B20AI013_PA1_Report

Note!

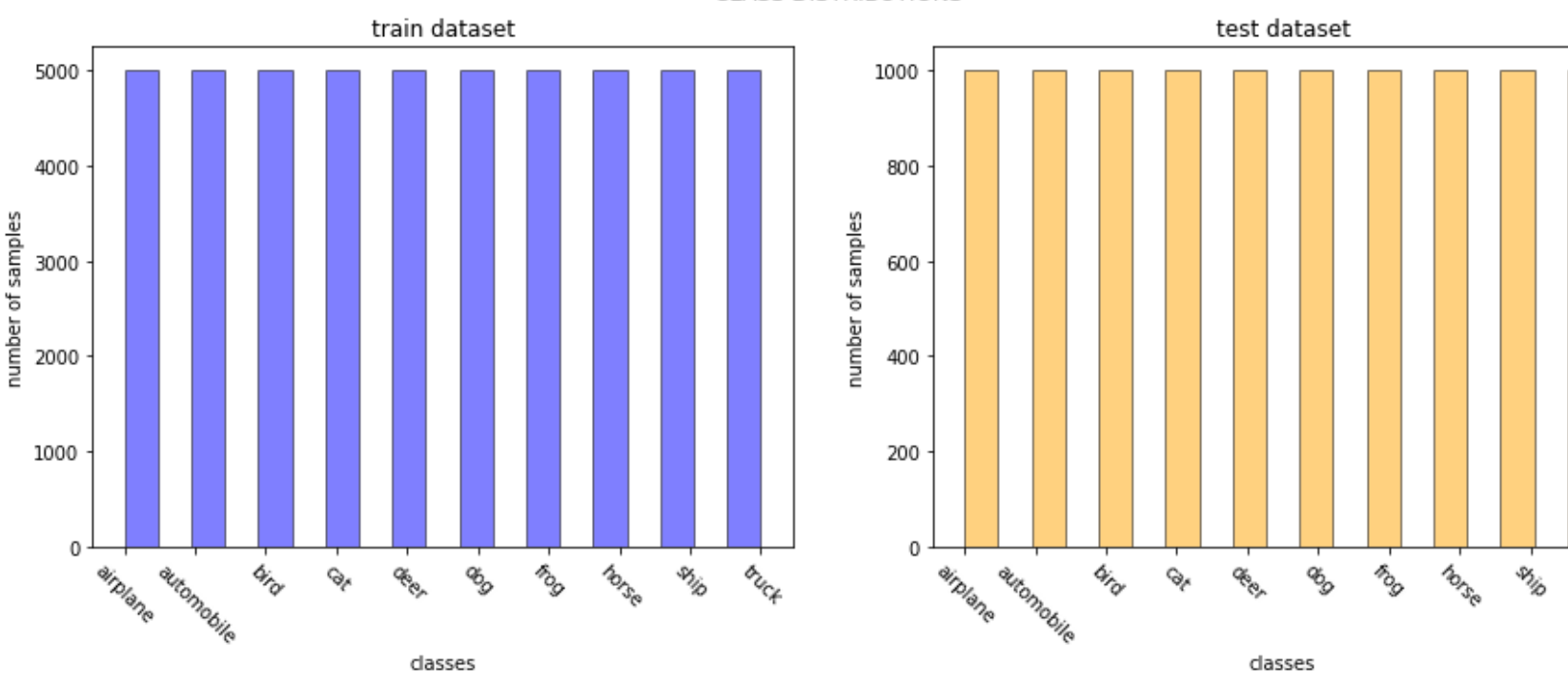
there are two .ipynb code files in the submission. this is because due to the lack of time, I had to resort to training on two different computers at once. Results of question 1 and 2 are mixed between both code files. apologies for this.

Question 1

Q1 Part 1 - implementing FGSM attack on Basic CNN

The CIFAR-10 dataset

The CIFAR-10 dataset is a collection of 60,000 32x32 color images of 10 different object classes, with 6,000 images per class. The classes include airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck. The dataset is split into 50,000 training images and 10,000 testing images. The images are labeled with their corresponding class.



