In-class HW: Implementing and Visualizing Adversarial Examples

Patrick Hurley

Prestin Bell

1. **Model Setup**:
    - Use an existing CNN model or modify the provided CNN architecture to classify the MNIST dataset. The model should include at least two convolutional layers and one fully connected layer.

2. **Adversarial Example Generation**:
    - Implement a method to generate adversarial examples. Start with the Fast Gradient Sign Method (FGSM), then explore more complex methods like Projected Gradient Descent (PGD).
    - Apply these methods to create adversarial examples from the test portion of the MNIST dataset.

3. **Model Testing and Evaluation**:
    - Evaluate the original model's performance on both clean and adversarial data.

The parameters we used for step 3 are as follows:

Epochs: 10

Classes: 10

Batch size: 256

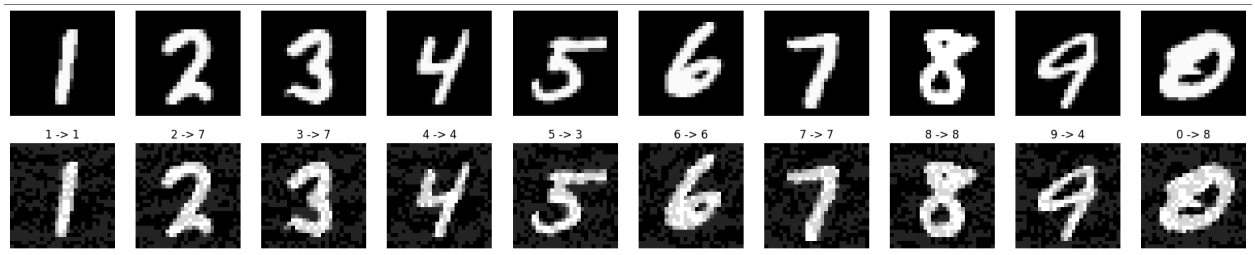
Learning Rate: 0.001

Below is a screenshot of a run with the non-adversarial data:

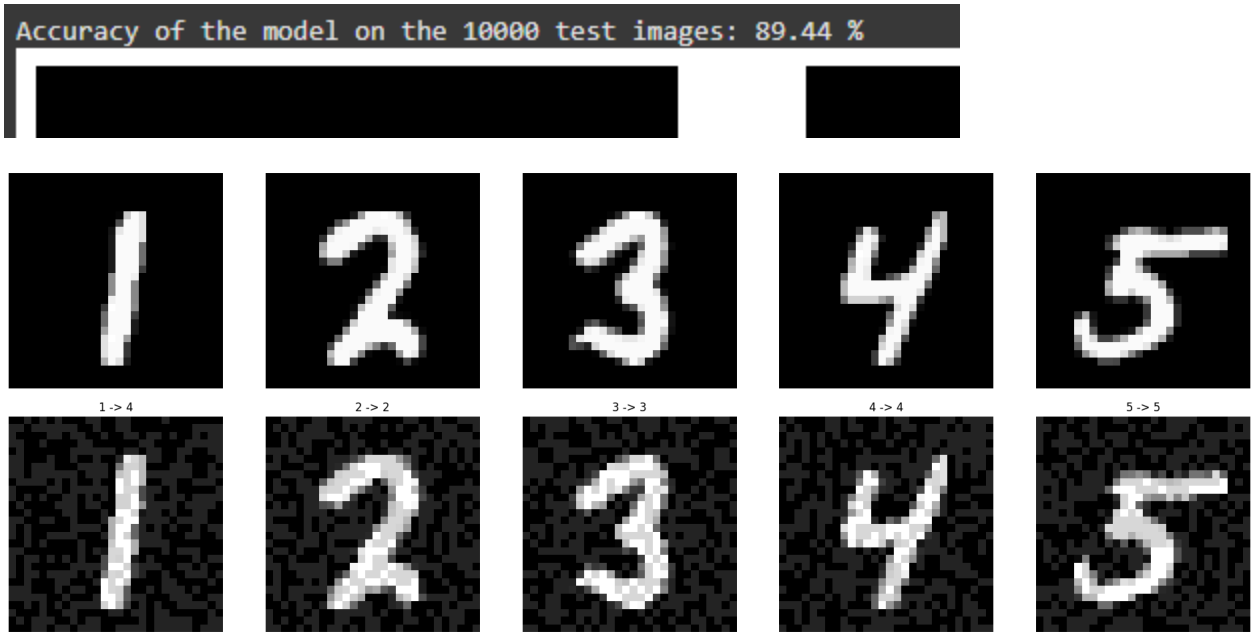
```
Epoch [1/10], Step [0/235], Loss: 2.3257, Accuracy: 6.25%
Epoch [1/10], Step [150/235], Loss: 0.0401, Accuracy: 98.44%
Epoch [2/10], Step [0/235], Loss: 0.0170, Accuracy: 99.22%
Epoch [2/10], Step [150/235], Loss: 0.0577, Accuracy: 98.44%
Epoch [3/10], Step [0/235], Loss: 0.0271, Accuracy: 98.83%
Epoch [3/10], Step [150/235], Loss: 0.0272, Accuracy: 99.22%
Epoch [4/10], Step [0/235], Loss: 0.0217, Accuracy: 99.61%
Epoch [4/10], Step [150/235], Loss: 0.0054, Accuracy: 100.00%
Epoch [5/10], Step [0/235], Loss: 0.0042, Accuracy: 100.00%
Epoch [5/10], Step [150/235], Loss: 0.0052, Accuracy: 99.61%
Epoch [6/10], Step [0/235], Loss: 0.0092, Accuracy: 99.61%
Epoch [6/10], Step [150/235], Loss: 0.0018, Accuracy: 100.00%
Epoch [7/10], Step [0/235], Loss: 0.0140, Accuracy: 99.22%
Epoch [7/10], Step [150/235], Loss: 0.0192, Accuracy: 98.83%
Epoch [8/10], Step [0/235], Loss: 0.0027, Accuracy: 100.00%
Epoch [8/10], Step [150/235], Loss: 0.0089, Accuracy: 99.61%
Epoch [9/10], Step [0/235], Loss: 0.0007, Accuracy: 100.00%
Epoch [9/10], Step [150/235], Loss: 0.0297, Accuracy: 98.44%
Epoch [10/10], Step [0/235], Loss: 0.0028, Accuracy: 100.00%
Epoch [10/10], Step [150/235], Loss: 0.0007, Accuracy: 100.00%
Accuracy of the model on the 10000 test images: 98.78 %
```

