NAPGuard: Towards Detecting Naturalistic Adversarial Patches

Supplementary Material

In this supplementary material, we provide additional details that we do not include in our main paper.

A. Overall Algorithm of NAPGuard

To effectively detect naturalistic adversarial patches (NAPs), we propose the NAPGuard, an elaborated critical feature modulation framework by aligning aggressive features and suppressing natural features during training and inference, respectively.

Our NAPGuard framework involves both training and inference processes, providing a more comprehensive detection framework against NAPs. During the training process, we calculate the pattern alignment loss $\mathcal{L}_p$, an MSE loss between feature maps obtained from normal images and aligned images extracted by $\mathcal{F}(\cdot)$. Further, we calculate the auxiliary loss $\mathcal{L}_a$ of the auxiliary branch $\mathcal{G}(x_i, \nabla^2)$. Finally, we jointly optimize the detection loss $\mathcal{L}_d$, pattern alignment loss $\mathcal{L}_p$ and auxiliary loss $\mathcal{L}_a$ to enhance the capability of the adversarial patch detection model ("**detector**") in capturing aggressive patterns. During the inference process, we universally mitigate the disturbance of natural features by utilizing the feature shield module $\mathcal{H}(\cdot)$ to improve generalization. The overall algorithm of NAPGuard can be seen in Algorithm 1.

Algorithm 1 NAPGuard Framework

Input: Training set $\mathcal{X} = \{x_1, x_2, ..., x_n\}$, ground truth labels $\{y_1, y_2, ..., y_n\}$, adversarial patch detection model $\mathbb{M}_\theta$ with a feature extractor $\mathcal{F}(\cdot)$, real-world images $\mathcal{X}^* = \{x_1^*, x_2^*, ..., x_n^*\}$, feature shield module $\mathcal{H}(\cdot)$.
Output: Detection Results $\{\hat{y}_1^*, \hat{y}_2^*, ..., \hat{y}_n^*\}$

```
 1: Initialize the model $\mathbb{M}_\theta$ and set it to training mode.
 2: for the number of epochs do
 3:     for $x_i$ in $\mathcal{X}$ do
 4:         Calculate $\mathcal{L}_p$ by Eqn. (2).
 5:         Calculate $\mathcal{L}_a$, $\mathcal{L}_d$ by Eqn. (3) and $\mathcal{L}_d$ is the original
            detection loss of the detector.
 6:         Update $\theta$ by backpropagating Eqn. (3).
 7:     end for
 8: end for
 9: Set the trained model $\mathbb{M}_\theta$ to evaluation mode.
10: for $x_i^*$ in $\mathcal{X}^*$ do
11:     $x_{i(s)}^* \leftarrow \mathcal{H}(x_i^*)$
12:     $\hat{y}_i^* \leftarrow \mathbb{M}_\theta(x_{i(s)}^*)$
13: end for
14: return $\{\hat{y}_1^*, \hat{y}_2^*, ..., \hat{y}_n^*\}$
```

Table S1. The ablation study results (AP@0.5↑) on different loss terms in our training strategy. "+AFAL" represents our training strategy, *i.e.*, $+\alpha\mathcal{L}_a + \beta\mathcal{L}_p$.

Method			Base	$+\mathcal{L}_p$	$+\mathcal{L}_a$	+AFAL
Patch Type	Non-NAPs	T-SEA [5]	99.42	99.33	99.45	98.32
		AdvPatch [16]	87.83	98.53	96.48	96.89
		AdvCloak [17]	59.94	87.47	72.64	91.22
		AdvTshirt [18]	42.07	83.70	57.34	65.95
		AdvTexture [4]	69.51	93.35	70.49	93.71
	NAPs	GNAP [3]	68.00	81.03	78.96	**88.00**
		DM-NAP [9]	54.48	90.25	91.14	**98.59**
		LAP [15]	36.23	67.93	52.44	**86.13**
	Mixture	Mixture	76.84	88.71	84.94	**91.89**

B. More Ablation Studies and Discussions

In this section, we provide more ablation studies on different loss terms and the choices of hyper-parameters. Further, we explore the defense potential of our method and provide more fine-grained results on specific types of adversarial patches.

