# Supplementary Material

## A. Baselines Details

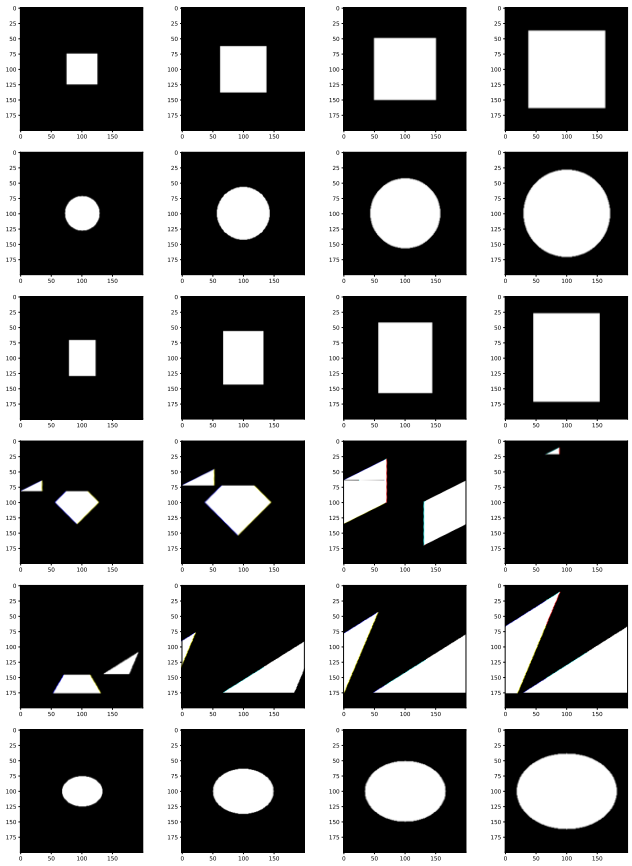
For JPEG compression [1], we set the quality parameter to 50. For spatial smoothing, we use window size 3. For LGS [2], we set the block size to 30, overlap to 5, threshold to 0.1, and smoothing factor to 2.3. For AT, we use PGD attacks with 30 iterations and step size 0.067, which takes around twelve hours per epoch on the xView training set and thirty-two hours on COCO using ten GPUs. We use SGD optimizers with an initial learning rate of 0.01, momentum 0.9, weight decay $5 \times 10^{-4}$, and batch size 10. We train each model with ten epochs. There is a possibility that the AT models would perform better if we train them longer or tune the training hyper-parameters. However, we were unable to do so due to the extremely expensive computation needed.

## B. SAC Details

### B.1. Training the Patch Segmenter

**COCO and xView datasets** We use U-Net [3] with 16 initial filters as the patch segmenter on the COCO and xView datasets. To train the patch segmenters, for each dataset we generate 55k fixed adversarial images from the training set with a patch size $100 \times 100$ by attacking base object detectors, among which 50k are used for training and 5k for validation. We randomly replace each adversarial image with its clean counterpart with a probability of 30% during training to ensure good performance on clean data. All images are cropped to squares and resized to $500 \times 500$ during training. We use RMSprop [4] optimizer with an initial learning rate of $10^{-4}$, momentum 0.9, weight decay $10^{-8}$, and batch size 16. We train patch segmenters for five epochs and evaluate them on the validation set five times in each epoch. We reduce the learning rate by a factor of ten if there is no improvement after two evaluations. For self adversarial training, we train each model for one epoch with $\lambda = 0.3$ using PGD attacks with 200 iterations and step size $\alpha = 0.01$, which takes around eight hours on COCO and four hours on xView using ten GPUs.

**APRICOT dataset** Detecting adversarial patches in the physical world can be more challenging, as the shape and appearance of patches can vary a lot under different viewing angles and lighting conditions. For the APRICOT dataset, We use U-Net [3] with 64 initial filters as the patch segmenter. We downscale each image by a factor of two during training and evaluation to save memory as each image is approximately 12 megapixels (*e.g.*, $4000 \times 3000$ pixels). We use 85% of the APRICOT test set (742 images) as the training set, and the rest (131 images) as the validation set. During training, we randomly crop $500 \times 500$ image patches from the downscale images, with a probability of $60\%$ that an image patch contain an adversarial patch and $40\%$ that it contain no patch. We use RMSprop [4] optimizer with an initial learning rate of $10^{-3}$, momentum 0.9, weight decay $10^{-8}$, and batch size 24. We train patch segmenters for 100 epochs, and reduce the learning rate by a factor of ten if the dice score on the validation set has no improvement after 10 epochs. After training, we pick the checkpoint that has the highest dice score on the validation set as our final model. The training takes around four hours on six gpus.



(a) $50 \times 50$ (b) $75 \times 75$ (c) $100 \times 100$ (d) $125 \times 125$

Figure 1. Different shapes used for evaluating SAC. From top to bottom: square, circle, rectangle, diamond, triangle, and ellipse. Shapes in each column have approximately the same $n \times n$ pixels, where $n \in \{50, 75, 100, 125\}$.

### B.2. Different Patch Shapes for Evaluating SAC

In the main paper, we demonstrate generalization to unseen patch shapes that were not considered in training the patch segmenter and in shape completion, obtaining surprisingly good robust performance. The shapes used for evaluating SAC are shown in Fig. 1.

### B.3. Evaluate SAC with SSD

We use Faster R-CNN [5] as our base object detector in the main paper. However, SAC is compatible with any object detector as it is a pre-processing defense. In this section, we show the performance of SAC using SSD [6] as the base object detector on the COCO dataset. The pre-trained SSD model is provided in `torchvision` [7]. We do not re-train the patch segmenter for SSD as the self adversarial training on the patch segmenter is object-detector agnostic. Results shown in Table 1 demonstrate that SAC can also provide strong robustness for SSD across different attack methods and patch sizes.

Table 1. mAP (%) under different attack methods using SSD as the base object detector. The mAP on clean images is 44.5% for the undefended model and 44.4% for the SAC defended model.

| Attack | Method | 75×75 | 100×100 | 125×125 |
|---|---|---|---|---|
| PGD [8] | Undefended | 18.3±0.4 | 11.4±0.2 | 7.0±0.1 |
| | SAC (**Ours**) | 39.1±0.3 | 38.8±0.2 | 34.2±0.1 |
| DPatch [9] | Undefended | 21.5±0.8 | 16.9±0.2 | 12.5±0.6 |
| | SAC (**Ours**) | 39.9±0.2 | 39.1±0.1 | 35.4±0.3 |
| MIM [10] | Undefended | 17.6±0.5 | 10.4±0.2 | 6.0±0.2 |
| | SAC (**Ours**) | 37.9±0.2 | 38.5±0.1 | 35.0±0.3 |

### B.4. Shape Completion Details

#### B.4.1 Dynamic Programming for Shape Completion

Recall that our shape-completed mask is defined as:

$$\hat{M}_{SC\,(i,j)} := \begin{cases} 1 & \text{if } \exists i',\, j' :\; M^{s,(i',j')}_{(i,j)} = 1 \text{ and} \\ & \frac{d_H(\hat{M}_{PS}, M^{s,(i',j')})}{s^2} \leq \gamma \\ 0 & \text{otherwise.} \end{cases} \tag{1}$$

Here, we give a dynamic-programming based $O(H \times W)$ time algorithm for computing this mask.

We first need to define the following $O(H \times W)$ time subroutine: for an $H \times W$ binary matrix $M$, let Cuml.$(M)$ be defined as follows:

$$\text{Cuml.}(M)_{(i,j)} := \sum_{i'=1}^{i} \sum_{j'=1}^{j} M_{(i',j')} \tag{2}$$

The entire matrix Cuml.$(M)$ can be computed in $O(H \times W)$ as follows. We first define Cuml.$^x(M)$ as:

$$\text{Cuml.}^x(M)_{(i,j)} := \sum_{i'=1}^{i} M_{(i',j)} \tag{3}$$

Note that Cuml.$^x(M)_{(1,j)} = M_{(1,j)}$ and that, for $i > 1$,

$$\text{Cuml.}^x(M)_{(i,j)} := M_{(i,j)} + \text{Cuml.}^x(M)_{(i-1,j)} \tag{4}$$

We can then construct Cuml.$^x(M)$ row-by-row along the index $i$, with each cell taking constant time to fill: therefore Cuml.$^x(M)$ is constructed in $O(H \times W)$ time. Cuml.$(M)$ can then be constructed through two applications of this algorithm as:

$$\text{Cuml.}(M) = (\text{Cuml.}^x((\text{Cuml.}^x(M))^T))^T \tag{5}$$

We now apply this algorithm to $\hat{M}_{PS}$ :

$$\text{Cuml}\hat{M}_{PS} := \text{Cuml.}(\hat{M}_{PS}). \tag{6}$$

Note that, for each $i, j$:

$$\begin{aligned}
&d_H(\hat{M}_{PS}, M^{s,(i,j)}) \\
&= \sum_{\substack{i' \in [i,i+s) \\ j' \in [j,j+s)}} (1 - \hat{M}_{PS,(i',j')}) + \sum_{\substack{i' \notin [i,i+s) \vee \\ j' \notin [j,j+s)}} \hat{M}_{PS,(i',j')} \\
&= s^2 - \sum_{\substack{i' \in [i,i+s) \\ j' \in [j,j+s)}} \hat{M}_{PS,(i',j')} + \sum_{\substack{i' \notin [i,i+s) \vee \\ j' \notin [j,j+s)}} \hat{M}_{PS,(i',j')} \\
&= s^2 + \sum_{(i',j')} \hat{M}_{PS,(i',j')} - 2 \sum_{\substack{i' \in [i,i+s) \\ j' \in [j,j+s)}} \hat{M}_{PS,(i',j')} \\
&= s^2 + \text{Cuml}\hat{M}_{PS(H,W)} - 2\Bigg( \sum_{\substack{i' \in [1,i+s) \\ j' \in [1,j+s)}} \hat{M}_{PS,(i',j')} - \\
&\sum_{\substack{i' \in [1,i) \\ j' \in [1,j+s)}} \hat{M}_{PS,(i',j')} - \sum_{\substack{i' \in [1,i+s) \\ j' \in [1,j)}} \hat{M}_{PS,(i',j')} + \sum_{\substack{i' \in [1,i) \\ j' \in [1,j)}} \hat{M}_{PS,(i',j')} \Bigg) \\
&= s^2 + \text{Cuml}\hat{M}_{PS,(H,W)} - 2\Big( \text{Cuml}\hat{M}_{PS,(i+s-1,j+s-1)} \\
&- \text{Cuml}\hat{M}_{PS,(i-1,j+s-1)} - \text{Cuml}\hat{M}_{PS,(i+s-1,j-1)} \\
&+ \text{Cuml}\hat{M}_{PS,(i-1,j-1)} \Big)
\end{aligned} \tag{7}$$

(We are disregarding edge cases where $i + s > H$ or $j + s > W$: these can be easily reasoned about.) Using a pre-computed Cuml$\hat{M}_{PS}$, we can then compute each of these Hamming distances in constant time. We can then, in $O(H \times W)$ time, compute the matrix $\hat{M}_\gamma$:

$$\hat{M}_{\gamma,(i,j)} := \mathbb{1}_{\frac{d_H(\hat{M}_{PS}, M^{s,(i,j)})}{s^2} \leq \gamma} \tag{8}$$

where $\mathbb{1}$ denotes an indicator function. We also pre-compute the cumulative sums of this matrix:

$$\text{Cuml}\hat{M}_\gamma := \text{Cuml.}(\hat{M}_\gamma) \tag{9}$$

Now, recall the condition of Eq. (1):

$$\begin{aligned}
&\exists\, i',\, j' :\ \ M^{s,(i',j')}_{(i,j)} = 1 \text{ and } \frac{d_H(\hat{M}_{PS}, M^{s,(i',j')})}{s^2} \leq \gamma \\
\iff &\exists\, i',\, j' :\ \ M^{s,(i',j')}_{(i,j)} = 1 \text{ and } \hat{M}_{\gamma,(i',j')} = 1 \\
\iff &\sum_{\substack{i'\in(i-s,i] \\ j'\in(j-s,j]}} \hat{M}_{\gamma,(i',j')} \geq 1 \\
\iff &\Bigg( \sum_{\substack{i'\in[1,i] \\ j'\in[1,j]}} \hat{M}_{\gamma,(i',j')} - \sum_{\substack{i'\in[1,i-s] \\ j'\in[1,j]}} \hat{M}_{\gamma,(i',j')} \\
&\quad - \sum_{\substack{i'\in[1,i] \\ j'\in[1,j-s]}} \hat{M}_{\gamma,(i',j')} + \sum_{\substack{i'\in[1,i-s] \\ j'\in[1,j-s]}} \hat{M}_{\gamma,(i',j')} \Bigg) \geq 1 \\
\iff &\big(\text{Cuml}\hat{M}_{\gamma,(i,j)} - \text{Cuml}\hat{M}_{\gamma,(i-s,j)} \\
&\quad - \text{Cuml}\hat{M}_{\gamma,(i,j-s)} + \text{Cuml}\hat{M}_{\gamma,(i-s,j-s)}\big) \geq 1
\end{aligned} \tag{10}$$

Again, this can be computed in constant time for each index. Let $\hat{C}_{\gamma,(i,j)} := \text{Cuml}\hat{M}_{\gamma,(i,j)} - \text{Cuml}\hat{M}_{\gamma,(i-s,j)} - \text{Cuml}\hat{M}_{\gamma,(i,j-s)} + \text{Cuml}\hat{M}_{\gamma,(i-s,j-s)}$, then Eq. (1) becomes simply:

$$\hat{M}_{SC\,(i,j)} := \mathbb{1}_{\hat{C}_{\gamma,(i,j)} \geq 1} \tag{11}$$

This gives us an overall runtime of $O(H \times W)$ as desired. Note that in our PyTorch implementation, we are able to use tensor operations such that no explicit iteration over indices is necessary at any point in the algorithm.

### B.4.2 Adjusting $\gamma$

In practice, the method described above can be highly sensitive to the hyperparameter $\gamma$. If $\gamma$ is set too low, then no candidate mask $M^{s,(i',j')}$ will be sufficiently close to $\hat{M}_{PS}$, so the detector will return nothing. However, if $\gamma$ is set too high, then the shape completion will be too conservative, masking a large area of possible candidate patches. (Note that $\gamma \geq 1$ is not usable, because it would cover an image entirely with a mask even when $\hat{M}_{PS} = \mathbf{0}$.) To deal with this issue, we initially use low values of $\gamma$, and then gradually increase $\gamma$ if no mask is initially returned – stopping when either some mask is returned or a maximum value is reached, at which point we assume that there is no ground-truth adversarial patch. Specifically, for iteration $t = 1, ..., T$, we set

$$\gamma_t := 1 - \alpha\beta^{(t-1)},$$

where $T \in \mathbb{N}$, and $\alpha, \beta < 1$. We then return the first nonzero $\hat{M}_{SC}(S, \gamma_t)$, or an empty mask if this does not occur. We set $\alpha = 0.9$, $\beta = 0.7$, $T = 15$. The values of $\alpha$, $\beta$ and $T$ are tuned using grid search on a validation set with 200 images from the xView dataset (See Figure 2).

| α | β | T | Benign mAP | Patch size 100: Adversarial mAP | Patch size 75: Adversarial mAP | Patch size 50: Adversarial mAP |
|---|---|---|---|---|---|---|
| 0.6 | 0.6 | 5 | 0.231 | 0.166 | 0.169 | 0.166 |
| 0.6 | 0.6 | 10 | 0.231 | 0.173 | 0.176 | 0.188 |
| 0.6 | 0.6 | 15 | 0.231 | 0.172 | 0.183 | 0.191 |
| 0.6 | 0.7 | 5 | 0.231 | 0.157 | 0.165 | 0.162 |
| 0.6 | 0.7 | 10 | 0.230 | 0.172 | 0.179 | 0.181 |
| 0.6 | 0.7 | 15 | 0.230 | 0.174 | 0.184 | 0.179 |
| 0.6 | 0.8 | 5 | 0.231 | 0.142 | 0.163 | 0.164 |
| 0.6 | 0.8 | 10 | 0.231 | 0.169 | 0.172 | 0.151 |
| 0.6 | 0.8 | 15 | 0.231 | 0.169 | 0.171 | 0.178 |
| 0.6 | 0.9 | 5 | 0.231 | 0.140 | 0.142 | 0.147 |
| 0.6 | 0.9 | 10 | 0.231 | 0.153 | 0.143 | 0.163 |
| 0.6 | 0.9 | 15 | 0.231 | 0.168 | 0.160 | 0.156 |
| 0.7 | 0.6 | 5 | 0.231 | 0.172 | 0.167 | 0.178 |
| 0.7 | 0.6 | 10 | 0.230 | 0.170 | 0.173 | 0.173 |
| 0.7 | 0.6 | 15 | 0.230 | 0.170 | 0.184 | 0.192 |
| 0.7 | 0.7 | 5 | 0.231 | 0.146 | 0.161 | 0.168 |
| 0.7 | 0.7 | 10 | 0.231 | 0.166 | 0.178 | 0.180 |
| 0.7 | 0.7 | 15 | 0.230 | 0.179 | 0.174 | 0.184 |
| 0.7 | 0.8 | 5 | 0.231 | 0.147 | 0.141 | 0.170 |
| 0.7 | 0.8 | 10 | 0.231 | 0.174 | 0.153 | 0.166 |
| 0.7 | 0.8 | 15 | 0.231 | 0.161 | 0.164 | 0.176 |
| 0.7 | 0.9 | 5 | 0.231 | 0.121 | 0.144 | 0.137 |
| 0.7 | 0.9 | 10 | 0.231 | 0.153 | 0.152 | 0.161 |
| 0.7 | 0.9 | 15 | 0.231 | 0.156 | 0.162 | 0.167 |
| 0.8 | 0.6 | 5 | 0.231 | 0.174 | 0.160 | 0.166 |
| 0.8 | 0.6 | 10 | 0.230 | 0.171 | 0.178 | 0.189 |
| 0.8 | 0.6 | 15 | 0.230 | 0.173 | 0.172 | 0.191 |
| 0.8 | 0.7 | 5 | 0.231 | 0.159 | 0.152 | 0.165 |
| 0.8 | 0.7 | 10 | 0.231 | 0.167 | 0.164 | 0.177 |
| 0.8 | 0.7 | 15 | 0.230 | 0.173 | 0.177 | 0.190 |
| 0.8 | 0.8 | 5 | 0.231 | 0.135 | 0.136 | 0.160 |
| 0.8 | 0.8 | 10 | 0.231 | 0.155 | 0.168 | 0.173 |
| 0.8 | 0.8 | 15 | 0.231 | 0.175 | 0.173 | 0.178 |
| 0.8 | 0.9 | 5 | 0.231 | 0.132 | 0.117 | 0.155 |
| 0.8 | 0.9 | 10 | 0.231 | 0.147 | 0.160 | 0.148 |
| 0.8 | 0.9 | 15 | 0.231 | 0.170 | 0.170 | 0.166 |
| 0.9 | 0.6 | 5 | 0.231 | 0.156 | 0.173 | 0.168 |
| 0.9 | 0.6 | 10 | 0.230 | 0.171 | 0.162 | 0.170 |
| 0.9 | 0.6 | 15 | 0.230 | 0.172 | 0.180 | 0.179 |
| 0.9 | 0.7 | 5 | 0.231 | 0.152 | 0.148 | 0.165 |
| 0.9 | 0.7 | 10 | 0.231 | 0.175 | 0.168 | 0.173 |
| 0.9 | 0.7 | 15 | 0.230 | 0.175 | 0.180 | 0.192 |
| 0.9 | 0.8 | 5 | 0.231 | 0.141 | 0.144 | 0.155 |
| 0.9 | 0.8 | 10 | 0.231 | 0.152 | 0.171 | 0.168 |
| 0.9 | 0.8 | 15 | 0.231 | 0.166 | 0.181 | 0.176 |
| 0.9 | 0.9 | 5 | 0.231 | 0.125 | 0.126 | 0.127 |
| 0.9 | 0.9 | 10 | 0.231 | 0.154 | 0.148 | 0.145 |
| 0.9 | 0.9 | 15 | 0.231 | 0.162 | 0.157 | 0.163 |