Here is an adversarial run example using the FGSM method:

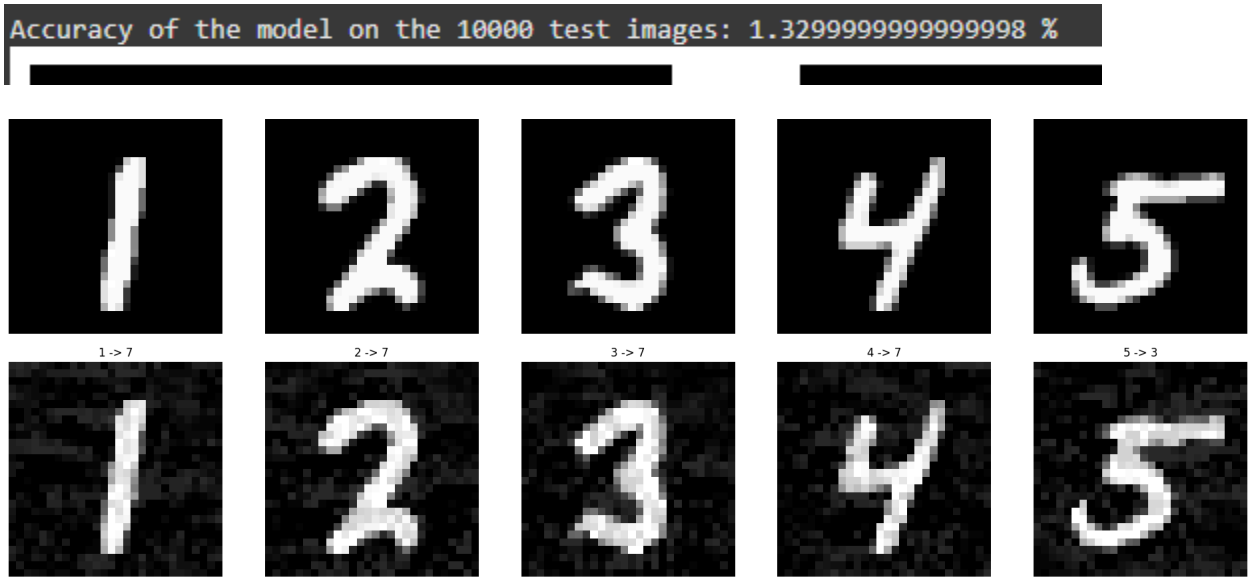
```
Accuracy of the model on the 10000 test images: 30.349999999999998 %
```



Here is an adversarial run example with random noise injection:



Here is an adversarial run example using the PGD Method



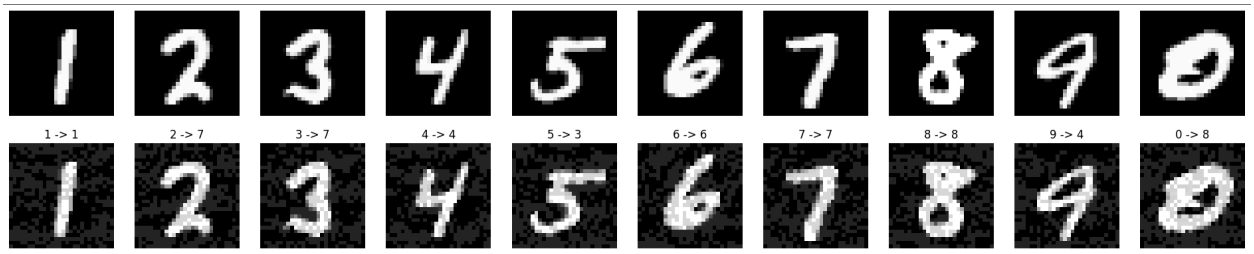
- Calculate and compare accuracy metrics for both sets.

For the nonadversarial data I was getting numbers around 98%. In the screenshot we gotten a run with an accuracy of 98.78%. The FGSM method dropped the accuracy to 30.34% which is over 60% from the nonadversarial test. The random noise injection dropped the accuracy by about 10%. To 89.44% The PGD Method Had destroyed the accuracy of the Learning Model by dropping it down to 1.32%.

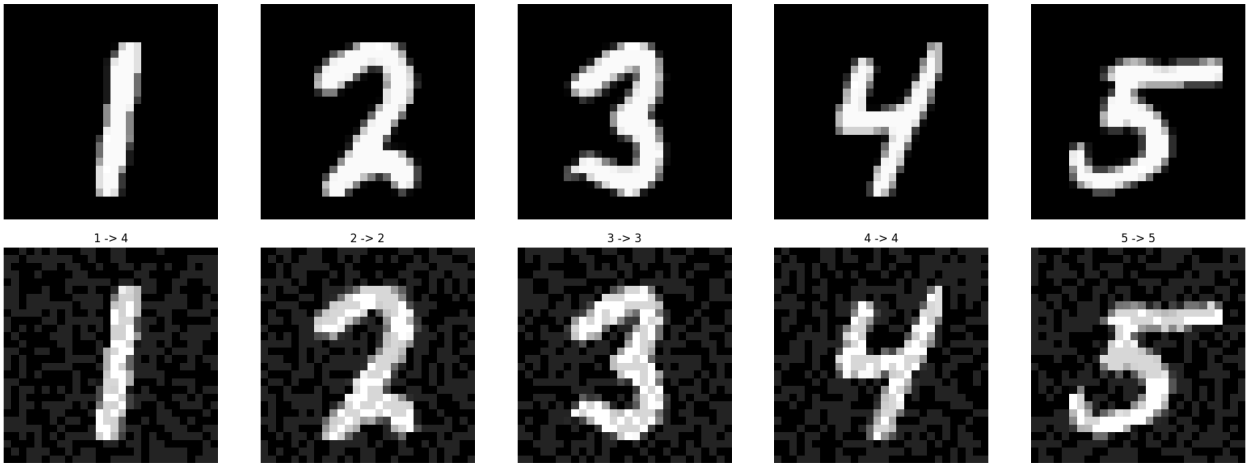
4. **Visualization and Analysis**:

    - Visualize the effects of adversarial modifications on the inputs.
    - Plot the original and adversarial images side-by-side to highlight the perturbations.