B.1. Different Loss Terms

First, we investigate the effect of loss terms in our training strategy. Specifically, we train the patch detector with loss function $\mathcal{L}_p$, $\mathcal{L}_a$ and $\alpha\mathcal{L}_a + \beta\mathcal{L}_p$ respectively (with $\mathcal{L}_d$ fixed). As shown in Tab. S1, the AP@0.5 shows a significant rise under $\mathcal{L}_p$ and $\mathcal{L}_a$ setting, while combining them shows further improvement (*i.e.*, for GNAP, +13.03% under $\mathcal{L}_p$ setting, +10.96% under $\mathcal{L}_a$ setting and +20.00% under $\alpha\mathcal{L}_a + \beta\mathcal{L}_p$ setting). These results empirically demonstrate that both of the loss terms could improve detector's performance on NAPs.

B.2. Hyper-parameters

In this part, we provide experimental results to support our choice of hyper-parameters in the training and inference strategy.

Hyper-parameters of Aggressive Feature Aligned Learning (AFAL) Strategy.

α **and** β**.** Regarding the hyper-parameters α and β, we argue that they control the balance of detector's alignment. In our experiments, we conduct a thorough analysis by varying the values of α and β to assess their impact on the detector's capability of detecting NAPs. Specifically, we set the α as 0.01, 0.2, 0.4, 0.5, 0.6, 0.8, 0.99 and β as 0.1, 1, 10, 100, respectively. As illustrated in Fig. S1, the highest AP is achieved when $\alpha = 0.4$ and $\beta = 10$. These results demonstrate our choice of α and β can achieve optimal per-

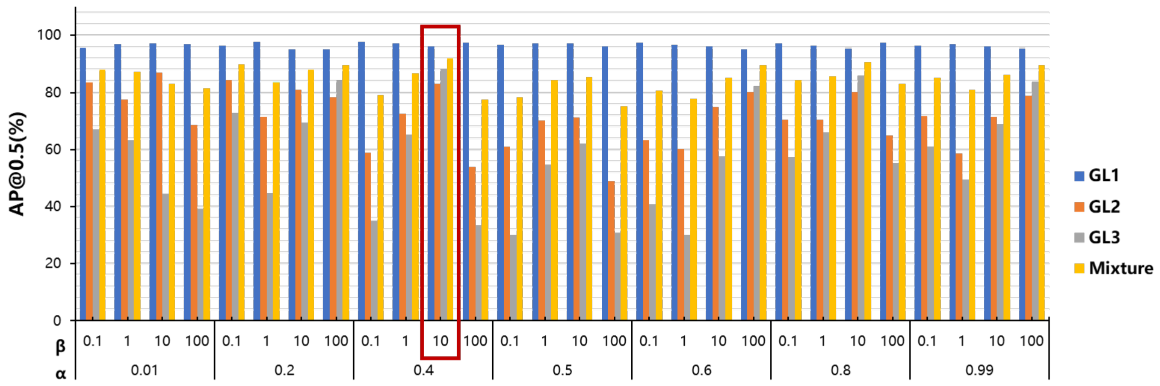

Figure S1. Hyper-parameters tuning experiments for α and β. The data in the red box represents the highest AP@0.5 values.

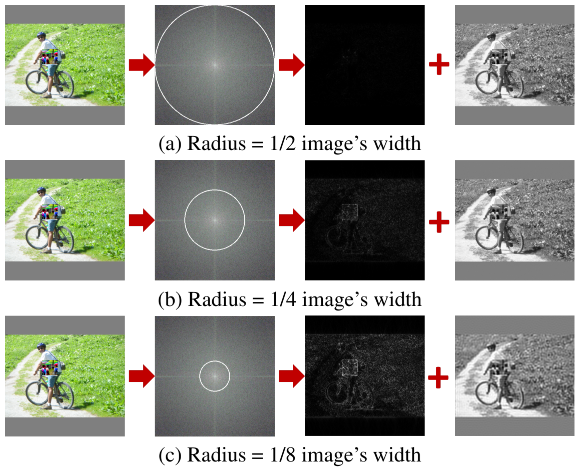
(a) Radius = 1/2 image's width
(b) Radius = 1/4 image's width
(c) Radius = 1/8 image's width

Figure S2. Visualization of the filtering results for different radii, from left to right: original adversarial examples, frequency domain images, high-pass filtered images, and low-pass filtered images.

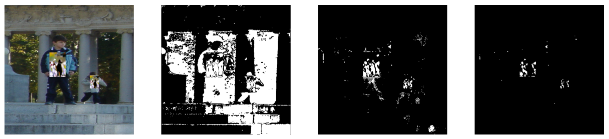

Figure S3. Original naturalistic adversarial example and the mask under different threshold γ, from left to right, 1.0, 2.0 and 3.0, respectively.

formance within our framework.

