EE6132: Assignment 2
Convolutional Neural Networks
Date: October 28, 2021

Contents

---

1 Brief Description of the Code

1.1 Libraries Used

The libraries used to run the python code are described below:

- numpy
- matplotlib.pyplot
- tqdm
- time
- os
- sklearn
- torch
- torchvision

1.2 Functions Defined

Firstly, the network is defined as a class CNN.
There are two networks described in the code, one with Batch Normalization and the other without.
The constructor of the network describes the structure of the network while the function *forward* establishes this relationship and also performs the forward pass.
Besides this, there are three main functions defined. One to test, one to train and lastly one to run the CNN for the required tasks.

Run CNN is like a wrapper function which implements training and testing if the required flags are set and saves the model.
It also hosts 7 different functions inside it, which together perform the required tasks in the assignment. The different functions it hosts are,

- Random Plots: To plot random test images along with their predicted class
- Plot Filters: This plots the filters in the convolutional layers.
- Visualize Activations: This plots the feature maps obtained from the convolutional layers before the activation function kicks in.
- Occlusion Exp: This performs the occlusion experiment on different test images.
- Non Targeted Attack
- Targeted Attack
- Add Noise

2 MNIST Classification using CNN

2.1 Training Statistics

Training Hyperparameters
I tried various combinations of hyperparameters on the network but later settled on the following.
Instead of cross entropy loss over the softmax function on the output layer, I implemented NLL loss on log softmax.
I read several papers suggesting that this helps in faster training and better results and hence with this arrangement.
However, as the NLL loss is implemented over the log of the softmax function, the functionality of the cross entropy loss is preserved.

- Learning Rate = 0.001
- Batch Size = 64
- Number of Epochs = 10
- Gradient Descent Algorithm = Adam
- Loss function = Negative Log Likelihood loss implemented on log softmax

Additionally, for more efficient training, there was some pre-processing done on the MNIST dataset. They were all normalized, i.e., mean subtracted and divided by standard deviation. **Training and Testing Plots**

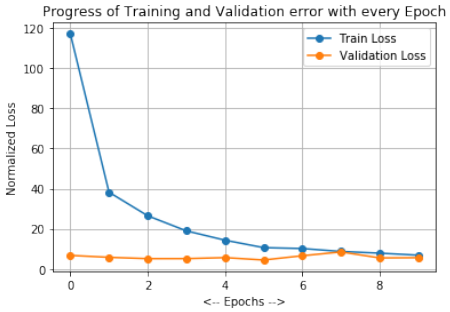

Figure 1: Train and Test loss plot across epochs

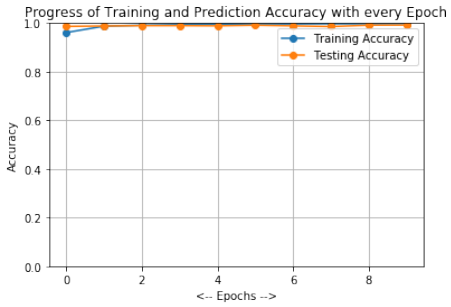

Figure 2: Train and Test accuracy plot across epochs

	precision	recall	f1-score	support
0 - zero	1.00	0.99	0.99	980
1 - one	0.99	1.00	1.00	1135
2 - two	0.99	0.99	0.99	1032
3 - three	0.96	1.00	0.98	1010
4 - four	0.99	0.99	0.99	982
5 - five	1.00	0.97	0.98	892
6 - six	0.99	0.99	0.99	958
7 - seven	1.00	0.99	0.99	1028
8 - eight	0.99	0.99	0.99	974
9 - nine	0.99	0.98	0.98	1009
accuracy			0.99	10000
macro avg	0.99	0.99	0.99	10000
weighted avg	0.99	0.99	0.99	10000

Figure 3: Statistics on the predictions after training

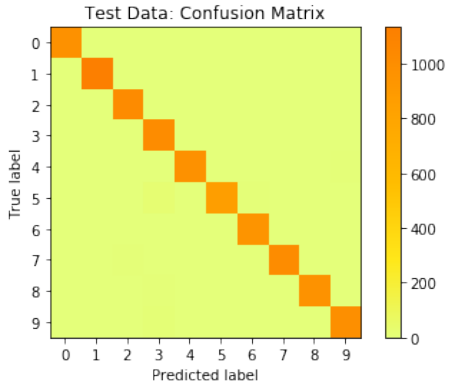

Figure 4: Confusion Matrix on the predictions after training

This experiment was repeated several times and the average prediction accuracy after training hovered about 99 %.
The model above had a final accuracy of 99.16%.
The model also had the corresponding average accuracies across epochs.

- Average prediction accuracy across epochs: 0.9888999999999999
- Average training accuracy across epochs: 0.9912483333333333

We see that the accuracies are very high when compared to the MLP experiment where the highest accuracy obtained was 98.9 %. Moreover, using the Adam Gradient Descent, the accuracies are much higher for the convolutional neural network starting from the first epoch! Thus, they prove to be much faster to train and more suitable for images.

Plots of Random Test Images:

The plots of random test images spread across all digits is shown below. The same test images seen here were used later for the adversarial and occlusion experiments.

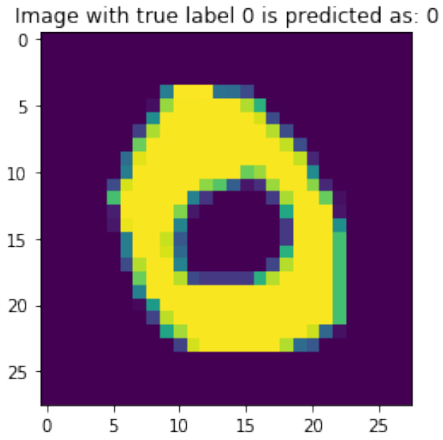

Figure 5: Random Test Image with true label 0

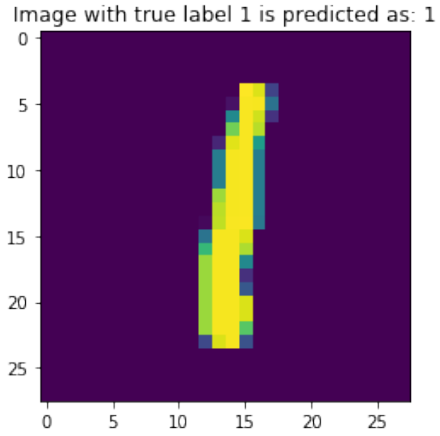

Figure 6: Random Test Image with true label 1

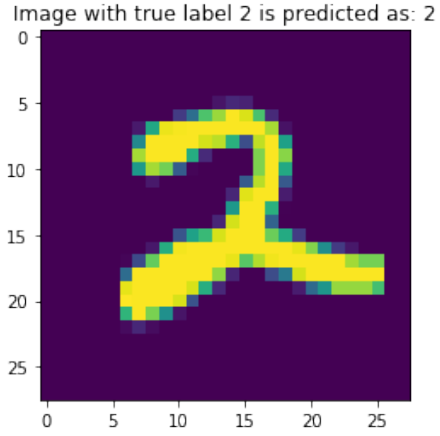

Figure 7: Random Test Image with true label 2

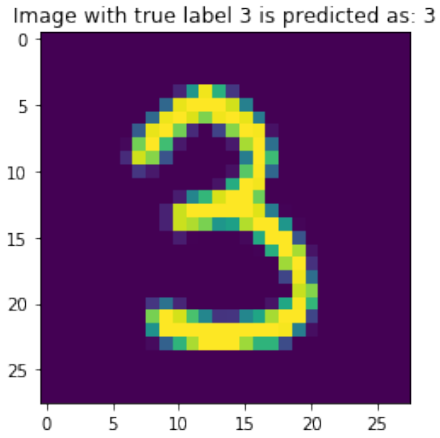

Figure 8: Random Test Image with true label 3

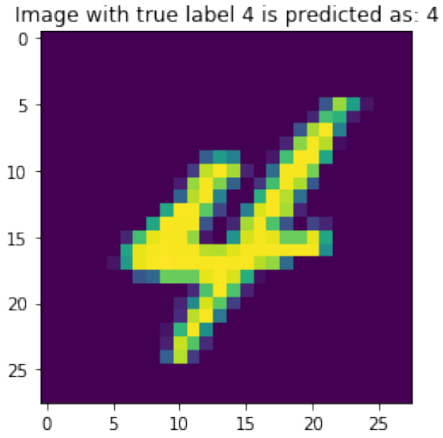

Figure 9: Random Test Image with true label 4

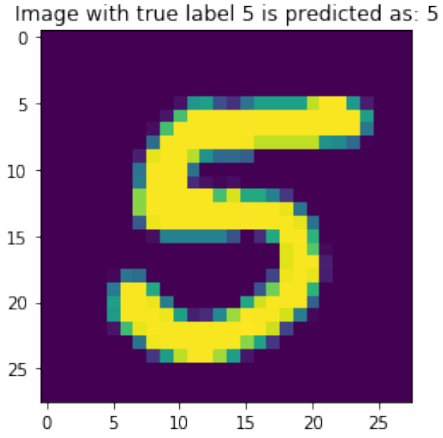

Figure 10: Random Test Image with true label 5

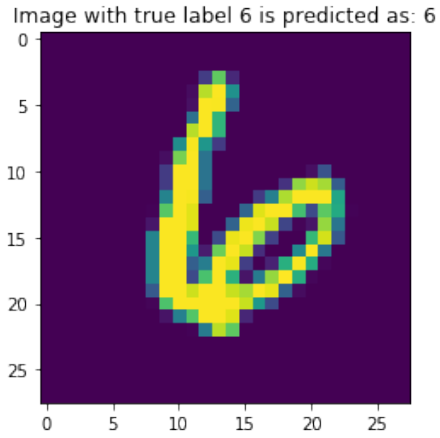

Figure 11: Random Test Image with true label 6

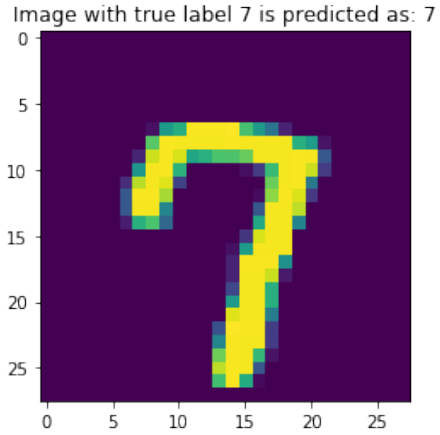

Figure 12: Random Test Image with true label 7

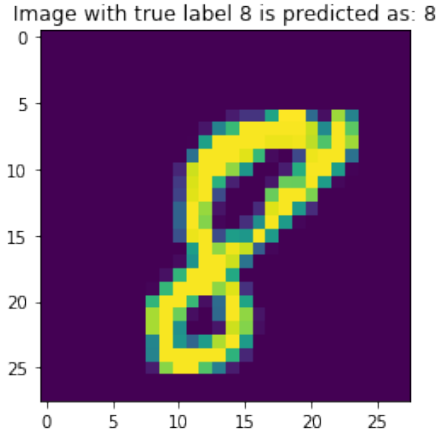

Figure 13: Random Test Image with true label 8

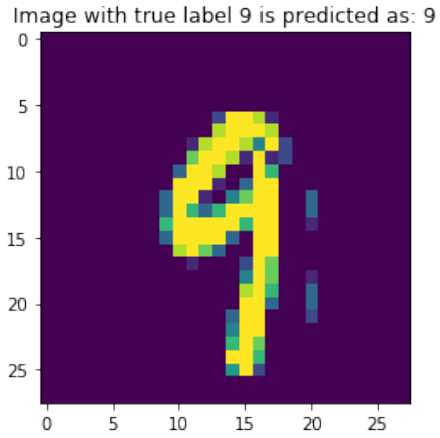

Figure 14: Random Test Image with true label 9

2.2 Network Architecture

The Architecture of the Network
I'm writing this below as 6 different layers although in the programming part, I've clubbed the convolutional layer and the max pooling into a single layer.