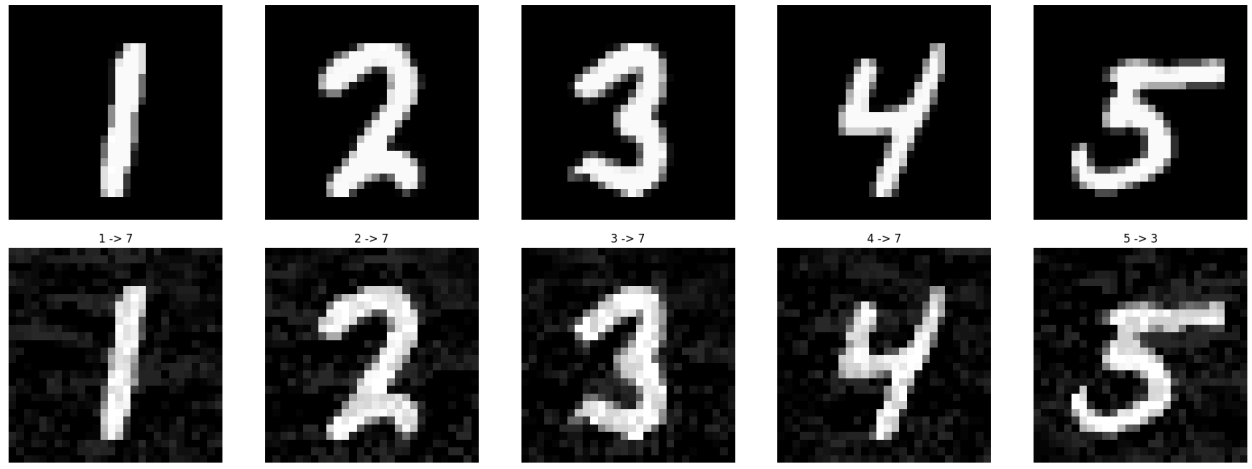
FGSM method:



Random noise injection:

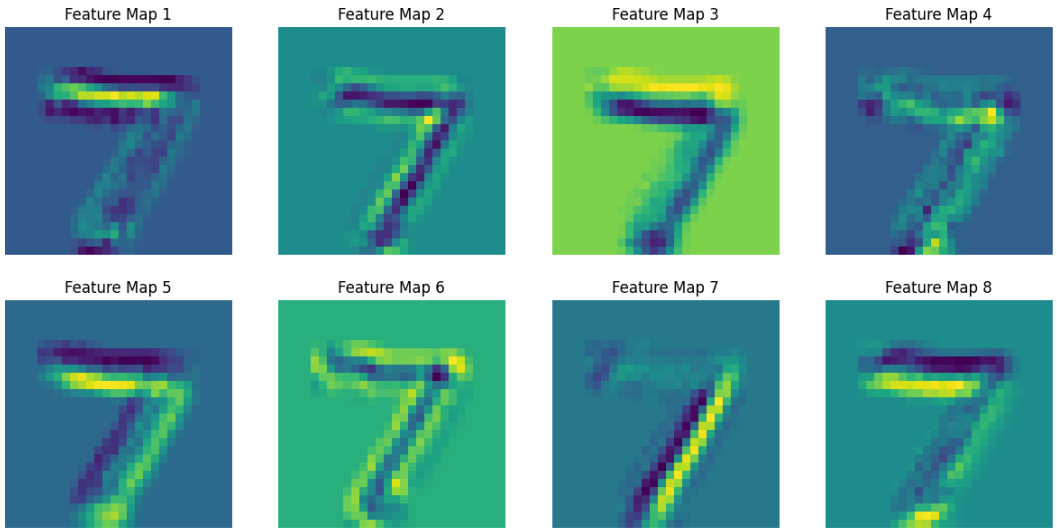


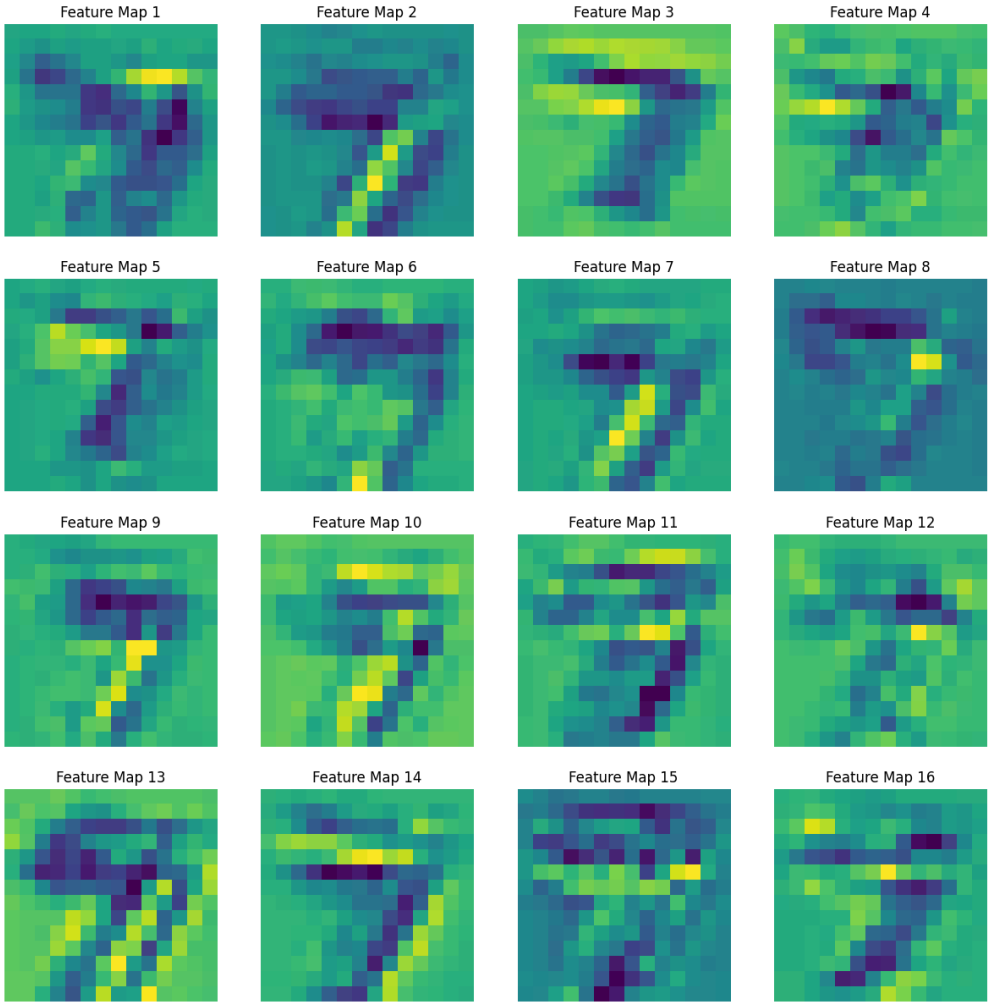
PGD Method:



- Visualize the model's feature maps for both clean and adversarial inputs to observe where the model focuses its attention for each type.