Figure 2. Validation set performance on xView under adaptive attack, as a function of defense hyperparameters $\alpha, \beta, T$ used for searching over $\gamma$. Within each column, more green shading indicates higher mAP.

### B.4.3 Adaptive attacks on Shape Completion

To attack the patch segmenter, we use a straight-through estimator (STE) [11] at the thresholding step: $\hat{M}_{PS} = PS_\theta(x) > 0.5$. To attack the shape completion algorithm, we have tried the following attacks:

**BPDA Attack** Note that the algorithm described in Section B.4.1 involves two non-differentiable thresholding steps (Eq. (8) and Eq. (11)). In order to implement an adaptive attack, at these steps, we use BPDA, using a STE for the gradient at each thresholding step. When aggregating masks

which assume patches of different sizes (Equation 9 in the main text) we also use a straight-through estimator on a thresholded sum of masks. This is the strongest adaptive attack for SAC that we found and we use this attack in the main paper.

**$\gamma$-Search STE Attack** There is an additional non-differentiable step in the defense, however: the search over values of $\gamma$ described in Section B.4.2. In order to deal with this, we attempted to use BPDA as well, using the following recursive formulation:

$$\begin{aligned}&\hat{M}_{SC}(S)_{\alpha,\beta,0} := \mathbf{0}\\&\hat{M}_{SC}(S)_{\alpha,\beta,T} := \hat{M}_{SC}(S, 1-\alpha)\\&+ \mathbb{1}_{\frac{\Sigma \hat{M}_{SC}(S,1-\alpha)}{C} < 1} \hat{M}_{SC}(S)_{\alpha*\beta,\beta,T-1} \quad (\text{for } T \geq 1)\end{aligned} \tag{12}$$

Where $C$ is the area of the smallest considered patch size in $S$ (i.e., the minimum nonzero shape completion output ).

We can then use a STE for the indicator function. However, this technique turns out to yield worse performance in practice than simply treating the search over $\gamma$ as non-differentiable (See Fig. 3). Therefore, in our main results, we treat this search over $\gamma$ as non-differentiable, rather than using an STE.

**Log-Sum-Exp Transfer Attack** We were also initially concerned that the simple straight-through estimation approach for the algorithm described in Section B.4.1 might fail, specifically at the point of Eq. (11), where the threshold takes the form (see Eq. (10)):

$$\sum_{\substack{i' \in (i-s,i] \\ j' \in (j-s,j]}} \hat{M}_{\gamma,(i',j')} \geq 1 \tag{13}$$

where $\hat{M}_{\gamma,(i',j')}$ is a 0/1 indicator of whether a patch should be added to the final output mask with upper-left corner $(i', j')$. We were concerned that a straight-through estimator would propagate gradients to *the sum directly*, affecting *every potential* patch which could cover a location $(i, j)$, rather than concentrating the gradient only on those patches that *actually* contribute to the pixel $(i, j)$ being masked.

To mitigate this, we first considered the equivalent threshold condition:

$$\max_{\substack{i' \in (i-s,i] \\ j' \in (j-s,j]}} \hat{M}_{\gamma,(i',j')} \geq 1 \tag{14}$$

While logically equivalent, the gradient propagated by the STE to the LHS would now only propagate on to the values $\hat{M}_{\gamma,(i',j')}$ which are equal to 1. However, unfortunately, this formulation is not compatible with the dynamic programming algorithm described in Section B.4.1: due to computational limitations, we do not want to compute the maximum over every pair $(i', j')$, *for each pair* $(i, j)$.

| α | β | T | Benign mAP | Patch size 100: Adversarial mAP | Patch size 75: Adversarial mAP | Patch size 50: Adversarial mAP |
|---|---|---|---|---|---|---|
| 0.6 | 0.6 | 5 | 0 | 0.003 | 0.028 | 0.036 |
| 0.6 | 0.6 | 10 | 0 | -0.006 | 0.018 | 0.007 |
| 0.6 | 0.6 | 15 | 0 | -0.002 | 0.011 | 0.002 |
| 0.6 | 0.7 | 5 | 0 | 0.012 | 0.030 | 0.033 |
| 0.6 | 0.7 | 10 | 0 | -0.005 | 0.014 | 0.020 |
| 0.6 | 0.7 | 15 | 0 | -0.006 | 0.011 | 0.014 |
| 0.6 | 0.8 | 5 | 0 | 0.025 | 0.029 | 0.032 |
| 0.6 | 0.8 | 10 | 0 | 0.001 | 0.018 | 0.053 |
| 0.6 | 0.8 | 15 | 0 | 0.001 | 0.020 | 0.016 |
| 0.6 | 0.9 | 5 | 0 | 0.025 | 0.046 | 0.046 |
| 0.6 | 0.9 | 10 | 0 | 0.013 | 0.050 | 0.031 |
| 0.6 | 0.9 | 15 | 0 | 0.002 | 0.028 | 0.037 |
| 0.7 | 0.6 | 5 | 0 | 0.003 | 0.018 | 0.025 |
| 0.7 | 0.6 | 10 | 0 | 0.010 | 0.013 | 0.025 |
| 0.7 | 0.6 | 15 | 0 | 0.000 | -0.002 | 0.003 |
| 0.7 | 0.7 | 5 | 0 | 0.026 | 0.028 | 0.038 |
| 0.7 | 0.7 | 10 | 0 | 0.002 | 0.008 | 0.022 |
| 0.7 | 0.7 | 15 | 0 | -0.009 | 0.009 | 0.017 |
| 0.7 | 0.8 | 5 | 0 | 0.022 | 0.047 | 0.033 |
| 0.7 | 0.8 | 10 | 0 | -0.001 | 0.034 | 0.036 |
| 0.7 | 0.8 | 15 | 0 | 0.015 | 0.022 | 0.025 |
| 0.7 | 0.9 | 5 | 0 | 0.052 | 0.043 | 0.060 |
| 0.7 | 0.9 | 10 | 0 | 0.018 | 0.036 | 0.034 |
| 0.7 | 0.9 | 15 | 0 | 0.016 | 0.020 | 0.029 |
| 0.8 | 0.6 | 5 | 0 | 0.001 | 0.023 | 0.035 |
| 0.8 | 0.6 | 10 | 0 | 0.004 | 0.009 | 0.006 |
| 0.8 | 0.6 | 15 | 0 | 0.002 | 0.017 | 0.012 |
| 0.8 | 0.7 | 5 | 0 | 0.017 | 0.034 | 0.033 |
| 0.8 | 0.7 | 10 | 0 | 0.013 | 0.025 | 0.034 |
| 0.8 | 0.7 | 15 | 0 | 0.004 | 0.012 | 0.008 |
| 0.8 | 0.8 | 5 | 0 | 0.038 | 0.054 | 0.037 |
| 0.8 | 0.8 | 10 | 0 | 0.018 | 0.019 | 0.025 |
| 0.8 | 0.8 | 15 | 0 | 0.000 | 0.016 | 0.017 |
| 0.8 | 0.9 | 5 | 0 | 0.045 | 0.073 | 0.045 |
| 0.8 | 0.9 | 10 | 0 | 0.025 | 0.028 | 0.049 |
| 0.8 | 0.9 | 15 | 0 | 0.004 | 0.019 | 0.032 |
| 0.9 | 0.6 | 5 | 0 | 0.024 | 0.022 | 0.025 |
| 0.9 | 0.6 | 10 | 0 | 0.003 | 0.030 | 0.038 |
| 0.9 | 0.6 | 15 | 0 | 0.001 | 0.002 | 0.028 |
| 0.9 | 0.7 | 5 | 0 | 0.029 | 0.045 | 0.030 |
| 0.9 | 0.7 | 10 | 0 | 0.011 | 0.024 | 0.020 |
| 0.9 | 0.7 | 15 | 0 | -0.002 | 0.011 | 0.004 |
| 0.9 | 0.8 | 5 | 0 | 0.038 | 0.041 | 0.043 |
| 0.9 | 0.8 | 10 | 0 | 0.029 | 0.018 | 0.028 |
| 0.9 | 0.8 | 15 | 0 | 0.011 | 0.006 | 0.028 |
| 0.9 | 0.9 | 5 | 0 | 0.052 | 0.068 | 0.073 |
| 0.9 | 0.9 | 10 | 0 | 0.025 | 0.046 | 0.063 |
| 0.9 | 0.9 | 15 | 0 | 0.016 | 0.037 | 0.035 |

