Robustness of Object Detection Models to Adversarial Attacks

Suryaansh Jain
CS21BTECH11057

Kushagra Gupta
CS21BTECH11033

Pranav Balasubramanian
AI21BTECH11023

Ankit Saha
AI21BTECH11004

Abstract

This is the final report for our project containing all the work we have done. We have used two kinds of object detectors: the one-stage, YOLO and the two-stage, Faster RCNN and analyzed their robustness to several adversarial attacks. We have implemented gradient-based adversarial attacks FGSM, DPAttack and PGD from literature for these models. In addition to that, we have proposed and implemented our own attack called Edge-DPAttack, based on edge detection. We also implemented an RL-based attack. We used the COCO 2017 object detection dataset for our experiments.

1. Architectures Tested

1.1. YOLO v5s [9]

Instead of generating region proposals, these one stage detector models directly predict bounding boxes and class probabilities from the image.

YOLO (You Only Look Once) treats object detection as a single regression problem, directly predicting bounding boxes and class probabilities from the input image in a single forward pass of the neural network.

The input image is divided into an $S \times S$ grid. Each grid cell is responsible for detecting objects whose center falls within it. For each grid cell, the model predicts:

- B bounding boxes, each with:
  - Coordinates (x, y, w, h)
  - An objectness score (confidence that an object exists in the box)
- Class probabilities for C object classes

The final prediction combines the objectness score and class probabilities to give class-specific confidence scores for each bounding box. YOLO outputs all predictions simultaneously, making it extremely fast and suitable for real-time applications.

1.2. Faster R-CNN [8]

Faster R-CNN is a two-stage object detection framework that improves both speed and accuracy over its predecessors. It first generates region proposals and then classifies and refines them.

The process involves two main stages:

1. **Region Proposal Network (RPN):** The input image is passed through a convolutional backbone (e.g., ResNet, VGG) to extract feature maps. The RPN slides a small network over these feature maps and generates a set of object proposals (regions likely to contain objects). Each proposal is defined by:
   - Bounding box coordinates
   - An objectness score (object vs. background)
2. **ROI Classification and Refinement:** The proposed regions are then pooled from the feature map using ROI Pooling (or ROI Align). Each ROI is passed through a classifier to:
   - Predict the object class
   - Refine the bounding box coordinates

This two-stage approach allows Faster R-CNN to focus on a smaller set of high-quality proposals, leading to more accurate detection compared to single-stage methods.

2. Adversarial Attacks from Literature

2.1. Fast Gradient Sign Method (FGSM) [11]

The Fast Gradient Sign Method (FGSM) is a white-box attack that uses the gradients of the model to form an adversarial input. The basic idea is to maximize the loss because higher loss generally indicates poorer model performance.

If $\mathbf{x}$ is the original input image, y is the model prediction, $\boldsymbol{\theta}$ is the set of model parameters, $\mathcal{L}$ is the loss function (to be maximized by the adversary), then the FGSM adversarial image corresponding to $\mathbf{x}$ is given by

$$\hat{\mathbf{x}}_{\text{FGSM}} = \mathbf{x} + \epsilon \operatorname{sign}\{\nabla_{\mathbf{x}} \mathcal{L}(\mathbf{x}, y; \boldsymbol{\theta})\} \tag{1}$$

where $\epsilon > 0$ is just used for scaling. Note that the gradient is with respect to $\mathbf{x}$. For those points in the image

where the gradient of the loss with respect to the image is positive, we would like to increase the pixel values at that location to increase the loss. And if the gradient is negative, we would like to decrease the pixel values at that location. This is essentially being captured by the sign function on the gradient.

We have chosen the negative of the confidence scores as the loss function to be maximized by the adversary, *i.e.*, the confidence score has to be minimized. We have experimented with various values of $\epsilon = 0.01, 0.05, 0.1$ to compare their performances.

2.2. Diffused Patch Attack (DPAttack) [10]

The Diffused Patch Attack (DPAttack) builds upon FGSM. Instead of perturbing the entire image, DPAttack only changes the values of a very small number of pixels. Patches are applied on every bounding box detected in an image, and the values of these patches are optimized iteratively using the gradients. If the binary mask corresponding to the positions of the patches is denoted by $\mathbf{M}$ and the patch image is $\boldsymbol{\delta}$, then the modified image corresponding to the original input image $\mathbf{x}$ is given by

$$\hat{\mathbf{x}}_{\text{DPAttack}}(\boldsymbol{\delta}, \mathbf{M}) = \mathbf{x} \odot (\mathbf{1} - \mathbf{M}) + \boldsymbol{\delta} \odot \mathbf{M} \tag{2}$$

where $\odot$ is the Hadamard (element-wise) product. The optimal $\boldsymbol{\delta}$ is found using an iterative FGSM-like process. At each iteration, $\boldsymbol{\delta}$ is updated to minimize the confidence-based loss $\mathcal{L}$. The goal of the adversary here is to suppress all detected boxes in an image by reducing their confidence scores below a given threshold t.

$$\boldsymbol{\delta}^{(t+1)} = \boldsymbol{\delta}^{(t)} - \epsilon \, \text{sign}\{\nabla_{\boldsymbol{\delta}^{(t)}} \mathcal{L}(\hat{\mathbf{x}}_{\text{DPAttack}}(\boldsymbol{\delta}^{(t)}, \mathbf{M}); \boldsymbol{\theta})\} \tag{3}$$

where $\epsilon > 0$ is used for scaling and θ is the set of model parameters. Note that the mask $\mathbf{M}$ remains constant throughout the optimization process.

We have used $\max\{0, \text{confidence} - t\}$ as the loss function to be minimized by the adversary. For the patch, we are using a 4×4 grid as the adversarial patch. For bounding boxes smaller than this size, we are accordingly reducing the grid size to 2×2. We have experimented with the $\epsilon = 0.3, 1.0$ and the number of iterations $T = 10, 20$ here.

2.3. Projected Gradient Descent (PGD) [6]

Projected Gradient Descent (PGD) can be viewed as an iterative extension of FGSM. While FGSM applies a single-step perturbation, PGD performs multiple iterations, each involving a small perturbation in the direction that maximizes the model's loss, followed by a projection back onto the allowed perturbation set to ensure the adversarial example remains within a specified norm constraint.

