Name: Pratikshit Singh
NetID: ps71

Basic Attack Methods

For each of Fast Gradient Sign Method (FGSM), Iterative Gradient Sign, and Least Likely Class methods:

(i) Briefly explain your implementation, including which pre-trained model you chose to attack and any hyperparameter choices

Pre-trained model used:
Resnet18 with IMAGENET1K_V1 weights. The last layer of this model was then changed to only have 10 units for our "imagenette2-320" dataset, using fine-tuning.

Fine-tuning step:

- criterion = nn.CrossEntropyLoss()
- optimizer = torch.optim.Adam(lr = 1e-3, params= model.parameters())
- train(model, criterion, optimizer, num_epochs=10, decay_epochs=15, init_lr=1e-3)

Fast Gradient Sign Method (FGSM):
Implementation:

1. If the initial prediction on the image is wrong, we don't bother attacking it, just move on.
2. Else, we calculate the cross_entropy loss on output(generated by the model on the image) and target for the image. Then zero all existing gradients followed by backward pass of the loss.
3. We then use the gradient by using the data grad sign for the below epsilon (multiplication of data grad sign and the epsilon) and adding to the original image, thus generating a perturbed image.
4. Then clamped the perturbed image values between 0 and 1.
5. We then predict the output again using the fine-tuned model on the perturbed image.

Epsilon values = [0, 2, 4, 8, 16, 20, 24, 28, 32, 40, 48, 56, 64, 80, 96, 112, 128] or normalized epsilon = [0.0, 0.0078125, 0.015625, 0.03125, 0.0625, 0.078125, 0.09375, 0.109375, 0.125, 0.15625, 0.1875, 0.21875, 0.25, 0.3125, 0.375, 0.4375, 0.5]

Iterative Gradient Sign Method:

Implementation:

1. If the initial prediction on the image is wrong, we don't bother attacking it, just move on.
2. Else, we attack a small amount in each iteration for the number of iteration defined.
3. In each iteration, we calculate the cross_entropy loss on output(generated by the model on the perturbed image) and target on the perturbed image. Then zero all existing gradients followed by backward pass of the loss.
4. We then use the gradient by using the data grad sign for the below alpha(multiplication of data grad sign and the alpha) and adding to the original image, thus generating a perturbed image.
5. We try to find the maximum between this “perturbed image” and “image + epsilon”, and then its minimum with “image - epsilon”. This limits the adversarial effect that we’re applying each iteration.
6. Then clamped the perturbed image values between 0 and 1.
7. We then predict the output again using the fine-tuned model on the perturbed image.
8. In the next iteration, this perturbed image acts as the base image.

Epsilon values = [0, 12.8, 25.6, 38.4, 51.2, 64.0, 76.8] or normalized epsilon = [0, .05, .1, .15, .2, .25, .3]
alpha = 64
Number of iterations = 10

Least Likely Class Method:

Implementation:

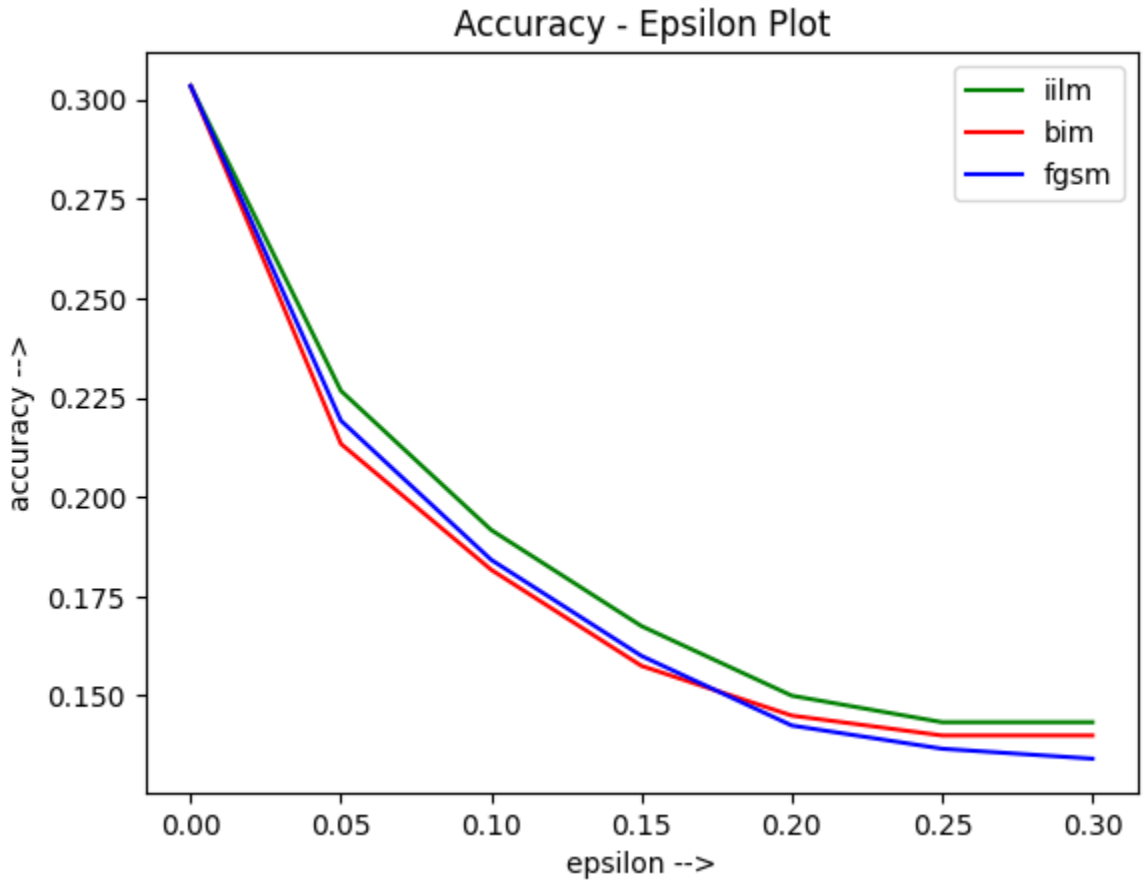
1. If the initial prediction on the image is wrong, we don't bother attacking it, just move on.
2. Else, we attack a small amount in each iteration for the number of iteration defined.

3. In each iteration, we calculate the cross_entropy loss on output(generated by the model on the image) and the least likely predicted class on the perturbed image. Then zero all existing gradients followed by backward pass of the loss.
4. We then use the gradient by using the data grad sign for the below alpha(multiplication of data grad sign and the alpha) and adding to the original image, thus generating a perturbed image.
5. We try to find the maximum between this “perturbed image” and “image + epsilon”, and then its minimum with “image - epsilon”. This limits the adversarial effect that we’re applying each iteration.
6. Then clamped the perturbed image values between 0 and 1.
7. We then predict the output again using the fine-tuned model on the perturbed image.
8. In the next iteration, this perturbed image acts as the base image.

