Evasion Attacks Report

* Initial Set Up

- We imported the necessary libraries in the 'Imports' section in our jupyter notebook like Pytorch, torchvision and some utility libraries like pandas, numpy and matplotlib.

- We loaded the CIFAR-10 data set and split it up into test sets, and our data loader handles the batches of images.

- Our chosen model was VGG-16. It had several convolutional layers, batch normalization and ReLU activation function followed by linear layers for the classification task.

- We loaded the pretrained weights for our basic model and adversarially trained (robust) model. We also did the testing function to see the clean performance (accuracy) on the test set by comparing the predicted labels against the true labels. We printed out their accuracies respectively for each. The basic model has an accuracy of **90 %** and the robust model has an accuracy of **75.25 %**. This is fine as the adversarially trained model typically reduces clean accuracy but improves robustness against some attacks. See *Figure 1* below.

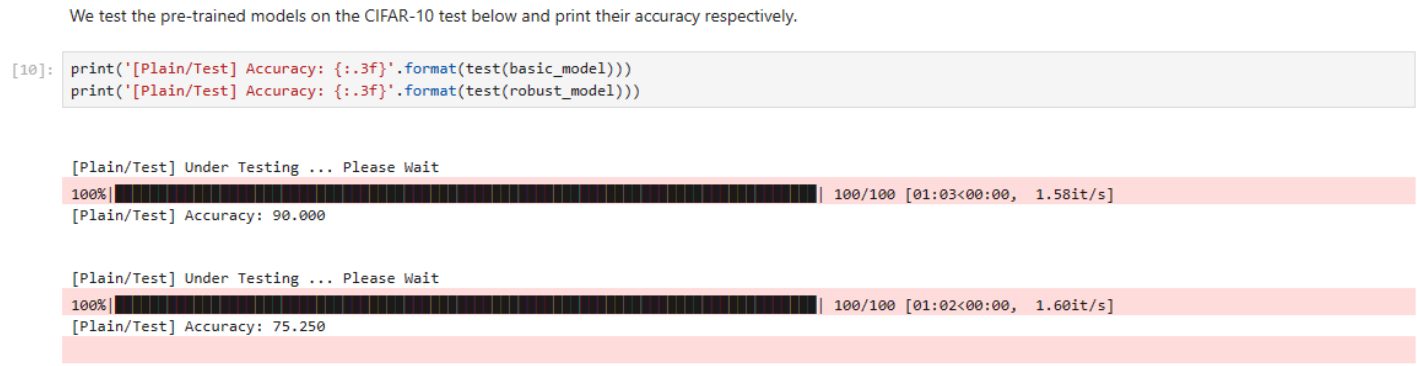

Figure 1. Models' accuracy results

1. FGSM Attack

a. Execute the untargeted and targeted version of the FGSM attack. Share your conclusions.

- In the **untargeted attack**, the goal is to cause the model to misclassify any class. We created an FGSM attack instance using the 'torchattacks' library and we were successful in our approach as the accuracy dropped down to **11.49 %**, indicating that the basic model was significantly vulnerable to our attack.

- In the **targeted attack**, the goal is to cause the model to misclassify inputs as a specific target class, which in our case was class 3, the cat image. In our efforts, the same way as the untargeted attack, we created an FGSM attack instance using the 'torchattacks' library and we were able to cause the model to make a mistake. When we performed the targeted attack and measured the accuracy, the result we obtained was **79.55 %**. This may seem counterintuitive at first but this accuracy percentage actually represents our success rate in deceiving the model to classifying different inputs as 'cat', representing how effectively the model was fooled and not the model's general accuracy. To put more explanation into this; in a standard setting, accuracy would be the proportion of correct predictions out of all predictions made. However, in the context of our targeted adversarial attack, the goal is to manipulate the model into making a specific incorrect prediction. Here, the "accuracy" reflects how successful the attack is at fooling the model into classifying various inputs as the designated target class "cat".

  See *Figure 2* below.

```
[19]: # creating the FGSM attack instance for untargeted and targeted attack on the basic model
      FGSM_attack_untargeted_basic = torchattacks.FGSM(basic_model, eps=8/255)
      FGSM_attack_targeted_basic = torchattacks.FGSM(basic_model, eps=8/255)

      # Testing with FGSM untargeted and targeted attacks on the basic model
      adv_test_untargeted(FGSM_attack_untargeted_basic, basic_model)
      adv_test_targeted(FGSM_attack_targeted_basic, basic_model)
```

```
[Plain/Test] Under Testing ... Please Wait
100%|██████████| 100/100 [02:38<00:00,  1.59s/it]
[Plain/Test] Accuracy: 11.490

[Plain/Test] Under Testing ... Please Wait
100%|██████████| 100/100 [02:42<00:00,  1.63s/it]
[Plain/Test] Accuracy: 79.550
```

Figure 2. Results of the untargeted and targeted attack on the basic model

b. Execute the untargeted and targeted version of the FGSM attack on the adversarially trained (robust) model. Share your conclusions.