Mathematically, given an original input $\mathbf{x}$, true label y, model parameters $\boldsymbol{\theta}$, loss function $\mathcal{L}$, perturbation budget ϵ, and step size α, the PGD adversarial example is generated through the following iterative process:

$$\mathbf{x}^{(0)} = \mathbf{x} + \boldsymbol{\delta}^{(0)}, \quad \text{where } \boldsymbol{\delta}^{(0)} \sim \mathcal{U}(-\epsilon, \epsilon)$$
$$\mathbf{x}^{(t+1)} = \Pi_{\mathcal{B}_\epsilon(\mathbf{x})} \left(\mathbf{x}^{(t)} + \alpha \cdot \text{sign} \left(\nabla_{\mathbf{x}} \mathcal{L}(\mathbf{x}^{(t)}, y; \boldsymbol{\theta}) \right) \right)$$

Here, $\Pi_{\mathcal{B}_\epsilon(\mathbf{x})}$ denotes the projection operator that ensures the perturbed input remains within the ϵ-ball (typically under the L_∞ norm) around the original input $\mathbf{x}$. The process is repeated for a predefined number of iterations or until convergence.

We have used two combinations of $(\epsilon, \alpha) = (\frac{8}{255}, \frac{2}{255})$ and $(\epsilon, \alpha) = (\frac{16}{255}, \frac{4}{255})$. These two combinations have each been used with $T = 10, 20$, thus giving us four combinations in total.

2.4. Targeted Adversarial Objectness Gradient Attack (TOG) [1]

The Targeted Objectness Gradient (TOG) attack is a white-box adversarial attack designed specifically for object detectors. Unlike untargeted attacks that aim to suppress object detection altogether, TOG is a targeted attack which seeks to make the detector misidentify one object class as another. This is achieved by directly manipulating the objectness gradients, which are central to how modern detectors localize and classify objects.

The key insight behind TOG is to modify the input in a way that causes the objectness score of a specific target class to increase, while suppressing the original class. Let $\mathbf{x}$ be the original input image, and let the adversary's goal be to mislead the model into detecting an object of class y_{target} instead of y_{true}. TOG modifies the image by computing gradients with respect to a specialized loss function:

$$\mathcal{L}_{\text{TOG}} = \mathcal{L}_{\text{objectness}}(y_{\text{target}}) - \mathcal{L}_{\text{objectness}}(y_{\text{true}}) \tag{4}$$

where $\mathcal{L}_{\text{objectness}}$ denotes the loss based on objectness scores for the respective classes. The adversarial image is then constructed using a single-step or iterative update:

$$\hat{\mathbf{x}}_{\text{TOG}} = \mathbf{x} + \epsilon \cdot \text{sign}\left(\nabla_{\mathbf{x}} \mathcal{L}_{\text{TOG}}\right) \tag{5}$$

where ϵ controls the strength of the perturbation.

We have used the TOG implementation available at this repository https://github.com/git-disl/TOG for the YOLO v3 and Faster RCNN models. Unfortunately, we could run it over the entire dataset and compute its metrics. But, we have performed the various kinds of attack on a few images, which are shown in the later sections.

3. Our Proposed Attacks:

3.1. Edge-DPAttack

If a human being is shown the silhouette of a dog, they would easily identify it as a dog even without any details

present in the image. Thus, we conjecture that the edges of an object are important for an object detection model as well. This forms the inspiration for our proposed attack: Edge-DPAttack.

Edge-DPAttack is a variant of DPAttack, where the mask $\mathbf{M}$ consists of the object edges detected in the image instead of a grid. We optimize the patch present along the edges iteratively using FGSM updates just like before. The edges are found by first applying a 5×5 Gaussian blur with $\sigma = (1, 1)$ and then using a Canny filter with thresholds 200 and 250. This binary edge mask is then multiplied element-wise with the bounding box masks to obtain the $\mathbf{M}$ to be used for a DPAttack.

$$\text{edges} = \text{Canny}(\text{GaussianBlur}(\mathbf{x})) \tag{6}$$

$$\mathbf{M} = \text{edges} \odot \text{boxes}(\mathbf{x}) \tag{7}$$

$$\hat{\mathbf{x}}_{\text{Edge-DPAttack}}(\boldsymbol{\delta}, \mathbf{M}) = \mathbf{x} \odot (\mathbf{1} - \mathbf{M}) + \boldsymbol{\delta} \odot \mathbf{M} \tag{8}$$

where $\boldsymbol{\delta}$ is obtained by iterative optimization just like in DPAttack.

We have used the same loss function as DPAttack and experimented with the $\epsilon = 0.3, 1.0$ and the number of iterations $T = 10, 20$.

3.2. RL Attack

This attack uses reinforcement learning, specifically, a Deep Q-Network (DQN) to train an agent to optimally pick a patch of the image and a certain perturbation to the patch so that the detection performance goes down. The states, actions, and rewards are defined as follows:

- **State:** The current image is as the state of the environment. All images are resized to a uniform size of 640×640.
- **Action:** The image is divided into 100 non-overlapping patches of size 64×64. An action is defined as selecting a specific patch and applying a perturbation to it. Possible perturbations include Gaussian noise, salt-and-pepper noise, Gaussian blur, median blur, sharpening, brightening, darkening, increasing contrast, descreasing contrast.
- **Reward:** The reward for a selected action is the sum of two components. A reward for reducing the object detection confidence score and a penalty for deviating too much from the original image, measured in terms of RMSE.

$$r(a) = (1 - o) - \sqrt{\frac{1}{N} \sum_{i,j} (s_{ij} - s'_{ij})^2} \tag{9}$$

where:
- $r(a)$ is the reward for the selected action a.
- o is the objectness score produced by the object detector.
- $\mathbf{s}$ is the original state (image).
- $\mathbf{s}'$ is the perturbed state after applying the action.

The term $(1 - o)$ provides a higher reward for a lower objectness score, encouraging successful attacks. The RMSE penalizes significant deviations from the original image.

Unfortunately, due to time and resource constraints, we could not train the DQN. Training with just one-timestep episodes for one epoch over the training set was alone taking about 20 minutes. We only have its implementation ready.

4. COCOeval [4]

To quantitatively evaluate the performance of the object detection models, we used the COCOeval evaluation tool from the `pycocotools` library. This tool computes standardized COCO-style metrics, providing a comprehensive understanding of detection quality across various object sizes and Intersection over Union (IoU) thresholds.

We formatted our model's predictions into the COCO JSON format, which includes fields such as `image_id`, `category_id`, `bbox`, and `score` for each detection. Here, `bbox` is in the format `[x_min, y_min, width, height]`, representing the top-left corner of the bounding box along with its width and height in pixels, while `score` denotes the confidence of the detection.