Hyper-parameters of Natural Feature Suppressed Inference (NFSI) Strategy.

The Radius of $\mathbf{R}_L$. In order to determine the appropriate radius of the low-pass filter, denoted as $\mathbf{R}_L$, we conduct experiments using three different radii: 1/8, 1/4, and 1/2 of the image's width. We visualize the filtering results to assess the effectiveness of each radius in separating high-frequency components within the adversarial patch. As shown in Fig. S2, we observe that using a too large radius prevents the filter from adequately separating high-frequency components, resulting in the retention of unwanted details. On the other hand, employing a too small radius leads to inaccurate separation, potentially removing essential natural features. Thus, we set the radius as 1/4 of the image's width to provide accurate separation of high-frequency features within the adversarial patch while maintaining the essential natural features required for further analysis.

The Weight γ. Regarding the weight γ that controls the threshold of region selection, it was set to three different values in our experiments: 1.0, 2.0, and 3.0. As shown in Fig. S3, a small threshold (*e.g.*, 1.0) would lead to failure of effectively identifying the regions that contain the richest natural features, while a large threshold may include excessively small regions that may have little effect on the overall result. Therefore, we set the weight γ as 2.0 to ensure that the resulting mask accurately selects the important areas within the adversarial example. This choice ensures that the resulting mask effectively captures the relevant regions containing rich natural features, contributing to the overall effectiveness of the approach.

The Standard Variation σ of Gaussian Kernel. As for the standard variation σ that controls the smoothing effect of Gaussian kernel, we conduct quantitative experiments to determine the optimal γ. Specifically, we test values of 0.5, 1.0, 2.0, 3.0, 4.0, 5.0 and 10.0. For each value, we evaluate the detection performance on subsets of the testing set from our GAP dataset, divided based on the attacking methods. It should be noted that in this experiment, the detector is enhanced by the AFAL strategy. This enables us to determine a more optimal σ value, which maximizes the synergistic performance improvement when using both strategies simultaneously. As shown in Tab. S2, we observe that the highest average detection performance is achieved

Table S2. The ablation study results (AP@0.5↑) on σ in our inference strategy. "Average" means the average value of AP@0.5 on NAPs.

σ			0	0.5	1.0	2.0	3.0	4.0	5.0	10.0
Patch Type	Non-NAPs	T-SEA [5]	98.32	98.35	98.37	98.37	98.37	98.63	98.63	98.63
		AdvPatch [16]	96.89	96.97	96.97	96.94	96.95	96.95	96.95	96.95
		AdvCloak [17]	91.22	92.02	92.30	92.25	92.24	92.22	92.22	92.22
		AdvTshirt [18]	65.95	93.98	68.53	69.35	69.53	69.57	69.58	69.60
		AdvTexture [4]	93.71	67.11	94.28	94.22	94.20	94.20	94.19	94.19
	NAPs	GNAP [3]	88.00	88.10	88.24	88.25	88.27	88.27	88.27	88.23
		DM-NAP [9]	98.59	98.61	98.63	98.67	98.66	98.66	98.66	98.66
		LAP [15]	86.13	86.09	86.07	86.00	86.07	86.00	86.00	86.00
		Average	90.91	90.93	90.98	90.97	**91.00**	90.98	90.98	90.96
	Mixture	Mixture	91.89	92.03	92.19	92.21	**92.24**	**92.24**	**92.24**	92.22

Table S3. The detection performance (AP@0.5↑) of YOLOV5 model before and after our defense. Here we use the normal AP@0.5 in object detection instead of patch detection.

	Patch Type (Defense AP↑)							
	T-SEA	AdvPatch	AdvCloak	AdvTshirt	AdvTexture	GNAP-Dog1	DM-NAP-Anime	LAP-Flower
No Defense	1.20	49.65	28.04	28.35	65.05	35.11	61.18	74.59
Ours+	**78.86**	**80.15**	**77.80**	**75.12**	**81.11**	**73.29**	**71.12**	**84.29**

when $\sigma = 3$, *i.e.*, 91.00% AP@0.5 for NAPs on average and 92.24% AP@0.5 for the mixture dataset.

B.3. Defense Potential

Since our method provides the patches' locations, it can serve as a pre-processing stage to defend against adversarial patches. Following the common ”locate-erase-detect” paradigm, we zero out the detected areas and evaluate the performance of YOLOV5 model before and after our defense on various patches (as shown in Tab. S3), using the normal AP for object detection. Results show that our method (**”Ours+”**) improves the average defense AP by 44.15% against Non-NAPs and 19.27% against NAPs, validating its potential in defense.