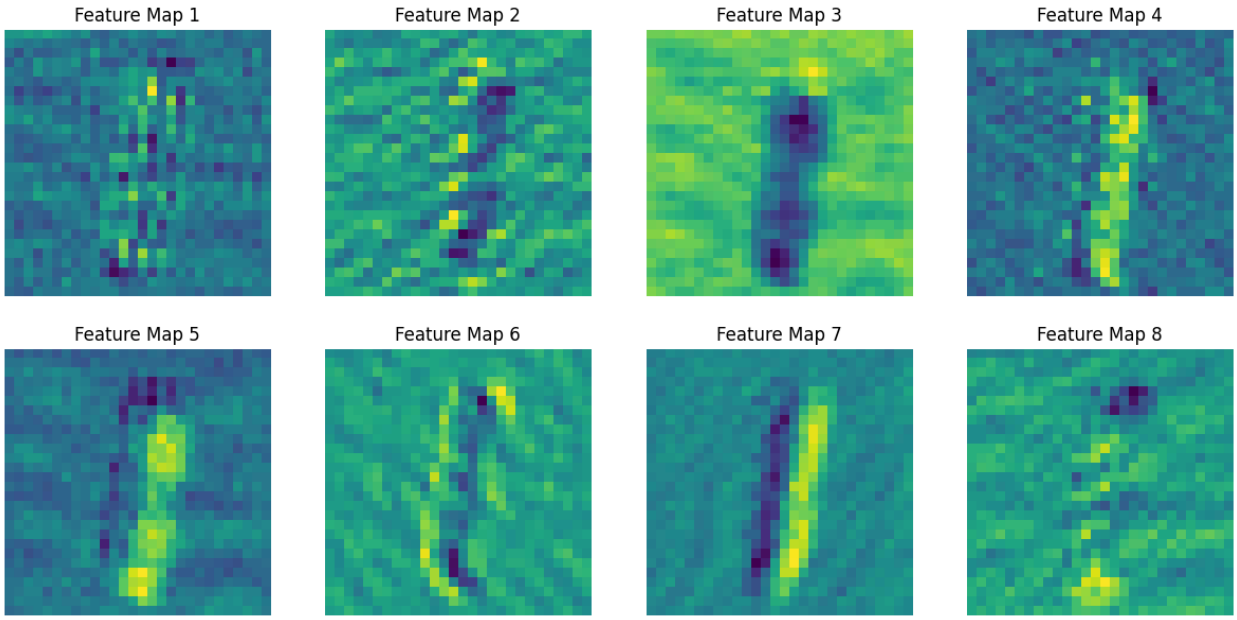
Below is a feature map of the nonadversarial input.