Training Methodology

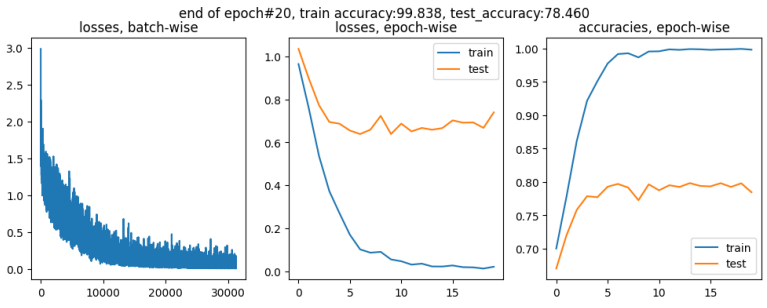
CNN Classifier

The architecture of the CNN classifier was based on my previous experimentation with the dataset. The schematics are as follows:

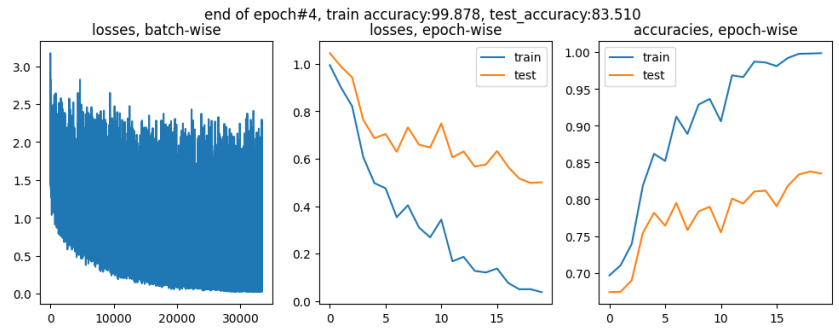
```
Output exceeds the size limit. Open the full output data in a text editor
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1          [-1, 64, 32, 32]           1,728
              ReLU-2          [-1, 64, 32, 32]               0
         MaxPool2d-3          [-1, 64, 16, 16]               0
            Conv2d-4         [-1, 128, 16, 16]          73,728
       BatchNorm2d-5         [-1, 128, 16, 16]             256
              ReLU-6         [-1, 128, 16, 16]               0
         MaxPool2d-7           [-1, 128, 8, 8]               0
            Conv2d-8           [-1, 256, 8, 8]         294,912
       BatchNorm2d-9           [-1, 256, 8, 8]             512
             ReLU-10           [-1, 256, 8, 8]               0
          Flatten-11               [-1, 16384]               0
           Linear-12                [-1, 1024]      16,777,216
     BatchNorm1d-13                [-1, 1024]           2,048
             ReLU-14                [-1, 1024]               0
           Linear-15                  [-1, 10]          10,240
     BatchNorm1d-16                  [-1, 10]              20
================================================================
Total params: 17,160,660
Trainable params: 17,160,660
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.01
...
Forward/backward pass size (MB): 2.46
Params size (MB): 65.46
Estimated Total Size (MB): 67.94
----------------------------------------------------------------
```

I have also used augmentation with AutoAugment [1] which is a dataset-specific augmentation policy learned for deep CNNs with reinforcement learning.

