ECE661 - HW5 Self-Contained Report

Elisa Chen (Duke ID: eyc11)

2023-11-21

1. True/False Questions

I did not receive a notification that the homework set was updated. I am responding to the homework set distributed prior to Nov 12th.

Problem 1.1: **False** - in an evasion attack, the attacker perturbs inputs from the users after the model has been deployed (not during training) fooling the DNN to output incorrect decisions.

Problem 1.2: **False** - adversarial defenses usually will improve the robustness of the model, but there will always be a trade-off between robustness and accuracy. Good adversarial training should be able to balance the trade-off, but never escape entirely from it.

Problem 1.3: **False** - in a backdoor attack, the attacker will first inject a specific noise trigger to a subset of training data points with the intention of forcing / fooling the model to learn a correlation between the trigger and a corrupted target label. However, the attacker doesn't need to use gradient-based perturbation during deployment as the backdoor attack itself involves the model being trained with the trigger, which would be activated during deployment.

Problem 1.4: **False** - both outlier exposure and ODIN detector use OOD data during training.

Problem 1.5: **True** - it has been empirically shown that white-box attack generated on one model is likely to be effective on another model, even with different model structure if they were trained on the same dataset.

Problem 1.6: **False** - it has been shown empirically that the steepest ascent on the local loss surface isn't necessarily the most efficient direction towards the decision boundary. Iterative methods such as basic iterative method (BIM) with smaller step size may find a better overall direction for the perturbation.

Problem 1.7: **True** - the purpose of the projection step of PGD attack is to prevent a misleading gradient due to gradient masking. Starting from a random point near the data sample is likely going to mitigate the gradient masking effect and the projection step ensures that the perturbed input remains within a specified epsilon distance from the original input.

Problem 1.8: **False** - attacks generated from intermediate layers transfer better than those generated at the output layer as shown through research in feature space attacks.

Problem 1.9: **True** - the DVERGE algorithm achieves significantly lower attack transferability between sub-models and increases the robustness of the model ensemble. The objective can be minimized even if sub-models use different set of features, including non-robust features.

Problem 1.10: **False** - The distribution of valid triggers isn't necessarily convex with one possible minima and could have a much more complicated search space. There are usually a number of valid triggers that could serve as triggers during inference time to generate a misclassification.

2. Lab 1

a): - The final accuracy of `NetA` model: 92.39% - The final accuracy of `NetB` model: 92.84%

The two models do **not** have the same architectures. When looking at the `models.py` file, we can tell that NetA and NetB models do not share the same architecture. NetA has 3 convolutional layers whereas NetB has 4 convolutional layers. The pooling is also applied at different stages of the model.

b): Please see the `PGD_attack` function in `attacks.py` file or the below screenshot for more details on the implementation of L_∞-constrained PGD adversarial attack. The below table describes what each of the input arguments is controlling:

Input Parameter	Description
`model`	neural net we train
`device`	machine we train our model on (cpu vs. gpu)
`dat`	features of data
`lbl`	ground truth labels
`eps`	max magnitude of perturbation that is applied to each sample. This is also used to enforce L-inf constraint
`alpha`	learning rate of training
`iters`	number of training steps
`rand_start`	boolean to indicate whether we choose a random point at the beginning to mitigate gradient masking effect

Please see the below screenshot for details of the PGD attack implementation:

```python
def PGD_attack(model, device, dat, lbl, eps, alpha, iters, rand_start):
    # TODO: Implement the PGD attack
    # - dat and lbl are tensors
    # - eps and alpha are floats
    # - iters is an integer
    # - rand_start is a bool

    # x_nat is the natural (clean) data batch, we .clone().detach()
    # to copy it and detach it from our computational graph
    x_nat = dat.clone().detach()

    # If rand_start is True, add uniform noise to the sample within [-eps,+eps],
    # else just copy x_nat
    if rand_start:
        x_nat += torch.FloatTensor(x_nat.shape).uniform_(-eps, eps).to(device)

    # Make sure the sample is projected into original distribution bounds [0,1]
    x_nat = torch.clamp(x_nat.clone().detach(), 0., 1.)

    # Iterate over iters
    for _ in range(iters):

        # Compute gradient w.r.t. data (we give you this function, but understand it)
        gradient = gradient_wrt_data(model, device, x_nat, lbl)

        # Perturb the image using the gradient
        x_nat = x_nat + alpha * torch.sign(gradient)

        # Clip the perturbed datapoints to ensure we still satisfy L_infinity constraint
        x_nat = torch.max(torch.min(x_nat, dat + eps), dat - eps)

        # Clip the perturbed datapoints to ensure we are in bounds [0,1]
        x_nat = torch.clamp(x_nat.clone().detach(), 0., 1.)

    # Return the final perturbed samples
    return x_nat
```

Figure 1: Coding implementation for PGD Attack

When we visualize perturbed samples with different values of ϵ, we notice that the noise becomes clearly visible to humans when $\epsilon = 0.1$. See below a subset of perturbed samples for different values of ϵ. For the most part, I'd imagine that humans would still be able to identify the rough silhouette of the items and make pretty accurate predictions despite of the noise. Notice than when $\epsilon = 0$, we obtain identical samples to the original input image as no perturbation is applied.