Example: COCO JSON Format for a Detection

```
{
"image_id": 42,
"category_id": 18,
"bbox": [5.34, 2.12, 180.0,
210.6],
"score": 0.932
}
```

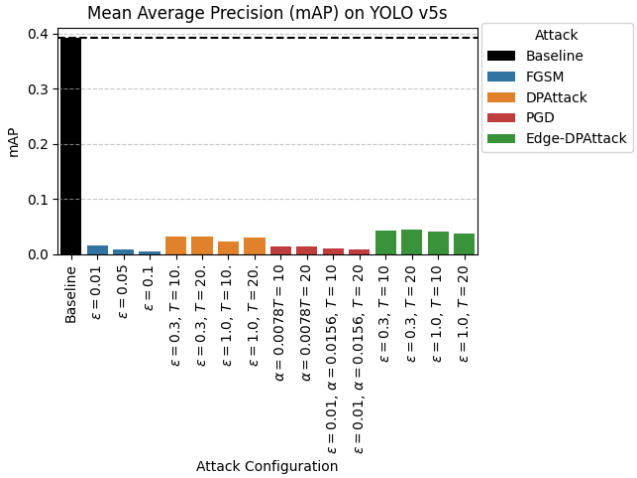
For ground truth annotations, we used the `instances_val2017.json` file, which is included with the COCO. dataset during installation. Unlike predictions, this file excludes the `score` field. Instead, it contains additional fields such as `area`, `iscrowd`, and `id`, which are required by COCOeval to properly compute metrics such as Average Precision and Recall.

Refer to table 3 for the interpretation of various metrics provided by COCOeval.

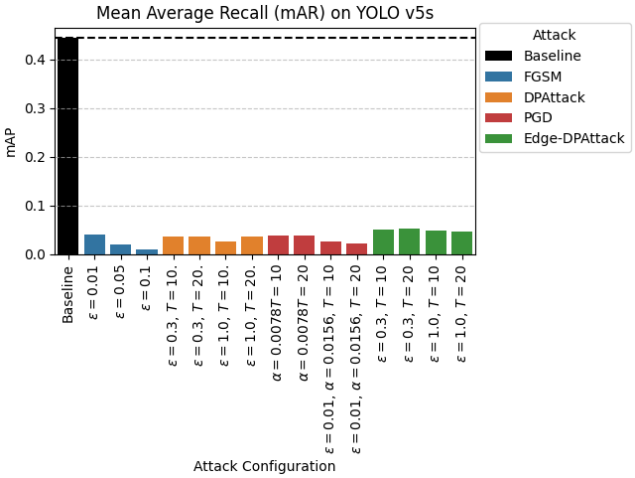
5. Expermiments and Results

The barplots 1 and 2 contain a comparison of the mean average precision (mAP) and the mean average recall (mAR) over IoU thresholds $0.50 : 0.95$ and maxDets $= 100$

Refer to table 1 and 2 to see COCO evaluations on the models YOLOv5s and Faster RCNN for each of the attacks.

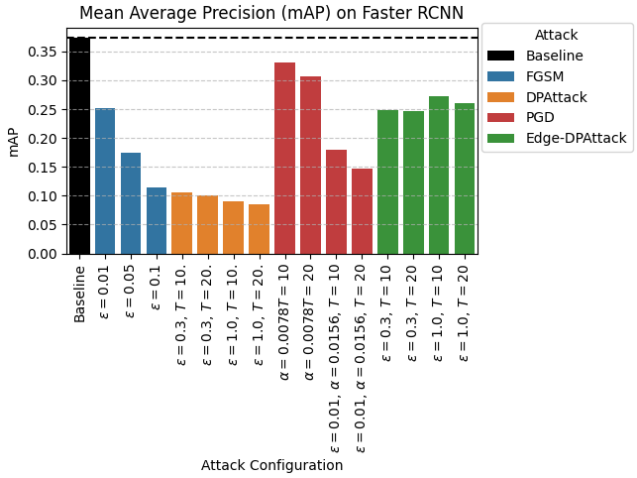


(a) mAP

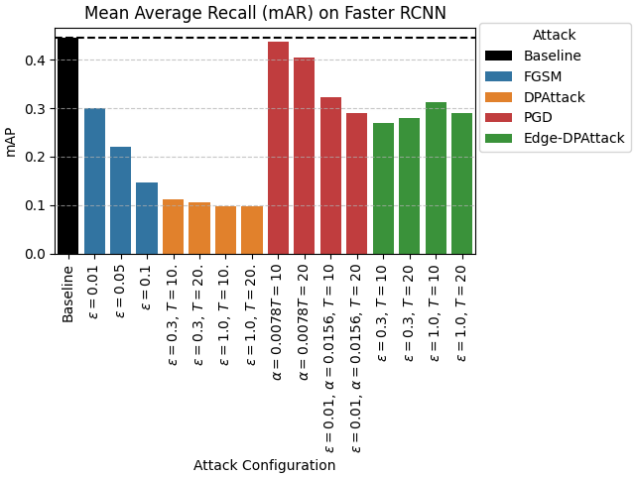


(b) mAR

Figure 1. Comparison of all attacks on YOLO v5s



(a) mAP



(b) mAR

Figure 2. Comparison of all attacks on Faster RCNN

We have also added some examples of adversarial images for each model and attack at the end of this report.

6. Conclusions

The most significant observation here is that Faster R-CNN is far more robust to adversarial attacks as compared to YOLO v5s. This is evident by the much larger decrease in mAP and mAR for every attack. This is likely due to the fact that Faster R-CNN is a two-stage detector whereas YOLO is a one-stage detector. Thus, the multi-stage nature of detection is making it more robust.

Another observation is that on YOLO, FGSM and PGD are performing the best (although it is very close), whereas on Faster R-CNN, DPAttack is performing the best. Also note that DPAttack performed so well despite perturbing very few pixels, as compared to attacks like FGSM and PGD. Our attack Edge-DPAttack also performed decently well, taking into consideration that we did not modify the internals of the objects (as opposed to DPAttack), only its edges.

Increasing the degree of perturbation also improves performance, as expected.

