Homework 5

Adversarial Examples for Facial Recognition and Beyond CNNs

IMPORTANT: For this homework, you will be graded on how well your code works. To get credit, your code must be able to run with no errors. If your code doesn't run, we will not be debugging it during grading. Please pay careful attention to make sure your code follows the format laid out in the starter file.

Submission deadline: **May 19th (Monday) at 11:59 pm CDT on Gradescope**.
Late submission deadline: **May 21th (Wednesday) at 11:59 pm CDT on Gradescope**.

In Homework 5, you will be applying adversarial machine learning to *facial recognition* models. These models have been widely deployed, despite security and privacy concerns on training data, inherent biases … etc. The goal of this assignment is to use existing knowledge developed over the quarter to generate adversarial images that prevent successful facial recognition.

Expected Run Times:

1. newt.cs.uchicago.edu: ~5 minutes
2. floo.cs.uchicago.edu: ~10 minutes
3. Mac M2 cpu: ~70 minutes

Setup

- Download the starter files for HW5 from Canvas:
    - This includes hw5_starter.py, test.py, utils.py
    - If you are running locally, please download model files from box and modify test.py to point to those files manually (if you are running on floo/newt, no action is necessary. All the files are already in /local/homework/data/hw5)
- Install packages
    - You are permitted to use any built-in library, along with any of the following packages: torch torchvision numpy einops pillow requests
    - **Please pip install open-clip-torch and lpips**
        - **Both package will be necessary for this homework**
- Check to make sure everything runs
    - Run the test.py file. As long as there are no errors, you are good to go
        - Parts 2 and 3 should correctly identify the given image as Adam Sandler

    - **The test.py file also contains helpful snippets for generating image features and getting outputs, so start by looking over that**.

Note, we will NOT be releasing the exact source images that we will use for grading. However, we encourage you to test your attacks on images of your choosing while you are building them.

Part 1: Face Recognition (10 points)

First, we need to better understand how facial recognition algorithms work. In part 1, you will be implementing a facial recognition system using two models (CosFace, and CLIP) as feature extractors. Unlike previous models we've dealt with (i.e., Resnet), the facial recognition models take images as input, and return a vector (image embedding). These embeddings are optimized such that images of the same people return vectors that have **high cosine/L2 similarity**. Thus, a very simple facial recognition system uses the feature extractor to generate an n x m matrix (n images, m is the embedding size) as a look-up table. Then when we need to "recognize" a new image, we can compute the cosine similarity of the new image's embedding to rows in our matrix, and return the identity of the image who's embedding has highest similarity to the input image.

Specifications:

Implement two facial recognition systems as described above, one with CosFace, and one with CLIP.

Feature extractor models:

- CosFace (stored as /local/homework/data/hw5/student-data/cosface.pth)
- CLIP (stored as /local/homework/data/hw5/student-data/clip.pth)

The test.py file has an example of how to get features from each model, so use that to get started.

Format:

Implement facial recognition in the function named "part_1_cosface" and "part_1_clip" in the starter file. Leave the function signatures exactly as it is. How exactly you implement is up to you, so long as the functions works properly when you run test.py.

- The functions should generate the n x m matrix for the training images only, and then evaluate on the testing images.
- NOTE: the testing images should not be used to generate the n x m matrix

Grading:

We will be evaluating your implementation of the facial recognition systems on a held out dataset with 50 identities.

Point Breakdown:

- Accuracy is between 90% and 100% → 10/10
- Accuracy is between 80% and 90% → 9/10
- … etc.

Part 2: Attacking Facial Recognition (70 points)

Your job is to design an attack that prevents accurate facial recognition of new images via adversarial perturbation. If done correctly, you will also show that perturbed images also fool generative models (VLMs) for captioning.

Specifications:

Implement a **SINGLE** white-box, **TARGETED** attack that prevents successful facial recognition for **BOTH** CosFace and CLIP feature extractors. This attack should **ALSO** transfer to a black box VLMs used for image captioning.

Feature extractor models (same as in part 1):

- CosFace (stored as /local/homework/data/hw5/student-data/cosface.pth)
- CLIP (stored as /local/homework/data/hw5/student-data/clip.pth)

You have an L_infinity perturbation budget of epsilon=16/255.