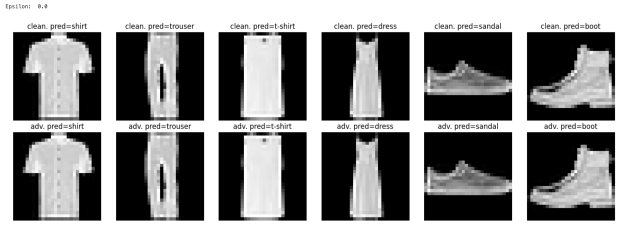

Figure 2: PGD Attack with Epsilon = 0 (No perturbation)

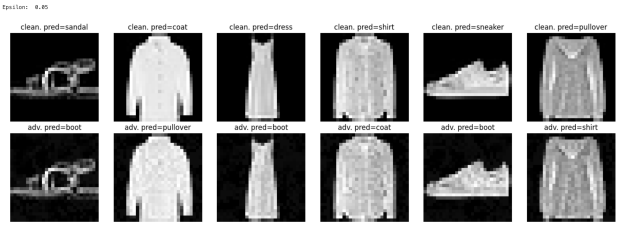

Figure 3: PGD Attack with Epsilon = 0.05

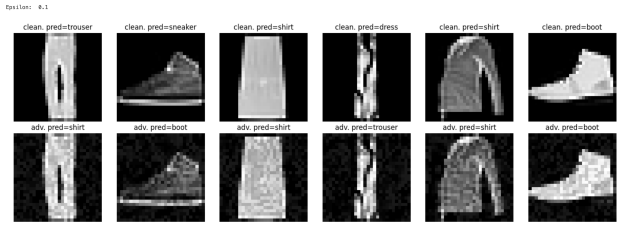

Figure 4: PGD Attack with Epsilon = 0.1

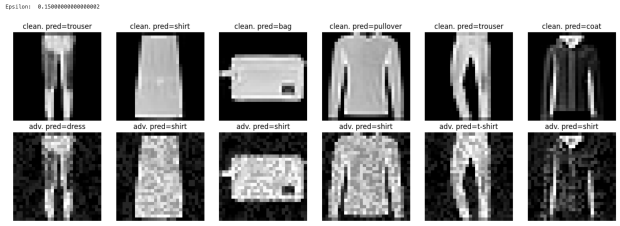

Figure 5: PGD Attack with Epsilon = 0.15

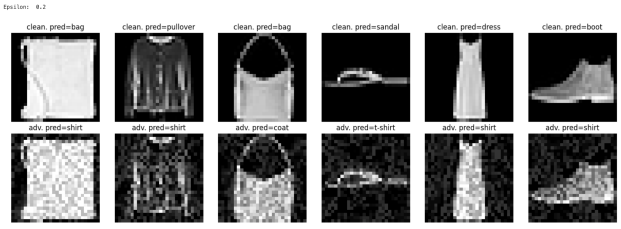

Figure 6: PGD Attack with Epsilon = 0.2

c): FGSM and rFGSM attacks are special cases of the PGD attack in the sense that alpha = eps and it is a one-step attack (iters = 1). The only difference between FGSM and rFGSM is that rFGSM uses random start and FGSM does not. See below for the implementations of FGSM and rFGSM attacks:

```python
def FGSM_attack(model, device, dat, lbl, eps, iters = 1):
    # TODO: Implement the FGSM attack
    # - Dat and lbl are tensors
    # - eps is a float

    # HINT: FGSM is a special case of PGD

    return PGD_attack(model, device, dat, lbl, eps, eps, iters, rand_start = False)


def rFGSM_attack(model, device, dat, lbl, eps, iters = 1):
    # TODO: Implement the FGSM attack
    # - Dat and lbl are tensors
    # - eps is a float

    # HINT: rFGSM is a special case of PGD

    return PGD_attack(model, device, dat, lbl, eps, eps, iters, rand_start = True)
```

Figure 7: FGSM and rFGSM code implementation

See below a subset of perturbed samples for different values of ϵ. Similar to PGD, when $\epsilon = 0$, we obtain identical samples to the original input image as no perturbation is applied. The noise seems to be less effective on darker shades than lighter shades for FGSM, whereas for PGD attack, the perturbation is applied evenly across lighter and darker shades. Additionally, it appears that FGSM would require a larger epsilon for the noise to be visible compared to PGD attack. Please see below figures for examples of FGSM and rFGSM for a range of epsilon values between 0 - 0.2:

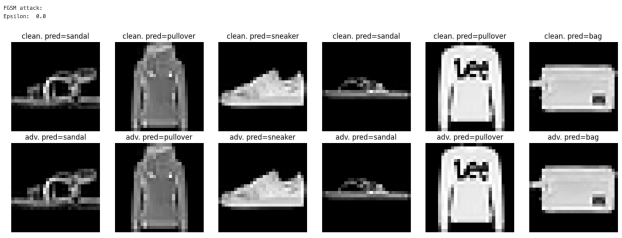

Figure 8: FGSM Attack with Epsilon = 0 (no perturbation)

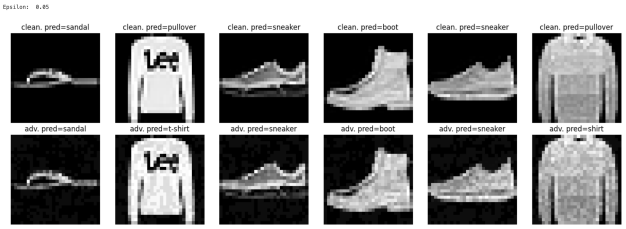

Figure 9: FGSM Attack with Epsilon = 0.05

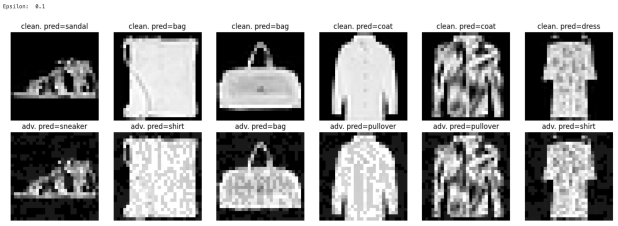

Figure 10: FGSM Attack with Epsilon = 0.1

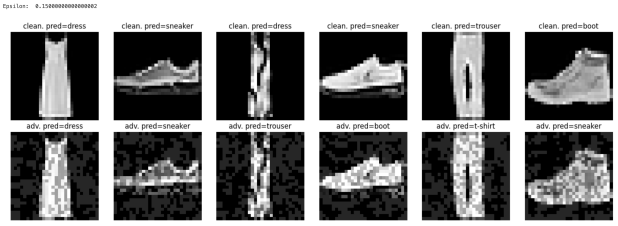

Figure 11: FGSM Attack with Epsilon = 0.15

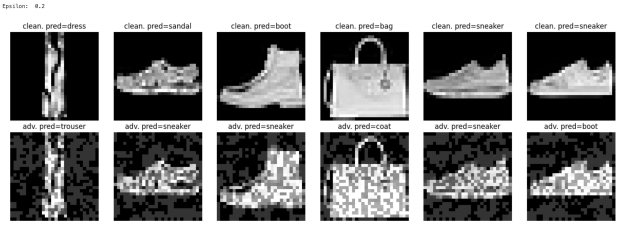

Figure 12: FGSM Attack with Epsilon = 0.2

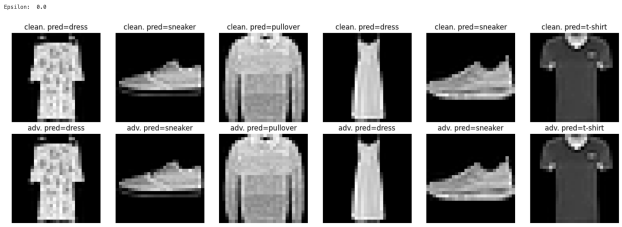

Figure 13: rFGSM Attack with Epsilon = 0 (no perturbation)

