Attack Project Report

Team Emoji

Name: Yifeng Xiong	UCI Net ID: yifengx4
Name: Monica Zhou	UCI Net ID: quanz13

Our Approach

Our attack is a white-box attack, and we do it in iterative ways. In each iteration, we get the gradient of the image and find out the top n largest and top n smallest coordinates. Then we subtract the gradient from the corresponding channel, and we do a clip to make sure every element of the image is still in [0, 1]. At the end of the iteration, we check whether the attack succeeds by calling the get_batch_out and check whether the target label has the largest score.

We decide to use the white-box attack in this project. Based on FGSM provided, we firstly implement BIM. Then we expand the BIM to PGD, by adding some noise at the beginning, and we add a check function, which will immediately break the iteration if our attack succeeds. Finally, we get some ideas from JSMA. In JSMA, the authors define a saliency map:

$$S(\mathbf{X}, t)[i] = \begin{cases} 0 \text{ if } \frac{\partial \mathbf{F}_t(\mathbf{X})}{\partial \mathbf{X}_i} < 0 \text{ or } \sum_{j \neq t} \frac{\partial \mathbf{F}_j(\mathbf{X})}{\partial \mathbf{X}_i} > 0 \\ \left(\frac{\partial \mathbf{F}_t(\mathbf{X})}{\partial \mathbf{X}_i}\right) \left|\sum_{j \neq t} \frac{\partial \mathbf{F}_j(\mathbf{X})}{\partial \mathbf{X}_i}\right| \text{ otherwise} \end{cases}$$

And they select the $i_{max} = argmax_i S(X, Y^*)[i]$ and modify X_i by θ. We think JSMA changes one number each iteration, so we would like to apply this strategy to PGD. But instead of changing one number, we make some changes: we pick the top n largest and top n smallest. Instead of directly changing the corresponding number to 1 or 0, we change it by minus the gradient. In summary, we are doing the iterative method, but we only change a small part. A better name may be Partial Iterative Method.

The following is the pseudocode:

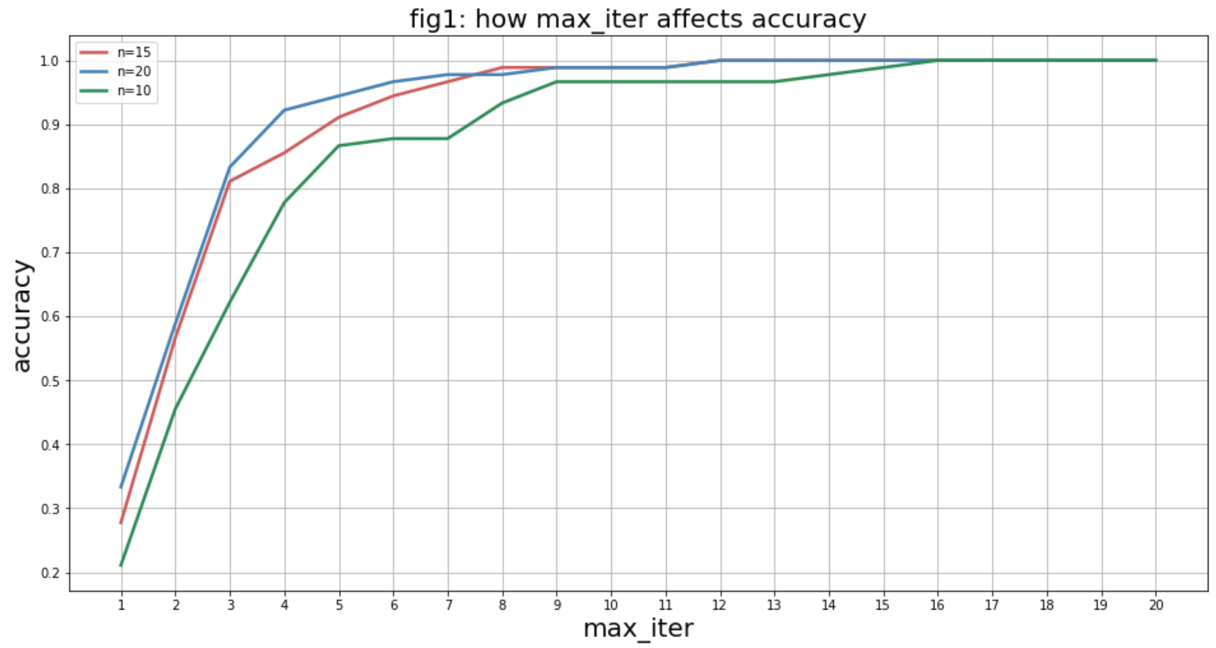
```
Input: X(original image), target_label, n(top n largest and n
smallest will be chosen), max_iter
Output: X* (perturbed image)
while iter < max_iter do
      Compute the gradient $\nabla F(X)$
      max_n = the top n largest components
      min_n = the top n smallest components
      X* = X - corresponding gradient
      Clip(X*)
      if attack succeed then
```

```
                break
        else
                X = X*
return X*
```