Without AutoAugment, test accuracy: 79.84%



With AutoAugment, test accuracy: 83.78%

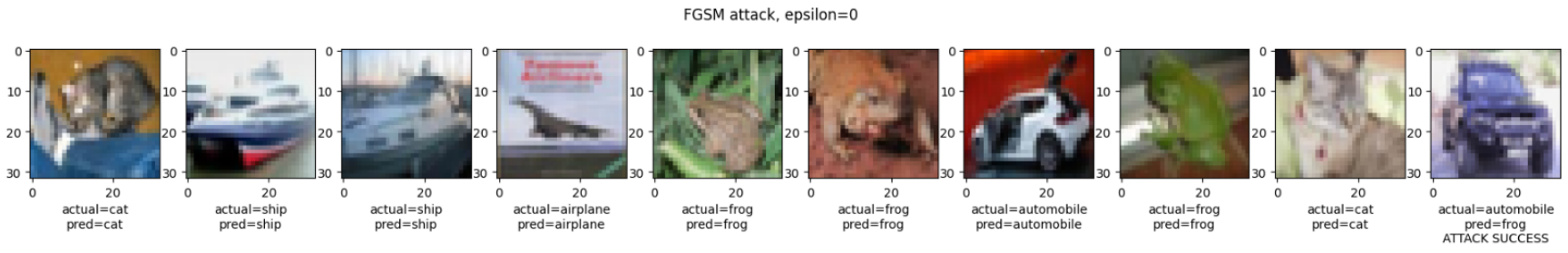


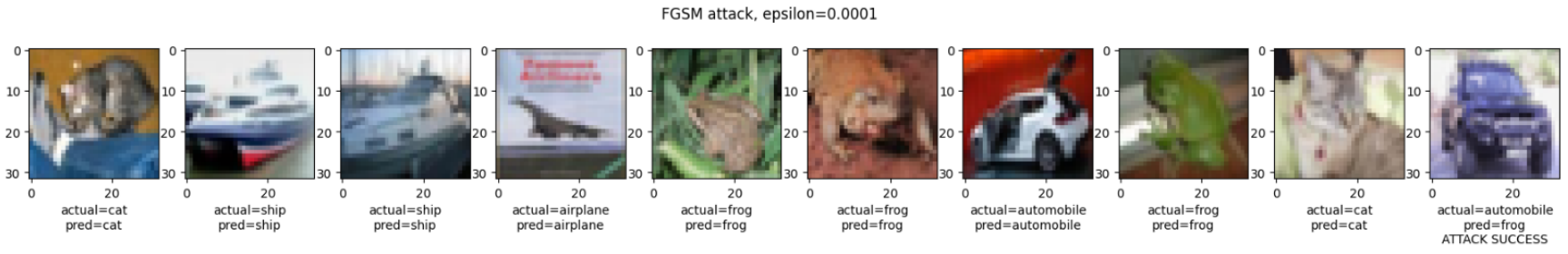
FGSM attack

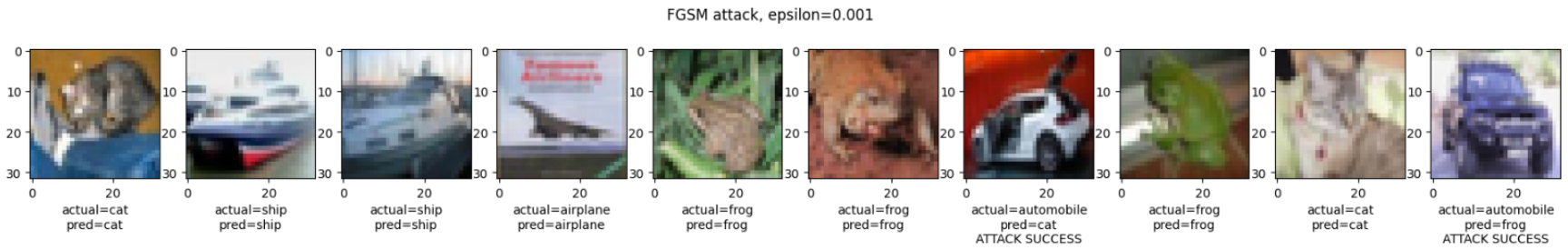
I have attacked the above trained CNN model with my implementation of FGSM. The FGSM attack is a supervised attack which perturbs the input images by the sign of its gradient with respect to the original classification loss function. The hyperparameter $\mathcal{E}$ controls the degree of the perturbation, and hence the tradeoff between attack detection and the rate of success of the attack (measured by whether the attack successfully fools the model to misclassify the input).

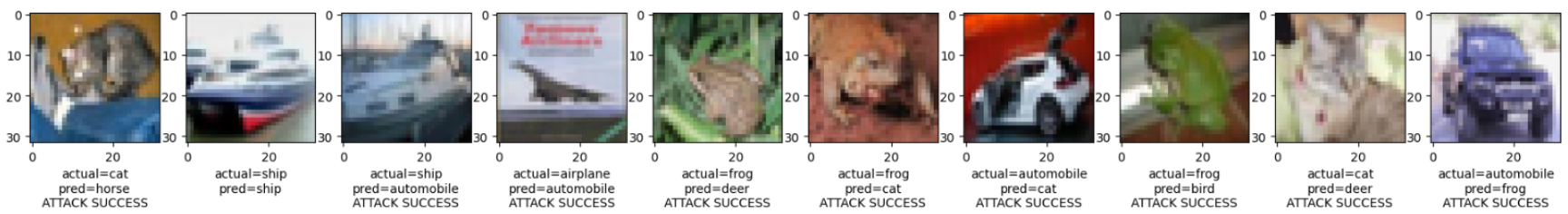
$$x^{adv} = x + \mathcal{E}.sign(\nabla_x J(x, y_{true}))$$

```
for epsilon= 0          after fgsm attack, test accuracy on attacked images:  84.31 %
for epsilon= 0.0001     after fgsm attack, test accuracy on attacked images:  83.72 %
for epsilon= 0.001      after fgsm attack, test accuracy on attacked images:  78.11 %
for epsilon= 0.01       after fgsm attack, test accuracy on attacked images:  26.00 %
for epsilon= 0.1        after fgsm attack, test accuracy on attacked images:   0.54 %
for epsilon= 1          after fgsm attack, test accuracy on attacked images:   7.69 %
```