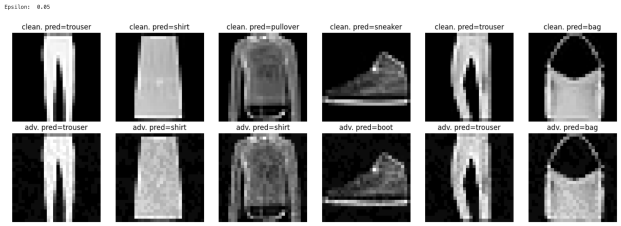

Figure 14: rFGSM Attack with Epsilon = 0.05

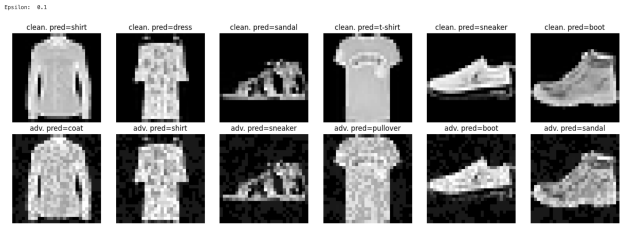

Figure 15: rFGSM Attack with Epsilon = 0.1

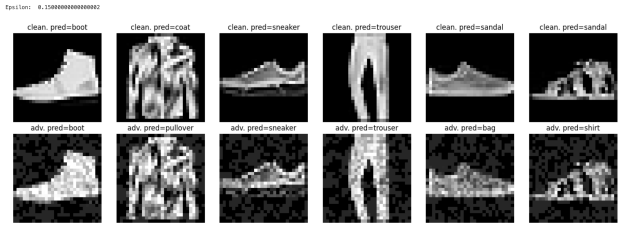

Figure 16: rFGSM Attack with Epsilon = 0.15

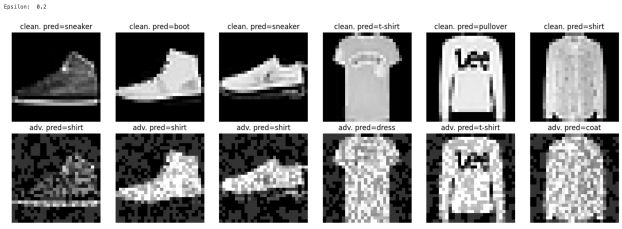

Figure 17: rFGSM Attack with Epsilon = 0.2

d): Please see code implementation of the L_2-constrained FGM attack in Figure 18. It seems like the L_2-constrained FGM attack requires a much larger ϵ value to have visible perturbation on the samples. The noise becomes perceptible to humans when $\epsilon = 2$, but is still not very effective in fooling the model. Visually, the L_2-constrained FGM attack is much more subtle compared to the FGSM and PGD attacks. The predictions on perturbed samples seem more accurate with L_2-constrained FGM attack. Please see examples of perturbed samples with varying ϵ values in the range of [0.0, 4.0]:

```python
def FGM_L2_attack(model, device, dat, lbl, eps):
    # x_nat is the natural (clean) data batch, we .clone().detach()
    # to copy it and detach it from our computational graph
    x_nat = dat.clone().detach()

    # Compute gradient w.r.t. data
    gradient = gradient_wrt_data(model, device, x_nat, lbl)

    # Compute sample-wise L2 norm of gradient (L2 norm for each batch element)
    # HINT: Flatten gradient tensor first, then compute L2 norm
    flattened_gradient = gradient.view(gradient.shape[0], -1)
    l2_norm = torch.norm(flattened_gradient, dim = 1) # check
    l2_norm = torch.clamp(l2_norm, min = 1e-12)
    l2_gradient = gradient / l2_norm.view(-1, 1, 1, 1)

    # Perturb the data using the gradient
    # HINT: Before normalizing the gradient by its L2 norm, use
    # torch.clamp(l2_of_grad, min=1e-12) to prevent division by 0

    # Add perturbation the data
    x_nat = x_nat + eps * l2_gradient

    # Clip the perturbed datapoints to ensure we are in bounds [0,1]
    x_nat = torch.clamp(x_nat.clone().detach(), 0., 1.)

    # Return the perturbed samples
    return x_nat
```

Figure 18: L2 FGM Attack Code Implementation

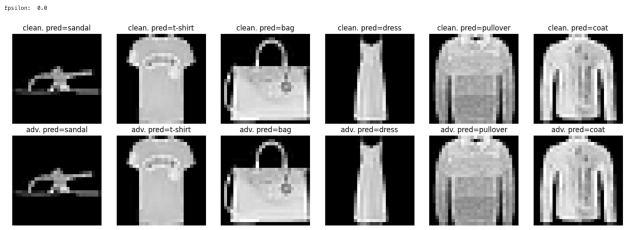

Figure 19: L2 FGM Attack with Epsilon = 0 (no perturbation)

