Sophia Sivula

CS 587

Dr. Vakanski

May 2nd, 2024

Adversarial Machine Learning Final Project

Project Overview: For my semester project, I will be comparing adversarial attacks on a biometric
system, specifically a fingerprint system. The dataset I will be using is the FVC2002 database, which is a collection of fingerprints spread throughout four different databases. Using database 1, these fingerprints were collected through the optical sensor "TouchView II" by Identix. I will be testing four different attacks on this dataset in an
attempt to misclassify the fingerprints. The four attacks will be a PGD attack, an FGSM attack, a HopSkipJump attack, and a universal perturbation attack. My plan is to use a VGG CNN for classification. My report will detail the success and/or failures of each attack, and will determine which attacks were most successful in misclassifying the fingerprints.

Project Plan:

- Step 1: Create data loader with FVC2002 database, using database 1.
- Step 2: Train the model using VGG16, and create a graph depicting accuracy and loss.
- Step 3-6: Generate the four attacks on the data and track the results and the accuracy of each attack.
- Step 7: Compare and discuss the attacks. For any unsuccessful attacks, determine why they may have been unsuccessful.

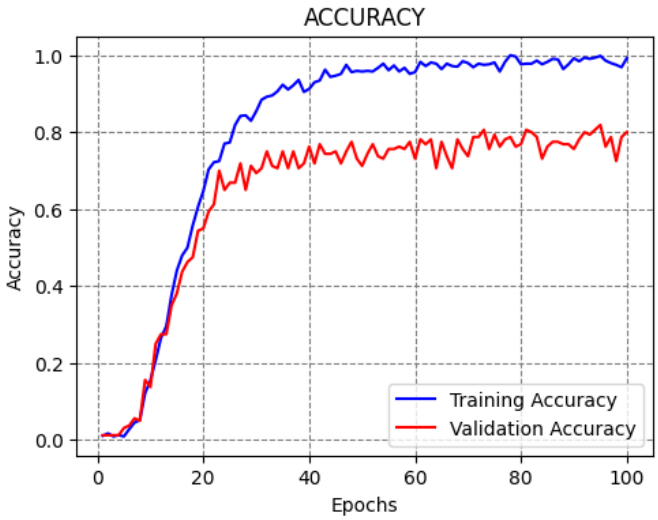
Step 1: Create dataloader.

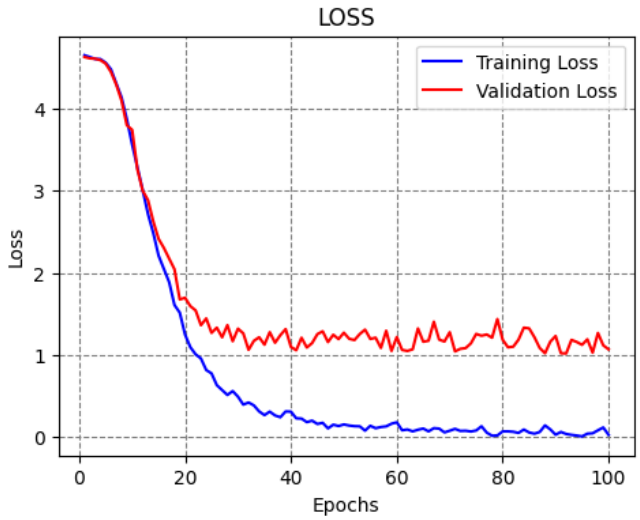
To create the dataloader, I followed the typical structure utilized in our previous class assignments. Since I am using a dataset for fingerprints that doesn't explicitly include labels but has detailed names for the files, I used a list directory to apply those names as the label. Within the dataset, the images were labeled with 1_1 to 1_8, up to 100_1 to 100_8. These segments of eight images each correspond to one finger, therefore it was

extremely important that they be grouped into classes to represent a finger. To do this, I used a for loop to read through the images and labels, and increase a label counter for each 8 images to keep track of each creation of a class. I then split into my train and test sets, and displayed the data type, the image train and test shapes, as well as the maximum, minimum, and average pixel value. I plotted 9 random images that identified the person and label, and then verified that this was correct in the dataset by looking through the pictures. This took a while to get right, as I had not realized that I needed to group the 8 images into one class, and therefore the label never matched the image displayed.