.

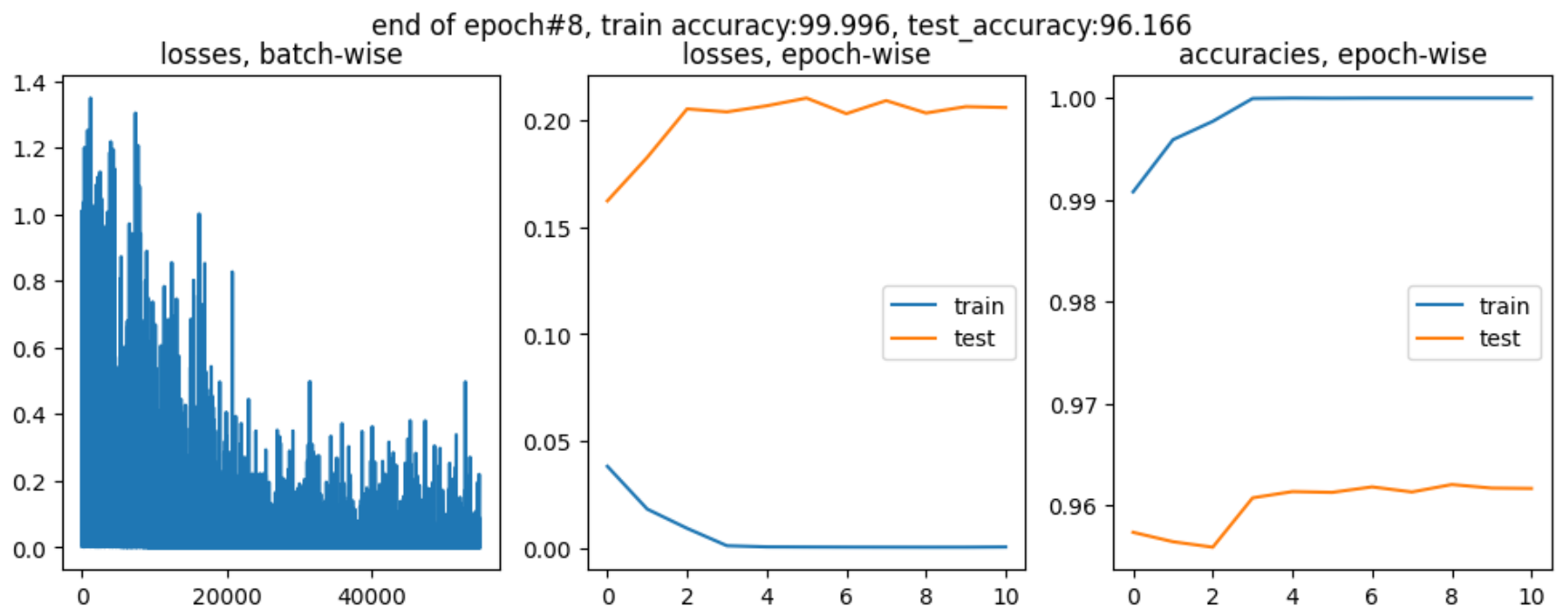
Q1 part 2 - two more classifiers, two more attacks

ResNet34 and **VGG11_bn** are some popular deep learning architectures that have been proven to perform well on various image classification tasks. These architectures are lighter to run than some of their heavier counterparts and offer a very good amount of performance for a low computational cost. They utilize tricks such as skip connections and stacked convolutions in order to reduce the number of parameters in the deep learning model and to push the depth further.