- A convolutional layer comprising 32 filters of dimensions 3x3 with stride 1 and zero padding of 1.
- A 2x2 max pool layer with stride 2.
- A convolutional layer comprising 32 filters of dimensions 3x3 with stride 1 and zero padding of 1.
- A 2x2 max pool layer with stride 2.
- A fully connected layer with 500 neurons.
- A fully connected layer with 10 neurons.

Dimensions of the Input and Output at the above layers
The dimensions are expressed as WxHxD wherever necessary. The Input and Output are for the layers as a whole.

- Conv1: Input Size = 28x28x1, Output Size = 28x28x32
- Maxpool1: Input Size = 28x28x32, Output Size = 14x14x32
- Conv2: Input Size = 14x14x32, Output Size = 14x14x32
- Maxpool2: Input Size = 14x14x32, Output Size = 7x7x32
- FC1: Input Size = (32x7x7) = 1568, Output Size = 500
- FC2: Input Size = 500, Output Size = 10

Number of Parameters in the network
Let us count the number of weights and biases required in a layer by layer fashion. The max pooling layers don't involve any parameters and thus can be omitted from this calculation.

- Conv1: Weights: 32x(3x3) ,Biases = 32, Total = 320
- Conv2: Weights: 32x(3x3x32), Biases = 32, Total = 9248
- FC1: Weights: 1568x500, Biases = 500, Total = 784500
- FC2: Weights: 500x10, Biases = 10, Total 5010

Therefore, total number of parameters = 320 + 9248 + 784500 + 5010 = 799078 parameters.
Out of this, 9568 or roughly just **1.12**% of the total parameters are present in the Convolutional Layers and 789510 or a whopping **98.80** % of the total parameters are present in the fully connected layers.

Number of Neurons in the network
As in the previous case, let us carefully compute this in a layer by layer fashion and also omit the max pooling layers from the calculation.

- Conv1: 32x3x3 = 288
- Conv2: 32x3x3x32 = 9216
- FC1: 500
- FC2: 10

Therefore, the total number of neurons = 288+9216+500+10 = 10014 neurons.
Out of this a whopping 9504 or **94.91** % of the total neurons are present in the Convolutional Layers and 510 or **5.09** % of the total neurons are present in the Fully Connected layers.

Thus, this confirms what we saw in class, roughly $\sim$ 5% of the parameters and $\sim$ 95% of the neurons are present in the convolutional layers while $\sim$ 95% of the parameters and $\sim$ 5% of the neurons are present in the fully connected layers.

2.3 Batch Normalization

For this part of the experiment, batch normalization was performed after the max pooling layers.All the other training parameters used before were maintained for a fair comparison. **Training and Testing Plots**

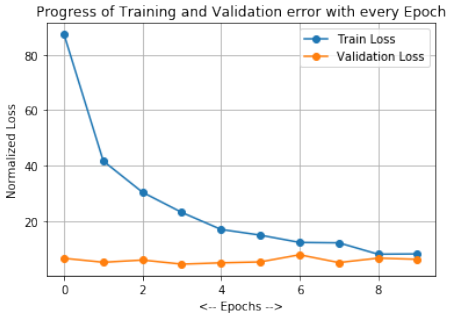

Figure 15: Train and Test loss plot across epochs

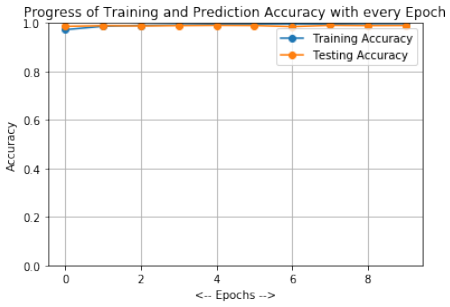

Figure 16: Train and Test accuracy plot across epochs

	precision	recall	f1-score	support
0 - zero	0.99	1.00	0.99	980
1 - one	0.99	1.00	0.99	1135
2 - two	0.99	0.99	0.99	1032
3 - three	0.99	1.00	0.99	1010
4 - four	0.99	0.99	0.99	982
5 - five	0.99	0.98	0.99	892
6 - six	0.99	0.99	0.99	958
7 - seven	0.99	0.99	0.99	1028
8 - eight	0.98	1.00	0.99	974
9 - nine	0.99	0.97	0.98	1009
accuracy			0.99	10000
macro avg	0.99	0.99	0.99	10000
weighted avg	0.99	0.99	0.99	10000

Figure 17: Statistics on the predictions after training

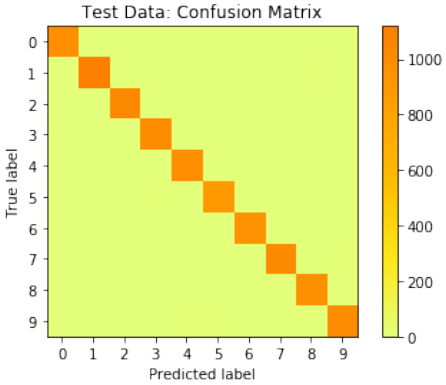

Figure 18: Confusion Matrix on the predictions after training

This experiment was repeated several times and the average prediction accuracy after training hovered about 99 % just like the previous case.
The model above had a final accuracy of 99.09%.
The model also had the corresponding average accuracies across epochs.

- Average prediction accuracy across epochs: 0.9895099999999999
- Average training accuracy across epochs: 0.99179

Comparing the models with and without Batch Normalization:

- **Accuracy metrics:** Both the models had similar statistics with respect to accuracy and training and testing losses.In this case, we observe that the batch normalized model has a marginally better average training loss and training accuracy, however, it is just a coincidence and not a conclusive effect as it wasn't observed every time the experiment was repeated. Shown below is a plot which illustrates this. Additionally, this could also be because we use the Adam Gradient Descent which could overpower the advantages of Batch Normalization or due to the pre-processing normalization done on the training data which could lead to a lesser internal covariance shift thereby reducing the need for Batch Normalization.

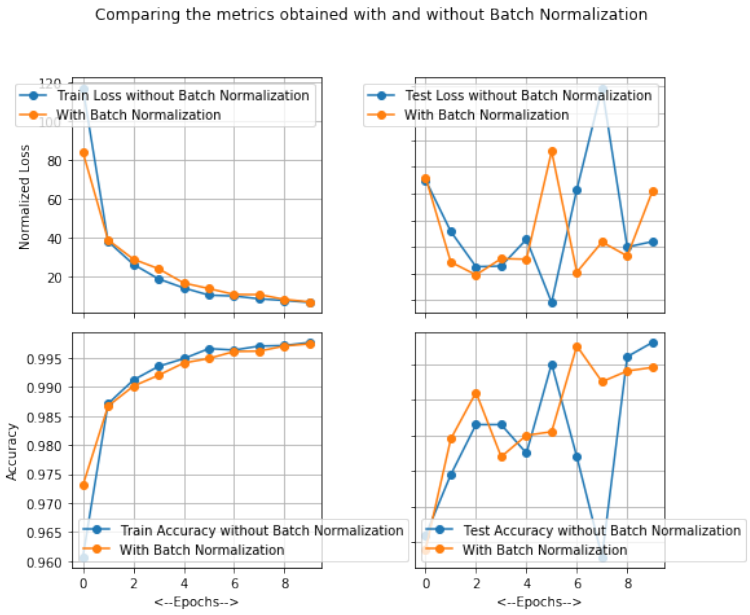

Figure 19: Comparing the two models

- **Time taken to train:** Even on this front, the batch normalized case has a comparable training time to that of the model without it. This could again be due to the two reasons listed above which help the normal case overcome its handicaps thus making it equal to the batch normalized model. The training time of the two models are given below.
  - For the normal case: Time taken to train and test model: 502.33562564849854
  - For the batch normalized case: Time taken to train and test model: 542.0460834503174

3 Visualizing CNNs

For all the experiments ahead, we use the pre-trained convolutional neural network without batch normalization described in the previous section.

3.1 Convolutional Layer Filters

Plots of the Filters of the first convolutional layer:

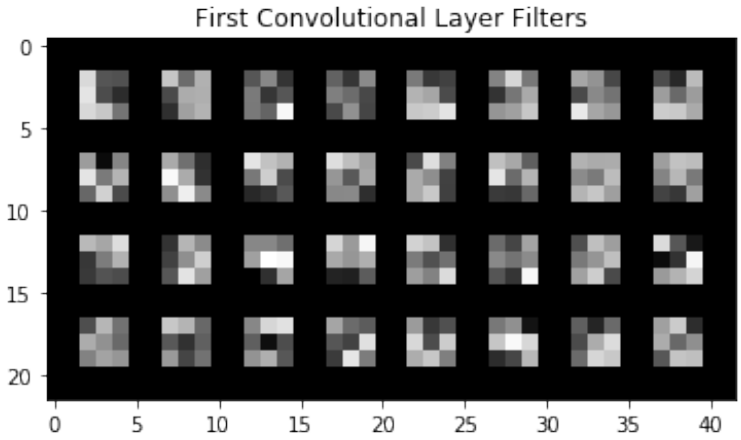

Figure 20: The filters of the first convolutional layer

Observations:

- We see that most of the filters are dark towards the edges while a few of these filters are dark towards the center.
- Some of these filters are reminiscent of the Gaussian or the Gabor filters discussed in class.
- We see that these filters detect basic features. The filters which are dark on the top or the bottom detect horizontal edges while the filters which are dark on the sides detect vertical edges. The filters which are dark at the centre and very light surrounding it , perform an image smoothening and the filters which are light at the centre and surrounded by grey patches perform an image sharpening operation.

Plots of the Filters of the second convolutional layer:
The second convolutional layer filters are box filters and hence it would be really hard to look at all 32 of them. Therefore, here are the plots of 5 random filters present in the second convolutional layer. The filters are indexed from 0 to 31.