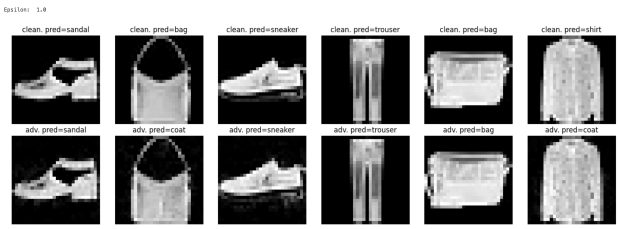

Figure 20: L2 FGM Attack with Epsilon = 1

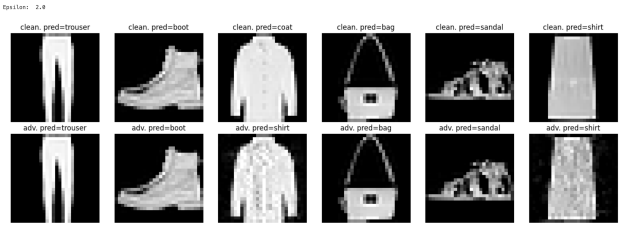

Figure 21: L2 FGM Attack with Epsilon = 2

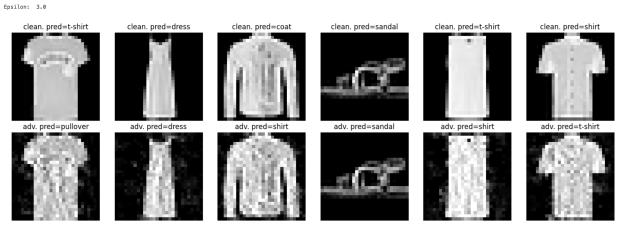

Figure 22: L2 FGM Attack with Epsilon = 3

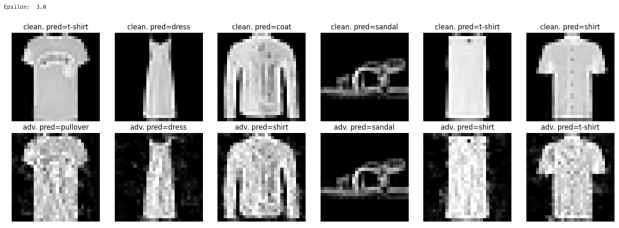

Figure 23: L2 FGM Attack with Epsilon = 4

3. Lab 2

a): In white-box attacks, the attacker assumes full access to the target model including knowledge of the architecture and weights of the neural network. On the contrary, black-box attackers only have query access to the target model (they have knowledge of input and output of models) and make no assumptions about model architecture or weights. A tactic in which we generate attacks on one model and input them into another model that has been trained on the same dataset is called **transfer attack**.

b): the `netA` was assigned as the whitebox model and `netB` was assigned as the blackbox model for this question. `np.linspace(0, 0.1, 11)` was used to generate 11 different epsilons within the range [0, 0.1] for the perturbations. Please see the below figure for details about the accuracy of each model (whitebox and blackbox) for each epsilon value:

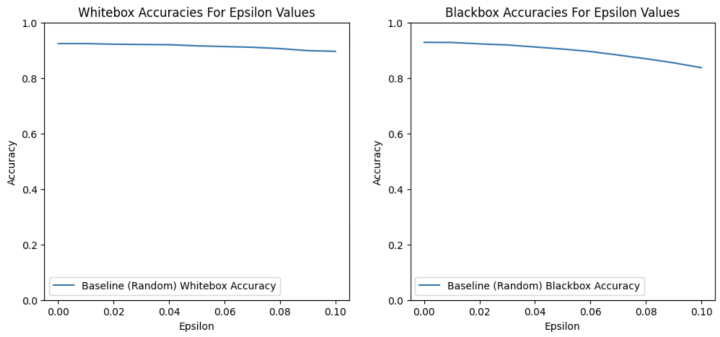

Figure 24: Accuracy vs. Epsilon for Whitebox and Blackbox Models (Random Attack)

Based on the information presented in the accuracy plots, random noise isn't particularly effective in attacking either of the models knowing that the accuracy is ~92% for each non-perturbed model. The accuracy doesn't change drastically for the whitebox model regardless of the ϵ value. The random noise is slightly more effective for the blackbox model than for the whitebox model when $\epsilon > 0.06$.

c): All methods (FGSM, rFGSM and PGD) perform significantly better than the random attack. rFGSM and FGSM attacks yield a final accuracy of ~30-35% when $\epsilon = 0.2$ and PGD yields a close to 0% accuracy when $\epsilon = 0.2$. It does appear that PGD is the most effective attack compared to the other methods. None of the attacks induce the equivalent of the "random guessing" accuracy except when $\epsilon = 0$, during which all attacks have the same accuracy as random guessing. Please see figure 25 for more details:

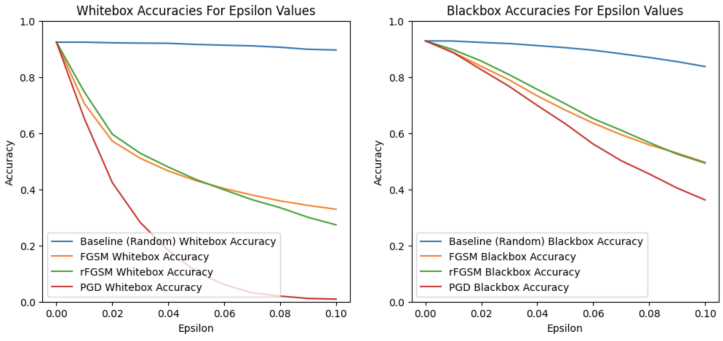

Figure 25: Accuracy vs. Epsilon for Whitebox and Blackbox Models (All Attacks)

d): All methods (FGSM, rFGSM and PGD) perform significantly better than the random attack for the blackbox model. rFGSM and FGSM attacks yield a final accuracy of ~50% when $\epsilon = 0.2$ and PGD yields ~40% accuracy when $\epsilon = 0.2$. It does appear that PGD is still the most effective attack compared to the

other methods even if overall the attacks aren't as effective on the blackbox model compared to the whitebox attacks. However, this is expected as we're using the whitebox model as a proxy for the blackbox model, and some information is lost in the transfer process. Similar to the whitebox model, none of the attacks induce the equivalent of the "random guessing" accuracy except when $\epsilon = 0$, during which all attacks have the same accuracy as random guessing. Please see figure 25 for more details.