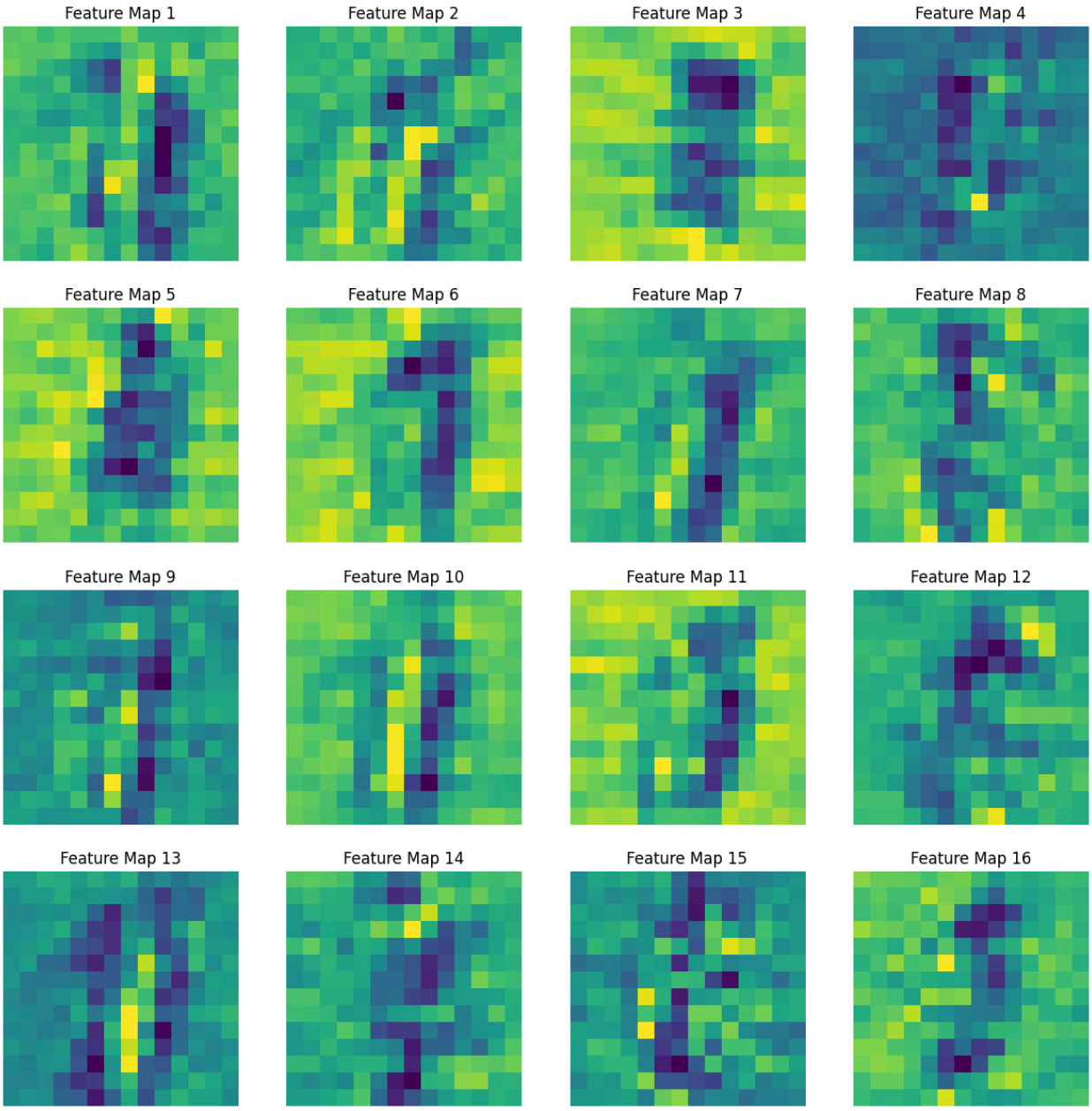




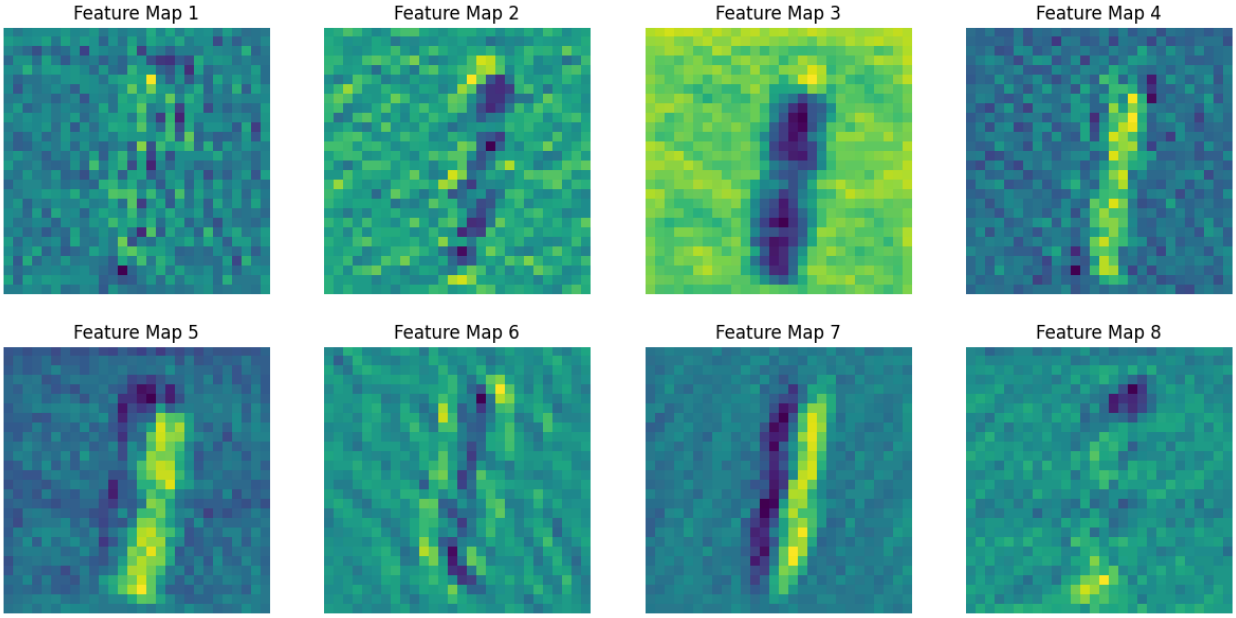
Below is a feature map of the FGSM

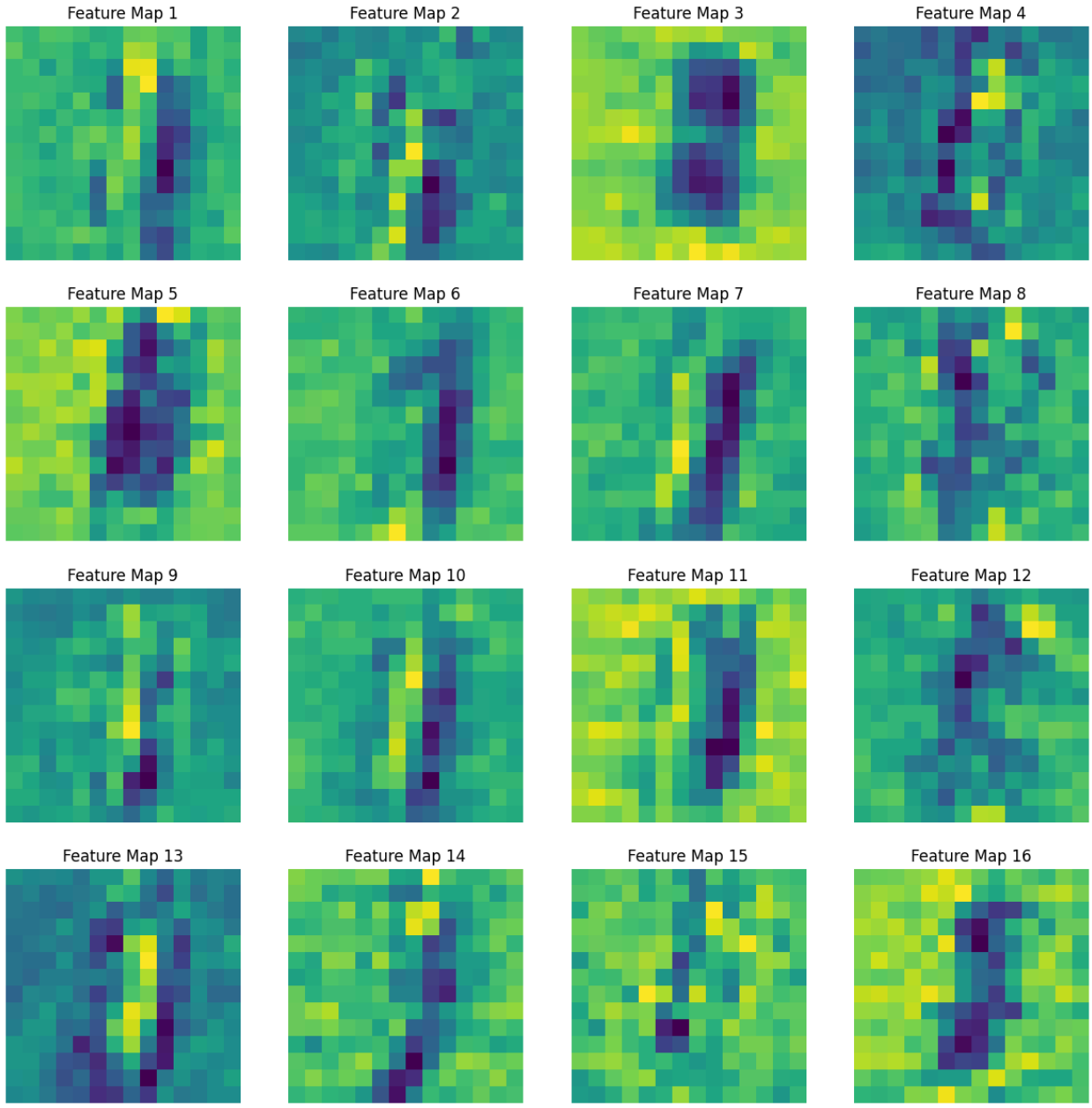
The stepsize is 0.2.