Metric	IoU	Area	Baseline	dpattack $\epsilon = 0.3$, iters=20, grid=3	dpattack $\epsilon = 0.3$, iters=10, grid=3	dpattack $\epsilon = 1.0$, iters=10, grid=3	dpattack $\epsilon = 1.0$, iters=20, grid=3	fgsm $\epsilon = 0.05$	fgsm $\epsilon = 0.1$	fgsm $\epsilon = 0.01$	edge attack $\epsilon = 0.3$, iters=20, grid=3	edge attack $\epsilon = 0.3$, iters=10, grid=3	edge attack $\epsilon = 1.0$, iters=20, grid=3	edge attack $\epsilon = 1.0$, iters=10, grid=3	pgdattack $\epsilon = 0.31$, alpha=0.007843, iters=10	pgdattack $\epsilon = 0.31$, alpha=0.007843, iters=20	pgdattack $\epsilon = 0.62$, alpha=0.015686, iters=10	pgdattack $\epsilon = 0.62$, alpha=0.015686, iters=20
Average Precision (AP)	0.50:0.95	all	0.392	0.032	0.031	0.022	0.030	0.015	0.009	0.004	0.044	0.042	0.037	0.040	0.014	0.014	0.010	0.009
	0.50	all	0.518	0.068	0.065	0.048	0.059	0.049	0.026	0.013	0.099	0.096	0.086	0.095	0.047	0.043	0.030	0.028
	0.75	all	0.428	0.023	0.024	0.018	0.019	0.005	0.004	0.002	0.034	0.030	0.030	0.030	0.005	0.005	0.004	0.004
	0.50:0.95	small	0.174	0.000	0.000	0.000	0.000	0.001	0.000	0.000	0.011	0.000	0.000	0.000	0.001	0.001	0.001	0.000
	0.50:0.95	medium	0.441	0.003	0.003	0.000	0.002	0.011	0.006	0.003	0.002	0.002	0.002	0.002	0.009	0.010	0.006	0.004
	0.50:0.95	large	0.530	0.068	0.059	0.047	0.059	0.039	0.021	0.009	0.083	0.084	0.079	0.083	0.037	0.038	0.022	0.022
Average Recall (AR)	0.50:0.95	all (maxDets=1)	0.311	0.030	0.028	0.023	0.028	0.026	0.015	0.008	0.038	0.034	0.031	0.033	0.026	0.026	0.018	0.017
	0.50:0.95	all (maxDets=10)	0.441	0.036	0.035	0.025	0.036	0.040	0.020	0.009	0.052	0.051	0.046	0.049	0.038	0.037	0.026	0.022
	0.50:0.95	all (maxDets=100)	0.444	0.036	0.035	0.025	0.036	0.040	0.020	0.009	0.052	0.051	0.046	0.049	0.038	0.038	0.026	0.022
	0.50:0.95	small	0.187	0.000	0.000	0.000	0.000	0.002	0.001	0.000	0.011	0.000	0.000	0.000	0.001	0.001	0.001	0.000
	0.50:0.95	medium	0.490	0.002	0.003	0.000	0.002	0.020	0.009	0.005	0.002	0.004	0.002	0.003	0.017	0.019	0.009	0.009
	0.50:0.95	large	0.597	0.077	0.068	0.053	0.070	0.093	0.043	0.018	0.103	0.103	0.100	0.102	0.089	0.089	0.059	0.051

Table 1. COCO evaluation results for yolov5s

Metric	IoU	Area	Baseline	dpattack $\epsilon = 0.3$, iters=20, grid=3	dpattack $\epsilon = 0.3$, iters=10, grid=3	dpattack $\epsilon = 1.0$, iters=10, grid=3	dpattack $\epsilon = 1.0$, iters=20, grid=3	fgsm $\epsilon = 0.05$	fgsm $\epsilon = 0.1$	fgsm $\epsilon = 0.01$	edge attack $\epsilon = 0.3$, iters=20, grid=3	edge attack $\epsilon = 0.3$, iters=10, grid=3	edge attack $\epsilon = 1.0$, iters=20, grid=3	edge attack $\epsilon = 1.0$, iters=10, grid=3	pgdattack $\epsilon = 0.31$, alpha=0.007843, iters=10	pgdattack $\epsilon = 0.31$, alpha=0.007843, iters=20	pgdattack $\epsilon = 0.62$, alpha=0.015686, iters=10	pgdattack $\epsilon = 0.62$, alpha=0.015686, iters=20
Average Precision (AP)	0.50:0.95	all	0.373	0.100	0.106	0.091	0.085	0.252	0.175	0.115	0.247	0.249	0.260	0.273	0.331	0.307	0.180	0.147
	0.50	all	0.564	0.159	0.158	0.154	0.151	0.389	0.284	0.185	0.389	0.385	0.401	0.413	0.526	0.496	0.317	0.260
	0.75	all	0.413	0.110	0.113	0.095	0.088	0.279	0.187	0.124	0.252	0.272	0.284	0.293	0.336	0.331	0.184	0.153
	0.50:0.95	small	0.198	0.006	0.010	0.015	0.010	0.113	0.064	0.042	0.095	0.094	0.106	0.115	0.272	0.268	0.132	0.120
	0.50:0.95	medium	0.398	0.029	0.033	0.080	0.042	0.255	0.169	0.115	0.234	0.230	0.262	0.252	0.361	0.303	0.210	0.164
	0.50:0.95	large	0.472	0.238	0.235	0.201	0.212	0.355	0.275	0.177	0.440	0.425	0.449	0.486	0.444	0.469	0.219	0.195
Average Recall (AR)	0.50:0.95	all (maxDets=1)	0.303	0.089	0.097	0.079	0.086	0.227	0.172	0.114	0.208	0.208	0.216	0.228	0.293	0.272	0.186	0.162
	0.50:0.95	all (maxDets=10)	0.438	0.106	0.111	0.098	0.097	0.299	0.221	0.145	0.280	0.270	0.291	0.313	0.432	0.397	0.309	0.277
	0.50:0.95	all (maxDets=100)	0.445	0.106	0.111	0.098	0.097	0.300	0.221	0.146	0.280	0.270	0.291	0.313	0.437	0.404	0.322	0.291
	0.50:0.95	small	0.228	0.009	0.010	0.015	0.010	0.122	0.071	0.046	0.106	0.095	0.117	0.123	0.318	0.315	0.210	0.209
	0.50:0.95	medium	0.466	0.035	0.034	0.083	0.045	0.296	0.208	0.140	0.254	0.244	0.287	0.278	0.472	0.390	0.360	0.318
	0.50:0.95	large	0.563	0.244	0.242	0.207	0.231	0.414	0.344	0.222	0.476	0.452	0.477	0.516	0.536	0.543	0.342	0.320

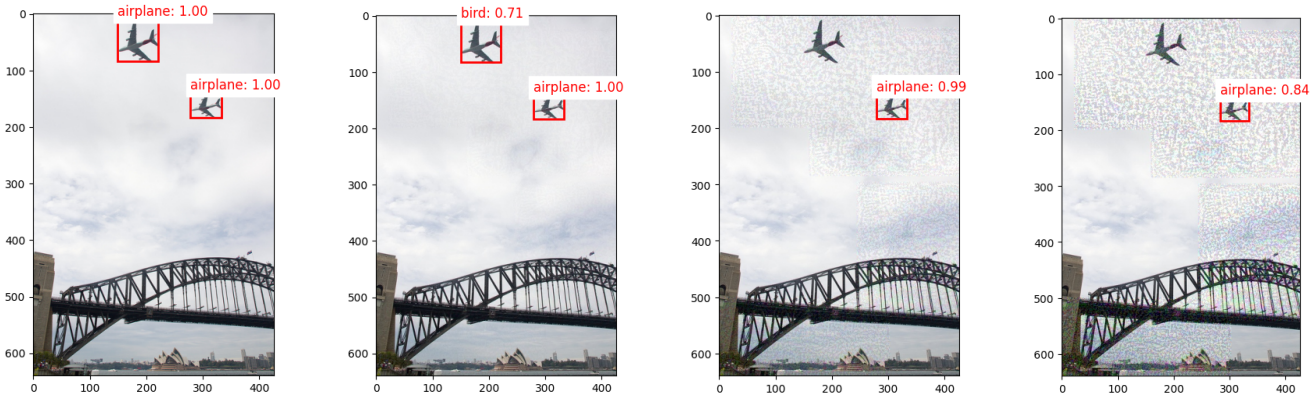
Table 2. COCO evaluation results for Faster R-CNN