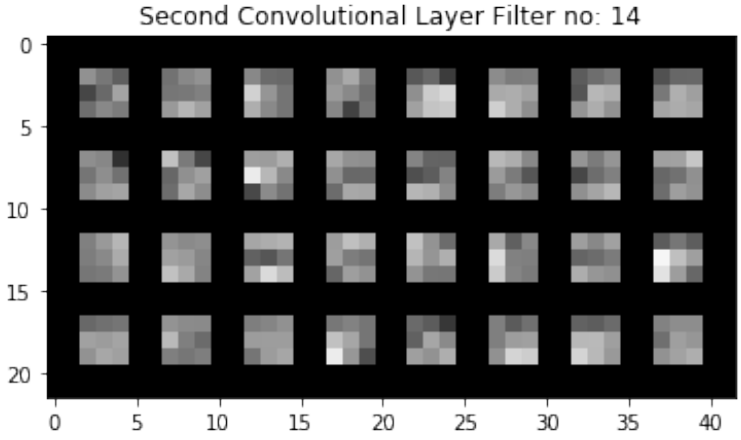

Figure 21: The fifteenth filter of the second convolutional layer

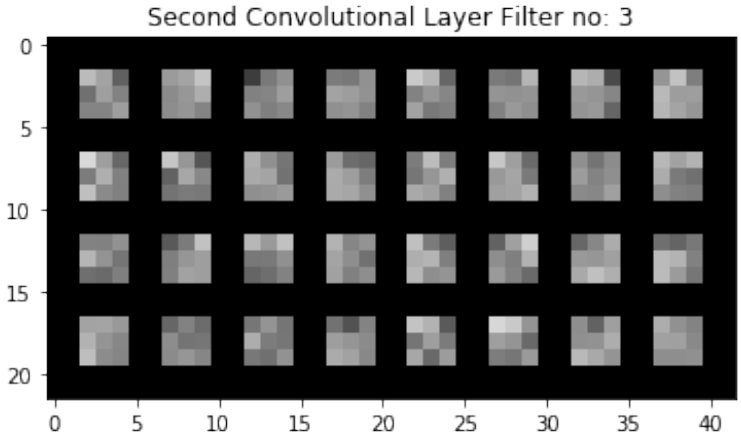

Figure 22: The fourth filter of the second convolutional layer

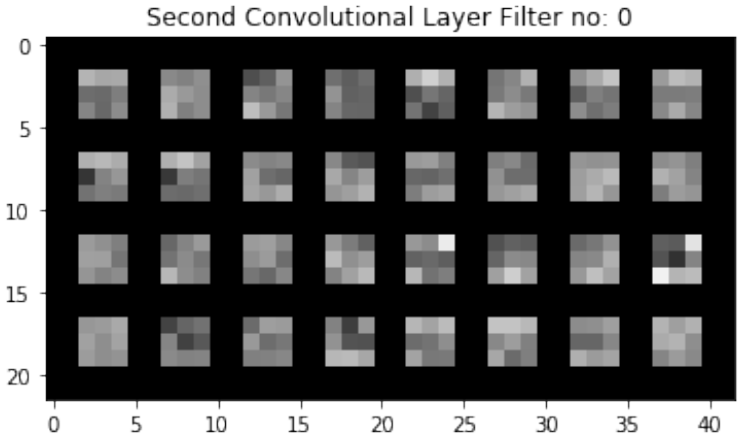

Figure 23: The first filter of the second convolutional layer

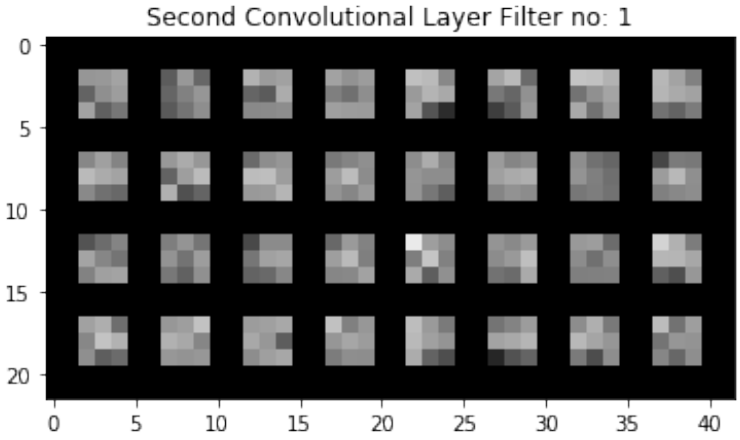

Figure 24: The second filter of the second convolutional layer

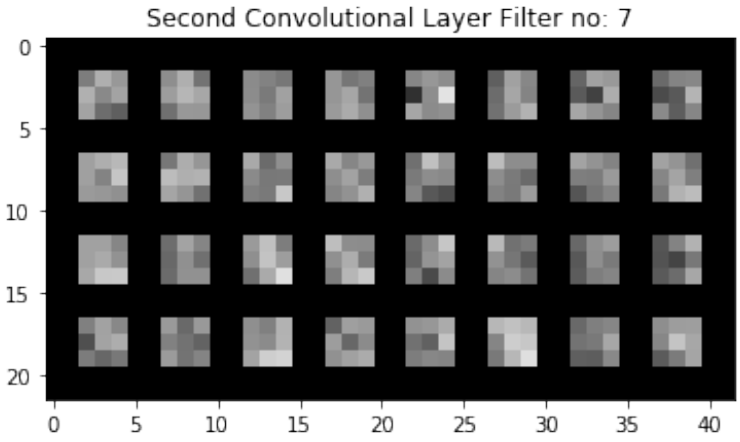

Figure 25: The eighth filter of the second convolutional layer

Observations:

- Unlike the filters of the first convolutional layer, we see that these filters are so sophisticated that we can't make any basic inferences also.

- We see that these filters have fewer black patches compared to those in the first layer. This could mean that they're trying to find joins or similar features.
- We see that there are some filters with white patches at the centre surrounded by grey patches. These filters could therefore try performing some sort of image sharpening or establish a connection between nearby pixels of the image.

3.2 Activations of the Convolutional Layers

The digits for this experiment were chosen at random. In this case I'm plotting the output of the activations of the convolutional layers for 2 digits, namely 0 and 5.
Plots of the activations of the first Convolutional Layer:

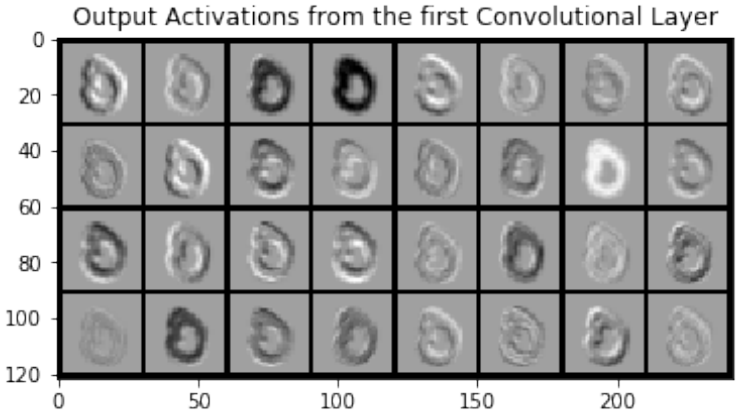

Figure 26: The activations for the digit 0

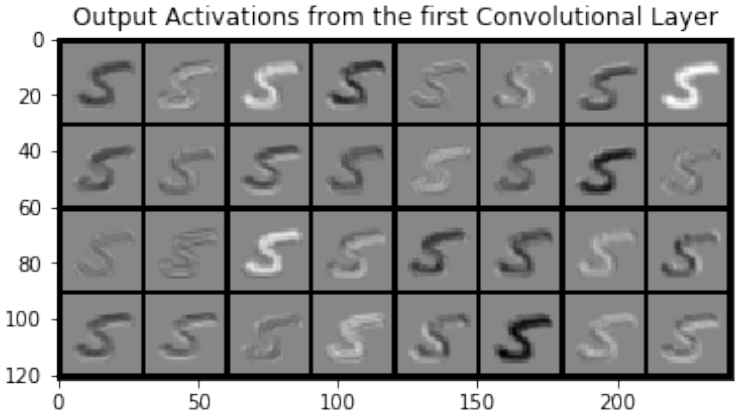

Figure 27: The activations for the digit 5

Observations

- As white is the highest and black the lowest value, we see that a lot of the lower level activations are either dark or light at the edges of the digit and therefore, we can assume that the neurons at the first convolutional layer primarily learn information about the edges which is what we had expected while looking at the plots for the filters.
- We also see that some of the feature maps just comprise of the digit coloured in grey which is a testament to the fact that a lot of the filters perform a pixel smoothening operation.
- We see that the output activations are very similar to the images which is also something we'd expect.

Plots of the activations of the second Convolutional Layer:

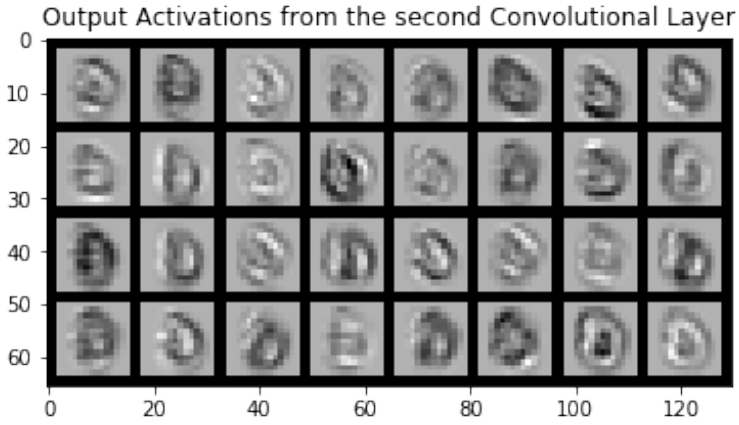

Figure 28: The activations for the digit 0

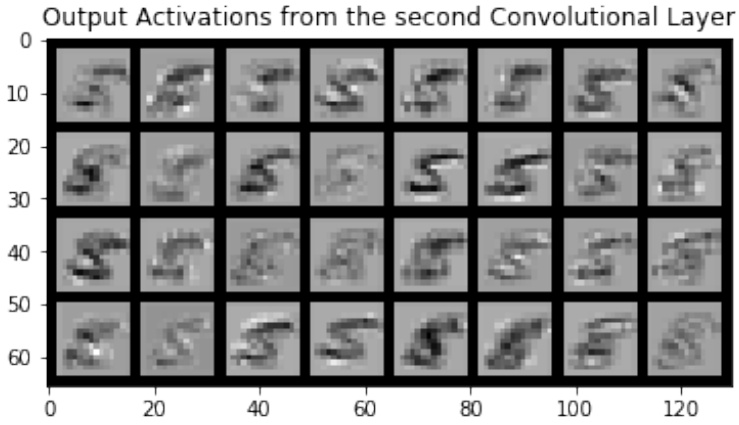

Figure 29: The activations for the digit 5

Observations

- As in the case with filters, we see that the activations are also more sophisticated looking compared to those at the first layer.
- We see that the dark and light regions are spread across specific localized places in the feature maps which signifies that the neurons are learning more sophisticated features.
- We see that there is no activation which is *absent* which reiterates the absence of any dead filter thus affirming that our learning rate to train the CNN wasn't too high.
- We also see that some of the feature maps have sharpened images which confirms the image sharpening operation that could've taken place at the filters.
- We see that the output activations are still resemblant to the digits although this resemblance is not as strong as that of the first layer feature maps.