Feature Map 1
Feature Map 2
Feature Map 3
Feature Map 4
Feature Map 5
Feature Map 6
Feature Map 7
Feature Map 8

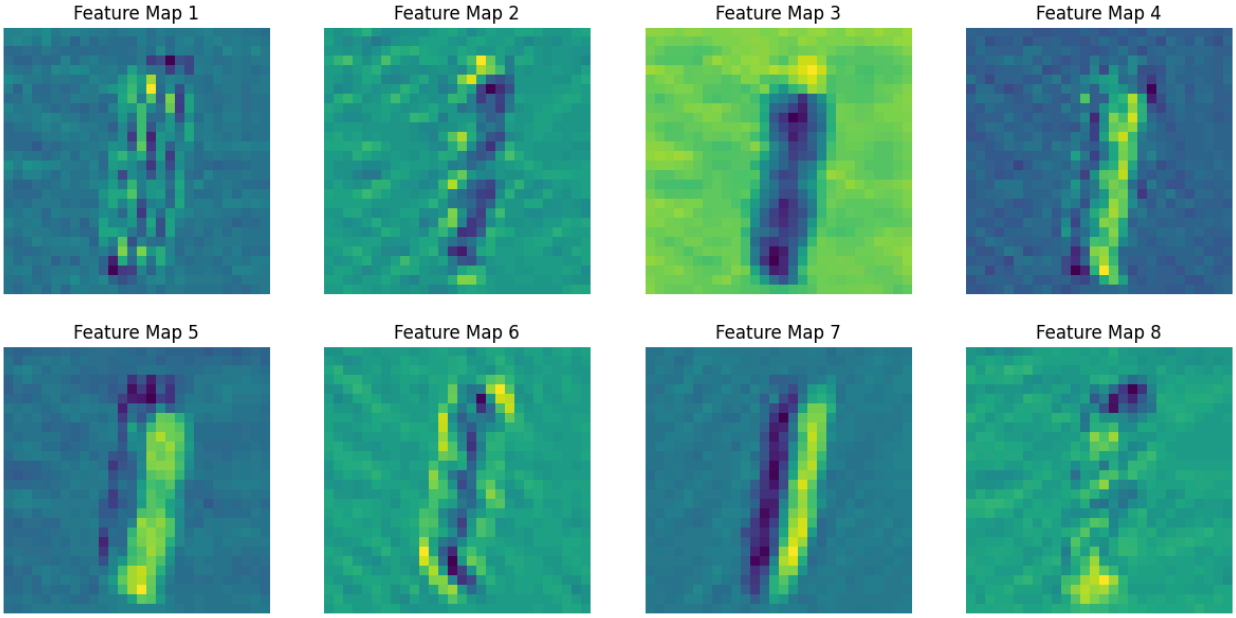


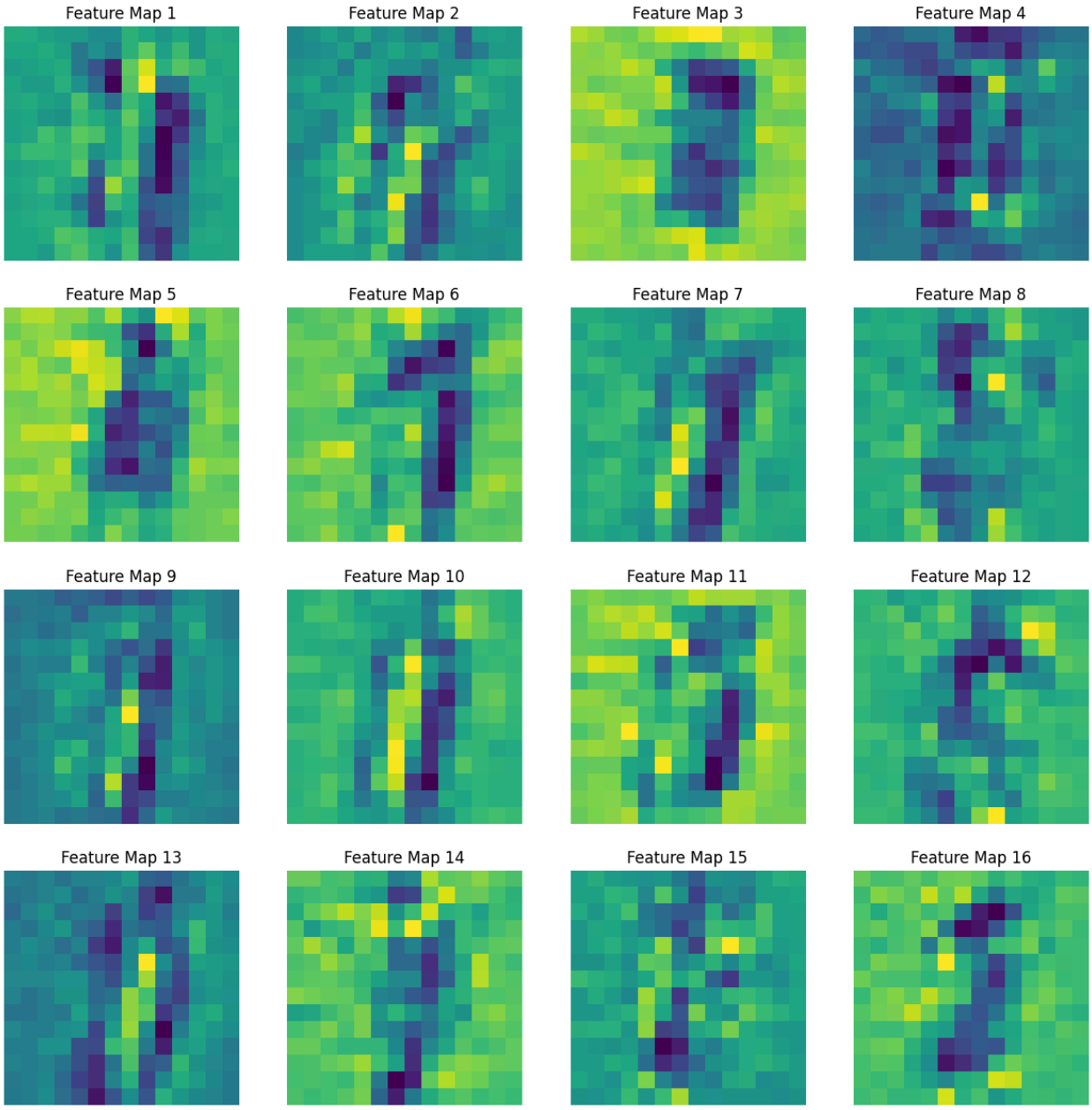
Below is a feature map of the Random Noise injection