References

[1] Ka-Ho Chow, Ling Liu, Mehmet Emre Gursoy, Stacey Truex, Wenqi Wei, and Yanzhao Wu. Tog: Targeted adversarial objectness gradient attacks on real-time object detec-

Table 3. Interpretation of various metrics provided by COCOeval

Metric	**Description**
Average Recall (AR)	Recall measures how many relevant (ground truth) objects are correctly detected by the model. Average Recall (AR) averages recall values over multiple IoU thresholds (from 0.50 to 0.95 in steps of 0.05).
Average Precision (AP)	AP is the area under the Precision-Recall (PR) curve over a set of IoU thresholds. It quantifies how well the model detects objects while avoiding false positives.
AP @[IoU=0.50:0.95]	The primary metric for COCO evaluation, averaging AP over 10 IoU thresholds from 0.50 to 0.95 (step size of 0.05). It balances precision and recall across different localization requirements.
AP @[IoU=0.50] (AP50)	Precision when the IoU threshold is fixed at 0.50. This is a more lenient metric, as detections only need to have 50% overlap with the ground truth.
AP @[IoU=0.75] (AP75)	Precision when the IoU threshold is fixed at 0.75, requiring higher localization accuracy. A significant drop from AP50 to AP75 indicates issues with precise localization.
AR (small)	Average Recall for small objects, where small is defined based on object area in pixels (COCO typically considers objects with an area less than 32^2 pixels).
AR (medium)	Average Recall for medium-sized objects (between 32^2 and 96^2 pixels).
AR (large)	Average Recall for large objects (area greater than 96^2 pixels). Large objects are generally easier to detect, resulting in higher recall.
AP (small)	Average Precision for small objects, reflecting how well the model detects tiny objects that are often occluded or lack texture.
AP (medium)	Average Precision for medium-sized objects.
AP (large)	Average Precision for large objects, where larger objects tend to have higher AP due to better feature extraction and localization.
AP (maxDets=1)	AP when only the top 1 detection per image is considered. This measures how well the model prioritizes the most confident detection.
AP (maxDets=10)	AP when considering up to 10 detections per image, reflecting performance in scenarios with multiple objects per image.
AP (maxDets=100)	AP when considering up to 100 detections per image, the standard setting in COCO evaluation. This allows full evaluation of the model's ability to detect multiple objects.



(a) Baseline (b) $\epsilon = 0.01$ (c) $\epsilon = 0.05$ (d) $\epsilon = 0.1$

Figure 3. Examples of FGSM attack on Faster RCNN

(a) Baseline (b) $\epsilon = 0.01$ (c) $\epsilon = 0.05$ (d) $\epsilon = 0.1$

Figure 4. Examples of FGSM attack on YOLOv5s

	DPAttack		Edge-DPAttack	
Model	$T = 10$	$T = 20$	$T = 10$	$T = 20$
YOLO v5s	#Px = 741073		#Px = 1482146	
	32.02	62.08	64.04	124.16
Faster R-CNN	#Px = 746228		#Px = 1492456	
	$T = 145.38$	$T = 304.03$	$T = 290.74$	$T = 608.07$

Table 4. Total time taken and total number of pixels modified

tion systems, 2020. 2

[2] Ross Girshick Ali Farhadi Joseph Redmon, Santosh Divvala. You only look once: Unified, real-time object detection, 2015.

[3] Jintang Li, Tao Xie, Liang Chen, Fenfang Xie, Xiangnan He, and Zibin Zheng. Adversarial attack on large scale graph, 2021.

[4] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollár. Microsoft coco: Common objects in context, 2015. 3

[5] Xin Liu, Huanrui Yang, Ziwei Liu, Linghao Song, Hai Li, and Yiran Chen. Dpatch: An adversarial patch attack on object detectors, 2019.

[6] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks, 2019. 2

[7] Khoi Nguyen Tiet Nguyen, Wenyu Zhang, Kangkang Lu, Yuhuan Wu, Xingjian Zheng, Hui Li Tan, and Liangli Zhen. A survey and evaluation of adversarial attacks for object detection. *arXiv preprint arXiv:2408.01934*, 2024.

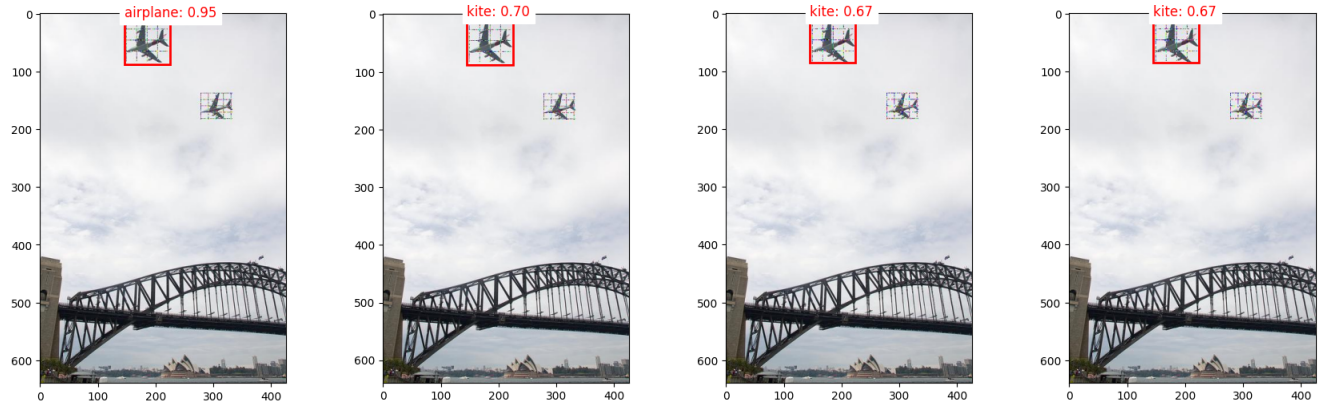
[8] Ross Girshick Jian Sun Shaoqing Ren, Kaiming He. Faster r-cnn: Towards real-time object detection with region proposal networks, 2016. 1

[9] Ultralytics. YOLOv5: A state-of-the-art real-time object detection system. https://docs.ultralytics.com, 2021. 1

[10] Shudeng Wu, Tao Dai, and Shu-Tao Xia. Dpattack: Diffused patch attacks against universal object detection, 2020. 2

[11] Xiaoyong Yuan, Pan He, Qile Zhu, and Xiaolin Li. Adversarial examples: Attacks and defenses for deep learning, 2018. 1

[12] Mu Zhu. Recall, precision and average precision. 2004.