Figure 3. Difference in mAP under BPDA attack using STE gradients for the search over $\gamma$ (as in Eq. (12)) versus simply treating the search as non-differentiable, on 200-image xView validation set. Positive numbers (green) indicate that the non-differentiable treatment yielded a more successful attack, while negative numbers (red) indicate that the STE treatment was more successful. We see that in most hyperparameter settings, the STE treatment of the search over $\gamma$ made the attack less successful, and in no setting did it make the attack substantially more successful.

To solve this problem, we instead used the following proxy function when generating attack gradients (including during the forward pass):

$$\log\left(\sum_{\substack{i' \in (i-s,i] \\ j' \in (j-s,j]}} e^{C \cdot \hat{M}_{\gamma,(i',j')}}\right) / C \geq 1 \tag{15}$$

where $C$ is a large constant (we use $C = 10\log(100)$). This is the "LogSumExp" softmax function: note that the LHS is approximately 1 if any $\hat{M}_{\gamma,(i',j')}$ is one and approximately zero otherwise. Also note that the derivative of the LHS with respect to each $\hat{M}_{\gamma,(i',j')}$ is similarly approximately 1 if $\hat{M}_{\gamma,(i',j')}$ is one and zero otherwise. Crucially, we can compute this in the above DP framework, simply by replacing $\hat{M}_{\gamma,(i',j')}$ with its exponent (and taking the log before thresholding).

However, in practice, the naive BPDA attack outperforms this adaptive attack (Fig. 4). This is likely because the condition in Eq. (15) is an inexact approximation, so the function being attacked differs from the true objective. (In both attacks, we treat the search over $\gamma$ as nondifferentiable, as described above.)

#### B.4.4 Patch Visualization

We find that adaptive attacks on models with SC would force the attacker to generate patches that have more structured noises trying to fool SC (see Fig. 5).

#### B.4.5 Visualization of Shape Completion Outputs

We provide several examples of shape completion outputs in Fig. 6. The outputs of the patch segmenter can be disturbed by the attacker such that some parts of the adversarial patches are not detected, especially under adaptive attacks. Given the output mask of the patch segmenter, the proposed shape completion algorithm generates a "completed patch mask" to cover the entire adversarial patches.

### B.5. Visualization of Detection Results

#### B.5.1 SAC under Adaptive Attacks

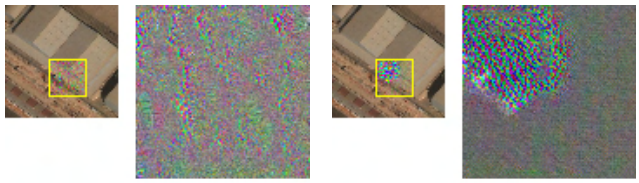
We provide several examples of SAC under adaptive attacks in Fig. 7 and Fig. 8. Adversarial patches create spurious detections, and make the detector ignore the ground-truth objects. SAC can detect and remove the adversarial patches even under strong adaptive attacks, and therefore restore model predictions.

#### B.5.2 SAC v.s. Baselines

In this paper, we compare SAC with JPEG [1], Spatial Smoothing [12], LGS [2], and vanilla adversarial training (AT) [8]. Visual comparisons are shown in Fig. 9 and Fig. 10. JPEG, Spatial Smoothing, LGS are pre-processing defenses that aim to remove the high-frequency information of adversarial patches. They have reasonable performance under non-adaptive attacks, but can not defend adaptive attacks where the adversary also attacks the pre-process functions. In addition, they degrade image quality, especially LGS, which degrades their performance on clean images. SAC can defend both non-adaptive and adaptive attacks. In addition, SAC does not degrade image quality, and therefore can maintain high performance on clean images.

| α | β | T | Benign mAP | Patch size 100: Adversarial mAP | Patch size 75: Adversarial mAP | Patch size 50: Adversarial mAP |
|---|---|---|---|---|---|---|
| 0.6 | 0.6 | 5 | 0 | 0.008 | 0.014 | 0.031 |
| 0.6 | 0.6 | 10 | 0 | 0.003 | 0.011 | 0.015 |
| 0.6 | 0.6 | 15 | 0 | 0.007 | -0.003 | 0.011 |
| 0.6 | 0.7 | 5 | 0 | 0.018 | 0.020 | 0.043 |
| 0.6 | 0.7 | 10 | 0 | 0.007 | 0.009 | 0.016 |
| 0.6 | 0.7 | 15 | 0 | -0.001 | 0.000 | 0.031 |
| 0.6 | 0.8 | 5 | 0 | 0.037 | 0.023 | 0.041 |
| 0.6 | 0.8 | 10 | 0 | 0.013 | 0.019 | 0.047 |
| 0.6 | 0.8 | 15 | 0 | 0.008 | 0.021 | 0.020 |
| 0.6 | 0.9 | 5 | 0 | 0.041 | 0.051 | 0.046 |
| 0.6 | 0.9 | 10 | 0 | 0.027 | 0.049 | 0.038 |
| 0.6 | 0.9 | 15 | 0 | 0.015 | 0.034 | 0.045 |
| 0.7 | 0.6 | 5 | 0 | 0.002 | 0.026 | 0.019 |
| 0.7 | 0.6 | 10 | 0 | 0.007 | 0.016 | 0.019 |
| 0.7 | 0.6 | 15 | 0 | 0.004 | 0.008 | 0.000 |
| 0.7 | 0.7 | 5 | 0 | 0.029 | 0.044 | 0.033 |
| 0.7 | 0.7 | 10 | 0 | 0.014 | 0.011 | 0.014 |
| 0.7 | 0.7 | 15 | 0 | 0.001 | 0.023 | 0.018 |
| 0.7 | 0.8 | 5 | 0 | 0.030 | 0.053 | 0.029 |
| 0.7 | 0.8 | 10 | 0 | -0.001 | 0.032 | 0.034 |
| 0.7 | 0.8 | 15 | 0 | 0.014 | 0.019 | 0.023 |
| 0.7 | 0.9 | 5 | 0 | 0.063 | 0.049 | 0.058 |
| 0.7 | 0.9 | 10 | 0 | 0.027 | 0.036 | 0.035 |
| 0.7 | 0.9 | 15 | 0 | 0.029 | 0.036 | 0.033 |
| 0.8 | 0.6 | 5 | 0 | 0.012 | 0.042 | 0.039 |
| 0.8 | 0.6 | 10 | 0 | 0.018 | 0.014 | 0.018 |
| 0.8 | 0.6 | 15 | 0 | 0.013 | 0.028 | 0.017 |
| 0.8 | 0.7 | 5 | 0 | 0.027 | 0.042 | 0.044 |
| 0.8 | 0.7 | 10 | 0 | 0.018 | 0.024 | 0.031 |
| 0.8 | 0.7 | 15 | 0 | 0.017 | 0.018 | 0.021 |
| 0.8 | 0.8 | 5 | 0 | 0.054 | 0.061 | 0.042 |
| 0.8 | 0.8 | 10 | 0 | 0.031 | 0.028 | 0.034 |
| 0.8 | 0.8 | 15 | 0 | 0.009 | 0.032 | 0.024 |
| 0.8 | 0.9 | 5 | 0 | 0.051 | 0.087 | 0.048 |
| 0.8 | 0.9 | 10 | 0 | 0.042 | 0.042 | 0.061 |
| 0.8 | 0.9 | 15 | 0 | 0.014 | 0.035 | 0.041 |
| 0.9 | 0.6 | 5 | 0 | 0.027 | 0.023 | 0.032 |
| 0.9 | 0.6 | 10 | 0 | 0.011 | 0.034 | 0.040 |
| 0.9 | 0.6 | 15 | 0 | 0.006 | 0.012 | 0.030 |
| 0.9 | 0.7 | 5 | 0 | 0.034 | 0.045 | 0.039 |
| 0.9 | 0.7 | 10 | 0 | 0.012 | 0.034 | 0.036 |
| 0.9 | 0.7 | 15 | 0 | 0.011 | 0.027 | 0.013 |
| 0.9 | 0.8 | 5 | 0 | 0.050 | 0.054 | 0.053 |
| 0.9 | 0.8 | 10 | 0 | 0.033 | 0.027 | 0.031 |
| 0.9 | 0.8 | 15 | 0 | 0.021 | 0.015 | 0.034 |
| 0.9 | 0.9 | 5 | 0 | 0.063 | 0.077 | 0.079 |
| 0.9 | 0.9 | 10 | 0 | 0.029 | 0.051 | 0.059 |
| 0.9 | 0.9 | 15 | 0 | 0.023 | 0.056 | 0.037 |