- In the **untargeted attack** against the adversarially trained (robust) model, our goal is the same; to induce misclassification of any class. Similarly, we utilized the FGSM attack instance and observed that the model's accuracy was **46.65%**, a significant improvement from the basic model's 11.49% under the same attack conditions. This shows us the effectiveness of adversarial training in enhancing the model's resilience against such attacks and the model's ability to maintain correct classifications.

- In the **targeted attack**, where the aim was to specifically misclassify inputs as the 'cat' class, the adversarially trained model displayed an accuracy of **72.44%**. While this number is lower than the basic model's 79.55% in the targeted attack, it was important for us that we interpreted this correctly. Again, the accuracy here reflects the attack's effectiveness, not the model's general performance. A lower percentage indicates that the adversarially trained model was more resistant to being fooled into misclassifying non-cat images as 'cat', showing the benefits of adversarial training in defending against targeted attacks. See *Figure 3* below.

```
[9]: # FGSM attack instance for untargeted and targeted attack on the adversarially robust model
FGSM_attack_untargeted_robust = torchattacks.FGSM(robust_model, eps=8/255)
FGSM_attack_targeted_robust = torchattacks.FGSM(robust_model, eps=8/255)

# Testing FGSM untargeted and targeted attacks on the adversarially robust model
adv_test_untargeted(FGSM_attack_untargeted_robust, robust_model)
adv_test_targeted(FGSM_attack_targeted_robust, robust_model)
```

```
[Plain/Test] Under Testing ... Please Wait
100%|██████████| 100/100 [02:57<00:00,  1.78s/it]
[Plain/Test] Accuracy: 46.650

[Plain/Test] Under Testing ... Please Wait
100%|██████████| 100/100 [02:59<00:00,  1.80s/it]
[Plain/Test] Accuracy: 72.440
```

Figure 3. Results of the untargeted and targeted attack on the robust model

For the results we obtained from part "a" and part "b", we graphed them together for better visualization. *Figure 4* below represents them and the blue bars (untargeted) show the model's accuracy under attack. The green bars (targeted) show the Attack Success Rate (ASR), with higher values indicating more successful attack.

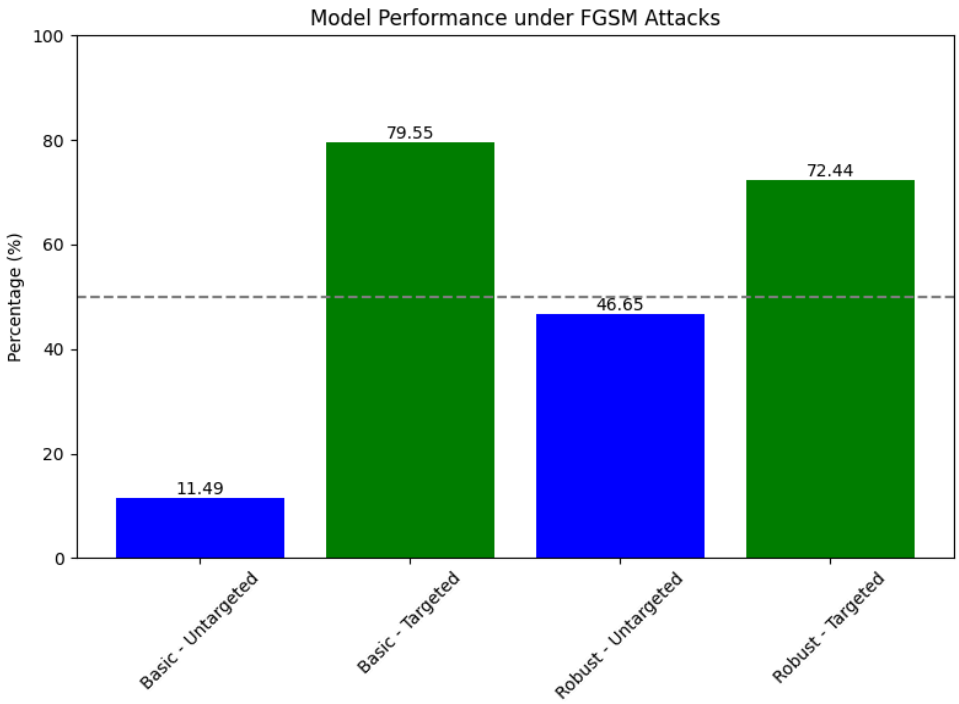

Figure 4. Results on the basic and robust model

To be honest, we were surprised by the small difference in success rate of our fooling efforts between the basic and robust models against the targeted attack and we thought it may be due to the architecture we chose or the single epsilon value we placed. We then tried with different epsilon values to see what the outcome we would receive.