3.3 Occlusion Experiment

Description of the experiment
The occlusion experiment is conducted by shifting a dark patch throughout the image and evaluating the probability and the predicted class as a function of the position of the dark patch.
This will enable us to figure out if the CNN is actually learning the correct features in the digits.

The occlusion experiment was repeated across different patch sizes, patch colours and stride length, but we finally settled on a 10x10 black patch moving across the image with a stride of 2 pixels.
The images used for the occlusion experiment are the same images depicted in the random plots category before.
Plot of the Occlusion Experiment on the digit 0

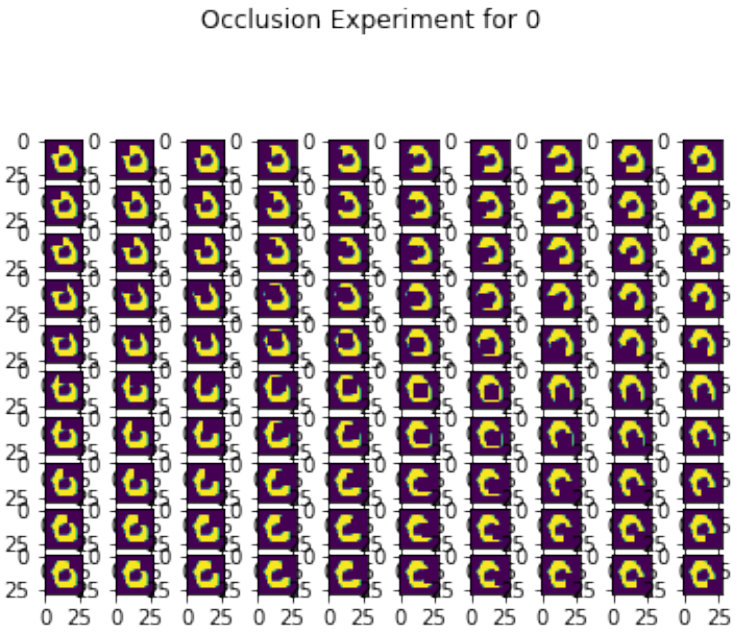

Figure 30: The experiment on the digit 0

We see that as we move along x, the patch moves below and as we move along y, the patch moves sideways. This is a consequence of the way the images are stored in pytorch.

Probability Heatmaps of all the digits:

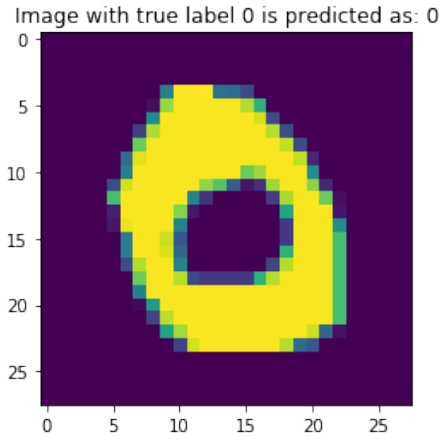

Figure 31: Random Test Image with true label 0

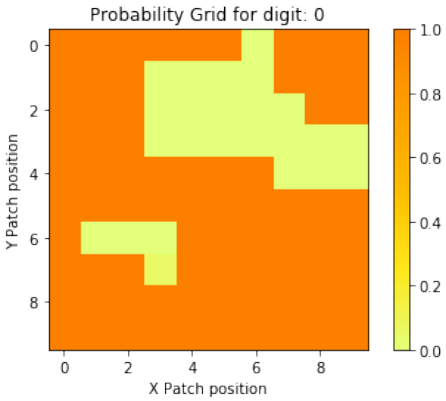

Figure 32: The probability heatmap for the digit 0

Corresponding Predicted Classes
[0. 0. 0. 0. 0. 0. 2. 0. 0. 0.

0. 0. 2. 3. 2. 2. 0. 0. 0.

0. 0. 2. 2. 2. 2. 7. 0. 0.

0. 0. 2. 2. 2. 2. 7. 7. 7.

0. 0. 0. 0. 0. 0. 7. 7. 7.

0. 0. 0. 0. 0. 0. 0. 0. 0.

6. 6. 6. 0. 0. 0. 0. 0. 0.

0. 0. 6. 0. 0. 0. 0. 0. 0.

0. 0. 0. 0. 0. 0. 0. 0. 0.

0. 0. 0. 0. 0. 0. 0. 0. 0.
]

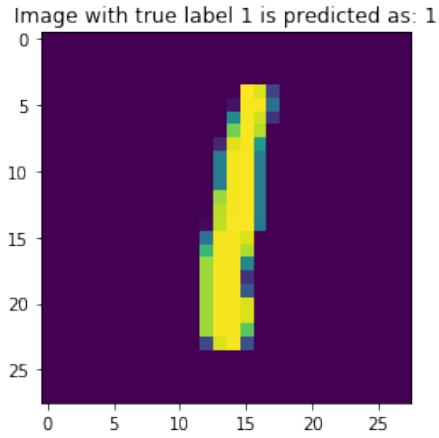

Figure 33: Random Test Image with true label 1

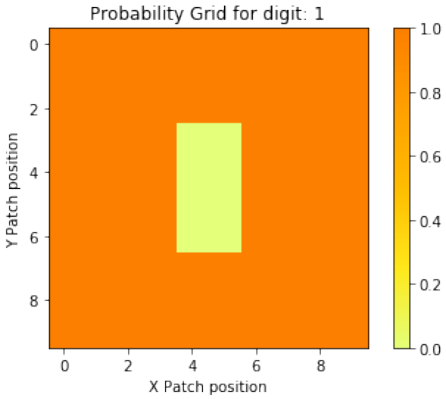

Figure 34: The probability heatmap for the digit 1

Corresponding Predicted Classes
[1. 1. 1. 1. 1. 1. 1. 1. 1. 1.

1. 1. 1. 1. 1. 1. 1. 1. 1.

1. 1. 1. 1. 1. 1. 1. 1. 1.

1. 1. 1. 3. 3. 1. 1. 1. 1.

1. 1. 1. 5. 5. 1. 1. 1. 1.

1. 1. 1. 5. 5. 1. 1. 1. 1.

1. 1. 1. 5. 5. 1. 1. 1. 1.

1. 1. 1. 1. 1. 1. 1. 1. 1.

1. 1. 1. 1. 1. 1. 1. 1. 1.

1. 1. 1. 1. 1. 1. 1. 1. 1.
]

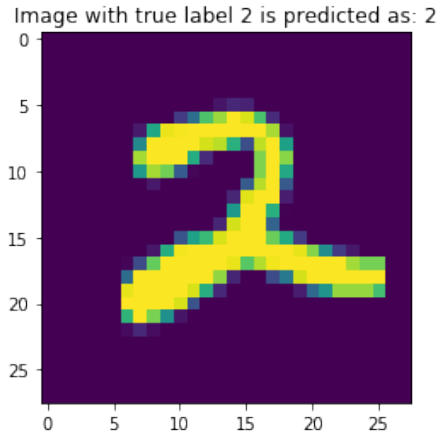

Figure 35: Random Test Image with true label 2

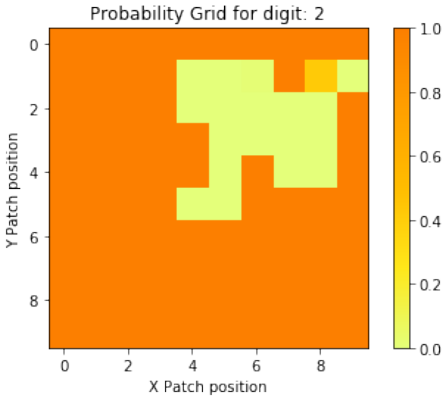

Figure 36: The probability heatmap for the digit 2

Corresponding Predicted Classes
[2. 2. 2. 2. 2. 2. 2. 2. 2. 2.

2. 2. 2. 3. 3. 3. 2. 3. 7.

2. 2. 2. 3. 3. 7. 7. 7. 2.

2. 2. 2. 2. 7. 7. 7. 7. 2.

2. 2. 2. 2. 7. 2. 3. 3. 2.

2. 2. 2. 1. 7. 2. 2. 2. 2.

2. 2. 2. 2. 2. 2. 2. 2. 2.

2. 2. 2. 2. 2. 2. 2. 2. 2.

2. 2. 2. 2. 2. 2. 2. 2. 2.

2. 2. 2. 2. 2. 2. 2. 2. 2.
]

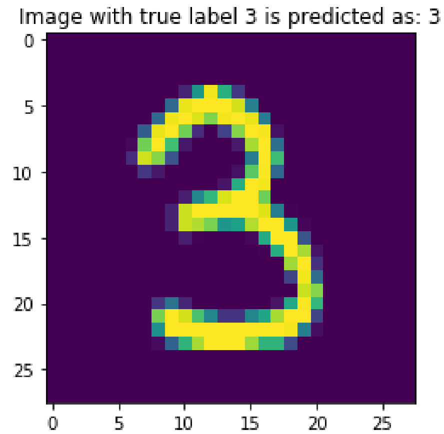

Figure 37: Random Test Image with true label 3

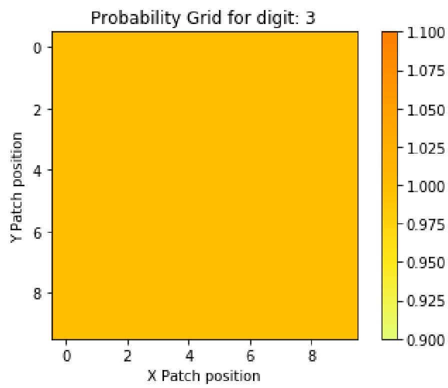

Figure 38: The probability heatmap for the digit 3

Corresponding Predicted Classes
[3. 3. 3. 3. 3. 3. 3. 3. 3. 3.

3. 3. 3. 3. 3. 3. 3. 3. 3.

3. 3. 3. 3. 3. 3. 3. 3. 3.

3. 3. 3. 3. 3. 3. 3. 3. 3.

3. 3. 3. 3. 3. 3. 3. 3. 3.

3. 3. 3. 3. 3. 3. 3. 3. 3.

3. 3. 3. 3. 3. 3. 3. 3. 3.

3. 3. 3. 3. 3. 3. 3. 3. 3.

3. 3. 3. 3. 3. 3. 3. 3. 3.

3. 3. 3. 3. 3. 3. 3. 3. 3.