Figure 4. Difference in mAP using Log-Sum-Exp approximation for Eq. (11) as described in Eq. (15) versus the naive BPDA attack we ultimately used, on 200-image xView validation set. Positive numbers (green) indicate that the naive BPDA attack yielded a more successful attack, while negative numbers (red) indicate that the Log-Sum-Exp treatment was more successful. We see that in most hyperparameter settings, the Log-Sum-Exp technique made the attack less successful, and in no setting did it make the attack substantially more successful.



(a) Patch for SAC without SC. (b) Patch for SAC with SC.

Figure 5. $100 \times 100$ adversarial patches generated by adaptive attacks on xView dataset. Patches for SAC without shape completion (SC) have widespread noises in the square bounded area, while patches for SAC with SC have structured noises.

#### B.5.3 SAC under Different Attack Methods

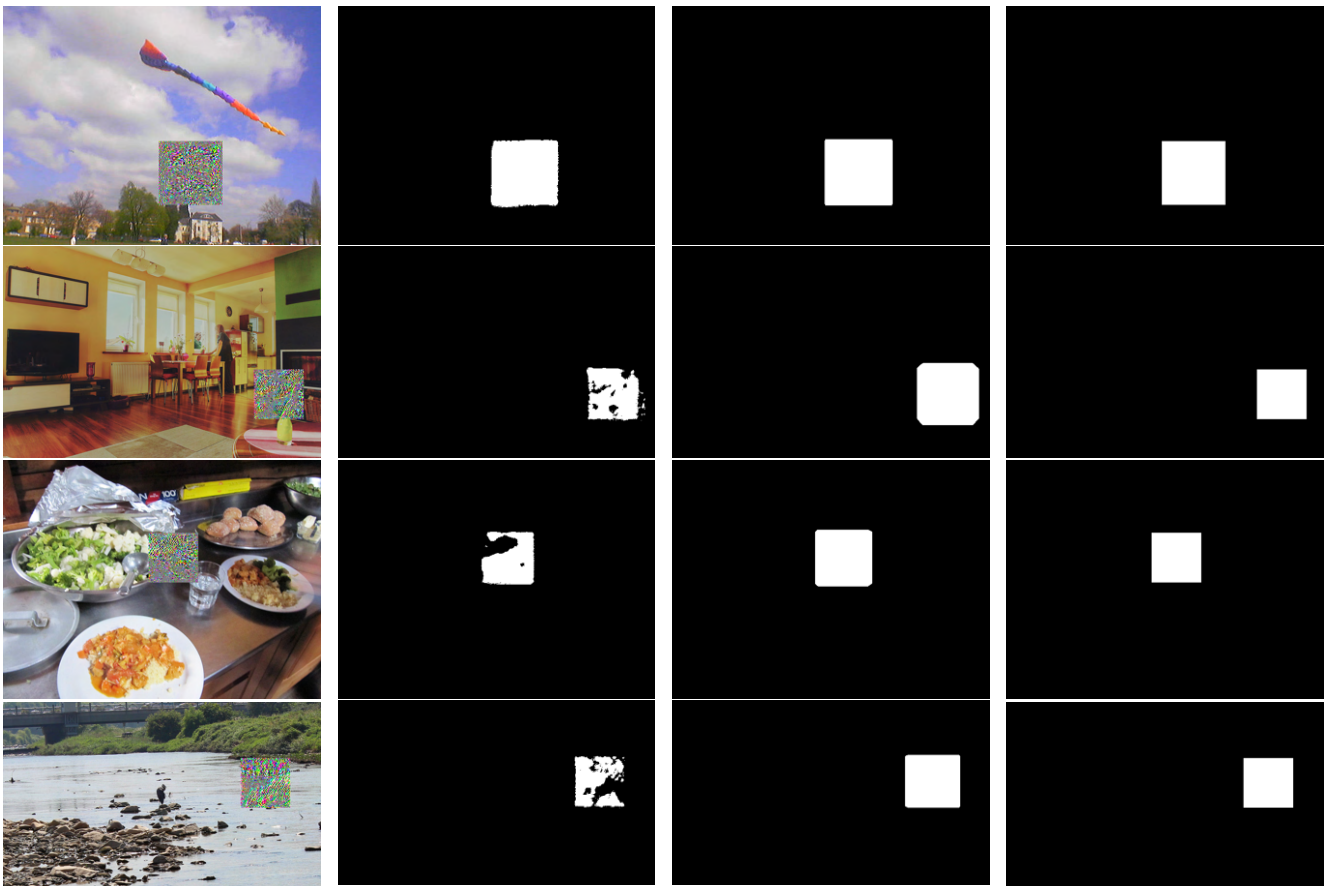
We visualize the detection results of SAC under different attacks in Fig. 11 and Fig. 12, including PGD [8], MIM [10] and DPatch [9]. SAC can effectively detect and remove the adversarial patches under different attacks and restore the model predictions. We also notice that the adversarial patches generated by different methods has different styles. PGD generated adversarial patches are less visible, even though it has the same $\epsilon = 1$ attack budget.

#### B.5.4 SAC under Different Patch Shapes

We visualize the detection results of SAC under PGD attacks with unseen patch shapes in Fig. 13 and Fig. 14, including circle, rectangle and ellipse. SAC can effectively detect and remove the adversarial patches of different shapes and restore the model predictions, even though those shapes are used in training the patch segmenter and mismatch the square shape prior in shape completion. However, we do notice that masked region can be larger than the original patches as SAC tries to cover the patch with square shapes.