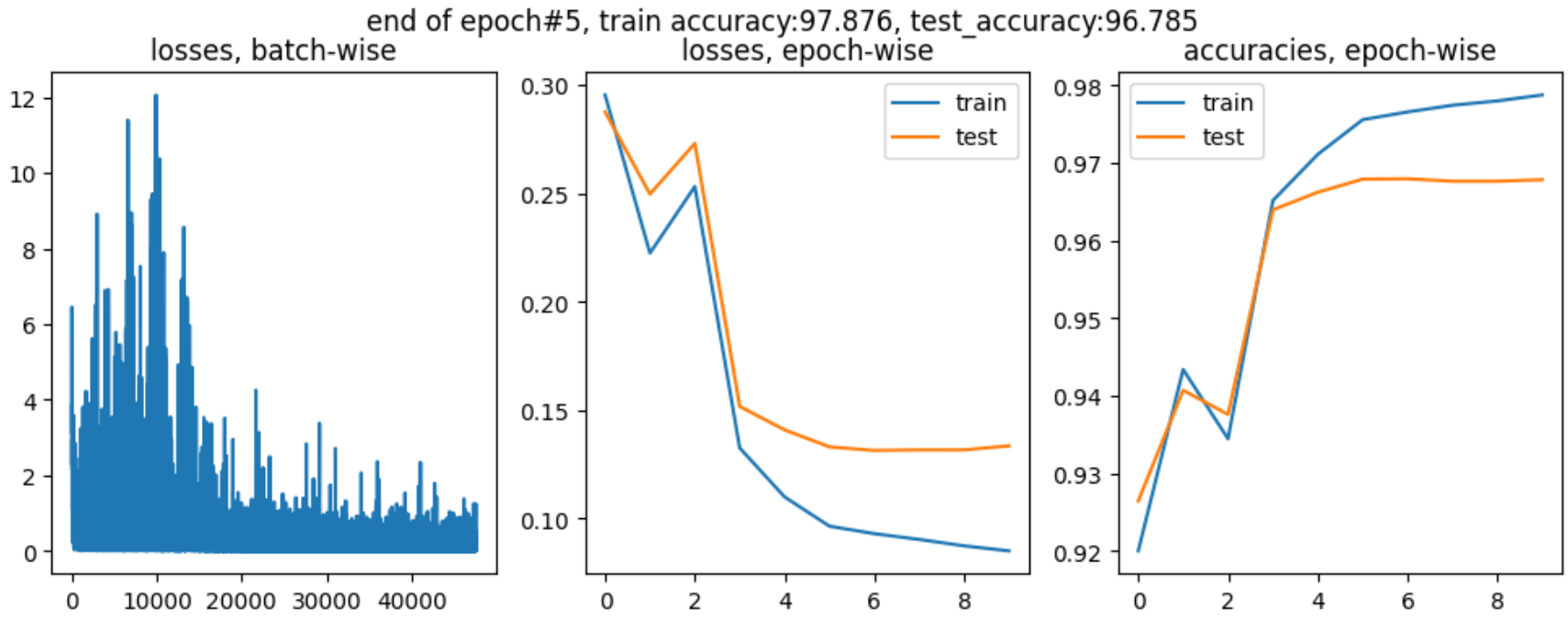
Overall, both ResNet34 and VGG11_bn are good choices for the SVHN dataset because they are well-suited to handle the low-resolution images with significant variations in orientation, scale, and lighting conditions.

I have attacked screenshots of the training of both classifiers on SVHN.

VGG11_bn training on SVHN



ResNet34 training on SVHN



The attacks I chose were:

1. **PGD** attack
2. **DeepFool** attack

PGD attack

Let x be the original input, y be the true label, and $f(x)$ be the output of the model.

The PGD attack starts by initializing the perturbed input x' as follows:

$$x' = x + \eta$$

where η is a small perturbation that is added to the input in the direction of the gradient of the loss function. The PGD attack then iteratively updates the perturbed input as follows:

$$x'_0 = x$$

$$x'_{t+1} = \Pi x + \epsilon(x'_t + \alpha \cdot sign(\nabla_x J(f(x'_t), y)))$$

where t is the iteration number, $\Pi_{x+\epsilon}$ is the projection operator that ensures that the perturbed input remains within a distance of ϵ from the original input x, α is the step size, $J(f(x), y)$ is the loss function, and $sign(\nabla_x J(f(x), y))$ is the sign of the gradient of the loss function with respect to the input.