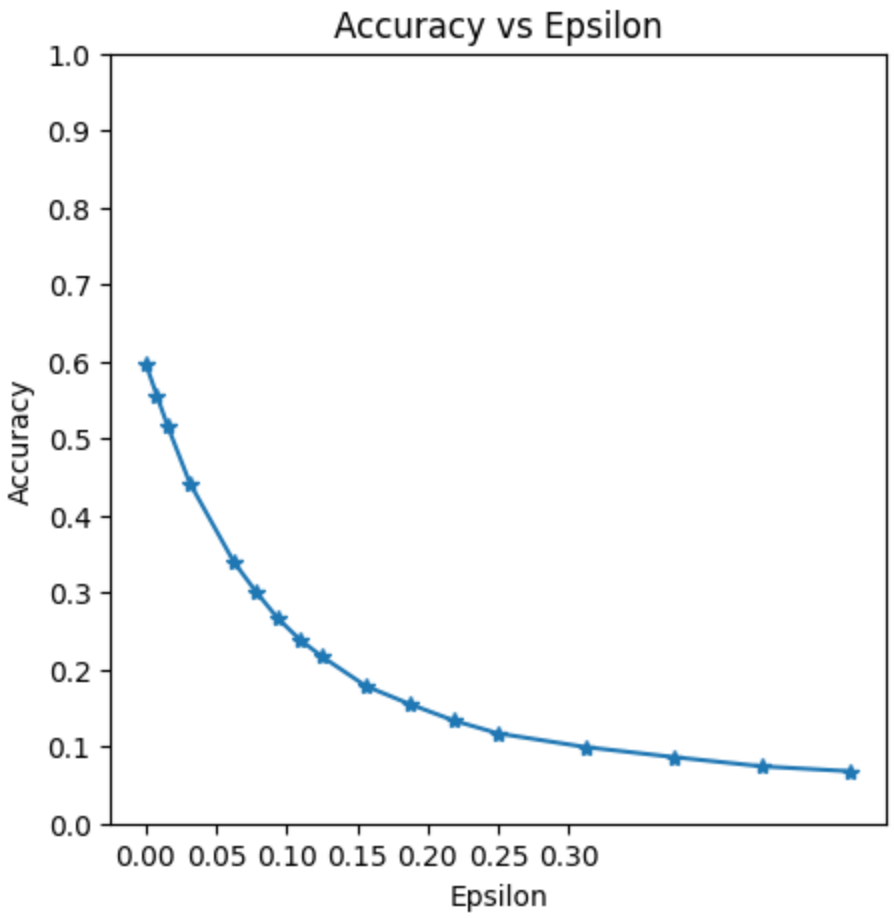
Epsilon values = [0, 12.8, 25.6, 38.4, 51.2, 64.0, 76.8] or [0, .05, .1, .15, .2, .25, .3]
alpha = 64
Number of iterations = 10

(ii) Plot accuracy vs. epsilon on the ImageNette or MNIST datasets. (ImageNette will get you full extra credit points, but if you cannot do ImageNette, you can do MNIST for fewer points. MNIST on top of IageNette will not get you any additional points.) If you implement multiple methods, you should show them all on the same plot.

Comparison plot for all 3 methods:



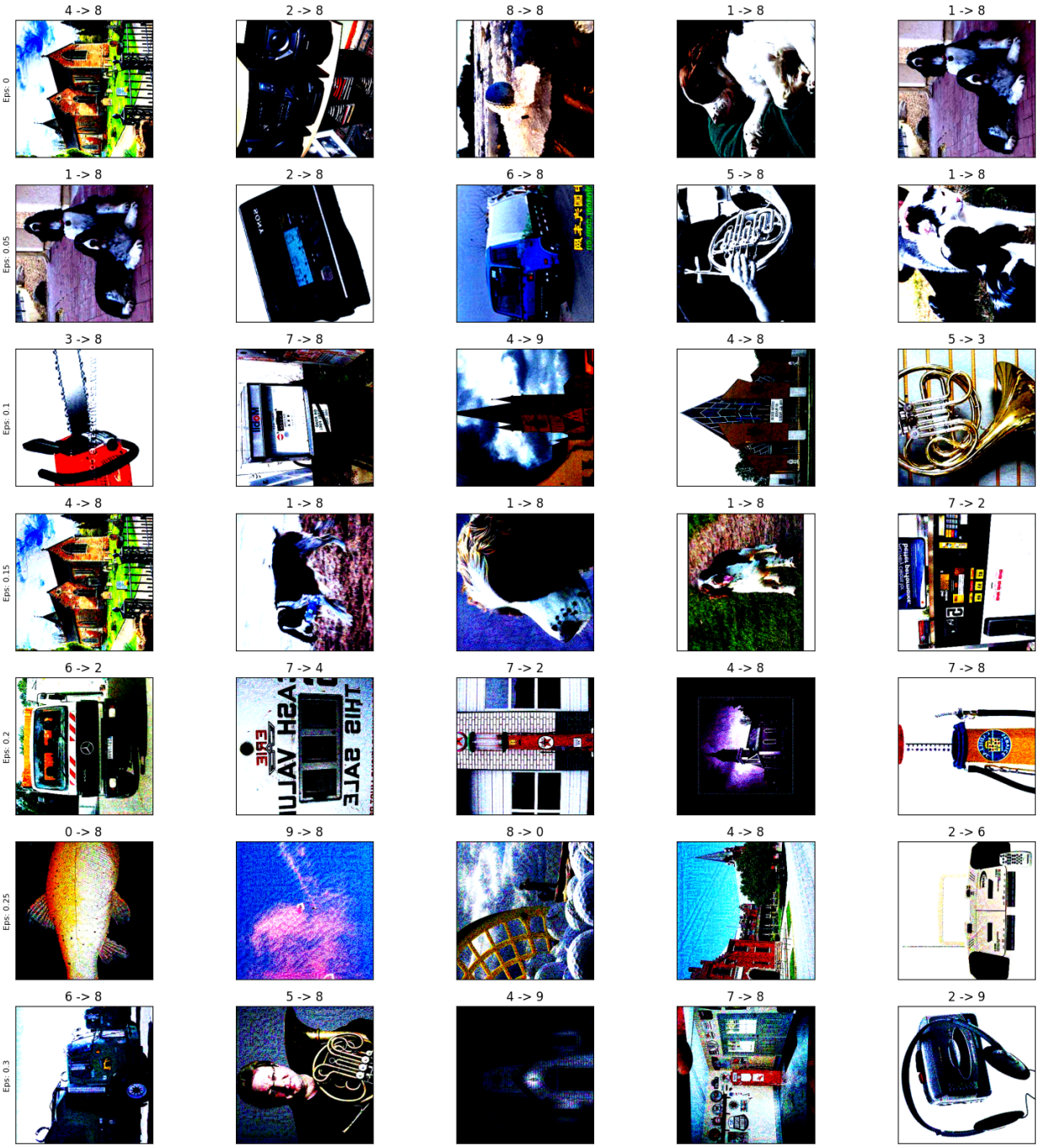
More detailed plot for Fast Gradient Sign Method (FGSM):