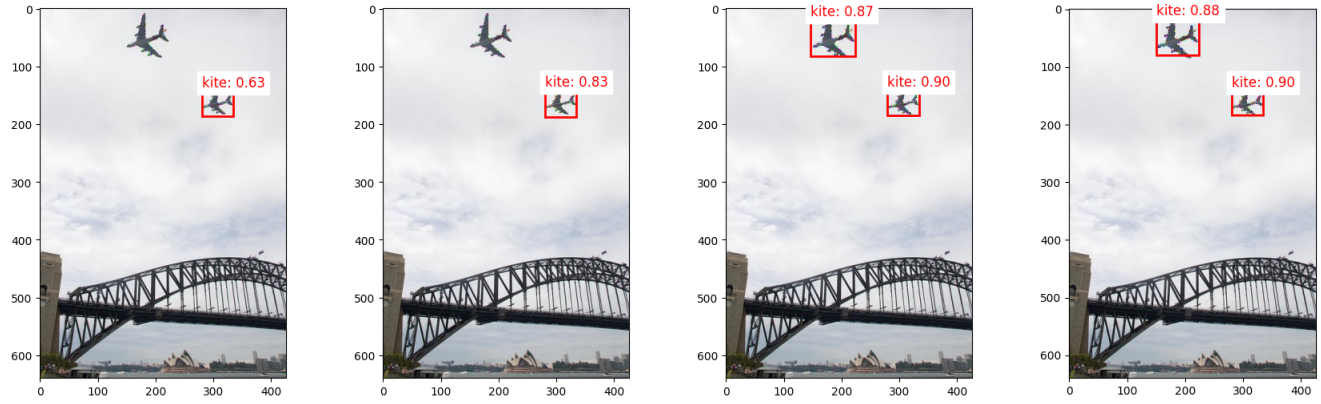


(a) $\epsilon = 0.3,\ T = 10$ (b) $\epsilon = 0.3,\ T = 20$ (c) $\epsilon = 1,\ T = 10$ (d) $\epsilon = 1,\ T = 20$

Figure 5. Examples of DPAttack on Faster RCNN

(a) $\epsilon = 0.3,\ T = 10$ (b) $\epsilon = 0.3,\ T = 20$ (c) $\epsilon = 1,\ T = 10$ (d) $\epsilon = 1,\ T = 20$

Figure 6. Examples of DPAttack on YOLOV5s

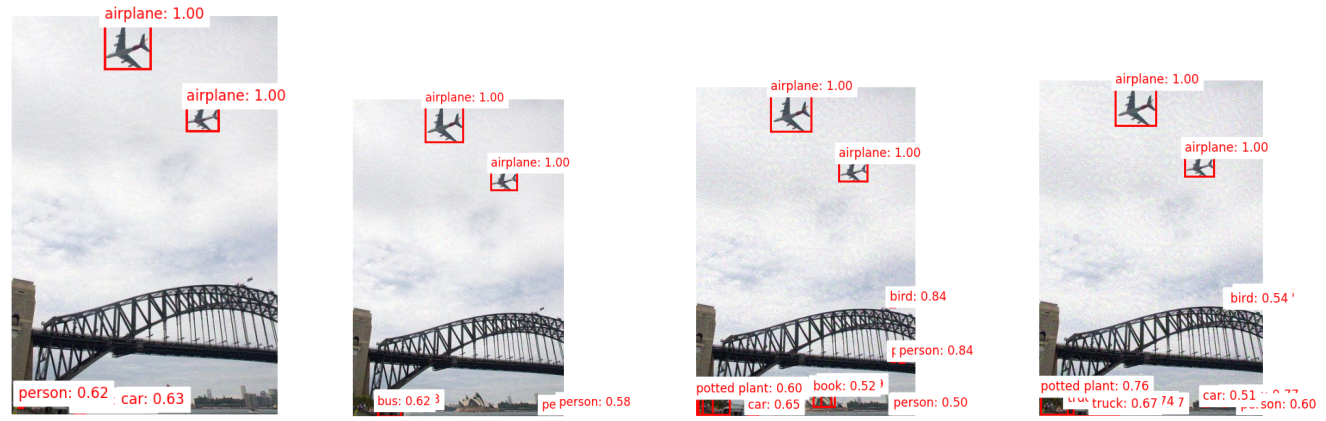


(a) $\epsilon = 0.3,\ T = 10$ (b) $\epsilon = 0.3,\ T = 20$ (c) $\epsilon = 1,\ T = 10$ (d) $\epsilon = 1,\ T = 20$

Figure 7. Examples of Edge-DPAttack on Faster RCNN

(a) $\epsilon = 0.3,\ T = 10$ (b) $\epsilon = 0.3,\ T = 20$ (c) $\epsilon = 1,\ T = 10$ (d) $\epsilon = 1,\ T = 20$

Figure 8. Examples of Edge-DPAttack on YOLOV5s

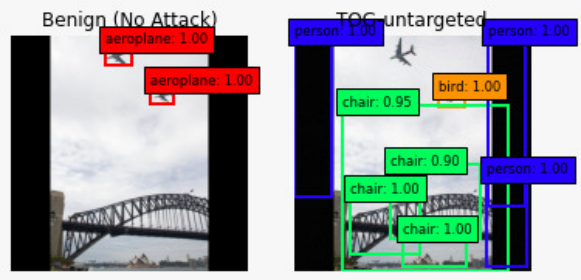


(a) $\epsilon = 0.03,\ \alpha = 0.007, T = 10$ (b) $\epsilon = 0.03,\ \alpha = 0.007, T = 10$ (c) $\epsilon = 0.06,\ \alpha = 0.015, T = 10$ (d) $\epsilon = 0.06,\ \alpha = 0.015, T = 20$

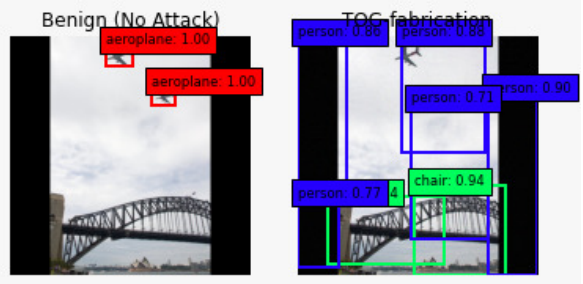
Figure 9. Examples of PGD Attack on Faster RCNN

(a) $\epsilon = 0.03,\ \alpha = 0.007, T = 10$ (b) $\epsilon = 0.03,\ \alpha = 0.007, T = 20$ (c) $\epsilon = 0.06,\ \alpha = 0.015, T = 10$ (d) $\epsilon = 0.06,\ \alpha = 0.015, T = 20$