The PGD attack terminates when the perturbed input is misclassified by the model, or the maximum number of iterations is reached.

DeepFool attack

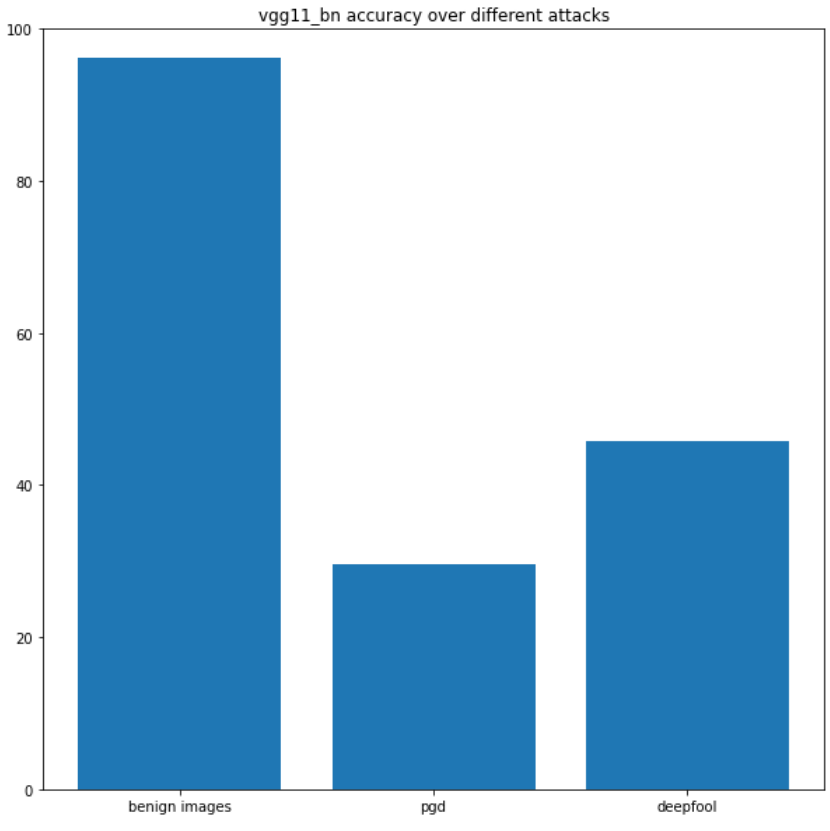
The attack proceeds as follows:

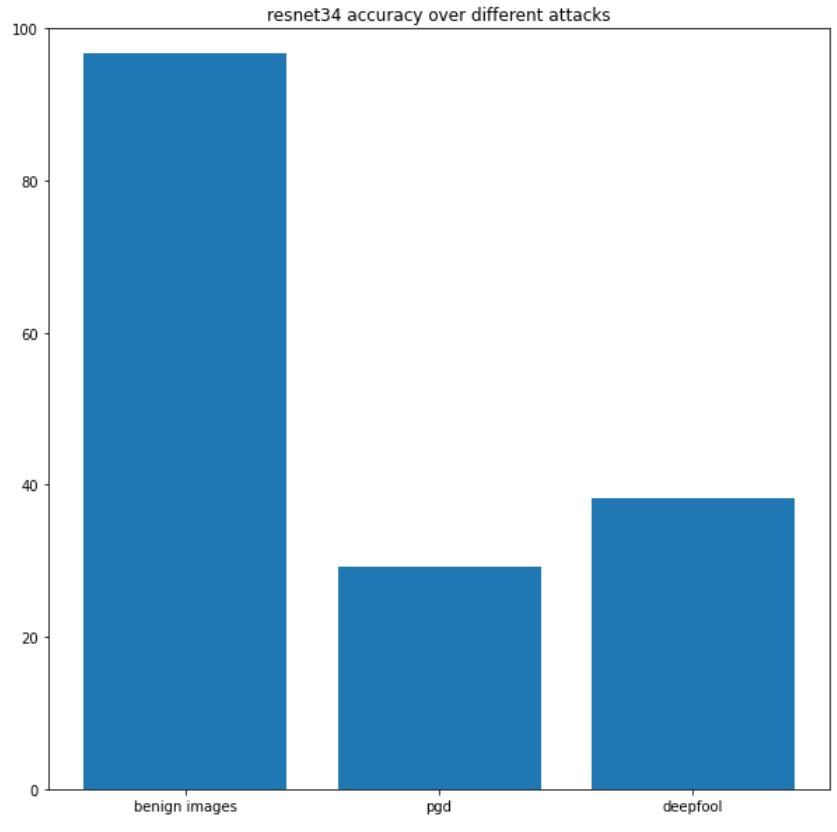
1. Given an input x and its true label y, initialize the perturbed input x' as x.
2. Compute the gradient of the model's output with respect to the input and find the index of the neuron that has the maximum gradient.
3. Compute the minimum perturbation needed to move the input across the decision boundary to a different class.
4. Update the perturbed input x' by adding the computed perturbation r to it.
5. Repeat steps 2-4 until the input x' is misclassified by the model.