Upon looking at the results when we have multiple epsilon values, they seemed counterintuitive to us that the success rate is dropping though we expected it to go higher as the perturbations get higher. One possibility we could think of was that the results weren't directly because of the usage of a single epsilon value, meaning the various perturbation amounts maybe made the images too distorted and made it harder to miss-classify the images as our target class, or maybe the model classified them into other incorrect categories which led to this reduction in the targeted attack's success rate. We are not exactly sure. It could also be due to the VGG-16 model we used that was not handling the images properly.

2. PGD and Auto-PDG Attack

Since the assignment description does not specify whether we should use the adversarially trained model or the basic model in this part, we chose the basic model to evaluate our attacks. We utilized the same two functions we created earlier in the FGSM part of the assignment. The functions "adv_test_untargeted" and "adv_test_targeted" do not do anything specific other than taking any attack instance and a model as input, then performing the attack and evaluating the model's performance on the adversarial examples.

a. Execute the untargeted version of the PGD and Auto-PGD attacks. Evaluate the results and compare both attacks in terms of success.

- In the **untargeted PGD attack**, the goal remains the same as in the previous part. We used the "torchattacks" library in the similar way, and the basic model's accuracy dropped to **0.69%** after the untargeted PGD attack. This drastic decrease shows us that the PGD attack was very effective in causing the model to misclassify the majority of the inputs.

- In the **untargeted Auto-PGD attack**, the goal is to provide enhancements to the PGD process while adjusting the attack parameters across iterations based on the attack's success. So, it aims to make PGD more effective. The results showed us that indeed the untargeted Auto-PGD attack was even more effective, with the model's accuracy plummeting to **0.04%**. This near-zero accuracy shows us that this was even more powerful in inducing widespread misclassification across various classes.

See *Figure 5* below for results we obtained.

```
[10]: PGD_attack_base = torchattacks.PGD(basic_model, eps=8/255, alpha=1/255, steps=10, random_start=True)
print("Basic Model - Untargeted PGD Attack")
adv_test_untargeted(PGD_attack_base, basic_model)

APGD_attack_base = torchattacks.APGD(basic_model, eps=8/255, steps=10)
print("Basic Model - Untargeted Auto-PGD Attack")
adv_test_untargeted(APGD_attack_base, basic_model)

Basic Model - Untargeted PGD Attack

[Plain/Test] Under Testing ... Please Wait
100%|██████████| 100/100 [18:26<00:00, 11.07s/it]
[Plain/Test] Accuracy: 0.690
Basic Model - Untargeted Auto-PGD Attack

[Plain/Test] Under Testing ... Please Wait
100%|██████████| 100/100 [20:16<00:00, 12.17s/it]
[Plain/Test] Accuracy: 0.040
```

Figure 4. Results of Untargeted PGD and Auto-PGD attack

b. Execute the targeted version of the PGD and Auto-PGD attacks. Evaluate the results and compare both attacks in terms of success. Use class cat (class index 3) as your target.

- In the **targeted version of the PGD attack**, our goal is again the same, to miss-classify the inputs as the target class "cat". Upon doing so, we were able to manipulate our model to classify **81.51%** of the inputs as the target class. This high percentage indicates our attack's success in achieving the targeted misclassification goal.

- **The targeted Auto-PGD attack** was also highly successful, even more than the PGD attack, causing the model to misclassify **83.04%** of inputs as our target class.

Figure 5 displays our results below.

```
[11]: print("Basic Model - Targeted PGD Attack")
      adv_test_targeted(PGD_attack_base, basic_model)

      # Auto-PGD - Targeted Attack Setup for Basic Model
      APGD_attack_targeted_base = torchattacks.APGD(basic_model, eps=8/255, steps=10, loss='ce')
      print("Basic Model - Targeted Auto-PGD Attack")
      adv_test_targeted(APGD_attack_targeted_base, basic_model)

Basic Model - Targeted PGD Attack

[Plain/Test] Under Testing ... Please Wait
100%|██████████| 100/100 [21:26<00:00, 12.86s/it]
[Plain/Test] Accuracy: 81.510
Basic Model - Targeted Auto-PGD Attack

[Plain/Test] Under Testing ... Please Wait
100%|██████████| 100/100 [05:47<00:00,  3.47s/it]
[Plain/Test] Accuracy: 83.040
```

Figure 5. Results of Targeted PGD and Auto-PGD attack

c. Explain why the PGD attack starts at a random point (rather than at the input point itself). Implement and execute the untargeted PGD attack that starts at the input point. Compare the results with those of the untargeted PGD attack from part (a).