e): As expected, the attack success rate curves are significantly lower for the whitebox model than the blackbox model, which is expected as we're using the whitebox as a proxy DNN for the blackbox model in this transfer attack. However, all methods are still significantly better at attacking the model than random guessing and PGD method is the most effective at attacking for both models. The whitebox attacks are more powerful, which makes sense intuitively, because the attacker has access to the gradients of the model with respect to the input. This information can be used to guide the generation of adversarial examples more efficiently, whereas we do not know the gradients of the blackbox model. In Lab 1b), the perceptibility threshold was found to be ~0.1 for all methods, which is quite alarming knowing that PGD is able to corrupt the prediction ~100% of the time for whitebox models and ~60% of the time for blackbox models.

4. Lab 3

a): The final accuracy of the model on clean test data when adversarially training `NetA` model with FGSM attack: 48.83%. The accuracy is significantly lower (~40% lower) than the accuracy for the standard trained model. The final accuracy of the model on clean test data when adversarially training `NetA` model with rFGSM attack: 88.72%. This is still lower than the accuracy of standard trained model, but significantly higher than the accuracy for the adversarially trained model with FGSM attack. We observe that rFGSM attack has a lot smoother training loss convergence compared to that of FGSM attack as shown in figure 26:

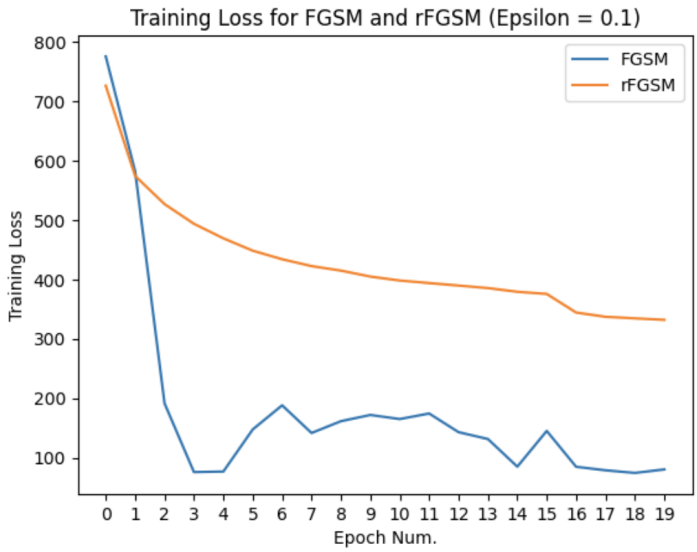

Figure 26: Training Loss for FGSM and rFGSM adversarially trained models

The training loss for FGSM is unstable but seems to reach a lower overall training loss compared to rFGSM. However, the instability in FGSM seems to indicate that the model isn't learning very effectively the attack patterns.

b): The final accuracy of the model on clean test data when adversarially training `NetA` model with PGD attack: 86.90%. It is about ~6% lower than the accuracy of standard trained model. The adversarially trained model with PGD attack follows a similar training convergence as the rFGSM model. They both seem to gradually converge as the epoch number decreases, but the PGD model seems to have a higher overall loss compared to the rFGSM model.

Figure 27: Training Loss for FGSM, rFGSM, and PGD adversarially trained models

c): Please see figure 28 for the accuracy vs. epsilon curves against all three attacks for adversarially trained FGSM and rFGSM model. From the accuracy curves we can observe that rFGSM seems to be relatively more robust to all types of attacks compared to FGSM. While the FGSM model appears to be performing very well for FGSM attack, it does not perform well on rFGSM and PGD attacks. However, this is expected as the FGSM model was able to learn the adversarially perturbed dataset and adjust the decision boundaries during training making itself less susceptible to FGSM attacks. The fact that the model was not robust to rFGSM and PGD attacks might suggest that it is overfitting. On the other hand, the adversarially trained rFGSM model is much more robust to all three attacks: while the overall accuracy for FGSM attack isn't as good as it was with the adversarially trained FGSM model, the accuracy is still >70% for all three attacks even when $\epsilon = 0.1$. Among the three attacks, PGD attack yielded the lowest accuracy while rFGSM and FGSM attacks had similar accuracy levels for all values of ϵ.