You can choose any learning rate and number of iterations you like, but we ask that you keep it reasonable. Any more than 2000 iterations is excessive.

Note: We are hosting our own facial recognition algorithm and VLM behind API access (black box), which we show how to query in test.py to test your solution. The facial recognition algorithm operates the same as your algorithm in part 1, but we made ours black box so you can't just copy the solution. If you are doing homework locally (not on floo or newt), you must be on VPN.

Note: you may use whatever source/target images you want to test your code. However, the black box facial recognition algorithm is only trained to identify these 50 identities.

Format:

Implement the attack in the function named “part_2” in the starter file. Leave the function signature exactly as it is. How exactly you implement this is up to you, so long as the function works properly when you run test.py.

Grading:

We will be evaluating your attack algorithm on 10 unknown images (of people), using both CosFace and CLIP-based facial recognition systems. Targets will be selected randomly.

Point Breakdown per Image (7 points):

- 4 points facial recognition fails (successful attack, 2pt CLIP, 2pt CosFace)
    - 1pt no longer same identity, 1pt achieving target identity
- 1 point if the perturbation stays within the perturbation budget
    - You can only earn these points if the attack is successful.
- 3 points fooling VLM (transfers to unseen VLM)
    - Output of VLM incorrectly describes the original image (doesn’t say Adam Sandler, man, … etc. when photo of Adam Sandler is perturbed)

Across all 10 images, this totals to 70 points.

Part 3: Better Visual Imperceptibility (20 points)

Unfortunately, the adversarial images you have generated (while successful), have too much visible noise. Fortunately, you have recently heard about a new method similar to PGD that allows you to generate adversarial images with less visible noise: using LPIPS as a function of visible change. You will be implementing an attack constrained by LPIPS as followed (hint, use penalty method):

$$\min L(x+\delta) \text{ subject to } LPIPS(x, x+\delta) \leq \epsilon$$

Specifications:

Implement a **SINGLE** white-box, **TARGETED** attack that is constrained by LPIPS that prevents successful facial recognition for **BOTH** CosFace and CLIP feature extractors. This attack should **ALSO** transfer to a black box VLMs used for image captioning.

Feature extractor models (same as in parts 1/2):

- CosFace (stored as /local/homework/data/hw5/student-data/cosface.pth)
- CLIP (stored as /local/homework/data/hw5/student-data/clip.pth)

LPIPS model

- LPIPS (stred as /local/homework/data/hw5/student-data/lpips.pth)

You have an LPIPS perturbation budget of 0.07

You can choose any learning rate and number of iterations you like, but we ask that you keep it reasonable. Any more than 2000 iterations is excessive.

Note: We are hosting our own facial recognition algorithm and VLM behind API access (black box), which you may query in test.py to test your solution.

Note, you can save the image and verify that LPIPS is in fact reducing visible perturbations in images.

Format:

Implement the attack in the function named “part_3” in the starter file. Leave the function signature exactly as it is. How exactly you implement this is up to you, so long as the function works properly when you run test.py.

Grading:

We will be evaluating your attack algorithm on 5 unknown images, using both CosFace and CLIP-based facial recognition systems. You may test these images against the VLM, though it is not required to completely fool the VLM for part 3 only.

Point Breakdown per Image (4 points):

- 2 points facial recognition fails (successful attack)
    - 1 pt for CosFace, 1 pt for CLIP (see part 2 for failure criteria)
- 2 points if the adversarial image stays within the LPIPS perturbation budget.
    - You can only earn these points if the attack is successful.

Across all 5 images, this totals to 20 points.

Submission

For this assignment, you will submit only hw5_starter.py to Gradescope. Do NOT change the name of this file.

You should **not** be loading any model files from the starter file, and you should **not** be loading any dataset in the starter file. We will do that separately and pass the model object and source images to your functions. Assume that your code will only have access to utils.py and the packages outlined in Setup.

There is no autograder for this assignment. We will be grading these assignments separately after the deadline. We will grade only for attack success, not code quality.