- Original assumption: The PGD attack starts from a random point instead of the input space so as to explore a wider range of the input space. This helps with finding adversarial examples that probably would not be found by starting at the input point. If the attack always starts from the same point, it may end up finding the same adversarial example for similar inputs or get stuck in a local optima. So, starting from random points helps in exploring the input space better and hence increasing the chances of finding effective adversarial examples that can fool the model.

- In part (a), we had the basic model's accuracy 0.69% (when we had a PGD attack starting at a random start). However, when we set it up to not start at a random input, the accuracy we received dropped down to **0.57%**. This doesn't make sense to us because this indicates that starting the attack from the random point was less effective than starting from the input point which contradicts with our original answer above. We would need to understand and go into more detail why this was the case but due to time constraints, we weren't able to do so.

Figure 6 below displays our results.

Figure 6. Result on Untargeted PGD attack with no random start

d. Explain the difference between the PGD and Auto-PGD attacks. Which shortcomings of PGD have been improved by Auto-PGD?

- Auto-PGD enhances PGD attack by taking an adaptive step size. This adjustment allows Auto-PGD to more effectively navigate the landscape, improving its ability to generate successful adversarial examples by adapting the step size based on the attack's progress and the specific input, leading to better performance in certain scenarios.

We've graphed the results for a clearer visualization below in *Figure 7*. Just as the targeted attack results show the Attack Success Rate (ASR), the untargeted attack results show the model's accuracy rate, similar to the one in the FGSM part.

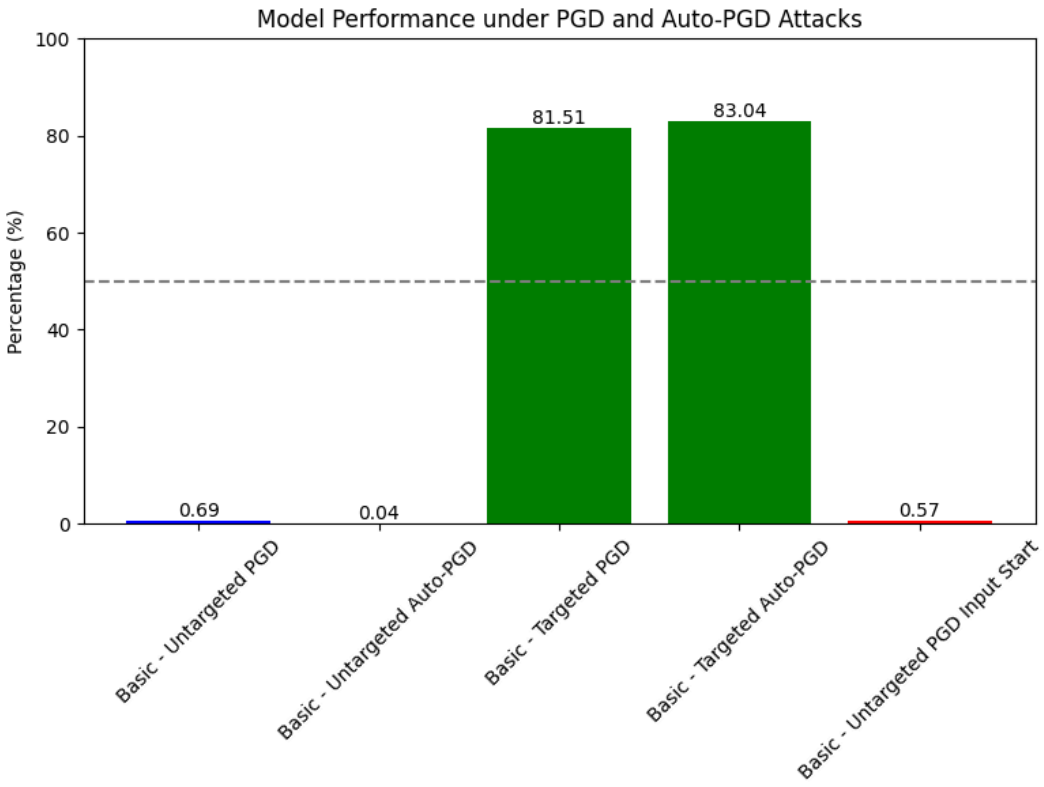

Figure 7. Results on PGD and Auto-PGD Attacks