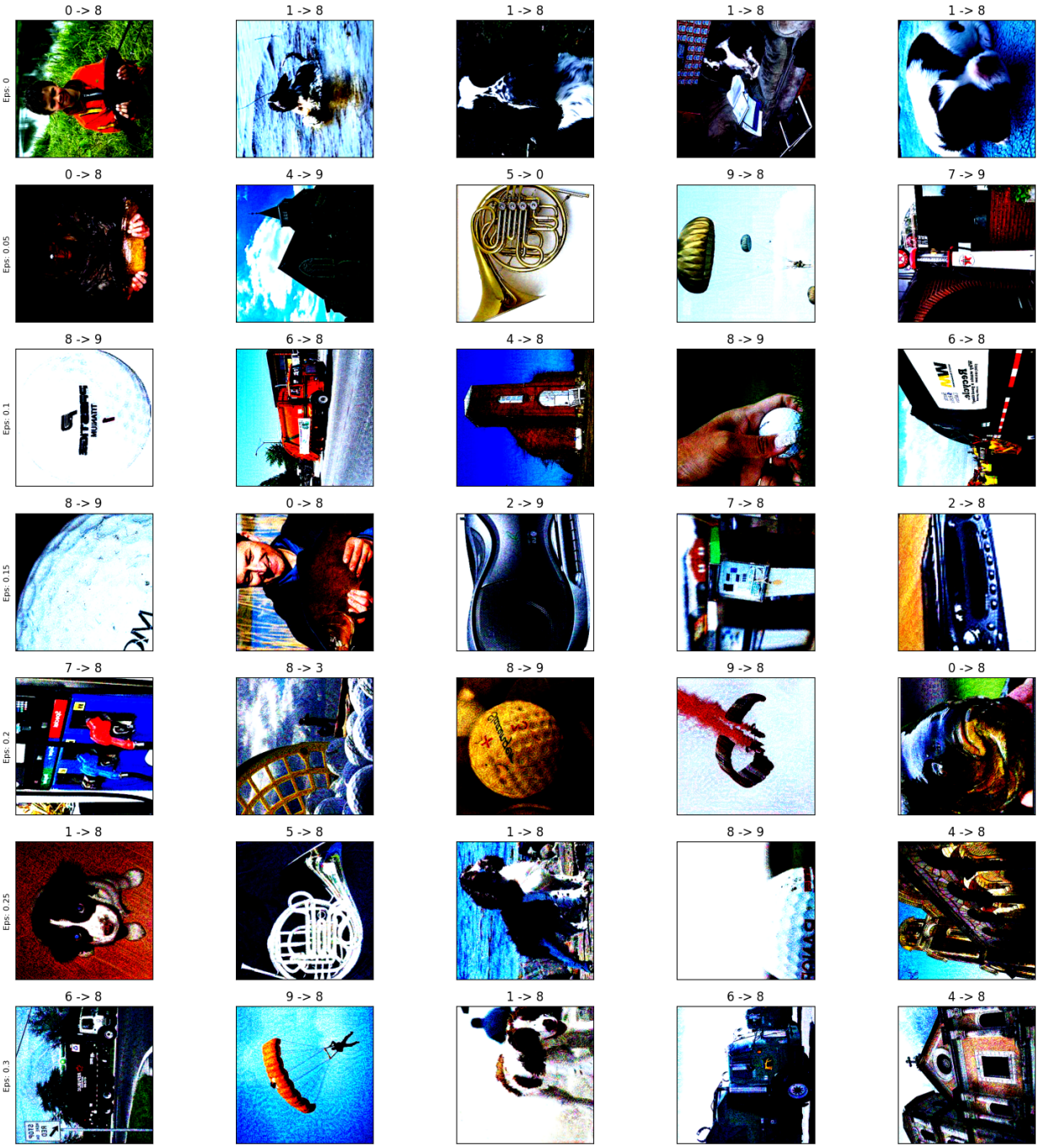
(iii) Visualize examples of attacks, where the original image and original (correctly predicted) label and the attacked image and falsely predicted label are shown for ImageNette or MNIST.