Here, r is the smallest perturbation needed to move x across the decision boundary to a different class. It is computed by finding the index of the class, other than the true class y, that is closest to the decision boundary and then computing the minimum perturbation in the direction of the gradient of the model's output.

RESULTS OF BOTH ATTACKS

Attack on ResNet34 **Attack on VGG11_bn**

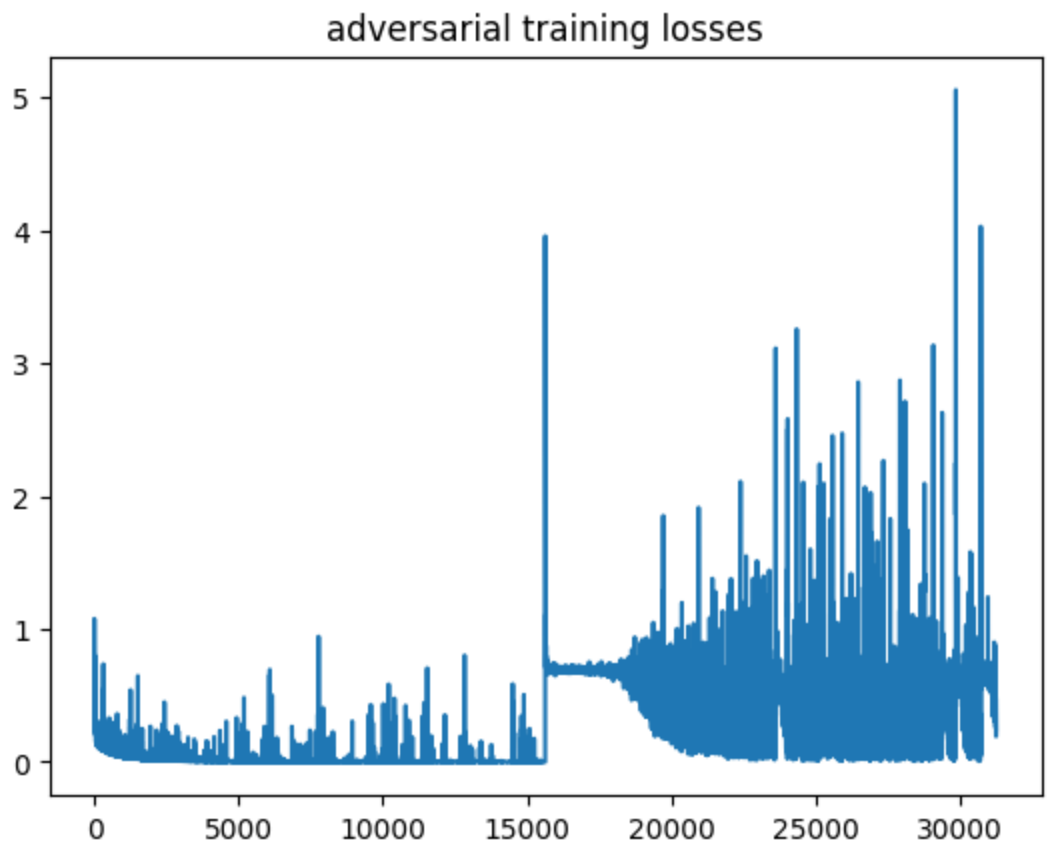




Q1 Part 3 - training a discriminator to distinguish between adversarially perturbed samples and benign

The victim model and dataset will be the initially trained CNN with Autoaugment, and the CIFAR10 dataset. The discriminator is a network with similar architecture as the CNN, with the difference of different classifier layer dimensions for distinguishing between positive (adversarially perturebd) and benign (unaffected) samples.