Step 2: Train the model.

To begin with, I trained the model with early stopping patience of 10, an Adam optimizer learning rate of 1e-4, and I fit the model with 100 epochs at a batch size of 32. This took 33 minutes to run and received 60.6% test accuracy and a test loss of 2.06. I reached out to Dr. Vakanski for some guidance on increasing my accuracy and retrained my model with an early stopping patience of 30. I changed the Adam optimizer learning rate to 1e-5, and I fit the model with 100 epochs at a batch size of 8. This took 58 minutes to run, and was much more successful. The test accuracy hit 80% and a test loss of 1.07.

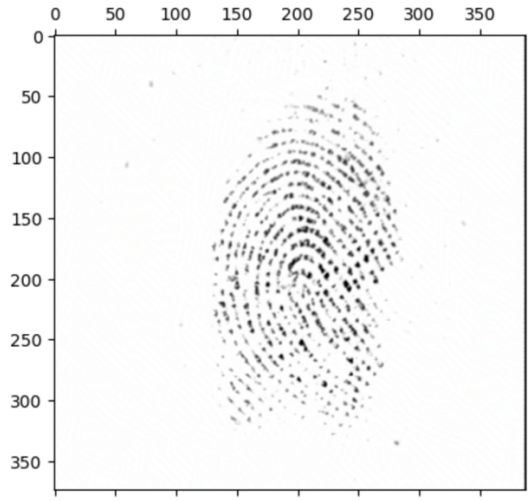




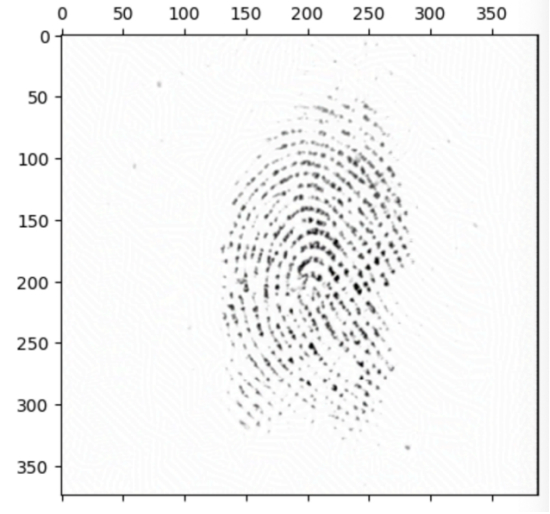
Step 3: Perform PGD attack.

A Projected Gradient Descent (PGD) attack works “by calculating the mapping between the predicted output by the model and the inputs. [It] uses backward propagated derivates of the loss function with respect to the input pixels. [1]” A PGD attack is an extension of the FGSM attack, and is white-box. The code for my PGD attack came from our previous assignments. I utilized epsilons 1./255, 3./255, 5./255, 8./255, 20./255, 50./255, and 80./255.

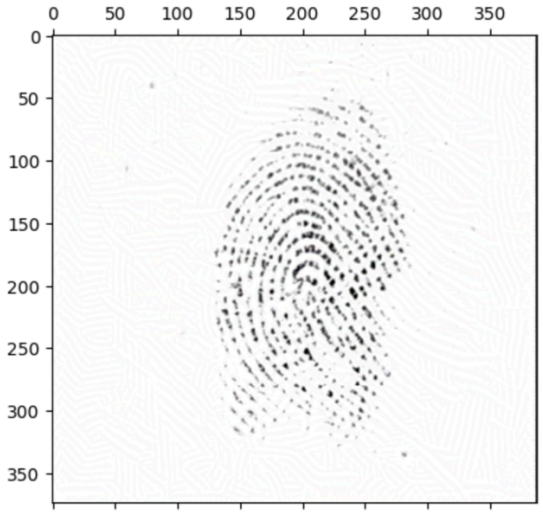
Accuracy on adversarial test data: 76.25%
Average perturbation: 0.00



