Lucas Guichard
50221623

CYBER PHYSICAL SYSTEM SECURITY

Prac Assignment #3

Deadline : 06.18 (Sun.) 23:59 pm
Report file type: Only PDF

Instructions

- (Basic 7pt)
  - Change the estimator to “FRCNN” to perform “Object Detection” and “Dpatch Attack” during the code by the 13th week of practice
- (Advanced 3pt)
  - Evaluate the impact of “DPatch” using “Average Precision” among YOLO's performance evaluation methods

You will find my code below with my results. All the code is documented for you to understand how I proceed this assignment.

Furthermore, I make different tests and find that I need to use a max_iter above or equal to 100 and a prediction threshold equal to 0.3. Otherwise, the two Average Precisions computed (one for the original image and one for the adversarial image) were two close.

This results show that FRCNN is not “that” vulnerable to the DPatch attack.

```
# -q for quiet (display only important logs)
!pip install -q adversarial-robustness-toolbox

# To use my Google Drive resources
from google.colab import drive
import os

# To use FRCNN and DPatch from ART library
#
https://adversarial-robustness-toolbox.readthedocs.io/en/latest/module
s/estimators/object_detection.html#object-detector-pytorch-faster-rcnn
from art.estimators.object_detection import PyTorchFasterRCNN
#
https://adversarial-robustness-toolbox.readthedocs.io/en/latest/module
s/attacks/evasion.html#dpatch
from art.attacks.evasion import DPatch

# To manipulate image
from PIL import Image

# For numpy computation
import numpy as np

# Create the ART PyTorchFasterRCNN estimator
frcnn = PyTorchFasterRCNN(
    # 3 channels (RGB) and size 640x640
    input_shape=(3, 640, 640),
    # Pixel values should be clipped between 0 and 255
    clip_values=(0, 255),
    # Color channels are the last dimension
    channels_first=False,
    # Use all the losses provided by Faster R-CNN model
    attack_losses=("loss_classifier", "loss_box_reg",
"loss_objectness", "loss_rpn_box_reg"),
    # Use the GPU for computation
    device_type="gpu"
)

# Create the RobustDPatch attack
attack = DPatch(
    # Our trained object detector, here Faster R-CNN model
    estimator=frcnn,
    # (height, width, nb_channels)
    patch_shape=(100, 100, 3),
    # Number of optimization steps
    max_iter=100
)

# GoogleDrive Mount
drive.mount('/content/drive')
```

```
image_path =
'/content/drive/MyDrive/{EPITECH}/tek4/Korea/CAU/SpringCourses/CyberPh
ysicalSystem/A3/image.jpg'
```

```
Drive already mounted at /content/drive; to attempt to forcibly
remount, call drive.mount("/content/drive", force_remount=True).
```

```
# Common Objects in Context (COCO) dataset for Faster R-CNN model
COCO_INSTANCE_CATEGORY_NAMES = [
    '__background__', 'person', 'bicycle', 'car', 'motorcycle',
'airplane', 'bus',
    'train', 'truck', 'boat', 'traffic light', 'fire hydrant', 'N/A',
'stop sign',
    'parking meter', 'bench', 'bird', 'cat', 'dog', 'horse', 'sheep',
'cow',
    'elephant', 'bear', 'zebra', 'giraffe', 'N/A', 'backpack',
'umbrella', 'N/A', 'N/A',
    'handbag', 'tie', 'suitcase', 'frisbee', 'skis', 'snowboard',
'sports ball',
    'kite', 'baseball bat', 'baseball glove', 'skateboard',
'surfboard', 'tennis racket',
    'bottle', 'N/A', 'wine glass', 'cup', 'fork', 'knife', 'spoon',
'bowl',
    'banana', 'apple', 'sandwich', 'orange', 'broccoli', 'carrot',
'hot dog', 'pizza',
    'donut', 'cake', 'chair', 'couch', 'potted plant', 'bed', 'N/A',
'dining table',
    'N/A', 'N/A', 'toilet', 'N/A', 'tv', 'laptop', 'mouse', 'remote',
'keyboard', 'cell phone',
    'microwave', 'oven', 'toaster', 'sink', 'refrigerator', 'N/A',
'book',
    'clock', 'vase', 'scissors', 'teddy bear', 'hair drier',
'toothbrush'
]
```

```
def extract_predictions(predictions):
    # Get the predicted class
    predictions_class = [COCO_INSTANCE_CATEGORY_NAMES[i] for i in
list(predictions[0]["labels"])]

    # Get the predicted bounding boxes
    predictions_boxes = [[(i[0], i[1]), (i[2], i[3])] for i in
list(predictions[0]["boxes"])]

    # Get the predicted score
    predictions_score = list(predictions[0]["scores"])

    # Get a list of index with score greater than threshold
    threshold = 0.3
```

```
    predictions_t = [predictions_score.index(x) for x in
predictions_score if x > threshold][-1]

    # Trim the predictions to include only the high-scoring objects
    predictions_boxes = predictions_boxes[: predictions_t + 1]
    predictions_class = predictions_class[: predictions_t + 1]

    return predictions_class, predictions_boxes, predictions_score

with Image.open(image_path).convert('RGB').resize((640, 640)) as
image:
    # Display original image
    image.show()
    # Image pre-processing
    image = np.expand_dims(image, axis=0).astype(np.float32)
    # Get original predictions
    original_predictions = extract_predictions(frcnn.predict(image))

    # Generate patch using DPatch attack
    patch = attack.generate(image)
    # Apply Patch
    adversarial_image = attack.apply_patch(image, patch)
    # Get adversarial predictions
    adversarial_predictions =
extract_predictions(frcnn.predict(adversarial_image))
    # Generate adversarial image
    adversarial_image =
Image.fromarray(np.squeeze(adversarial_image.astype(np.uint8)))
    # Display adversarial image
    adversarial_image.show()
```