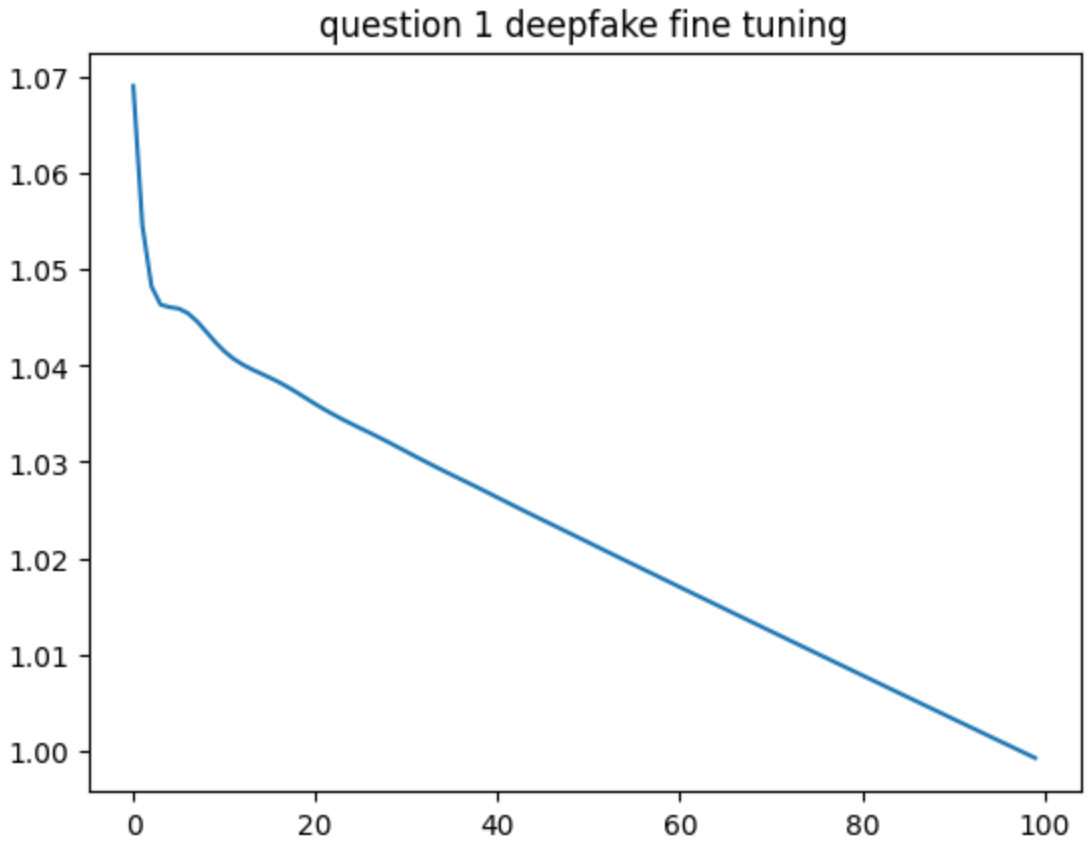
I have executed an FGSM attack on the dataset with $\epsilon = 0.01$ and done adversarial training of the discriminator in order to show its efficacy as a detector against adversarial samples.




```
   1  train_detection_accuracy, test_detection_accuracy
 ✓ 0.0s
(0.89374, 0.8895)
```

Question 2 -fine tuning onto deepfakes

I have finetuned the model in q1(iii) as a discriminator between deepfakes which I have constructed myself. They are provided in the zip file.




```
100%|██████████| 1/1 [00:00<00:00,  4.18it/s]
100%|██████████| 1/1 [00:00<00:00,  2.74it/s]

1.0 0.9803921568627451
```

The train accuracy is 100% and the test accuracy is 98% which means that the detector is excellent and the fine tuning is successful.

References

[1] AutoAugment: Learning Policies from Data