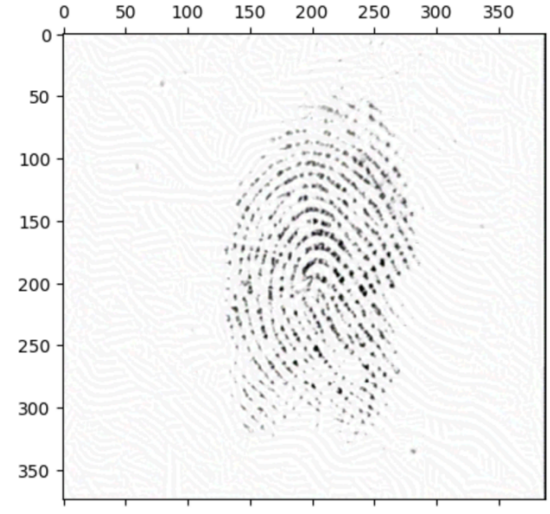
Accuracy on adversarial test data: 61.25%
Average perturbation: 0.01

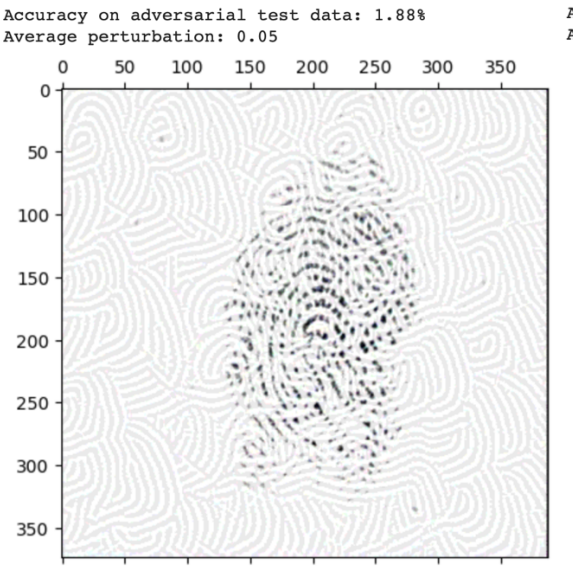


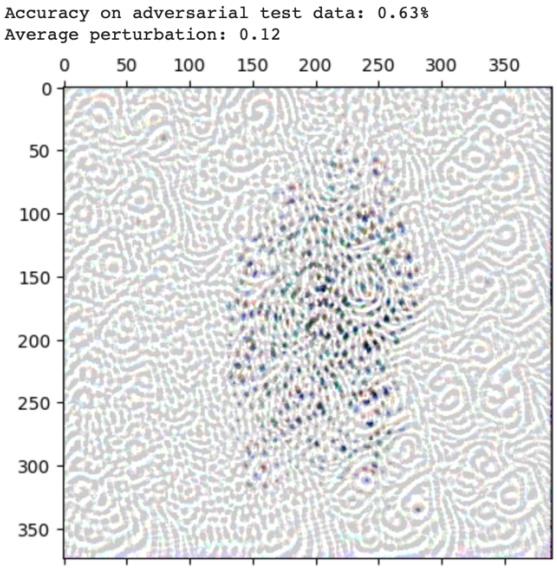
Accuracy on adversarial test data: 43.13%
Average perturbation: 0.01