Furthermore, our strategies hold the potential to be integrated into robustness-enhancing techniques such as adversarial training. By aligning aggressive features and suppressing natural features of generated adversarial examples, we can encourage the model to focus on robust and generalized features, thereby improving its robustness to unseen adversarial examples.

B.4. Fine-grained Results

While our method achieves a high average precision (AP), there are still variations in the detection of certain patch types. Results in Tab. S4 show that the AP on GNAP-Penguin, AdvTshirt, GNAP-Cliffing, LAP-Ivysaur are 68.17%, 69.53%, 80.90% and 82.56% respectively. This indicates that our method still has progress room on fine-grained types of adversarial patches.

C. Adaptability Analysis

In this section, we validate the adaptability of our NAPGuard framework by applying it to other models. Instead of using YOLOV5 [7] as the base model, we train the patch detector based on YOLOV3 [13] and YOLOV3 tiny [1] on our GAP dataset. Then, we apply the AFAL and NFSI strategies to these detectors and evaluate their performance before and after using our strategies.

Specifically, due to the limitation of GPU, we reduce the batch size to 4 for both YOLOV3 and YOLOV3 tiny. Other settings (hyper-parameters, loss functions and augmentations) are aligned with the settings described in Sec. E.1. As for AFAL strategy, we set $\alpha = 0.4$ and $\beta = 10$, which is consistent with the settings used in our main paper. For YOLOV3, we use the backbone as the feature extractor $\mathcal{F}(\cdot)$ and utilize its output to calculate the pattern alignment loss $\mathcal{L}_p$. For YOLOV3 tiny, we utilize the output of the last convolution layer in the backbone to calculate the pattern alignment loss $\mathcal{L}_p$. As for NFSI strategy, we set $\gamma = 2$, the standard deviation $\sigma = 3$ of a 3×3 Gaussian kernel and the radius of the circular low-pass filter $\mathbf{R}_L$ is set as 1/4 of the image's width, which are also consistent with the settings used in our main paper.

Experimental results shown in Tab. S5 demonstrate the effectiveness of our proposed NAPGuard on detecting NAPs when applied to other models. Specifically, we observe a significant increase in AP@0.5 for NAPs based on YOLOV3, with a notable improvement of **+15.79**%. Similarly, when applied to YOLOV3 Tiny, our framework yields a positive impact with an increase of +2.61% AP@0.5 for NAPs. These results confirm that our framework can adapt

Table S4. The detailed experimental results (AP@0.5↑) of our proposed NAPGuard on fine-grained types of adversarial patches.

Non-NAPs	T-SEA	98.37	AdvPatch	96.95	AdvCloak	92.24	AdvTshirt	69.53	AdvTexture	94.20
NAPs	GNAP-Peacock1	95.65	GNAP-Dog2	99.03	GNAP-Penguin	68.17	DM-NAP-Anime	99.04	LAP-Flower	86.87
	GNAP-Peacock2	96.94	GNAP-ATM1	86.62	GNAP-Bulbul	98.94	DM-NAP-Dog	98.53	LAP-Ivysaur	82.56
	GNAP-Dog1	95.21	GNAP-ATM2	91.29	GNAP-Cliffing	80.90	DM-NAP-Princess	98.11	LAP-Shaymin	89.59

Table S5. The experimental results (AP@0.5↑) of our proposed NAPGuard based on YOLOV3 and YOLOV3 tiny. "Base" represents the base detector directly trained on our GAP dataset.

Method		Patch Type								
		Non-NAPs					NAPs			Mixture
		T-SEA [5]	AdvPatch [16]	AdvCloak [17]	AdvTshirt [18]	AdvTexture [4]	GNAP [3]	DM-NAP [9]	LAP [15]	
YOLOV3	Base	**99.49**	90.92	49.57	53.70	33.59	70.59	59.39	13.17	74.05
	+Ours	99.47	**98.72**	**80.18**	**96.09**	**78.11**	**72.69**	**75.30**	**42.53**	**81.77**
YOLOV3 tiny	Base	98.80	50.56	**27.79**	59.18	**56.48**	33.36	27.47	16.18	62.83
	+Ours	**98.90**	**66.42**	26.50	**62.72**	45.29	**34.19**	**32.38**	**18.28**	**64.03**

to other object detection models to improve their detection capabilities for NAPs.