]

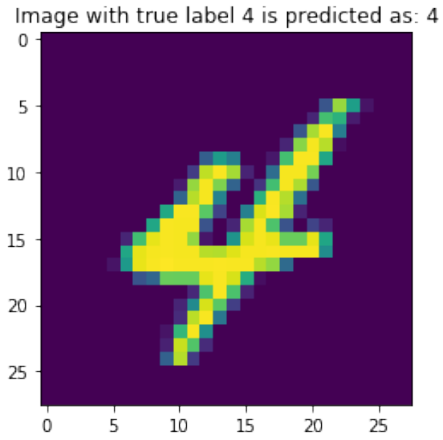

Figure 39: Random Test Image with true label 4

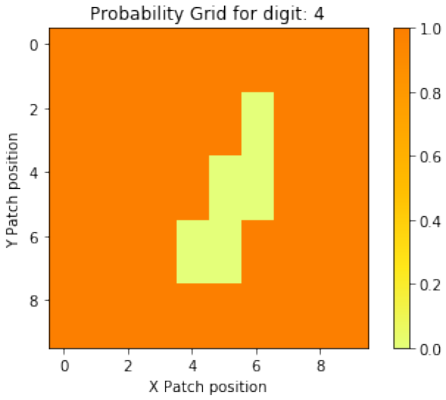

Figure 40: The probability heatmap for the digit 4

Corresponding Predicted Classes
[4. 4. 4. 4. 4. 4. 4. 4. 4. 4.

4. 4. 4. 4. 4. 4. 4. 4. 4.

4. 4. 4. 4. 4. 7. 4. 4. 4.

4. 4. 4. 4. 4. 7. 4. 4. 4.

4. 4. 4. 4. 7. 7. 4. 4. 4.

4. 4. 4. 4. 5. 5. 4. 4. 4.

4. 4. 4. 5. 5. 4. 4. 4. 4.

4. 4. 4. 5. 5. 4. 4. 4. 4.

4. 4. 4. 4. 4. 4. 4. 4. 4.

4. 4. 4. 4. 4. 4. 4. 4. 4.
]

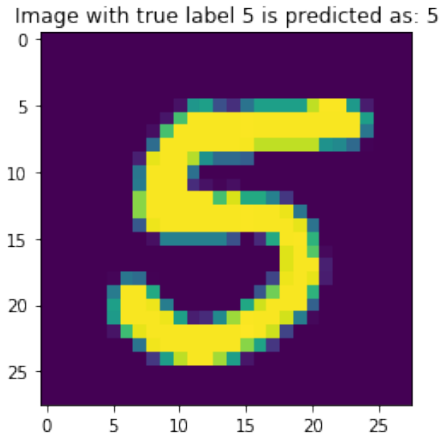

Figure 41: Random Test Image with true label 5

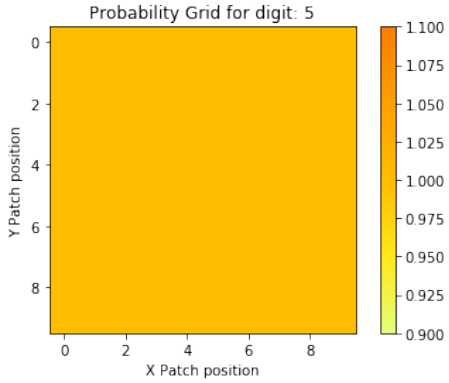

Figure 42: The probability heatmap for the digit 5

Corresponding Predicted Classes
[5. 5. 5. 5. 5. 5. 5. 5. 5. 5.

5. 5. 5. 5. 5. 5. 5. 5. 5. 5.

5. 5. 5. 5. 5. 5. 5. 5. 5. 5.

5. 5. 5. 5. 5. 5. 5. 5. 5. 5.

5. 5. 5. 5. 5. 5. 5. 5. 5. 5.

5. 5. 5. 5. 5. 5. 5. 5. 5. 5.

5. 5. 5. 5. 5. 5. 5. 5. 5. 5.

5. 5. 5. 5. 5. 5. 5. 5. 5. 5.

5. 5. 5. 5. 5. 5. 5. 5. 5. 5.

]

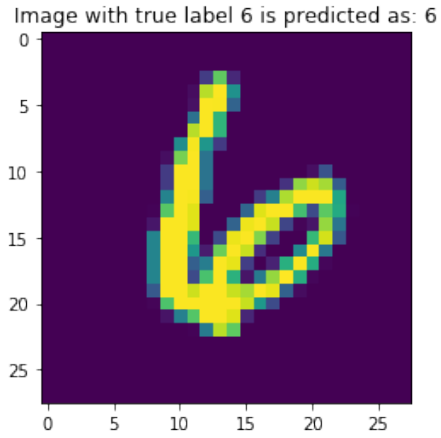

Figure 43: Random Test Image with true label 6

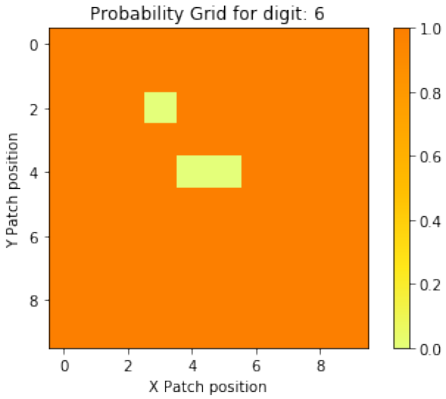

Figure 44: The probability heatmap for the digit 6

Corresponding Predicted Classes
[6. 6. 6. 6. 6. 6. 6. 6. 6. 6.

6. 6. 6. 6. 6. 6. 6. 6. 6.

6. 6. 2. 6. 6. 6. 6. 6. 6.

6. 6. 6. 6. 6. 6. 6. 6. 6.

6. 6. 6. 3. 5. 6. 6. 6. 6.

6. 6. 6. 6. 6. 6. 6. 6. 6.

6. 6. 6. 6. 6. 6. 6. 6. 6.

6. 6. 6. 6. 6. 6. 6. 6. 6.

6. 6. 6. 6. 6. 6. 6. 6. 6.

6. 6. 6. 6. 6. 6. 6. 6. 6.

]

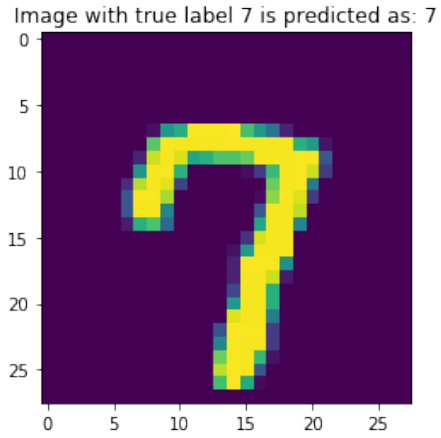

Figure 45: Random Test Image with true label 7

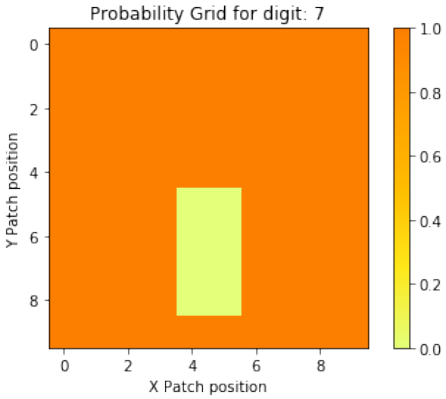

Figure 46: The probability heatmap for the digit 7

Corresponding Predicted Classes
7. 7. 7. 7. 7. 7. 7. 7. 7.

7. 7. 7. 7. 7. 7. 7. 7. 7.

7. 7. 7. 7. 7. 7. 7. 7. 7.

7. 7. 7. 7. 7. 7. 7. 7. 7.

7. 7. 7. 7. 7. 7. 7. 7. 7.

7. 7. 7. 9. 9. 7. 7. 7. 7.

7. 7. 7. 9. 5. 7. 7. 7. 7.

7. 7. 7. 9. 5. 7. 7. 7. 7.

7. 7. 7. 5. 5. 7. 7. 7. 7.

7. 7. 7. 7. 7. 7. 7. 7. 7.

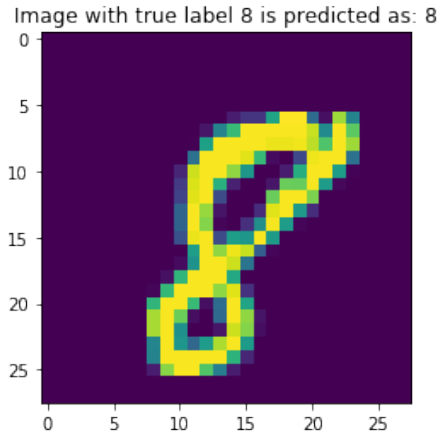

Figure 47: Random Test Image with true label 8

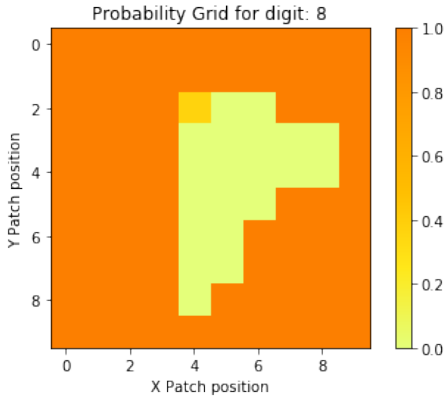

Figure 48: The probability heatmap for the digit 8

Corresponding Predicted Classes
8. 8. 8. 8. 8. 8. 8. 8. 8.

8. 8. 8. 8. 8. 8. 8. 8. 8.

8. 8. 8. 2. 3. 3. 8. 8. 8.

8. 8. 8. 3. 3. 3. 3. 9. 8.

8. 8. 8. 3. 3. 3. 3. 9. 8.

8. 8. 8. 3. 3. 3. 8. 8. 8.

8. 8. 8. 5. 5. 8. 8. 8. 8.

8. 8. 8. 5. 5. 8. 8. 8. 8.

8. 8. 8. 5. 8. 8. 8. 8. 8.

8. 8. 8. 8. 8. 8. 8. 8. 8.

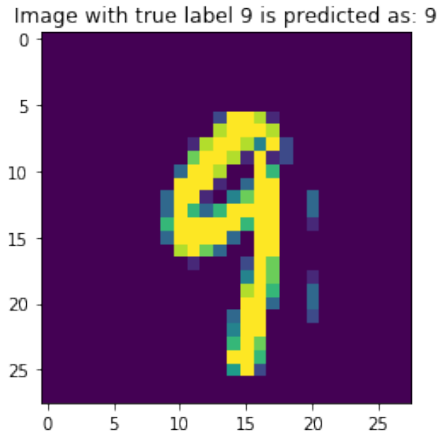

Figure 49: Random Test Image with true label 9

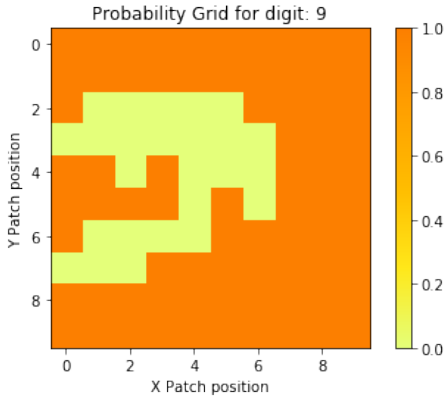

Figure 50: The probability heatmap for the digit 9

Corresponding Predicted Classes
9. 9. 9. 9. 9. 9. 9. 9. 9.