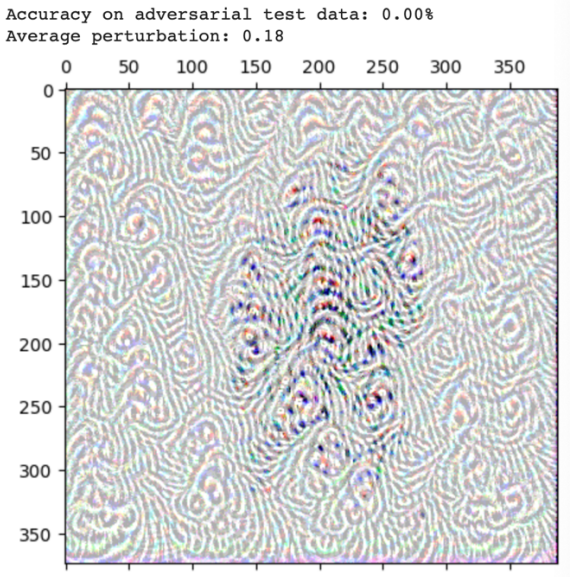


Accuracy on adversarial test data: 20.00%
Average perturbation: 0.02





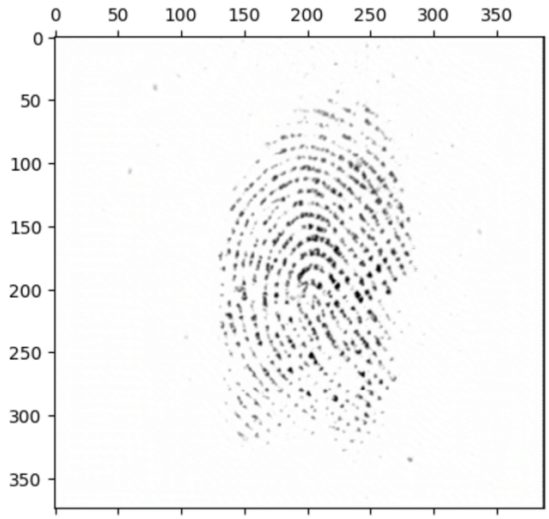




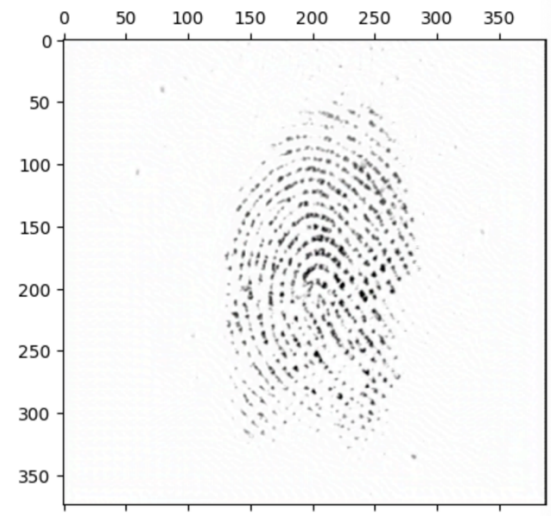
Step 4: Perform FGSM attack.

A Fast Gradient Method (FGSM) attack is similar to a PGD attack, in that it works "by calculating the mapping between the predicted output by the model and the inputs. [1]" However, it is a significantly faster attack and therefore used more often. I utilized the FGSM attack code provided to us in previous assignments. I used the same epsilons as the PGD attack (./255, 3./255, 5./255, 8./255, 20./255, 50./255, and 80./255).

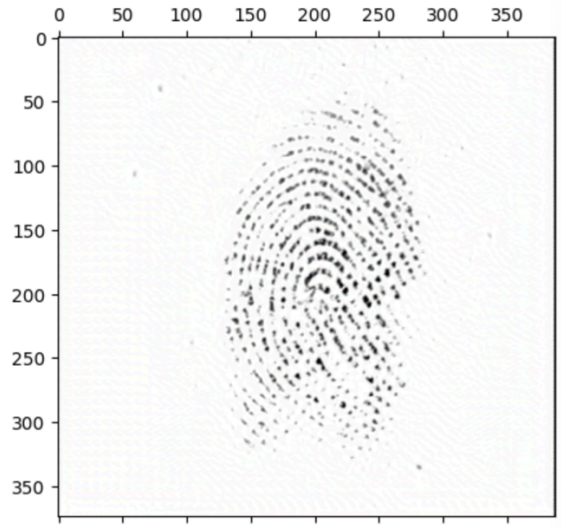
Accuracy on adversarial test data: 75.63%
Average perturbation: 0.00



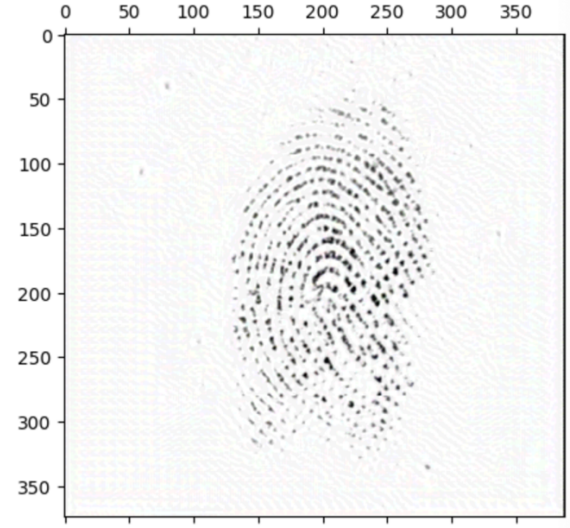
Accuracy on adversarial test data: 63.13%
Average perturbation: 0.01