Feature Map 1
Feature Map 2
Feature Map 3
Feature Map 4
Feature Map 5
Feature Map 6
Feature Map 7
Feature Map 8



Below is a feature map of the PGD:


Feature Map 1
Feature Map 2
Feature Map 3
Feature Map 4
Feature Map 5
Feature Map 6
Feature Map 7
Feature Map 8



5. **Hyperparameter Impact**:
    - Experiment with different values of the perturbation magnitude/step size (0.05, 0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4) in the FGSM and observe how it affects the success of adversarial attacks.
    - Discuss the trade-offs between perturbation magnitude and perceptibility.

Epochs: 10

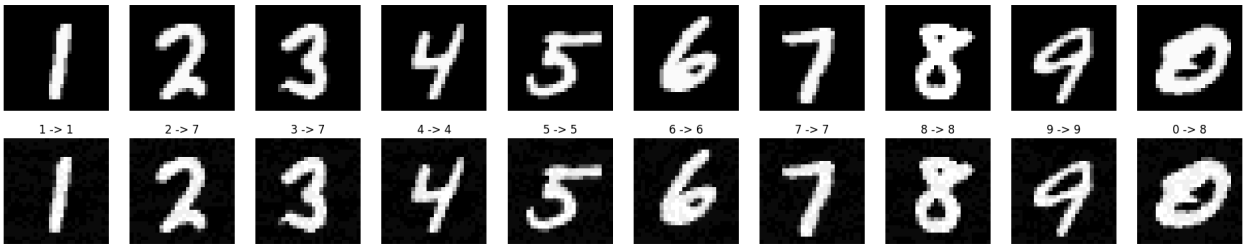
Classes: 10

Batch size: 256

Learning Rate: 0.001

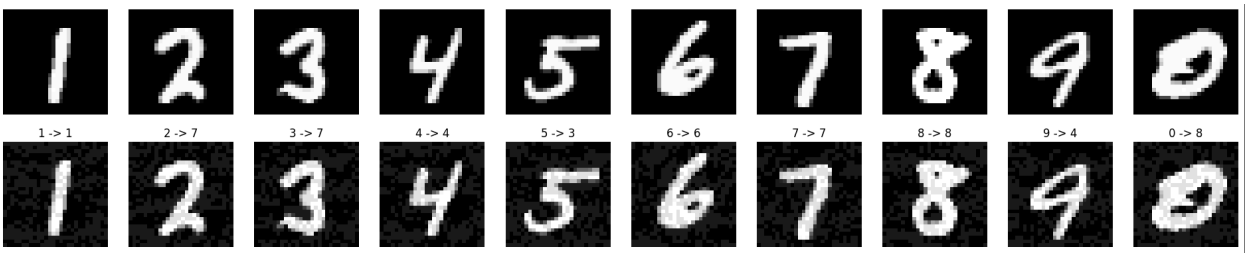
StepSize 0.05

```
Accuracy of the model on the 10000 test images: 81.43 %
```



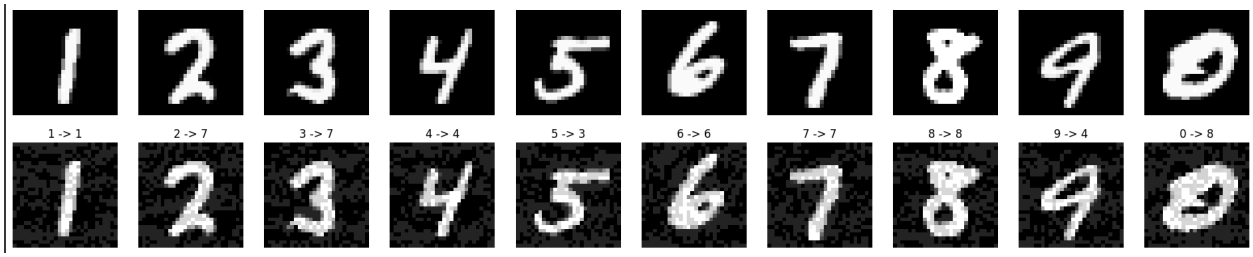
StepSize 0.1

```
Accuracy of the model on the 10000 test images: 51.13999999999999 %
```