Further, during our experiments, we observe an interesting phenomenon: the effect of our framework on YOLOV3 tiny is comparatively less pronounced than on YOLOV3. This can be attributed to the limited model complexity of YOLOv3 Tiny, which has fewer layers and parameters, restricting its ability to fully benefit from our framework.

D. More Details of GAP Dataset

D.1. Motivation

Necessity. Inspired by Ad-YOLO [6], we consider to harness the potential of object detection models for detecting physical adversarial patches. However, since the adversarial patch detection dataset used in Ad-YOLO is not publicly available and there are no other open source alternatives, we begin by constructing our own dataset to train a patch detector. Furthermore, it is worth noting that the training set of Ad-YOLO lacks NAPs, which may result in limited generalization capabilities. To reduce this generalization gap between NAPs and Non-NAPs, our GAP dataset's training set encompasses both of them, which allows the detector to learn the characteristics of NAPs.

Benchmarking Generalization. Moreover, to study the generalized detection ability of patch detectors, we include a greater variety of adversarial patch types in the testing set compared to the training set. If the training and testing sets have the same or similar adversarial patch types, the model may overly rely on the features of these seen patches. This deliberate choice aims to thoroughly assess the model's performance when encountering unseen adversarial patches, thus closely simulating real-world scenarios. By incorporating diverse adversarial patch types in the testing set, we can provide a more comprehensive benchmark for the model's generalized detection ability, promoting the development of more robust and generalized detection methods.

D.2. Data Details

D.2.1 Data Acquisition

Datasets. All images used in this dataset are derived from the testing set of two widely used datasets in object detection: INRIA-Person [2] and MS COCO [10]. The testing set of INRIA-Person consists of 288 images. Specifically, since the adversarial patches are all based on pedestrian detection task, we only select images containing person from COCO's validation set (named COCO-person), which contains 1684 images.

Adversarial Patches. GAP contains 25 types of distinct adversarial patches from 8 methods including 15 NAPs and 10 Non-NAPs, demonstrating the extensive diversity. In practice, we select 9 patches from GNAP [3], 6 patches from T-SEA [5], 3 patches each from DM-NAP [9] and LAP [15], and 1 patch each from [16], AdvCloak [17], AdvTshirt [18] and AdvTexture [4]. The overall statistics of our dataset are shown in Fig. 3 in the main paper. More details of these patches are provided as follows:

- **T-SEA** [5]: We select 6 types of patches from this method based on the victim model, namely, T-SEA YOLOV2, T-SEA YOLOV3, T-SEA YOLOV3 tiny, T-SEA YOLOV4, T-SEA YOLOV4 tiny and T-SEA YOLOV5. In practice, for each patch type, we generate 288 adversarial examples on INRIA-Person and 1000 on COCO-person. Among these, the adversarial examples generated by the first five patches are allocated to both the training set and testing set, following a 4:1 ratio, while those generated by the last patch are only put into the testing set to evaluate the generalization.
- **AdvPatch** [16]: We select the patch generated by minimizing the objectness score, which achieves the best attacking performance in the paper. We generate 100 adversarial examples on INRIA-Person and 150 adversarial examples on COCO-person. All of these images are in-

cluded in the testing set for evaluation.

- **AdvCloak** [17]: We select the patch trained to attack the YOLOV2 model, which exhibits the best attacking performance. We generate 100 adversarial examples on INRIA-Person, and all of them are placed in the testing set.
- **AdvTshirt** [18]: We select the patch from this method that demonstrates the best attacking performance. We generate 100 adversarial examples on INRIA-Person, and all of these examples are included in the testing set.
- **AdvTexture** [4]: We select the TC-EGA patch from this method, which has the best attacking performance. We generate 100 adversarial examples on INRIA-Person and 150 adversarial examples on COCO-person. All of these images are included in the testing set for evaluation.
- **GNAP** [3]: We select 9 patches from this method based on different initial class, namely, GNAP-Dog1, GNAP-Dog2, GNAP-Peacock1, GNAP-Peacock2, GNAP-ATM1, GNAP-ATM2, GNAP-Cliffing, GNAP-Bulbul and GNAP-Penguin. For each of GNAP-Dog1 and GNAP-Dog2, we generate 288 adversarial examples on INRIA-Person and put them to both the training set and testing set, following a 4:1 ratio. For the other patches, we generate 100 adversarial examples for each type on INRIA-Person and put them into the testing set.
- **DM-NAP** [9]: We select 3 patches from this method based on different initial class, namely, DM-NAP-Dog, DM-NAP-Princess, DM-NAP-Anime. We generate 100 adversarial examples for each type on INRIA-Person and all of these examples are included in the testing set.
- **LAP** [15]: We select 3 patches from this method based on different initial class, namely, LAP-Flower, LAP-Ivysaur, LAP-Shaymin. We generate 100 adversarial examples for each type on INRIA-Person and all of these examples are placed in the testing set.