Figure 10. Examples of PGD-Attack on YOLOV5s



TOG untargeted

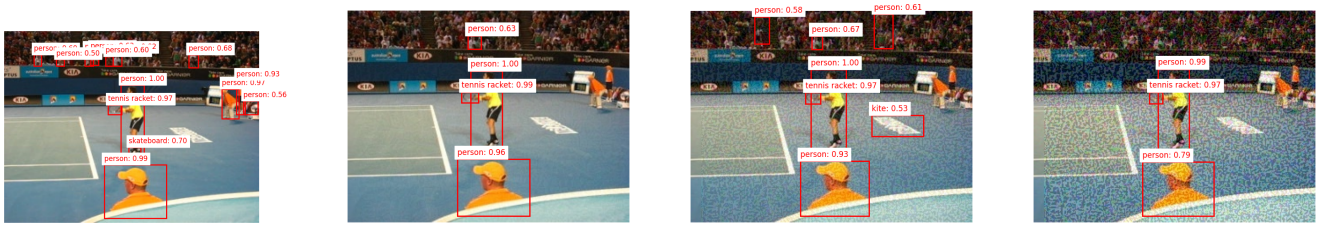


TOG fabrication

TOG mislabeling

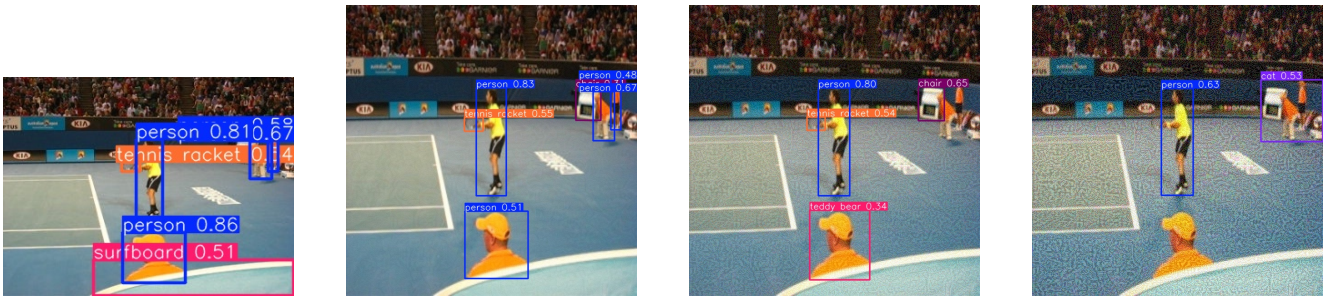
TOG vanishing

Figure 12. Examples of TOG Attack on Faster RCNN



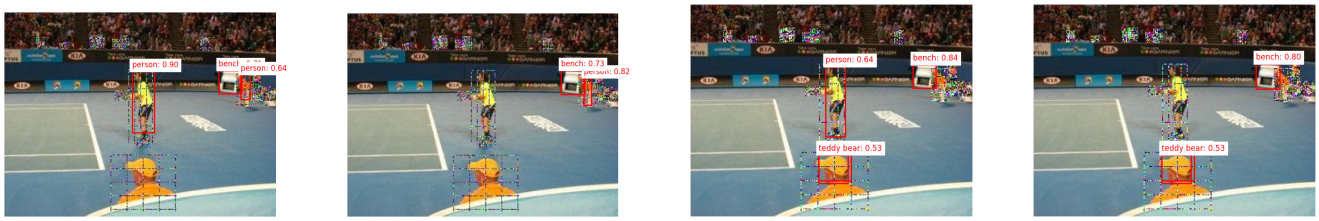
(a) Baseline (b) $\epsilon = 0.01$ (c) $\epsilon = 0.05$ (d) $\epsilon = 0.1$

Figure 13. Examples of FGSM attack on Faster RCNN



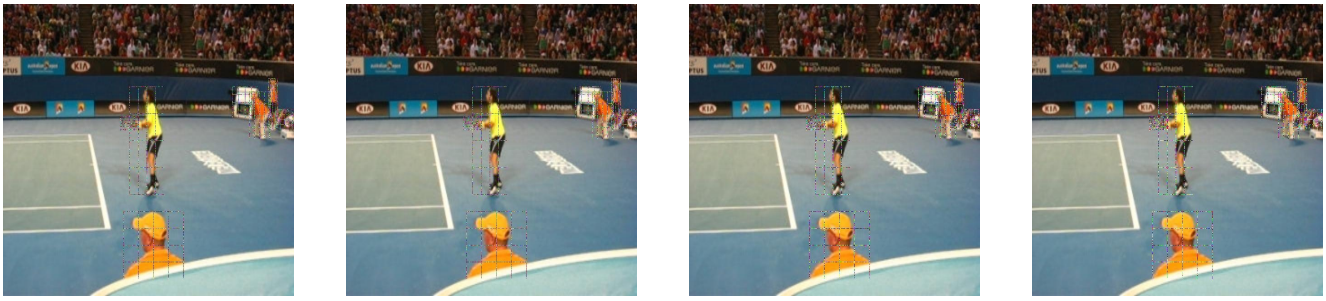
(a) Baseline (b) $\epsilon = 0.01$ (c) $\epsilon = 0.05$ (d) $\epsilon = 0.1$

Figure 14. Examples of FGSM attack on YOLOv5s



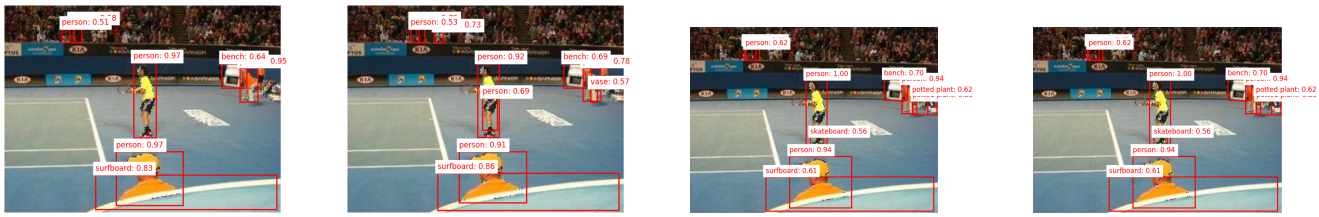
(a) $\epsilon = 0.3,\ T = 10$ (b) $\epsilon = 0.3,\ T = 20$ (c) $\epsilon = 1,\ T = 10$ (d) $\epsilon = 1,\ T = 20$