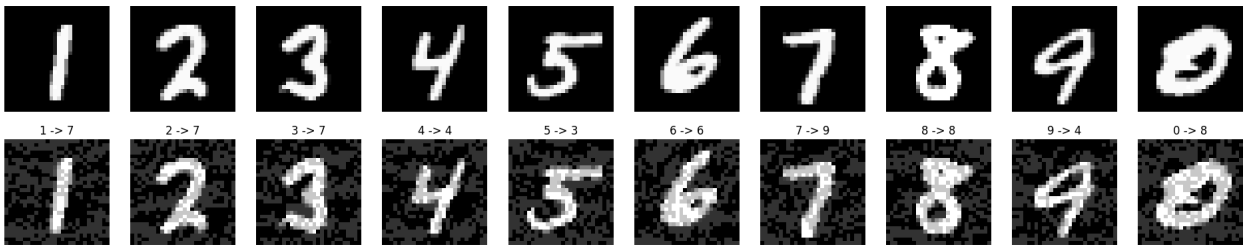
StepSize 0.15

```
Accuracy of the model on the 10000 test images: 30.349999999999998 %
```



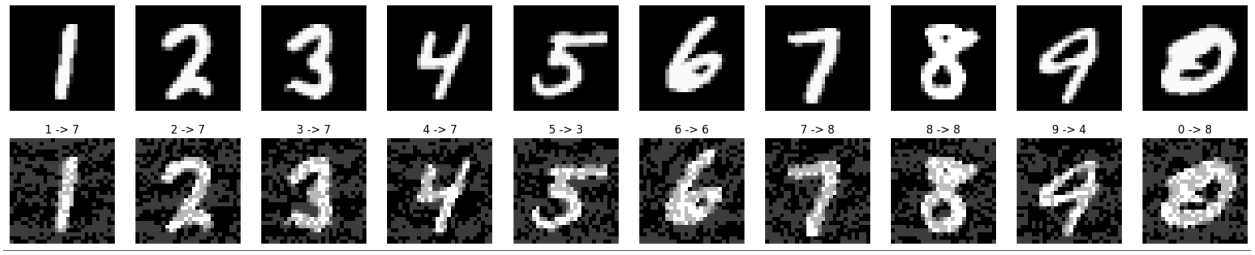
StepSize 0.2

```
Accuracy of the model on the 10000 test images: 18.44 %
```



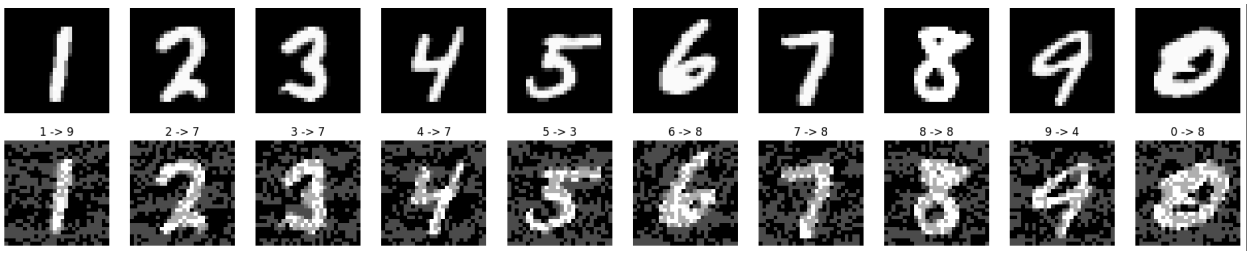
StepSize 0.25

```
Accuracy of the model on the 10000 test images: 13.13 %
```



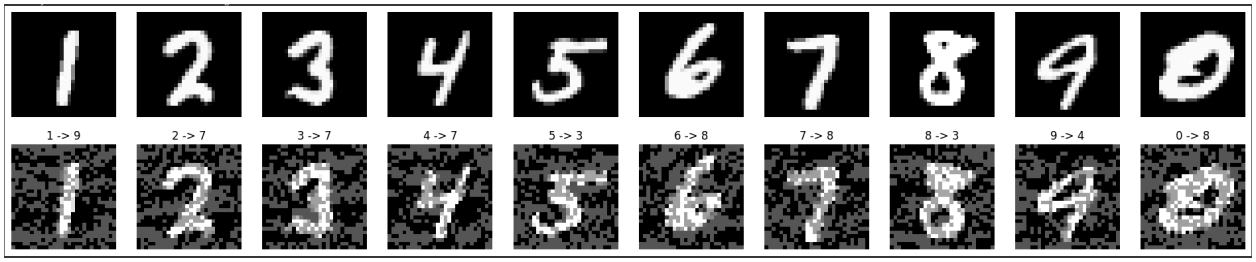
StepSize 0.3

```
Accuracy of the model on the 10000 test images: 10.24 %
```

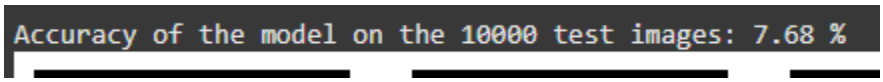


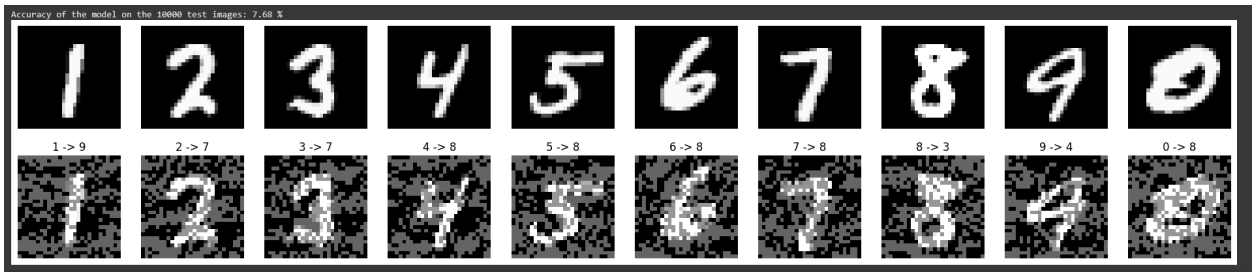
StepSize 0.35

```
Accuracy of the model on the 10000 test images: 8.690000000000001 %
```



StepSize 0.4





It seems LIke the higher the perturbation magnitude/step size, the less perceptible they will become. Values in the negatives also seem to behave similarly to positive values. The closer the value is to Zero, the more perceptible the image will be, and the farther from zero the less perceptible it will become. Even in extreme examples like -2,1 and 1, where it is impossible to determine what the numbers are to humans, somehow the machine can determine what the numbers are at low rates.