)^{th}$ class is for adversarial inputs. In this way, an attacker can directly attack G to break F and D at the same time. G is defined as,

$$G(x)_i = \begin{cases} F(x)_i & \text{if } i \le L \\ (D(x)+1) \cdot max_j F(x)_j & \text{if } i = L+1 \end{cases} \tag{19}$$

where if x is adversarial then $D(x) > 0$, so we have $G(x)_{L+1} > max(F(x))$ and $argmax_i G(x)_i = L + 1$; if x is clean then $D(x) < 0$, so we have $argmax_i G(x)_i = argmax_i F(x)_i$. An attacker will optimize this joint objective (target $t \neq l$ and $t \neq L+1$) to construct adversarial examples on G.

In our experiment, an attacker needs to fool all the detectors in X-Ensemble. So we modify G into an L+4 classifier as,

$$G(x)_i = \begin{cases} F(x)_i & \text{if } i \le L \\ (D_k(x)+1) \cdot max_j F(x)_j & \text{if } i = L+k \end{cases} \tag{20}$$

where D_k is one part of X-DET and $k = 1, 2, 3, 4$. Notice that here we remove the LRP detector since it is not differentiable in the second-order. So that a targeted attacker can generate examples on the classifier and the detectors to perform a white-box attack.

[1] https://github.com/BorealisAI/advertorch

[2] https://github.com/openai/pixel-cnn