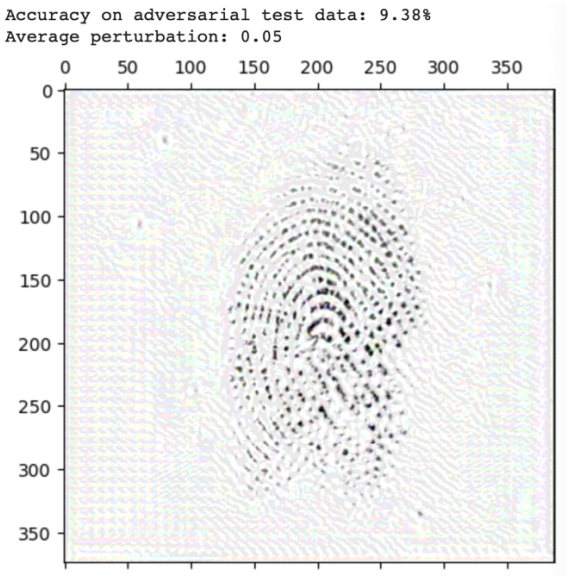


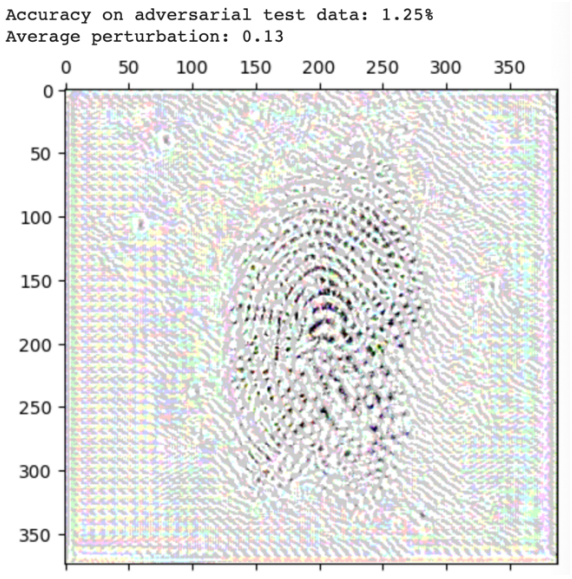
Accuracy on adversarial test data: 50.00%
Average perturbation: 0.01