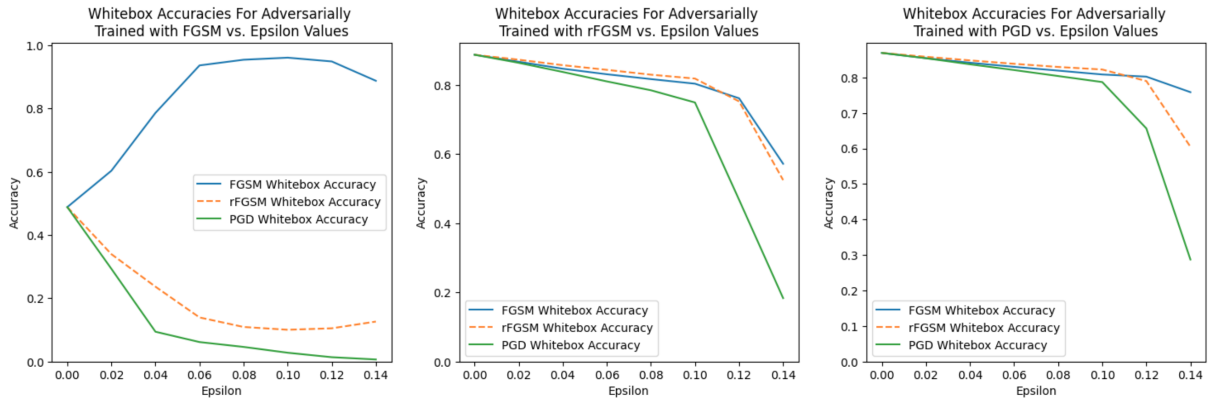

Figure 28: Accuracy vs. Epsilon Curves for FGSM, rFGSM, and PGD adversarially trained models

d): Similar to the adversarially trained rFGSM model, the adversarially trained PGD model is quite robust to all three forms of attacks. Please see figure 28 for more details. Overall, the PGD model seems to have a higher accuracy for a given ϵ value compared to the rFGSM model. The accuracy is >75% for all attacks when $\epsilon = 0.1$. The iterative nature of PGD helps the model adapt to various levels of perturbation making it

more robust overall against all forms of attacks. While the PGD model seems to provide the highest accuracy overall, it can be argued that the adversarially trained rFGSM model is able to yield comparable accuracy, but with much less training cost (because rFGSM only takes one step as opposed to multiple steps like PGD during training), thus making it a better training method than the others. In the paper, the training time for PGD model was ~8x larger than for rFGSM model, so if training time is a major consideration for the model developer, the rFGSM model might be the best training method against the three attacks.

e): Three additional PGD-based AT models were created with ϵ equal to 0.05, 0.15 and 0.2 values. Please see the below table for the clean data accuracy for various PGD-based AT models with varying ϵ values:

Training Epsilon Value	Clean Test Accuracy
0 (non-AT)	92.39%
0.05	89.62%
0.1	86.90%
0.15	85.08%
0.2	84.32%

It seems that the higher the epsilon value, the lower the clean test accuracy. This aligns with our expectations of a tradeoff between robustness and accuracy. The more perturbation we account for in the model, the lower the clean test accuracy. However, the larger the training ϵ value, the more robust the model seems to be against various attacks. Looking at Figure 29, we see that the accuracies are the highest when $\epsilon = 0.2$ and lowest when $\epsilon = 0.05$. Whenever the attack ϵ exceeds the training ϵ, we see a sharp decline in the accuracies for all attacks, and PGD attack seems to decline the most indicating that it is hard to create a robust model against PGD attacks. The PGD-based AT Models hold fairly well against all forms of attacks when training $\epsilon <$ attack ϵ.

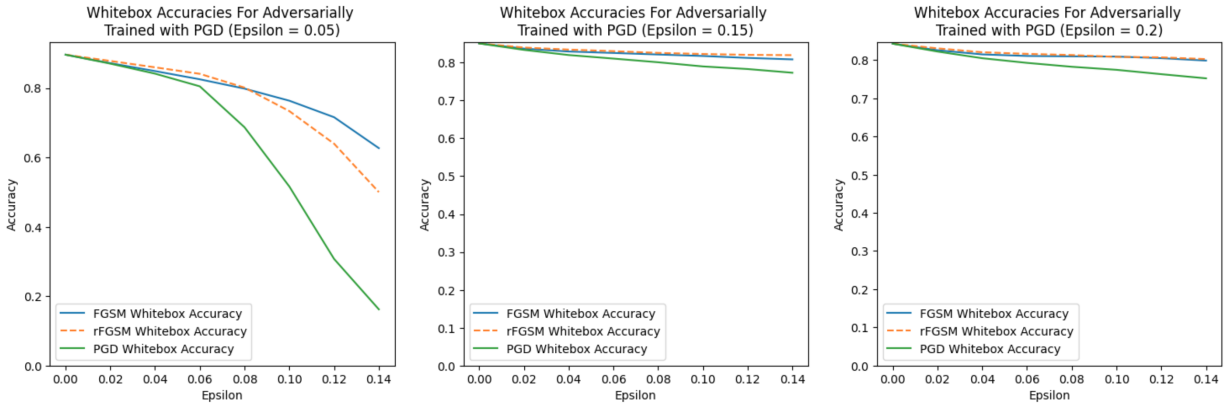

Figure 29: Accuracy vs. Epsilon Curves for Three PGD-AT models

f): Please see Figure 30 for the saliency maps for 6 samples from the dataset for non-AT and PGD-AT models.