D.2.2 Data Properties

Patch Scale. For AdvTexture [4], GNAP [3], DM-NAP [9] and LAP [15], we set the patch scale as 0.2. For T-SEA [5], AdvPatch [16], AdvCloak [17] and AdvTshirt [18], we set the patch scale as 0.15.

Image Size. All images are stored in PNG format with a fixed size of 416×416 pixels. For Non-NAPs, image padding is applied to ensure alignment with the fixed size. For NAPs, image resizing is performed to achieve alignment. These settings are consistent with the methodologies outlined in the respective papers.

Annotations. Instead of heavy human annotations, all of the adversarial examples in our GAP dataset are automatically annotated during the generation process, where we can directly obtain and save their positions within the image. Every adversarial patch is located with a bounding-box location.

E. More Experimental Details

In this section, we first list out the training details of the patch detector to facilitate reproducibility. Then, we provide more details about our compared baselines.

E.1. Implementation Details

Models. We train the patch detector utilizing the common used object detection model (*e.g.*, YOLOV5 [7]). First, we convert the model to a single-class model and set the 0 category as "patch". Then, we train the model on our GAP dataset and obtain a patch detector. Specifically, the backbone of YOLOV5 is based on the Darknet-53 architecture and we use YOLOV5s model, which has a smaller model size and fewer parameters. For YOLOV5, we use the backbone as the feature extractor $\mathcal{F}(\cdot)$ and utilize its output to calculate the pattern alignment loss $\mathcal{L}_p$.

Hyper-parameters. The training stage involves a 3 epochs warm-up phase to gradually increase the learning rate from a lower value to the initial learning rate. During training, an SGD optimizer is used with an initial learning rate of 0.01, a momentum value of 0.937, a weight decay of 0.0005 and a final learning rate 0.01. We set the batch size is 16, the image size as 416 to align with the GAP dataset, and the detector is trained for a maximum of 200 epochs.

Loss Functions. The original detection loss consists of three parts: the bounding box loss $\mathcal{L}_{bbox}$, the class loss $\mathcal{L}_{cls}$ and the objectness loss $\mathcal{L}_{obj}$. The weights for these loss are 0.05, 0.5 and 1.0, respectively, to determine the relative importance of each loss component during the training process. Since the detector is a single-class model, the class loss $\mathcal{L}_{cls}$ will always be equal to zero.

Augmentations. The parameters hsv_h, hsv_s, and hsv_v are used to control the augmentation of image HSV (Hue, Saturation, Value) values during training, which are set to 0.015, 0.7 and 0.4, respectively. Additionally, image shifting is applied during training, with a magnitude equal to 0.1 times the image size. Moreover, we introduce the hyper-parameter $scale = 0.5$ to control scale jitter, and augment the data using Mosaic, which combines the original image with three random images.

It is important to clarify that the loss functions and augmentations mentioned in our work already exist in the YOLOv5 framework. We have not introduced any new operations or modifications to the existing implementation, except for our method. The purpose of providing a list of these settings is solely to enhance the reproducibility of our experiments.

E.2. Compared Baselines Details

In this paper, we choose four compared baselines: LGS [12], Ad-YOLO [6], SAC [11] and PatchZero [19]. Here, we provide more details about these methods.

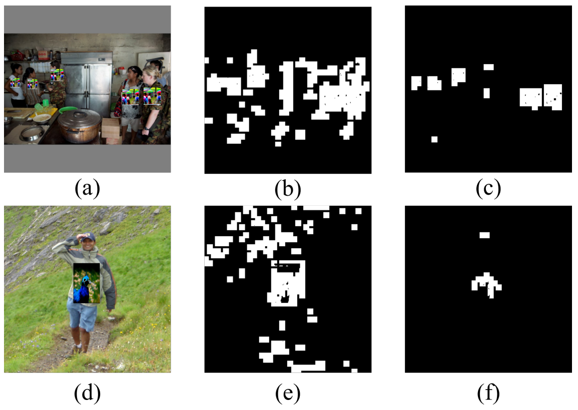
Figure S4. Generated masks by LGS [12] under different threshold. (a) and (d): Original adversarial examples, sampled from GAP. (b) and (e): Masks under 0.1 threshold. (c) and (f): Masks under 0.17 threshold. Our adjustment improves the precision for detecting adversarial patches in our dataset. Best in view.