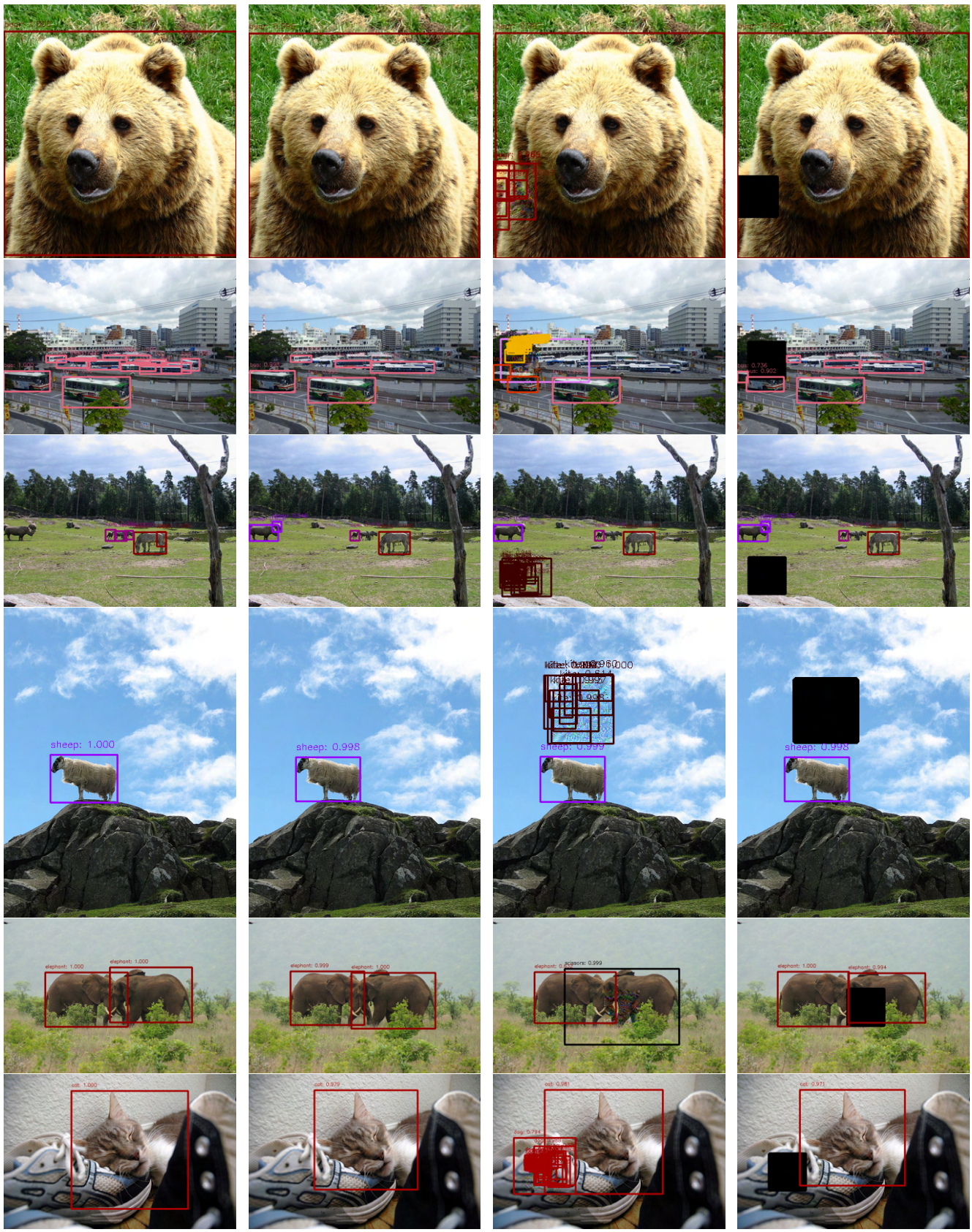
#### B.5.5 Failure Cases

There are several failure modes in SAC: 1) SAC completely fails to detect a patch (*e.g.*, Fig. 15 row 1), which happens very rarely; 2) SAC successfully detects and removes a patch, but the black blocks from patch removing causes misdetection (*e.g.*, Fig. 15 row 2), which happens more often on the COCO dataset since black blocks resemble some object categories in the dataset such as TV, traffic light, and suitcase; 3) SAC successfully detects and removes a patch, but the patch covers foreground objects and thus the object detector fails to detect the objects on the masked image (*e.g.*, Fig. 15 row 3). We can potentially mitigate the first issue by improving the patch segmenter, such as using more advanced segmentation networks and doing longer self adversarial training. For the second issue, we can avoid it by fine-tuning the base object detetor on images with randomly-placed black blocks. For the third issue, if the attacker is allowed to arbitrarily distort the pixels and destroy all the information within the patch such as in physical patch attacks, there is no chance that we can detect the objects hiding behind the adversarial patches. However, in the case where the patches are less visible, some information may be preserved in the patched area. We can potentially impaint or reconstruct the content within the patches to help detection.



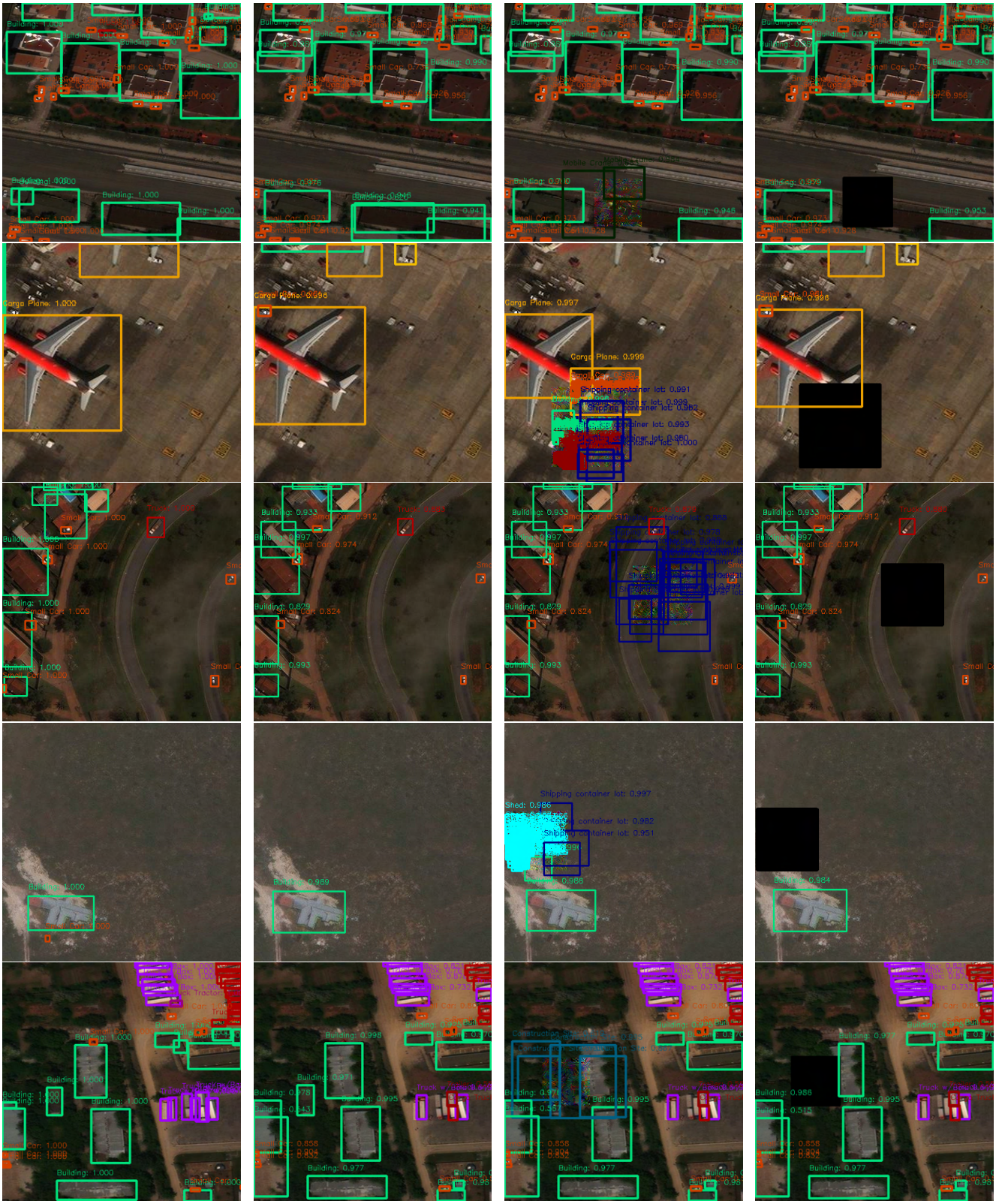
(a) Adversarial images. (b) Outputs of patch segmentation $\hat{M}_{PS}$. (c) Outputs of shape completion $\hat{M}_{SC}$. (d) Ground-truth patch masks $M$.

Figure 6. Visualization of shape completion outputs. Given the output of the patch segmenter, the proposed shape completion algorithm generates a "completed patch mask" to cover the entire adversarial patches.




(a) Ground-truth on clean images. (b) Predictions on clean images (c) Predictions on adversarial images. (d) Predictions on SAC masked images.

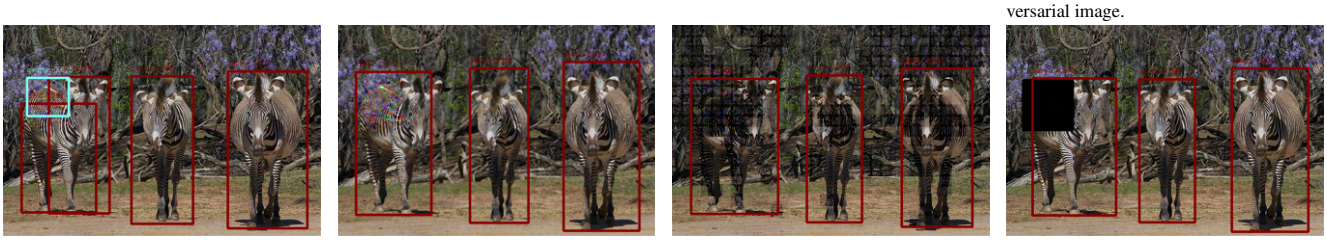
Figure 7. Examples on the COCO dataset. The adversarial patches are $100 \times 100$ squares generated by PGD adaptive attacks.



(a) Ground-truth on clean images. (b) Predictions on clean images (c) Predictions on adversarial images. (d) Predictions on SAC masked images.

Figure 8. Examples on the xView dataset. The adversarial patches are $100 \times 100$ squares generated by PGD adaptive attacks. Adversarial patches create spurious detections, and make the detector ignore the ground-truth objects. SAC can detect and remove the patches even under strong adaptive attacks, and therefore restore model predictions.

(a) Ground-truth on the clean image. (b) Predictions on the clean image. (c) Predictions on the adversarial image. (d) Predictions of the AT model on the adversarial image.

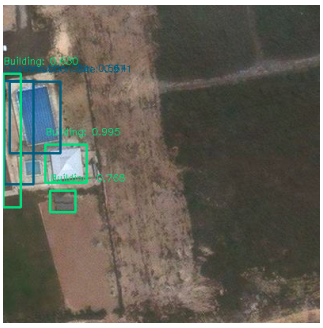


(e) Predictions on the JPEG [1] processed adversarial image (non-adaptive attack). (f) Predictions on the Spatial Smoothing [12] processed adversarial image (non-adaptive attack). (g) Predictions on the LGS [2] processed adversarial image (non-adaptive attack). (h) Predictions on the SAC masked adversarial image (non-adaptive attack).