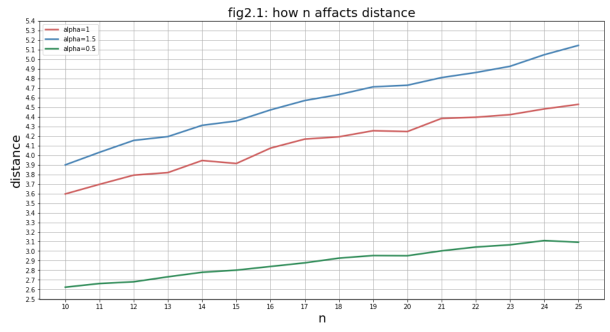
Analysis

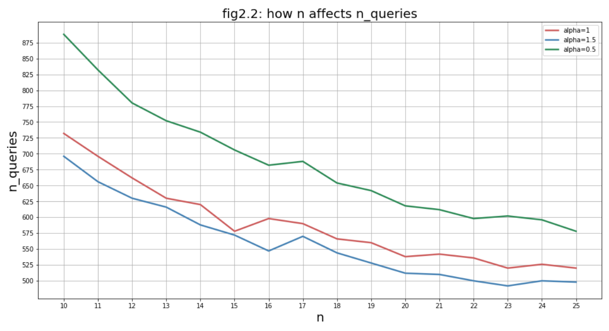
We believe Test Accuracy is much more important than Distance and Number of Queries. And in our approach, there are three numbers we can change: n (how many numbers will change), max_iter (how many iterations), and alpha (how much should we modify in each iteration).

Firstly, we notice that when we fix n and alpha, and increase max_iter, the test accuracy will increase. Then if we fix n and alpha, and continue to increase max_iter, the distance and queries will stop increasing at some points. By breaking the iteration after the attack succeeds, we are able to ensure the running time will not be affected by a large max_iter number. So we can set max_iter to a large number and focus on the other two.