Figure 15. Examples of DPAttack on Faster RCNN



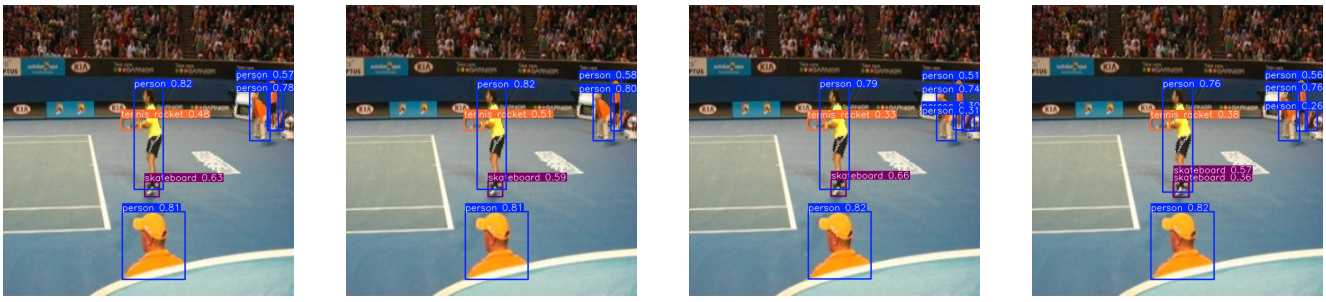
(a) $\epsilon = 0.3,\ T = 10$ (b) $\epsilon = 0.3,\ T = 20$ (c) $\epsilon = 1,\ T = 10$ (d) $\epsilon = 1,\ T = 20$

Figure 16. Examples of DPAttack on YOLOV5s



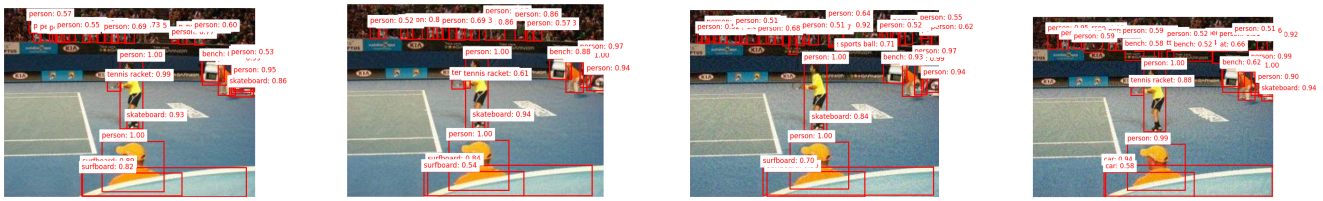
(a) $\epsilon = 0.3,\ T = 10$ (b) $\epsilon = 0.3,\ T = 20$ (c) $\epsilon = 1,\ T = 10$ (d) $\epsilon = 1,\ T = 20$

Figure 17. Examples of Edge-DPAttack on Faster RCNN



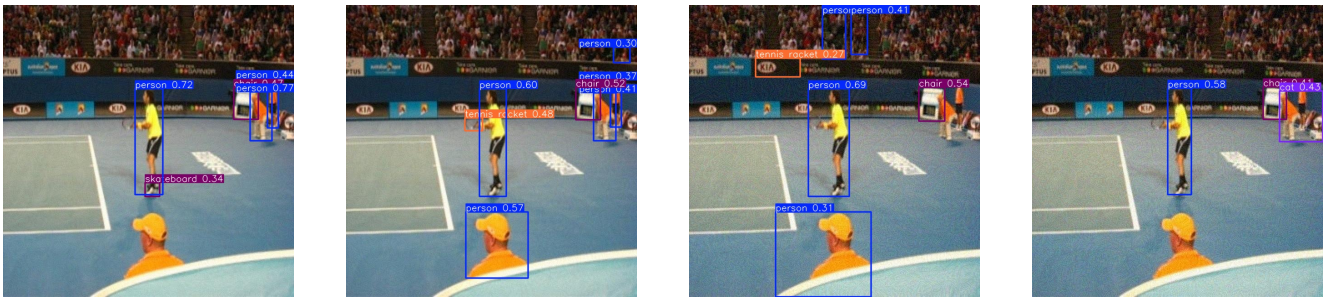
(a) $\epsilon = 0.3,\ T = 10$ (b) $\epsilon = 0.3,\ T = 20$ (c) $\epsilon = 1,\ T = 10$ (d) $\epsilon = 1,\ T = 20$

Figure 18. Examples of Edge-DPAttack on YOLOV5s



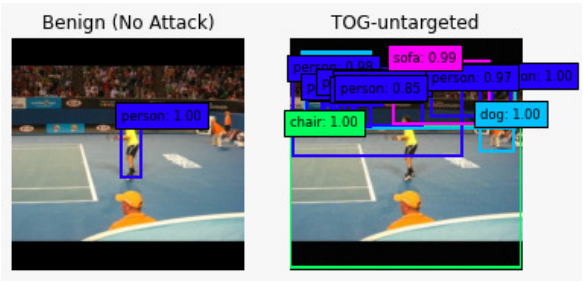
(a) $\epsilon = 0.03, \alpha = 0.007, T = 10$ (b) $\epsilon = 0.03, \alpha = 0.007, T = 10$ (c) $\epsilon = 0.06, \alpha = 0.015, T = 10$ (d) $\epsilon = 0.06, \alpha = 0.015, T = 20$

Figure 19. Examples of PGD Attack on Faster RCNN

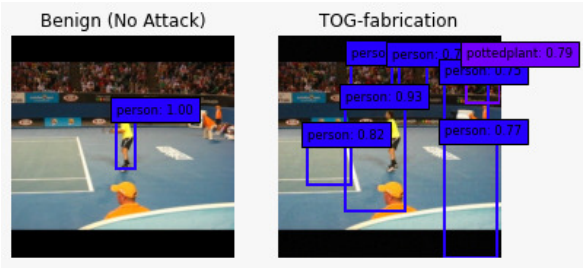


(a) $\epsilon = 0.03, \alpha = 0.007, T = 10$ (b) $\epsilon = 0.03, \alpha = 0.007, T = 20$ (c) $\epsilon = 0.06, \alpha = 0.015, T = 10$ (d) $\epsilon = 0.06, \alpha = 0.015, T = 20$

Figure 20. Examples of PGD-Attack on YOLOV5s

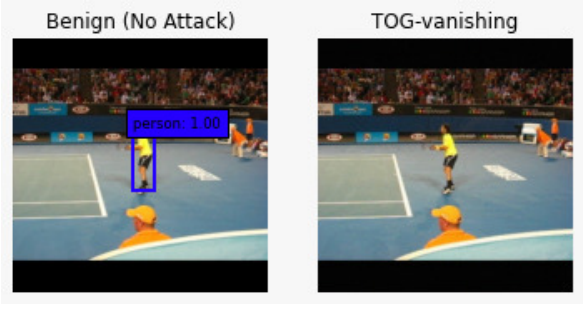


TOG untargeted



TOG fabrication

TOG mislabeling



TOG vanishing

Figure 22. Examples of TOG Attack on Faster RCNN