9. 9. 9. 9. 9. 9. 9. 9. 9.

4. 4. 1. 1. 7. 9. 9. 9. 9.

4. 4. 7. 7. 7. 7. 9. 9. 9.

9. 7. 9. 7. 7. 3. 9. 9. 9.

9. 9. 9. 7. 9. 5. 9. 9. 9.

4. 4. 4. 5. 9. 9. 9. 9. 9.

4. 4. 9. 9. 9. 9. 9. 9. 9.

9. 9. 9. 9. 9. 9. 9. 9. 9.

9. 9. 9. 9. 9. 9. 9. 9. 9.

Observations:
Some common observations are given below.

- On repeating the experiment for smaller patches, we see that the low probability regions in the heat maps were much lower as lesser part of the image was being occluded.
- The probability heatmap remains high when the portion of the image occluded has nothing to do with the digit.
- We also see that the probability heatmap remains high at the regions when only a small portion of the image or edges are occluded.
- This says that the classifier is robust even when it is shown an occluded image.
- When there is a patch covering the whole digit, or there is a central patch, we see that the prediction is very bad and consequently the probability is also low.

Digit specific observations are given below.

- We see that 0 gets predicted as 2 or 7 during the occlusion experiment. This makes sense, as two also shares the side lobe present in 0 and when only that portion of 0 is visible, it could be mistaken for 2. Likewise, as this zero is slightly tilted the right side portion is resemblant to that of a 7.
- Here we cannot make any conclusion about the wrong predictions.It is very reassuring to see that the low probability region is concentrated around the region where 1 is present.
- The 2 looks like a 3 if the bottom portion of the image is occluded and that explains all the mispredictions involving 3. Likewise, the two also looks like 7 should the right lower portion of the image be occluded and this could explain the presence of 7 amongst the mispredictions.
- The 3 and 5 are predicted everywhere with high confidence even with occlusion. This could mean that the network has learnt specific features to distinguish 3s and 5s from other numbers at a very high precision or it could mean that the occlusion patch is not big enough to occlude large areas of the image. The latter is more likely.
- The 4 gets mistaken for 7 which makes sense as the 4 is slightly angled. Covering the left side of 4 makes for a slightly passable 7.
- We see that the 8 gets mistaken for 3 and 5 quite often. This makes a lot of sense as, removing the top right and the bottom right region from 8 give us 3 while removing the top left and the bottom right region from 8 gives us 5.
- We see that 9 gets mistaken for 7 quite often. This is because 9 without the bottom part of the lobe is a 7.

Thus with all these observations, we can positively state that our network is learning the correct features for the images.

4 Adversarial Examples

For each of the adversarial examples, Adam is yet again used for gradient ascent. This is done in the code by taking the loss function in the code to be negative of the actual function in order to convert the gradient descent into an ascent.
As in the occlusion experiment, all the digits considered were from the database of digits we saw in the random plots section.
As I trained with log softmax, an exponent function has been applied to recover the softmax probability from there.

4.1 Non-Targeted Attack

The step size used for the non-targeted attack was 0.05 and it was run for 15000 iterations to get the contours of the image properly. The plots of the images are shown below.
Plots:

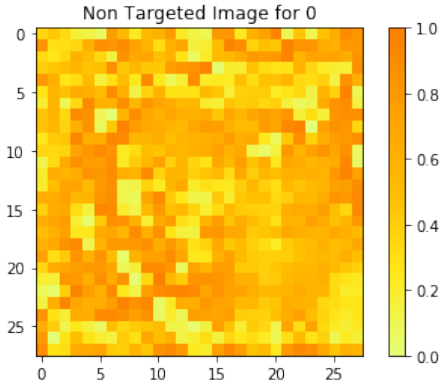

Figure 51: Adversarial Image generated for 0

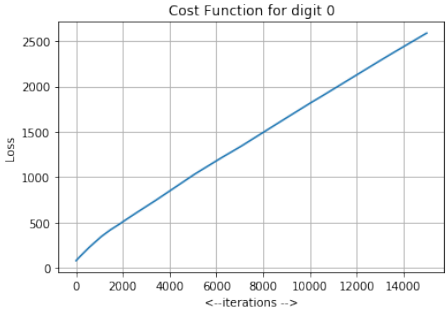

Figure 52: Cost incurred in generating the adversarial image for 0

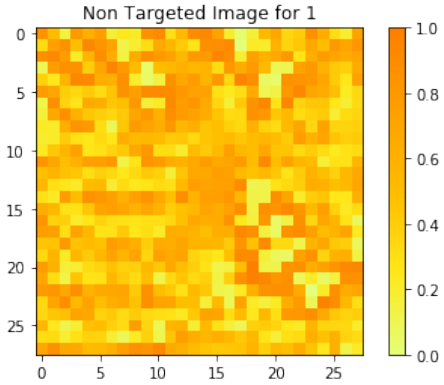

Figure 53: Adversarial Image generated for 1

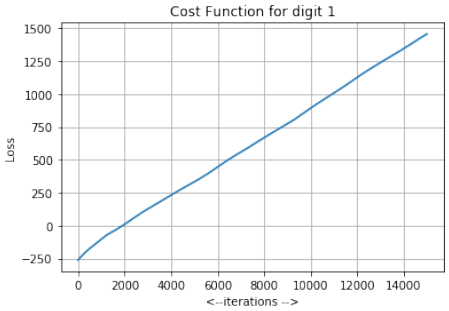

Figure 54: Cost incurred in generating the adversarial image for 1

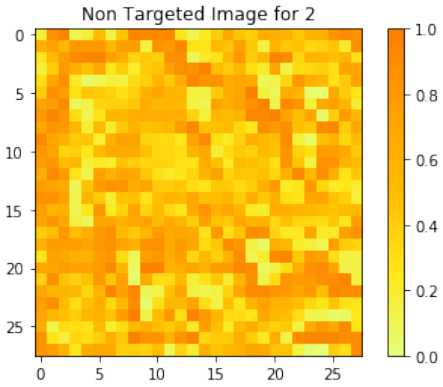

Figure 55: Adversarial Image generated for 2

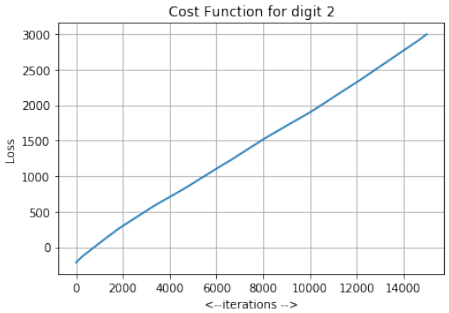

Figure 56: Cost incurred in generating the adversarial image for 2

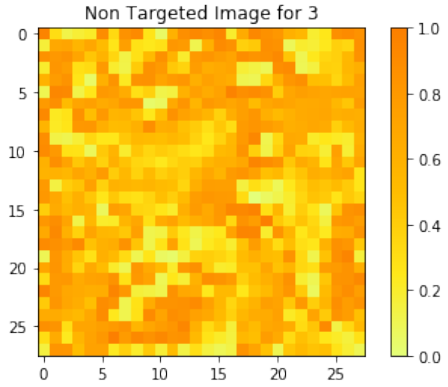

Figure 57: Adversarial Image generated for 3

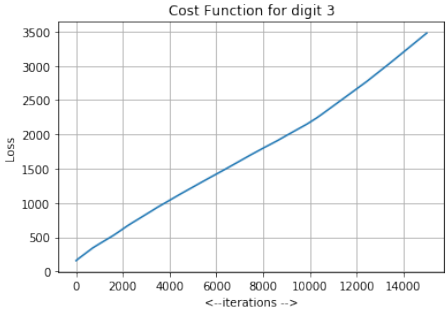

Figure 58: Cost incurred in generating the adversarial image for 3

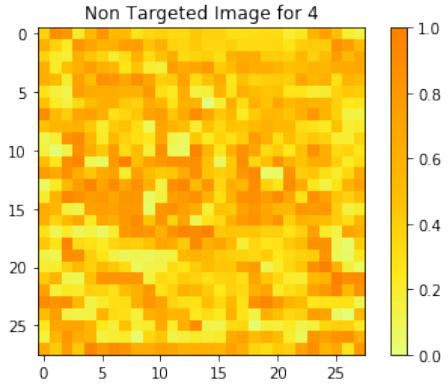

Figure 59: Adversarial Image generated for 4

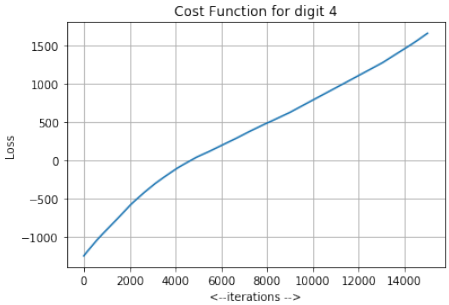

Figure 60: Cost incurred in generating the adversarial image for 4

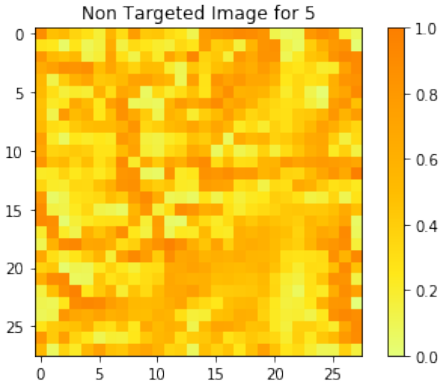

Figure 61: Adversarial Image generated for 5

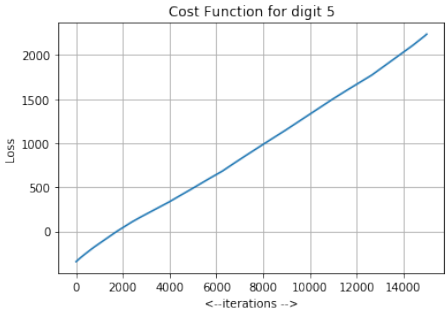

Figure 62: Cost incurred in generating the adversarial image for 5

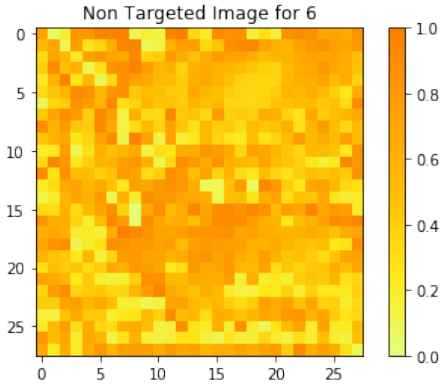

Figure 63: Adversarial Image generated for 6

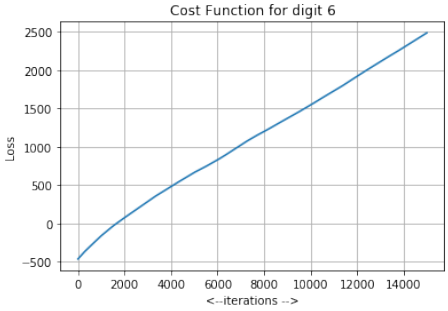

Figure 64: Cost incurred in generating the adversarial image for 6

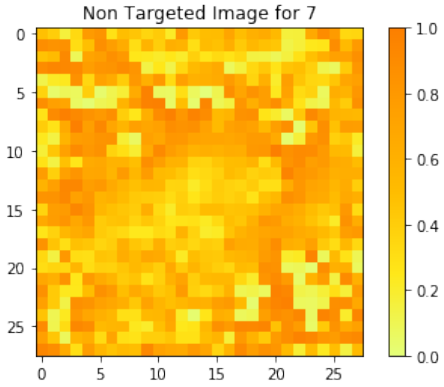

Figure 65: Adversarial Image generated for 7

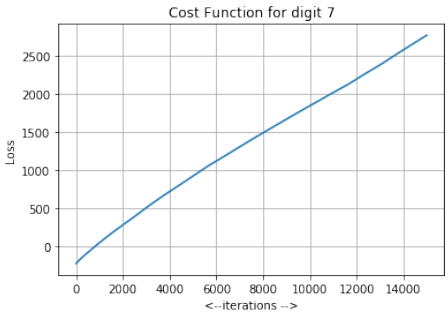

Figure 66: Cost incurred in generating the adversarial image for 7

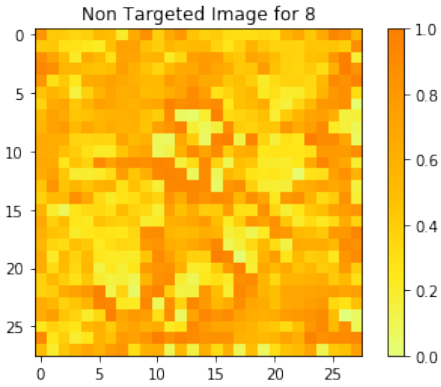

Figure 67: Adversarial Image generated for 8

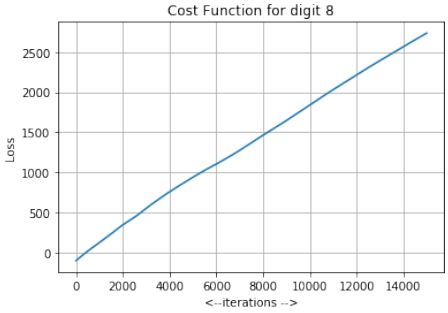

Figure 68: Cost incurred in generating the adversarial image for 8

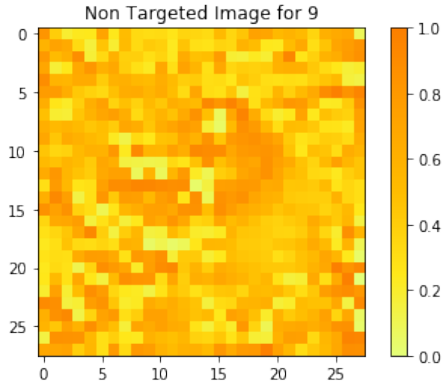

Figure 69: Adversarial Image generated for 9

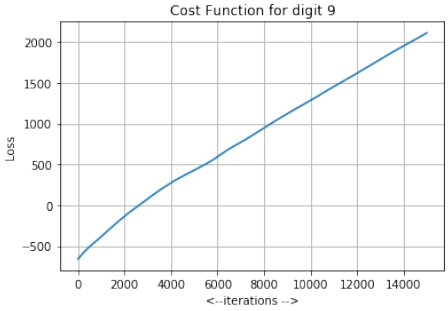

Figure 70: Cost incurred in generating the adversarial image for 9

Observations

- The networks predict the adversarial examples with very high probability = 1. The convergence to this probability is also very fast, approximately a few thousand iterations. The additional iterations are just to make the contours in noise perceptible to the human eye to some extent.
- As seen in all the cases, the cost is increasing. This is what we'd expect to happen as we want the logits to be higher, as then only the softmax probability will be higher which is necessary for the example to be predicted to the corresponding class. Thus the cost goes up.
- As we'd expect, the generated images look nothing like actual digits as the neural network learns in strange ways unbeknownst to us. However, we can see that the general characteristic of some of the digits can be observed in the generated images.
- The generated image for 8 has two lobes faintly visible.
- The generated image for 9 has a top lobe faintly visible.
- The bottom lobe for 6 is also faintly visible.
- We see the side edges for 5.
- We can actually make out 3 from the contour.
- The top curve of 2 is visible. The bottom portion is a bit too shaky though.
- We can also make out a rough circle from the plot for 0.

4.2 Targeted Attack

The value of beta used was 0.0001 upon experimenting with multiple values. I saw that smaller the beta, larger was the resemblance to the target image while a larger beta implied that the resulting image was much closer to the actual image.
This was run for 1000 iterations to generate the adversarial image.
The initial value of X was taken to be the chosen image as this saves on iterations and as the final image would have to lie closer to the original image. I tried initializing X with noise and these results were successful to. However, they took too long to generate a proper adversarial image.
In the code, all the digits are chosen based on the user's preference, thus we can also generate combinations not shown in the plots below.
Plots