(i) Predictions on the JPEG [1] processed adversarial image (adaptive attack). (j) Predictions on the Spatial Smoothing [12] processed adversarial image (adaptive attack). (k) Predictions on the LGS [2] processed adversarial image (adaptive attack). (l) Predictions on the SAC masked adversarial image (adaptive attack).

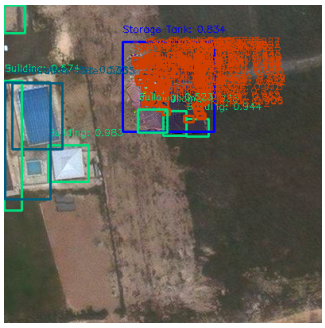
Figure 9. Detection results of different defense methods on the COCO dataset. The adversarial patches are $100 \times 100$ squares and placed at the same location. JPEG [1], Spatial Smoothing [12], LGS [2] have reasonable performance under non-adaptive attacks, but can not defend adaptive attacks where the adversary also attacks the pre-processing functions. In addition, they degrade image quality, especially LGS. SAC can defend both non-adaptive and adaptive attacks and maintains high image quality.

(a) Ground-truth on the clean image.

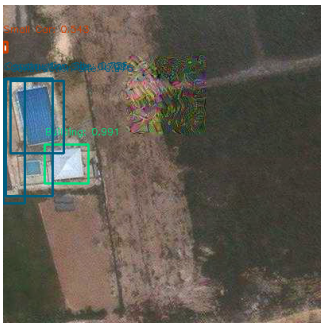


(b) Predictions on the clean image.

(c) Predictions on the adversarial image.



(d) Predictions of the AT model on the adversarial image.

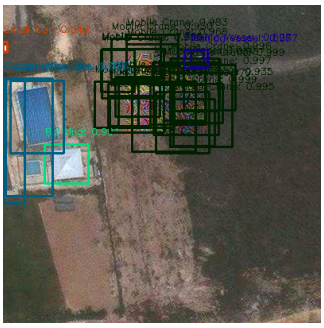


(e) Predictions on the JPEG [1] processed adversarial image (non-adaptive attack).

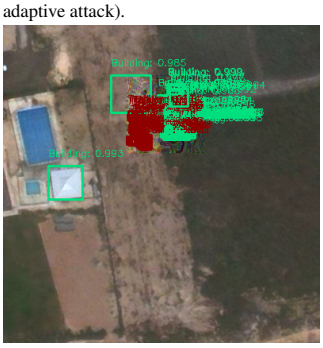
(f) Predictions on the Spatial Smoothing [12] processed adversarial image (non-adaptive attack).

(g) Predictions on the LGS [2] processed adversarial image (non-adaptive attack).

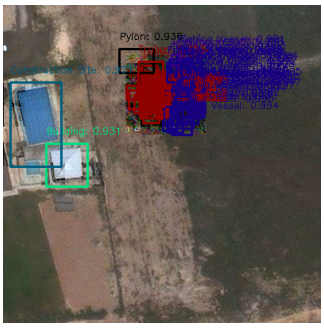
(h) Predictions on the SAC masked adversarial image (non-adaptive attack).



(i) Predictions on the JPEG [1] processed adversarial image (adaptive attack).



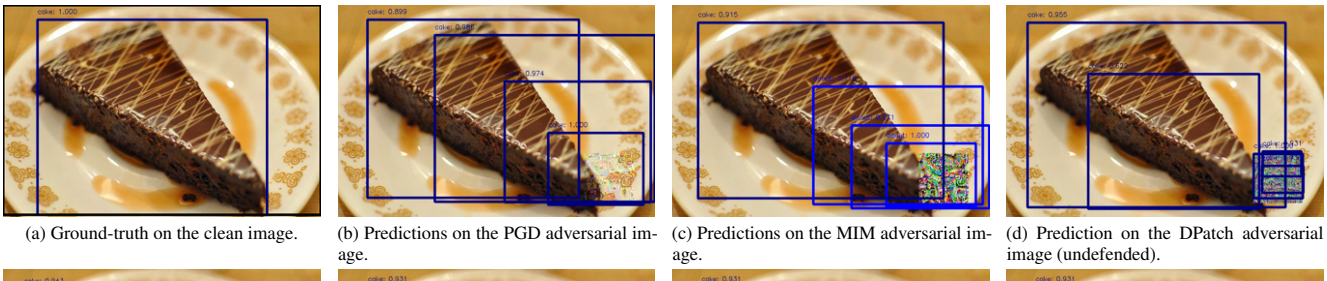
(j) Predictions on the Spatial Smoothing [12] processed adversarial image (adaptive attack).



(k) Predictions on the LGS [2] processed adversarial image (adaptive attack).

(l) Predictions on the SAC masked adversarial image (adaptive attack).

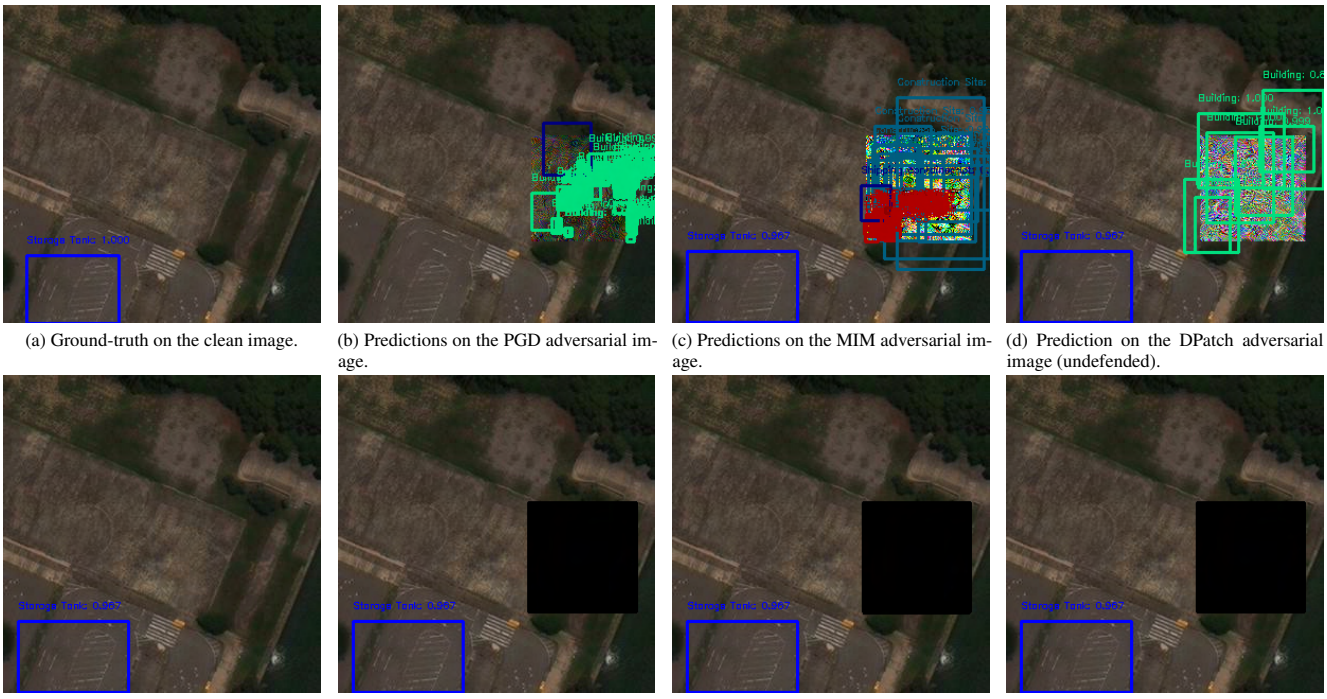
Figure 10. Detection results of different defense methods on the xView dataset. The adversarial patches are $100 \times 100$ squares and placed at the same location. JPEG [1], Spatial Smoothing [12], LGS [2] have reasonable performance under non-adaptive attacks, but can not defend adaptive attacks where the adversary also attacks the pre-processing functions. In addition, they degrade image quality, especially LGS. SAC can defend both non-adaptive and adaptive attacks and maintains high image quality.



(a) Ground-truth on the clean image. (b) Predictions on the PGD adversarial image. (c) Predictions on the MIM adversarial image. (d) Prediction on the DPatch adversarial image (undefended).

(e) Predictions on the clean image. (f) Predictions on the SAC masked PGD adversarial image. (g) Predictions on the SAC masked MIM adversarial image. (h) Predictions on the SAC masked DPatch adversarial image.