- **LGS** [12]. LGS (Local Gradients Smoothing) is an image analysis method to estimate noise location in gradient domain and erase their influence. In practice, we set the block size to 15, overlap to 5 and smoothing factor to 2.3, which aligns with the experimental setting of the paper. We find that the threshold of 0.1 specified in the experimental setting is too small for our dataset, resulting in the inclusion of large irrelevant regions in the generated masks. To address this issue and improve performance on our dataset, we make an adjustment by increasing the threshold to 0.17 (as shown in Fig. S4).
- **APE** [8]. APE (Adversarial Patch-Feature Energy) is an image analysis method to defend against adversarial patches by exploiting common deep feature characteristics. In practice, we use the YOLOV2 model as the vanilla detector and select 1, 4, 5 layers to generate masks, which aligns with the experimental setting of the paper.
- **Ad-YOLO** [6]. Ad-YOLO (Adversarial YOLO), similar to our method, utilizes an object detection model to detect adversarial patches. However, Ad-YOLO obtains a patch detector by including a new category called "patch" at the end of the category list. As a result, the detector encompasses a total of 81 classes.
  Given that the training process of Ad-YOLO incorporates the original labels of the clean dataset, we make adjustments to our GAP dataset accordingly and obtain the GAP-adjusted dataset. Thus, in the GAP-adjusted dataset, we add the original labels of the clean images that correspond to the adversarial examples. Further, we include the corresponding clean images in our training set, which aligns with the construction of training set of Ad-YOLO. The training settings of Ad-YOLO are consistent with our approach (as shown in Sec. E.1).
- **SAC** [11]. SAC (Segment and Complete) utilizes an image segmentation model (*i.e.*, the U-Net [14]) to detect and remove adversarial patches. We train the patch segmenter according to the process provided in the original paper. First, we generate a set of adversarial examples by attacking the base object detector (*i.e.*, Faster R-CNN). Since our GAP dataset is based on INRIA-Person and MS COCO, we generate adversarial examples on both of these dataset. Then, we use these images to train the patch segmenter. Further, we robustify the segmenter with the self adversarial training. Following the settings in the original paper, we train the patch segmenters for five epochs by using RMSprop optimizer with an initial learning rate of 10^{-4}, momentum 0.9, weight decay 10^{-8} and batch size 16. For self adversarial training, we train the segmenter for five epochs with batch size 10 (reduce due to GPU constraints), $\lambda = 0.3$ using PGD attacks with 200 iterations, step size $\alpha = 0.01$ and patch size 100×100, where λ controls the weight between clean and adversarial images.
- **PatchZero** [19]. Similar to SAC, PatchZero also utilizes an image segmentation model (*i.e.*, the PSPNet [20]) to detect and remove adversarial patches. We train the patch detector according to the two-stage training provided in the original paper. First, we attack the downstream detector (*i.e.*, Faster R-CNN) and generate a mixture of benign and adversarial images to train the patch detector. Since our GAP dataset is based on INRIA-Person and MS COCO, we generate adversarial examples on both of these dataset. Then, we generate adversarial patches at every training step with updated model weights. For the two-stage training, we use the Adam optimizer with a learning rate of 0.0001 and set the batch size as 12 (reduce due to GPU constraints). Since the authors did not report the training epoch in their paper, we train both stages for five epochs. For attacking, we use the Masked PGD attack with a perturbation strength of 0.3, step size of 0.1, patch size 120×120 and 100 iterations, which aligns with the original paper.

