Adversarial Patch Attacks:

CSE XXXX

Presenter - Amelia McMahan

Overview:

Adversarial Patch Attacks are a way to fool object detection models into either not recognizing an object is there, or misclassifying it completely. This can be done both physically and digitally - anything from a simple white sheet of paper duct-taped to a stop sign, or onto more sophisticated digital attacks that specifically target object recognition algorithms in things like self-driving cars or security cameras. The examples we'll review here are relatively simple, but illustrate how these attacks can be implemented, as well as some common techniques behind them. We'll be looking at two main attacks: a basic overlay, and a robust (adaptive) patch.

0. Setting up the Environment:

Before you begin, please make sure that you download and unzip the *Adv_Patch* folder for this assignment, and upload it directly to your Google Drive. We'll be using this for our images and patches to overlay.
Ensure that the file path in cell #4 matches the path to your drive folder. You can check by clicking on the file icon on the far left pane, then clicking on the 3 dots on the right of that pane. From there, there should be an option to "copy file path".
Additionally, run the import cells in the Colab notebook. These contain important libraries such as *adversarial-robustness-toolbox (art)* and *Yolo*, that are an attack library and object recognition model, respectively. We'll be using *art* for our RobsutPatch attack and the original patch generation.*Yolo* is the detection model we'll be attacking.
If you have the time, you can look up documentation for art here. There's an almost overwhelming amount of potential, but it gives a good outlook into the power behind art.
Yolo documentation is found here. There are numerous versions and pretrained models; we are using yolov5.

1. Running the Initial Object Detector

In cells 5-8, you'll find a number of helper functions. These are to help in the later image classification portion. Look over these - comments are there to provide some context, but these functions are the core part of the attack actions. You'll need to understand which function works where for the later exercises.

2. Initial Image Predictions

This is just to give a baseline on what the Yolo model predicts on the image set.

Attack 1: Basic Patch Overlay

This merely slaps a preexisting patch on top of a set of images to obscure or confuse the presence of certain objects. It's a great way to understand some of the mechanics going on and the general concept behind such an attack. These patches are more similar to what you'd see in real-life, but can be done with any image. Later on, you'll apply an image of your choice as your own adversarial patch, in a sense.
As you can see, this merely causes loss of detection for the yolo model by getting in the way of the image. While it's a great illustration of the principles behind these patch attacks, this attack's true power is in Robust (Adaptive) Patches:

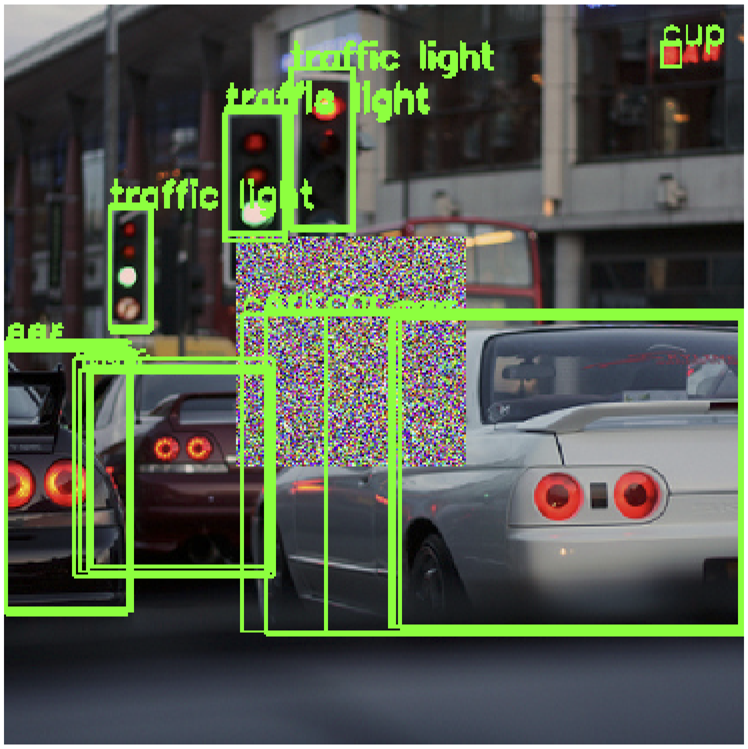
Attack 2: RobustPatch

This attack focuses on iterations: slowly altering the patch to further fool the model into misclassifying certain objects. RobustPatches can be incredibly powerful, with the downside of increased computational power.. The example here is relatively simple, with minimal alterations, and still takes around 15-20 minutes to run on a typical Colab notebook.