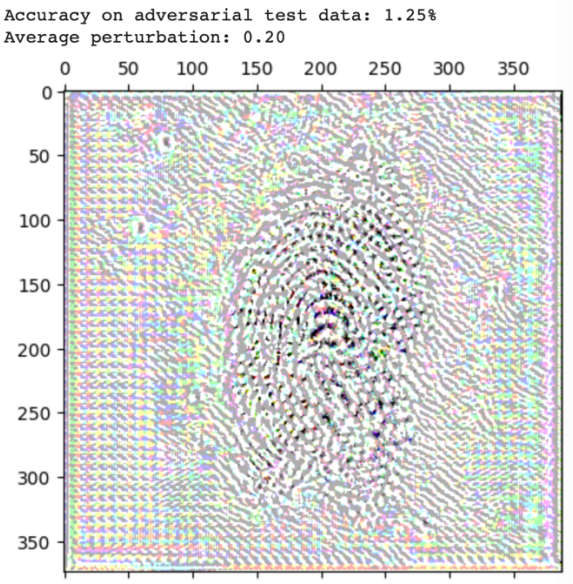


Accuracy on adversarial test data: 33.13%
Average perturbation: 0.02



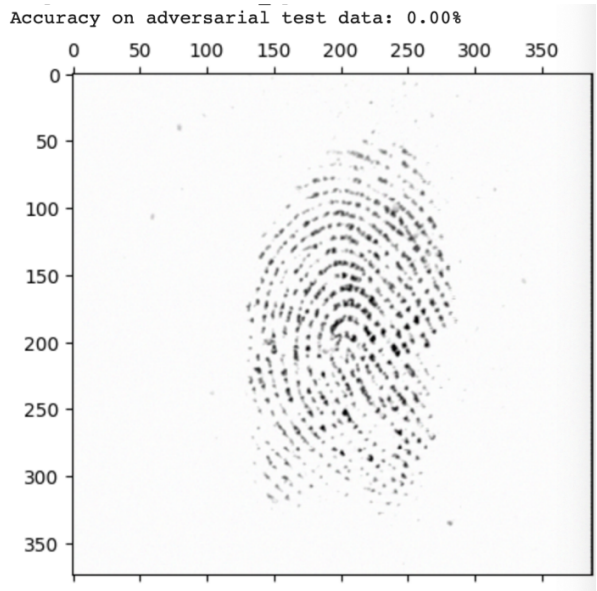






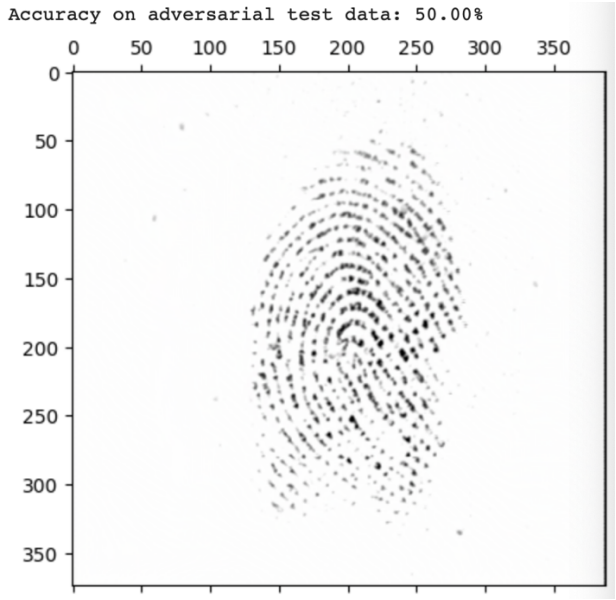
Step 5: Perform HopSkipJump attack.

A HopSkipJump attack is a black-box attack that is more advanced version of a boundary attack and only requires class predictions. [2] Due to RAM and GPU constraints, I tested the HSJA on two adversarial samples, per Dr. Vakanski's suggestion. I used the same code format for testing from assignment 4, with the use of ChatGPT to resolve any errors that arose that I couldn't solve myself. This test took 38 minutes to perform, and yielded a 0% accuracy on adversarial test data.



Step 6: Perform Universal Perturbation attack.

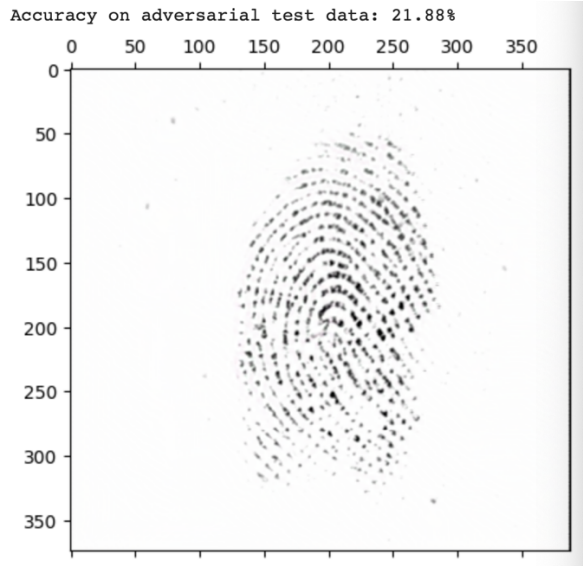
A Universal Perturbation attack is a white-box attack. I created the attack with the same code format as utilized in the HSJ attack and tested it on two adversarial samples. This took about 22 minutes to perform and achieved an accuracy on adversarial test data of 50%.