Fast Gradient Sign Method (FGSM):

Iterative Gradient Sign Method:



Least Likely Class Method:



Finally, if you implemented multiple methods, discuss any differences between them in terms of efficacy, visibility of perturbation, computational efficiency, etc.

Fast Gradient Sign Method (FGSM), Iterative Gradient Sign Method, and Least Likely Class Method are popular techniques used for generating adversarial examples in image classification tasks. These methods aim to generate imperceptible perturbations to the input image that can lead to incorrect predictions by the target model.

The FGSM is a simple and efficient method that involves computing the gradient of the loss function with respect to the input image and then perturbing the image in the direction of the sign of the gradient. The magnitude of the perturbation is controlled by a hyperparameter, which determines the strength of the attack. The FGSM has been shown to be effective in generating adversarial examples with high success rates, but the generated perturbations can be visible and easily detected by humans.

The Iterative Gradient Sign Method (IGSM) is a variant of the FGSM that applies the perturbation iteratively. Instead of generating a single perturbation, the IGSM applies multiple small perturbations in the direction of the gradient until the desired misclassification is achieved. The IGSM is more powerful than the FGSM and can generate adversarial examples with higher success rates while reducing the visibility of the perturbations. However, the IGSM is computationally more expensive than the FGSM as it requires multiple iterations.

The Least Likely Class Method (LLCM) is another technique for generating adversarial examples. Instead of targeting the most likely class of the input image, the LLCM targets the least likely class. This method generates perturbations that cause the target model to predict the input image as the least likely class. The LLCM can be more effective than the FGSM and IGSM in scenarios where the target model is highly confident in its predictions. However, the LLCM can also generate more visible perturbations than the other methods.

In terms of efficacy, the IGSM is generally considered to be the most effective technique for generating adversarial examples as it can achieve higher success rates than the FGSM and LLCM. In terms of visibility of perturbations, the IGSM is also the most effective method, followed by the LLCM and FGSM. In terms of computational efficiency, the FGSM is the most efficient method as it requires only one gradient computation per image, while the IGSM requires multiple gradient computations per image.

In conclusion, the choice of adversarial example generation method depends on the specific requirements of the task. The FGSM is a simple and efficient method that can

generate adversarial examples quickly, while the IGSM is a more powerful method that can generate less visible perturbations but requires more computational resources. The LLCM is a useful technique in scenarios where the target model is highly confident in its predictions.