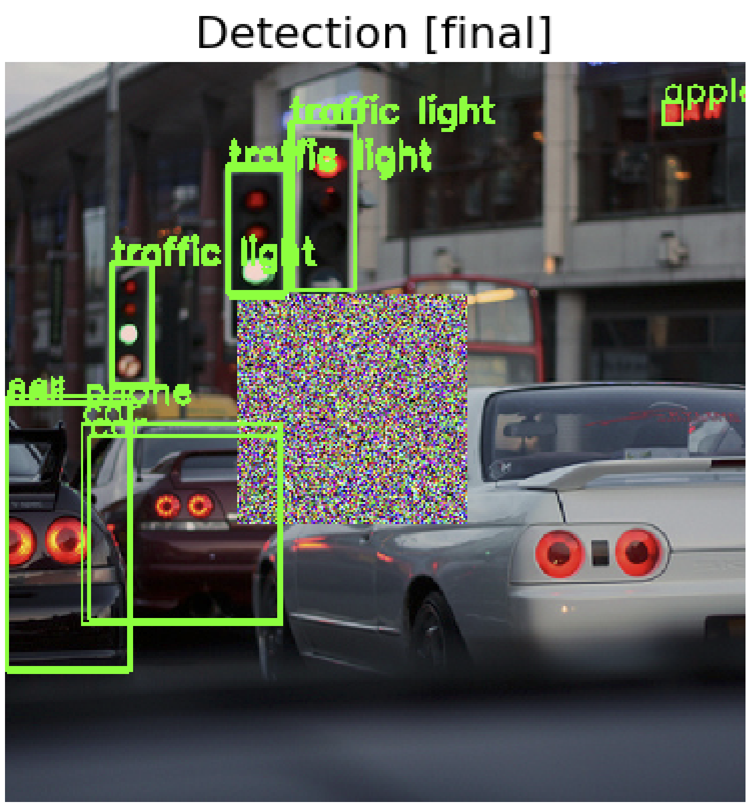
You can see some of the patch's capabilities here:

Example after the first iteration:

Detection [iter: 0]



Example after 20 iterations:



As you can see, it's begun to misclassify, or even ignore some objects completely. As the patch undergoes different iterations, you may get different classifications, as the random value it starts with may change its approach. Take note that on a surface level, the patch itself doesn't appear different, yet after 20 interactions, it's already significantly messed with the detection model.

Defense Against Patch Attacks:

The most basic defense is physical security. Like mentioned in the introduction, patch attacks can be as simple as a sheet obscuring a traffic sign.
On the software side, there's some defense tools available such as PAD that use adversarial examples during training to help the detector learn to spot such attacks. This isn't that effective against pretrained models, such as the yolo5 one in this case..
Other mitigations include reducing image noise, as most patches (like the ones we used) are in the form of noise clusters. Once detected, this noise can be blocked out and removed from the image.
The upside of patch attacks is that - due to requiring a visible alteration on top of the image - they're easy to identify once that abnormality is spotted.

TODO:

Patch settings: Play around with the patch settings for a bit. Some ideas include altering the scale value in the *addPatch* function, or changing the patch's location. Its location is currently set to (0,0) - the top left corner. *Is there a more efficient place to put it?*

Add your own: *Add an image of your own as the patch*! It can be anything - just make sure it's uploaded as a png into the Adv_Patch folder in your Google Drive.The cells to do this are provided for you - you simply need to correctly format and paste your chosen image.

- *Please submit a screenshot of your photo on top of the existing images.*

Additionally, answer the following questions for the RobustPatch:

1. Which misclassifications occurred?
2. How do you think these would affect an autonomous vehicle's judgment?
3. Was there anything you thought interesting about these misclassifications?
4. *OPTIONAL*: If you have the time, try messing with the RobustPatch settings. What happens if you change the targeted parameter to *False*?