References

[1] Pranav Adarsh, Pratibha Rathi, and Manoj Kumar. Yolo v3-tiny: Object detection and recognition using one stage improved model. In *2020 6th international conference on advanced computing and communication systems (ICACCS)*, pages 687–694. IEEE, 2020. 3

[2] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, 1: 886–893 vol. 1, 2005. 4

[3] Yuqing Hu, Jun-Cheng Chen, Bo-Han Kung, K. Hua, and Daniel Stanley Tan. Naturalistic physical adversarial patch for object detectors. *2021 IEEE/CVF International Confer-*

ence on Computer Vision (ICCV), pages 7828–7837, 2021. 1, 3, 4, 5

[4] Zhan Hu, Siyuan Huang, Xiaopei Zhu, Xiaolin Hu, Fuchun Sun, and Bo Zhang. Adversarial texture for fooling person detectors in the physical world. *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 13297–13306, 2022. 1, 3, 4, 5

[5] Hao Huang, Ziyan Chen, Huanran Chen, Yongtao Wang, and Kevin Zhang. T-sea: Transfer-based self-ensemble attack on object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 20514–20523, 2023. 1, 3, 4, 5

[6] Nan Ji, YanFei Feng, Haidong Xie, Xueshuang Xiang, and Naijin Liu. Adversarial yolo: Defense human detection patch attacks via detecting adversarial patches. *ArXiv*, abs/2103.08860, 2021. 4, 5, 6

[7] Glenn Jocher, Ayush Chaurasia, Alex Stoken, Jirka Borovec, NanoCode012, Yonghye Kwon, Kalen Michael, TaoXie, Jiacong Fang, imyhxy, Lorna, (Zeng Yifu), Colin Wong, Abhiram V, Diego Montes, Zhiqiang Wang, Cristi Fati, Jebastin Nadar, Laughing, UnglvKitDe, Victor Sonck, tkianai, yxNONG, Piotr Skalski, Adam Hogan, Dhruv Nair, Max Strobel, and Mrinal Jain. ultralytics/yolov5: v7.0 - YOLOv5 SOTA Realtime Instance Segmentation, 2022. 3, 5

[8] Taeheon Kim, Youngjoon Yu, and Yong Man Ro. Defending physical adversarial attack on object detection via adversarial patch-feature energy. In *Proceedings of the 30th ACM International Conference on Multimedia*, pages 1905–1913, 2022. 6

[9] Shuo-Yen Lin, Ernie Chu, Che-Hsien Lin, Jun-Cheng Chen, and Jia-Ching Wang. Diffusion to confusion: Naturalistic adversarial patch generation based on diffusion model for object detector, 2023. 1, 3, 4, 5

[10] Tsung-Yi Lin, M. Maire, Serge J. Belongie, James Hays, P. Perona, Deva Ramanan, Piotr Dollár, and C. L. Zitnick. Microsoft coco: Common objects in context. pages 740–755, 2014. 4

[11] Jiangjiang Liu, Alexander Levine, Chun Pong Lau, Ramalingam Chellappa, and S. Feizi. Segment and complete: Defending object detectors against adversarial patch attacks with robust patch detection. *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 14953–14962, 2021. 5, 6

[12] Muzammal Naseer, Salman Hameed Khan, and F. Porikli. Local gradients smoothing: Defense against localized adversarial attacks. *2019 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 1300–1307, 2018. 5, 6

[13] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018. 3

[14] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18*, pages 234–241. Springer, 2015. 6

[15] Jia Tan, Nan Ji, Haidong Xie, and Xueshuang Xiang. Legitimate adversarial patches: Evading human eyes and detection models in the physical world. In *Proceedings of the 29th ACM international conference on multimedia*, pages 5307–5315, 2021. 1, 3, 4, 5

[16] Simen Thys, Wiebe Van Ranst, and T. Goedemé. Fooling automated surveillance cameras: Adversarial patches to attack person detection. *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 49–55, 2019. 1, 3, 4, 5

[17] Zuxuan Wu, Ser-Nam Lim, Larry S Davis, and Tom Goldstein. Making an invisibility cloak: Real world adversarial attacks on object detectors. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part IV 16*, pages 1–17. Springer, 2020. 1, 3, 4, 5

[18] Kaidi Xu, Gaoyuan Zhang, Sijia Liu, Quanfu Fan, Mengshu Sun, Hongge Chen, Pin-Yu Chen, Yanzhi Wang, and Xue Lin. Adversarial t-shirt! evading person detectors in a physical world. pages 665–681, 2019. 1, 3, 4, 5

[19] Ke Xu, Yao Xiao, Zhaoheng Zheng, Kaijie Cai, and Ram Nevatia. Patchzero: Defending against adversarial patch attacks by detecting and zeroing the patch. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 4632–4641, 2023. 5, 6

[20] Hengshuang Zhao, Jianping Shi, Xiaojuan Qi, Xiaogang Wang, and Jiaya Jia. Pyramid scene parsing network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2881–2890, 2017. 6