Extra: Perform NewtonFool attack.

Lastly, I performed a white-box NewtonFool attack. I was having issues running attacks so I tested a variety of attacks outside of HSJ and UP, and the results for the

NewtonFool were decent so I kept it in the project. I used the same code format as used for the last two attacks and used all 160 test images for the attack. It achieved 21.88% accuracy on the adversarial test data.



Step 7: Analyze and compare results.

The PGD attack was extremely successful. After the first round the accuracy on the adversarial test data was 76.25% with a 0.0 average perturbation, followed by an average perturbation of 0.01 with an accuracy of 61.25% and 43.13%, an average perturbation of 0.02 and with an accuracy of 20%, an average perturbation of 0.05 with an accuracy of 1.88%, an average perturbation of 0.12 with an accuracy of 0.63%, and finally an average perturbation of 0.18 with an accuracy of 0%. Even though the attack was able to reach 0% accuracy, a perturbation of 0.18 is extremely noticeable and obvious to the human eye that the image has been tampered with. The average perturbation of .01 (with accuracy of 43.13%) was the greatest drop in accuracy while still not being immediately noticeable to anyone looking at the image. An accuracy of 43% is still half the accuracy on the clean, trained model.

The FGSM attack began with an average perturbation of 0 and accuracy of 75.63%, then an average perturbation of 0.01 with an accuracy of 63.13% and 50%, an average perturbation of 0.02 with an accuracy of 63.13%, an average perturbation of 0.05 with an accuracy of 9.38%, an average perturbation of 0.13 with an accuracy of 1.25%, and finally an average perturbation of 0.20 with an accuracy of 1.25%. The image with an average perturbation of 0.02 and an accuracy of 63.13% was the largest level of perturbation that was not obviously noticeable to the human eye. This is a higher perturbation than the PGD attack, as that became noticeable at 0.01. This attack was also able to achieve a lower

accuracy before becoming obvious that it was being tampered with. However, the PGD attack was able to reach an accuracy of 0% at a perturbation of 0.18 and even at a perturbation of 0.20 the FGSM never hit 0% accuracy, instead reaching 1.25%. All of this being said, I still think the FGSM was a better attack for the fingerprint recognition dataset because it was significantly quicker than the PGD attack and was able to achieve a lower accuracy with less obvious perturbation than the PGD attack had.

The HopSkipJump attack was extremely successful at misclassifying the data, and achieved a 0% accuracy when tested on two adversarial samples. The image looks very minimally perturbed, unlike the PGD and FGSM 0% and 1.125% accuracy images which looked very heavily altered. Result wise, this test is a better option than the first two but took nearly as long as the PGD attack did to perform.

The Universal Perturbation attack was quicker than the HSJ attack, and produced a fingerprint image that looks slightly less tampered with than the HSJA. However, it was only able to achieve an accuracy on the adversarial test data of 50%. This is still better than the initial accuracy on the clean model, but not nearly as successful as the other attacks performed. Both the PGD attack and FGSM attack were able to breach an accuracy of 50% within the average perturbation of 0.01, so this test does not seem like a good fit for this fingerprint recognition dataset. I believe this is because fingerprints are individualistic and a successful change to one fingerprint may not be effective against another.

Finally, the NewtonFool attack was fairly quick to perform, and the generated adversarial image is not obviously modified. However, this attack only achieved an accuracy of 21.88%. This is better than that of the Universal Perturbation attack but much worse than the HopSkipJump attack. Compared to the FGSM, its image looks most similar to the average perturbation of 0.01 with an accuracy of 63.13% and for the PGD it looks most similar to the image with an average perturbation of 0.01 with an accuracy of 61.25%. Compared to these results, the NewtonFool is more successful than the PGD and FGSM attack. Overall, between all of the attacks, I found the HopSkipJump to be the most successful, followed by the NewtonFool, PGD, FGSM, and finally the Universal Perturbation attack as the least successful.

References:

[1] Dr. Vakanski's CS 587 class notes: "Lecture 4 – Evasion Attacks against White-box Models"

[2] https://github.com/Trusted-AI/adversarial-robustness-toolbox/blob/main/art/attacks/evasion/hop_skip_jump.py