The saliency maps between non-AT and PGD-AT models are drastically different. We see that for non-AT models, the saliency map is very noisy and hardly any patterns / representations were learned. On the contrary, the saliency maps for PGD-AT models are significantly less noisy and able to recognize / trace the silhouettes of the clothing. As an example, when $\epsilon = 0.05$, we can clearly see how the outlines of the clothing have been represented.

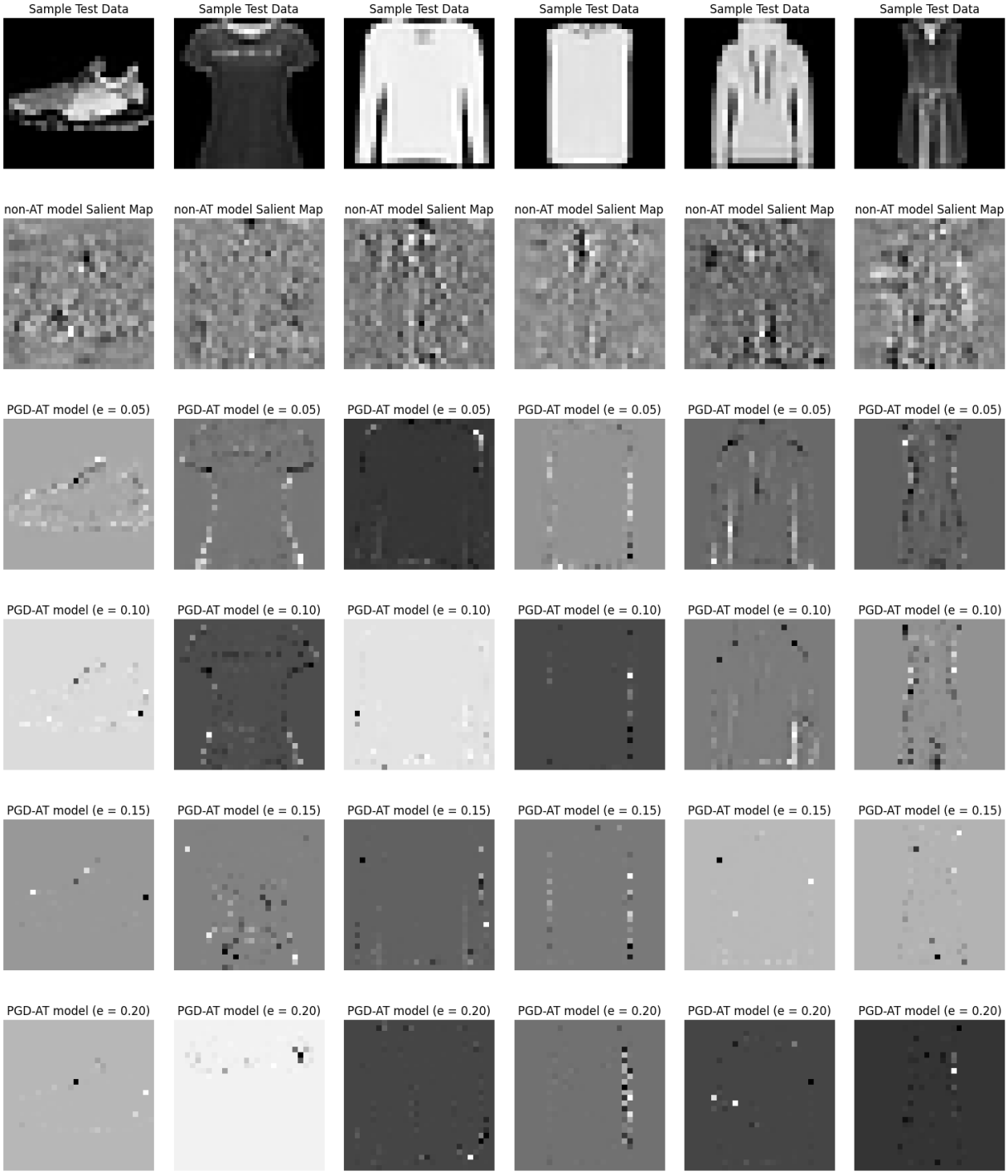

Figure 30: Saliency Maps for non-AT and PGD-AT Models