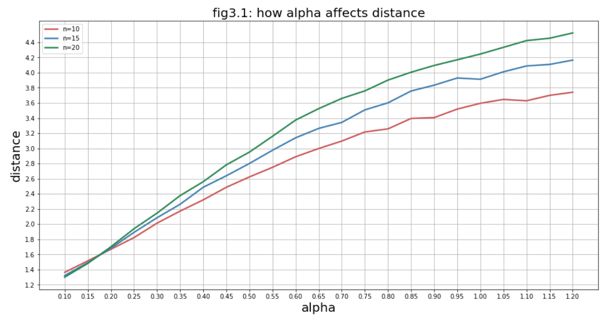


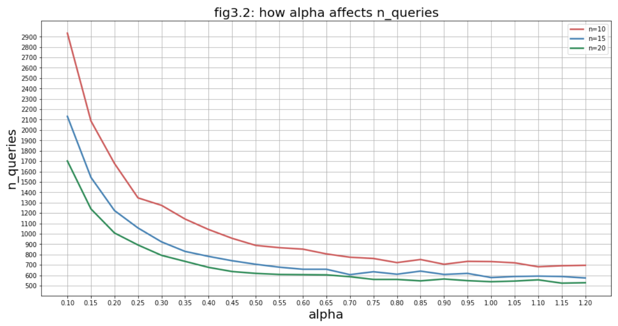
Then, we set max_iter to 100, and alpha to 0.5, 1, and 1.5. We notice that when n increase, the distance will increase, but the number of queries will decrease. This is because the increase of n means in every iteration, we will change more numbers.





Finally, we set max_iter to 100, and n to 10, 15, 20. We notice that when alpha increases, the distance will increase, but the number of queries will decrease. This is because the increase of alpha will make the change of every iteration larger.





What we want is small distance and small n_queries. But based on fig 3.1and fig 3.2, we get a smaller distance if alpha is smaller, and we get a smaller n_queries if alpha is larger. Instead of doing a tradeoff here, we decide to increase n. As we can see from fig 3.2, n_queries decrease if n increases.
So, we fix max_iter = 100, alpha = 0.1, and here is the data we get:

	Test Accuracy	Distance	Number of Queries
n = 20	90 / 90	1.3502601	1640
n = 100	90 / 90	1.244008	682
n = 300	90 / 90	1.3042741	504
n = 500	90 / 90	1.3080025	454
n = 1000	90 / 90	1.403391	456

We choose n = 500 for further exploration. Then we fix n = 500, max_iter = 100, and we would change alpha to get the final result.

	Test Accuracy	Distance	Number of Queries
alpha = 0.02	90 / 90	0.7733395	1384

alpha = 0.05	90 / 90	0.9298556	666
alpha = 0.1	90 / 90	1.3080025	454
alpha = 0.15	90 / 90	1.8408854	402

The best result we can find is n = 500, max_iter = 100, alpha = 0.05.
Compare to FGSM and PGD, our method has some significant advantages.

	Test Accuracy	Distance	Number of Queries
FGSM	84 / 90	15.322199	180
PGD	90 / 90	4.604026	1578
Ours(n=500, max_iter=100, alpha=0.05)	90 / 90	0.9298556	666

Compared to FGSM, our method has higher test accuracy and a much smaller distance. Compared to PGD, our method has nearly one-fourth of the distance, and one-third of the number of queries.

Concluding Thoughts

We think we have succeeded with this project, and we are satisfied with our approach. We have good results in local testing, as well. Before starting the project, we read a lot of papers for preparation. We have a full understanding of black-box attacks and white-box attacks. In fact, we also implemented another approach based on genetic attack, but each time it takes too long to test, and the result is relatively poor, so we finally decided to do a white-box attack. While implementing JSMA, we encountered some difficulties: after we completed the code according to their idea, we could not achieve the desired result. But we did not give up and started to think about the difference between JSMA and other methods. Finally, we combined the advantages of each white-box attack and designed our own method.
We think we can do better with more time. But this is also because we are working together for the first time, and we have not found a very efficient way. On the other hand, we spend too much time reading papers and ignoring hands-on implementation. We will improve in the next project.

Work Cited

Kurakin, Alexey, et al. “Adversarial Examples in the Physical World.” *ArXiv.org*, 11 Feb. 2017, https://arxiv.org/abs/1607.02533.

Madry, Aleksander, et al. “Towards Deep Learning Models Resistant to Adversarial Attacks.” *ArXiv.org*, 4 Sept. 2019, https://arxiv.org/abs/1706.06083.

Papernot, Nicolas, et al. “The Limitations of Deep Learning in Adversarial Settings.” *ArXiv.org*, 24 Nov. 2015, https://arxiv.org/abs/1511.07528.