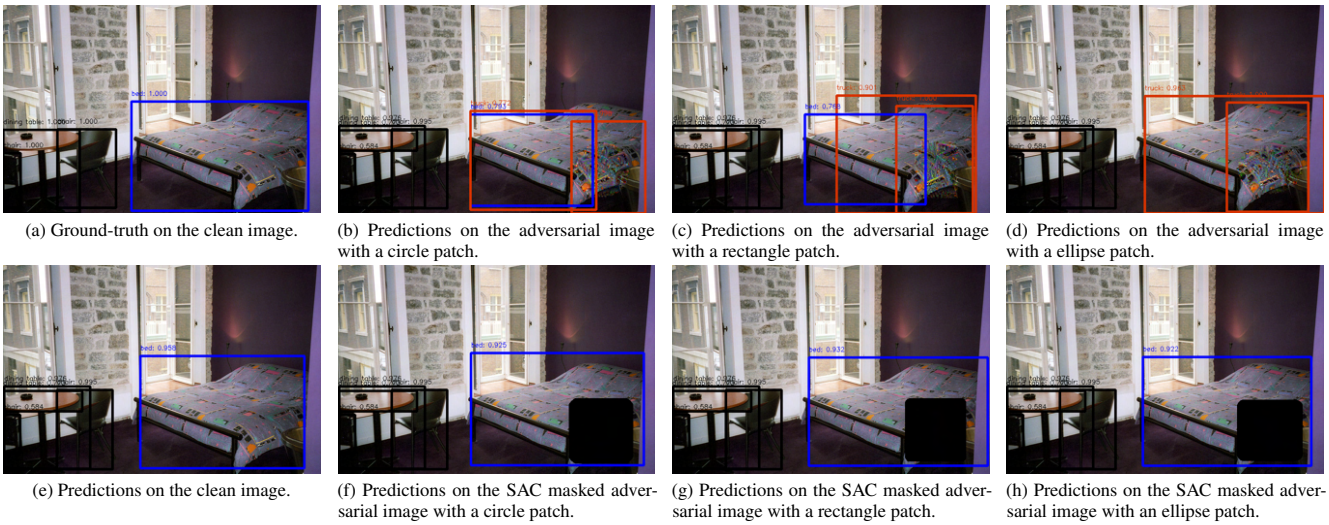
Figure 11. Detection results on a clean image and corresponding adversarial images generated by different attack methods. The image is taken from the COCO dataset. The adversarial patches are $100 \times 100$ squares and placed at the same location.



(a) Ground-truth on the clean image. (b) Predictions on the PGD adversarial image. (c) Predictions on the MIM adversarial image. (d) Prediction on the DPatch adversarial image (undefended).

(e) Predictions on the clean image. (f) Predictions on the SAC masked PGD adversarial image. (g) Predictions on the SAC masked MIM adversarial image. (h) Predictions on the SAC masked DPatch adversarial image.

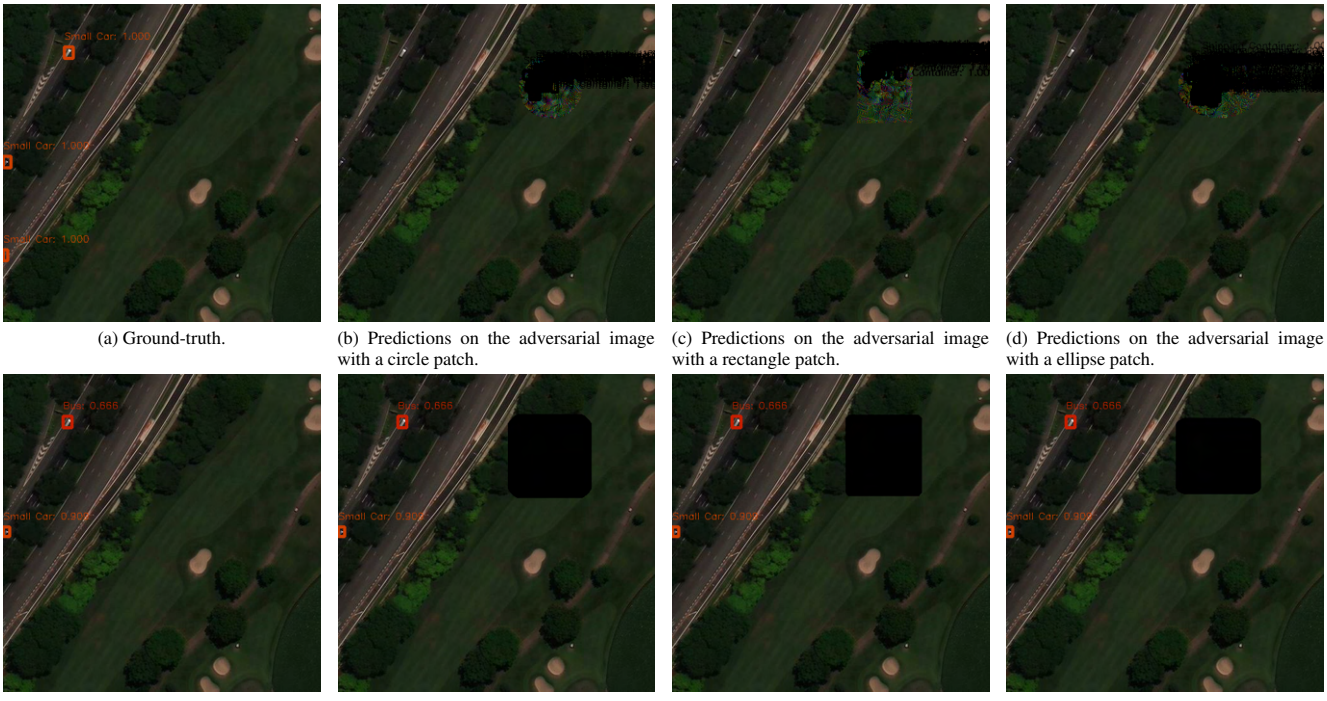
Figure 12. Detection results on a clean image and corresponding adversarial images generated by different attack methods. The image is taken from the xView dataset. The adversarial patches are $100 \times 100$ squares and placed at the same location.



(a) Ground-truth on the clean image. (b) Predictions on the adversarial image with a circle patch. (c) Predictions on the adversarial image with a rectangle patch. (d) Predictions on the adversarial image with a ellipse patch.

(e) Predictions on the clean image. (f) Predictions on the SAC masked adversarial image with a circle patch. (g) Predictions on the SAC masked adversarial image with a rectangle patch. (h) Predictions on the SAC masked adversarial image with an ellipse patch.

Figure 13. Detection results on adversarial images with different patch shapes. The image is taken from the COCO dataset. The adversarial patches have $100 \times 100$ pixels and placed at the same location.



(a) Ground-truth. (b) Predictions on the adversarial image with a circle patch. (c) Predictions on the adversarial image with a rectangle patch. (d) Predictions on the adversarial image with a ellipse patch.

(e) Predictions on the clean image. (f) Predictions on the SAC masked adversarial image with a circle patch. (g) Predictions on the SAC masked adversarial image with a rectangle patch. (h) Predictions on the SAC masked adversarial image with an ellipse patch.

Figure 14. Detection results on adversarial images with different patch shapes. The image is taken from the xView dataset. The adversarial patches have $100 \times 100$ pixels and placed at the same location.

(a) Ground-truth on clean images. (b) Predictions on clean images (c) Predictions on adversarial images. (d) Predictions on SAC masked images.

Figure 15. Examples of failure cases. Row 1: SAC fails to detect and remove the adversarial patch, which happens very rarely. Row 2: the black block from masking out the patch creates a false detection of "TV". Row 3: the black block from masking out the patch cover foreground objects. See the discussion in Section B.5.5.

## References

[1] Gintare Karolina Dziugaite, Zoubin Ghahramani, and Daniel M Roy. A study of the effect of JPG compression on adversarial images. *arXiv preprint arXiv:1608.00853*, 2016. 1, 5, 10, 11

[2] Muzammal Naseer, Salman Khan, and Fatih Porikli. Local gradients smoothing: Defense against localized adversarial attacks. In *2019 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 1300–1307. IEEE, 2019. 1, 5, 10, 11

[3] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-Net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 234–241. Springer, 2015. 1

[4] Geoffrey Hinton, Nitish Srivastava, and Kevin Swersky. Neural networks for machine learning lecture 6a overview of mini-batch gradient descent. *Cited on*, 14(8), 2012. 1

[5] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'15, page 91–99, Cambridge, MA, USA, 2015. MIT Press. 2

[6] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. SSD: Single shot multibox detector. In *European Conference on Computer Vision*, pages 21–37. Springer, 2016. 2

[7] Sébastien Marcel and Yann Rodriguez. Torchvision the machine-vision package of torch. In *Proceedings of the 18th ACM International Conference on Multimedia*, MM '10, page 1485–1488, New York, NY, USA, 2010. Association for Computing Machinery. 2

[8] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083*, 2017. 2, 5, 6

[9] Xin Liu, Huanrui Yang, Ziwei Liu, Linghao Song, Hai Li, and Yiran Chen. DPatch: An adversarial patch attack on object detectors. *arXiv preprint arXiv:1806.02299*, 2018. 2, 6

[10] Yinpeng Dong, Fangzhou Liao, Tianyu Pang, Hang Su, Jun Zhu, Xiaolin Hu, and Jianguo Li. Boosting adversarial attacks with momentum. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 9185–9193, 2018. 2, 6

[11] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013. 3

[12] Weilin Xu, David Evans, and Yanjun Qi. Feature squeezing: Detecting adversarial examples in deep neural networks. *arXiv preprint arXiv:1704.01155*, 2017. 5, 10, 11