{"model_id":"8687f810e13a444e9e7f17cb254b588c","version_major":2,"version_minor":0}

```
print("original_predictions: ", original_predictions)
print("adversarial_predictions: ", adversarial_predictions)
```

```
original_predictions:  (['elephant', 'elephant', 'person', 'person',
'person', 'person', 'elephant', 'person', 'person', 'person',
'person', 'elephant', 'person'], [[(113.70426, 293.4304), (405.972,
549.2706)], [(286.96158, 304.71335), (470.05215, 535.3393)],
[(567.1975, 412.1783), (589.3996, 456.32285)], [(532.03455,
427.14035), (543.67377, 451.26114)], [(544.37115, 406.37656),
(563.9235, 452.2059)], [(545.01776, 425.7192), (562.69037,
456.19653)], [(46.17589, 327.82056), (121.84545, 459.15088)],
[(546.8743, 398.70886), (563.62354, 432.09772)], [(533.761,
426.18155), (550.49255, 452.5009)], [(592.4627, 405.9861), (605.8766,
436.93604)], [(596.72046, 412.58005), (613.19653, 437.68512)],
[(88.444115, 332.33493), (119.93784, 452.9946)], [(541.6527,
427.56567), (555.1032, 452.54782)]], [0.99801826, 0.9975617,
0.95922613, 0.94342023, 0.8373061, 0.8017515, 0.7798114, 0.70904416,
```

```
0.43768758, 0.41843945, 0.36958683, 0.33718556, 0.3239281, 0.2952294,
0.25224367, 0.21383898, 0.19356671, 0.1291906, 0.11477267,
0.112597935, 0.11156301, 0.11129278, 0.11052208, 0.09048176,
0.08728262, 0.074121945, 0.07336877, 0.07098442, 0.06788332,
0.06038555, 0.05296946])
adversarial_predictions:  (['elephant', 'elephant', 'umbrella',
'person', 'person', 'person', 'person', 'elephant', 'person',
'person', 'person', 'person', 'elephant', 'person'], [[(113.70426,
293.4304), (405.972, 549.2706)], [(286.96158, 304.71335), (470.05215,
535.3393)], [(0.0, 0.5874451), (108.65326, 106.312744)], [(567.1975,
412.1783), (589.3996, 456.32285)], [(532.03455, 427.14035),
(543.67377, 451.26114)], [(544.37115, 406.37656), (563.9235,
452.2059)], [(545.01776, 425.7192), (562.69037, 456.19653)],
[(46.17589, 327.82056), (121.84545, 459.15088)], [(546.8743,
398.70886), (563.62354, 432.09772)], [(533.761, 426.18155),
(550.49255, 452.5009)], [(592.4627, 405.9861), (605.8766, 436.93604)],
[(596.72046, 412.58005), (613.19653, 437.68512)], [(88.444115,
332.33493), (119.93784, 452.9946)], [(541.6527, 427.56567), (555.1032,
452.54782)]], [0.99801826, 0.9975617, 0.98269224, 0.95922613,
0.94342023, 0.8373061, 0.8017515, 0.7798114, 0.70904416, 0.43768758,
0.41843945, 0.36958683, 0.33718556, 0.3239281, 0.2952294, 0.25224367,
0.21383898, 0.19356671, 0.1291906, 0.11477267, 0.112597935,
0.11156301, 0.11129278, 0.11052208, 0.09048176, 0.08728262,
0.074121945, 0.07336877, 0.07098442, 0.06788332, 0.06038555,
0.05296946])

def calculate_ap(predictions):
    # Sort the predictions by score in descending order
    sorted_predictions = sorted(predictions[2], reverse=True)

    # Initialize variables
    true_positives = 0
    false_positives = 0
    precision = []
    recall = []

    # Calculate precision and recall at each threshold
    for prediction in sorted_predictions:
        if prediction in predictions[2]:
            true_positives += 1
        else:
            false_positives += 1
        precision.append(true_positives / (true_positives +
false_positives))
        recall.append(true_positives / len(predictions[2]))

    # Calculate Average Precision
    ap = sum(precision[i] * (recall[i] - recall[i - 1]) for i in
range(1, len(precision)))
```

```
    return ap

# Calculate AP for original predictions
original_ap = calculate_ap(original_predictions)
print("AP for original predictions = ", original_ap)

# Calculate AP for adversarial predictions
adversarial_ap = calculate_ap(adversarial_predictions)
print("AP for adversarial predictions = ", adversarial_ap)

# Compare the AP values
if original_ap != adversarial_ap:
    print("The DPatch attack had an impact on the Average Precision.")
else:
    print("The DPatch attack has not significantly affected the
Average Precision.")
```

```
AP for original predictions =  0.967741935483871
AP for adversarial predictions =  0.96875
The DPatch attack had an impact on the Average Precision.
```