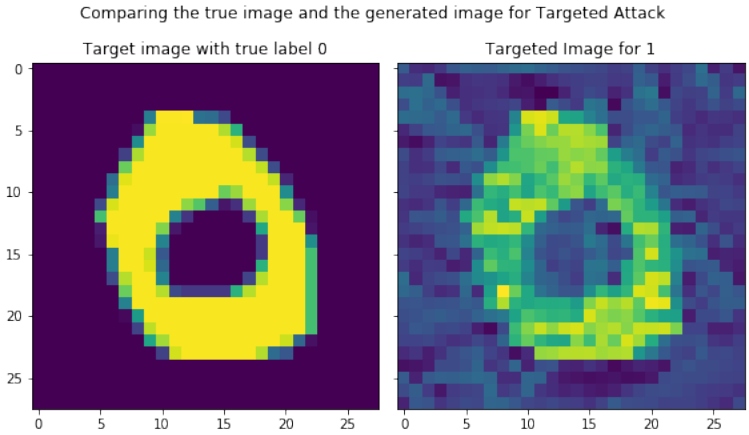

Figure 71: Adversarial Image generated from 0

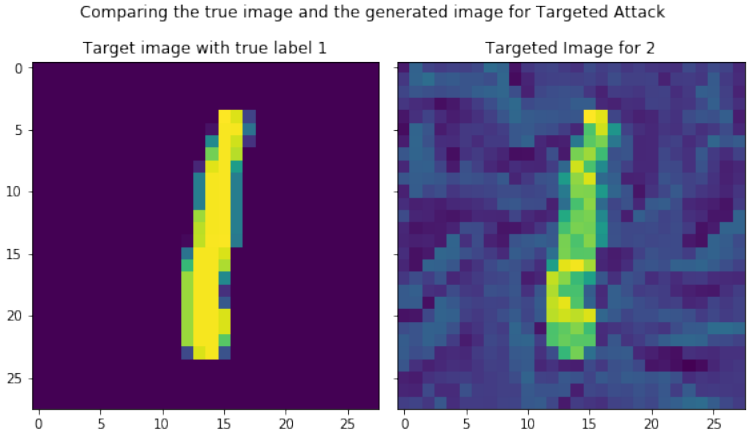

Figure 72: Adversarial Image generated from 1

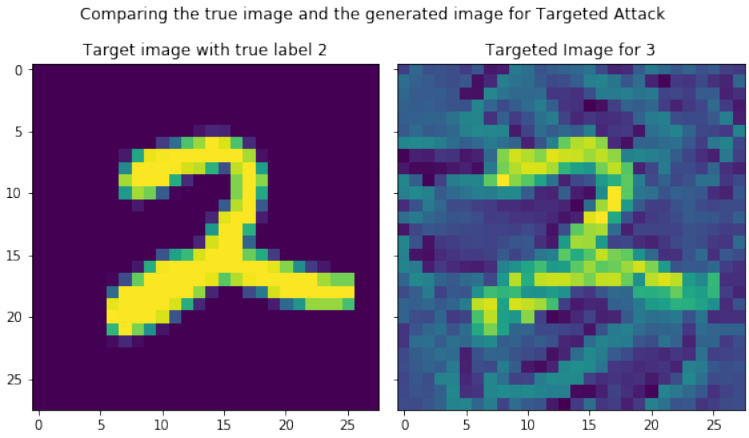

Figure 73: Adversarial Image generated from 2

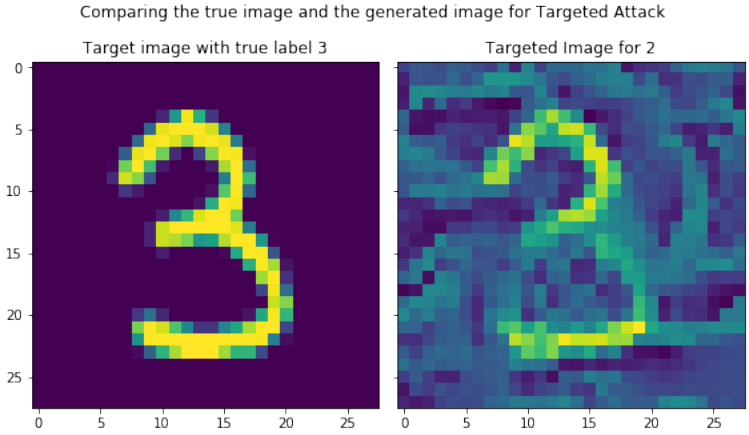

Figure 74: Adversarial Image generated from 3

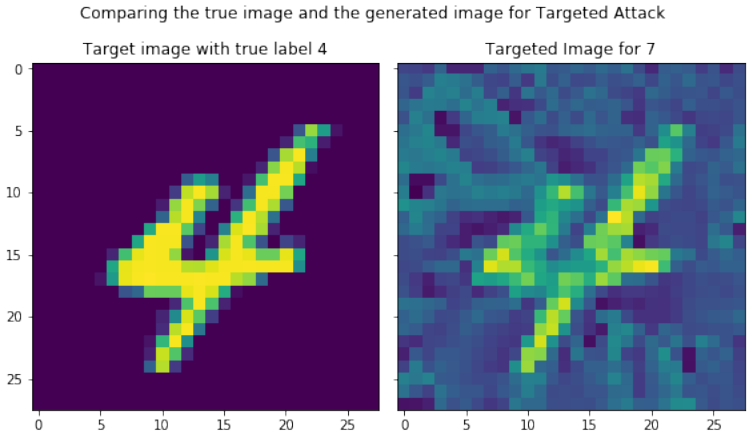

Figure 75: Adversarial Image generated from 4

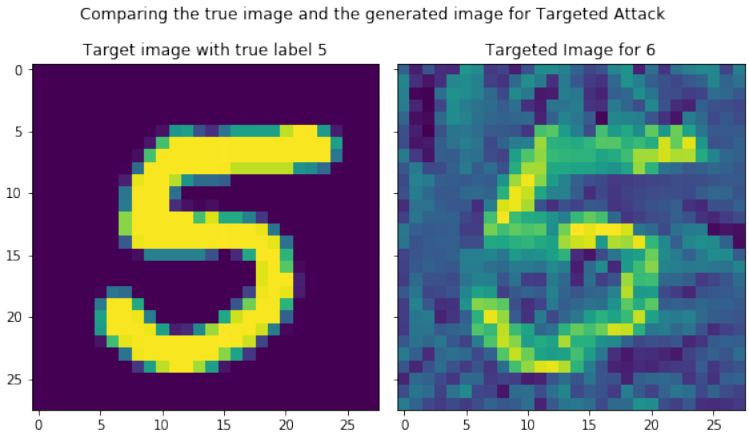

Figure 76: Adversarial Image generated from 5

Figure 77: Adversarial Image generated from 6

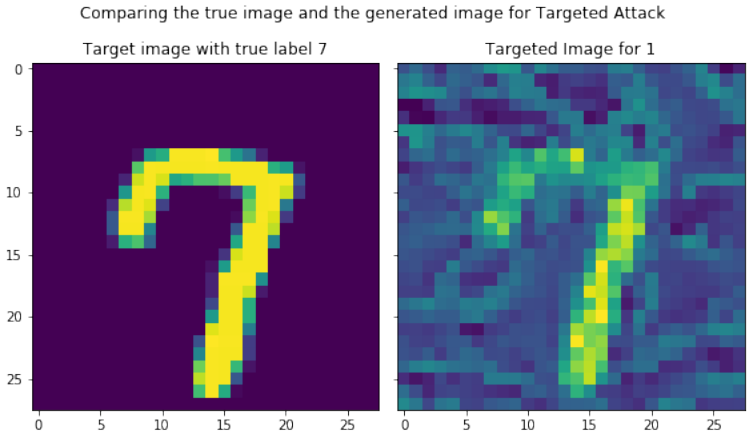

Figure 78: Adversarial Image generated from 7

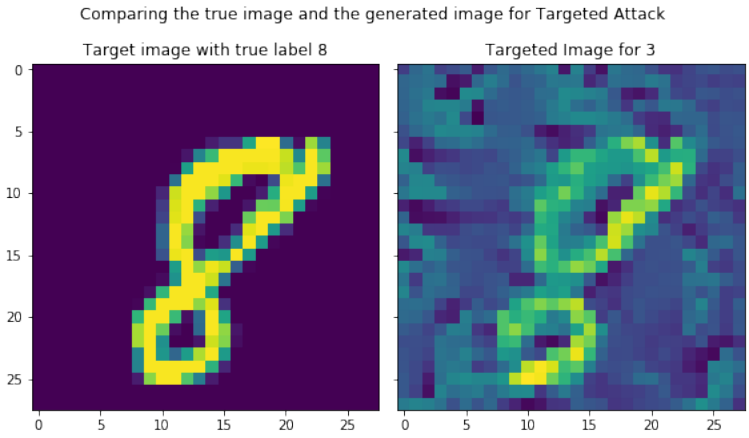

Figure 79: Adversarial Image generated from 8

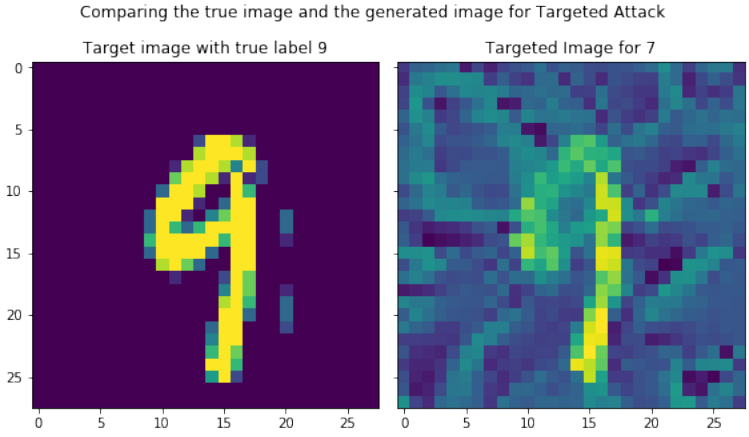

Figure 80: Adversarial Image generated from 9

Observations

- As in the previous case, the confidence in the predictions were astoundingly high.
- We see that as we increase the value of beta, the generated image is very close to the true image with very minimal characteristics of the target class. However, the confidence in the predictions still remain astoundingly high.
- We also see that if we choose a very low value for beta, then the noise term overpowers the logits term and thus, we end up getting a blurry image which has nothing to do with the original image.
- We see that the resulting adversarial images here are much closer to numbers as we had expected.
- Some adversarial images are captured very well and possess the desired features of the target class. These can be seen in the adversarial images generated from 1, where we can see a 2 lurking in the background.
- We see that the adversarial image generated from 2 is also very resemblant to that of 3.The bottom curve is visibly constructed by the gradient ascent based updates.
- We can see that this is the case for when we try to pass 3 off as 2 as well. The bottom curve in 3 is faint and there is a 2 which is overlapping it.
- We can see that the adversarial image generated from 5 also bears a good resemblance to 6 with the addition of the bottom lobe.
- We see that even in 7, the top edge is cut off by the updates to make the adversarial image to rightfully look like 1.

- We see that in 8, the side portions described in the occlusion experiment have been cut off for the resulting image to look like a 3. However, the image looks like 2 disjoint lobes. Maybe with more iterations, the image would've looked like 3.
- The top lobe of 9 is chopped off to make it look like a 7.

All these observations just reinforce our belief that our CNN has learnt the correct features.

4.3 Adding Noise

The learning rate used for noise addition was very small as recommended. It was kept at about 0.005 and was run for about 1500 iterations to produce the noise matrix. This noise matrix was later added to another random test image to check if the noise matrix succeeded in fooling the network yet again.
In the code, all the digits are chosen based on the user's preference, thus we can also generate combinations not shown in the plots below.
Plots

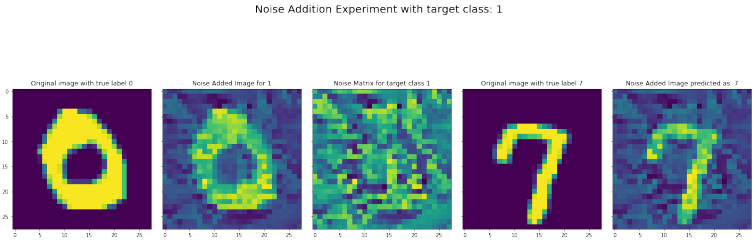

Figure 81: Adversarial Images generated from 0

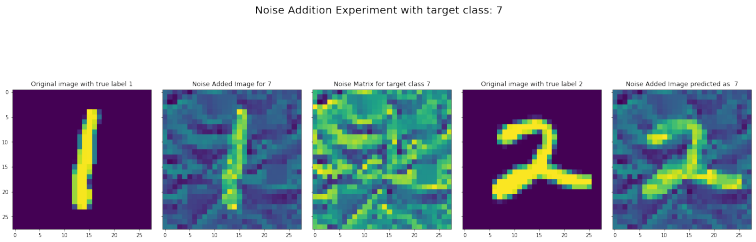

Figure 82: Adversarial Images generated from 1

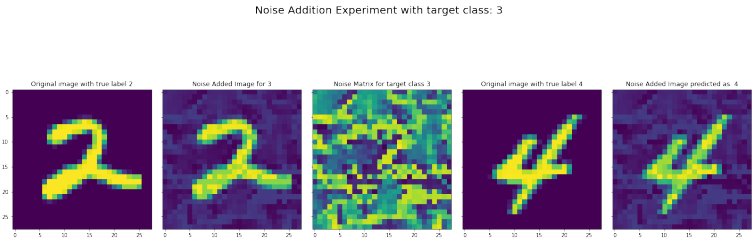

Figure 83: Adversarial Images generated from 2

Figure 84: Adversarial Images generated from 3

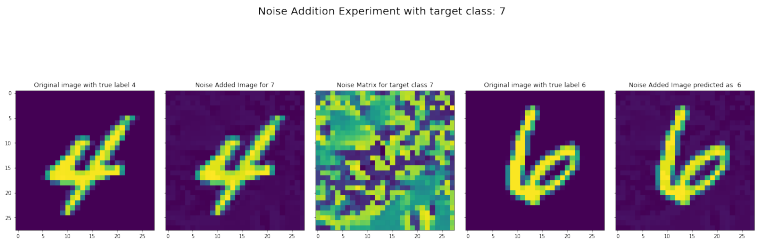

Figure 85: Adversarial Images generated from 4

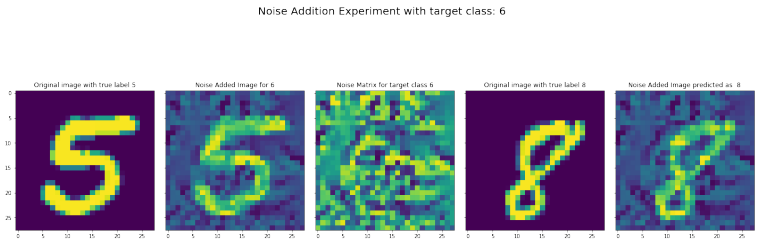

Figure 86: Adversarial Images generated from 5

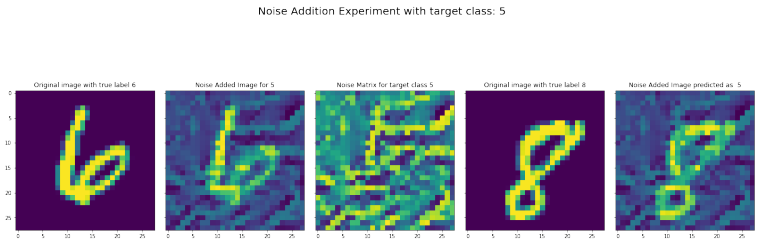

Figure 87: Adversarial Images generated from 6

Figure 88: Adversarial Images generated from 7

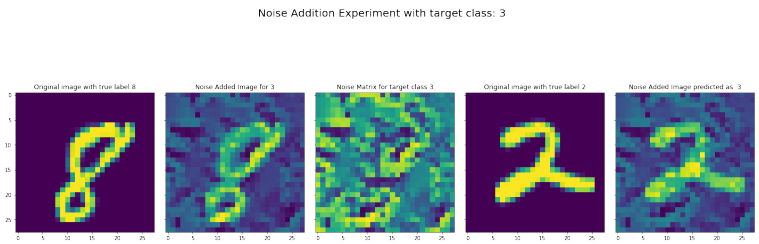

Figure 89: Adversarial Images generated from 8

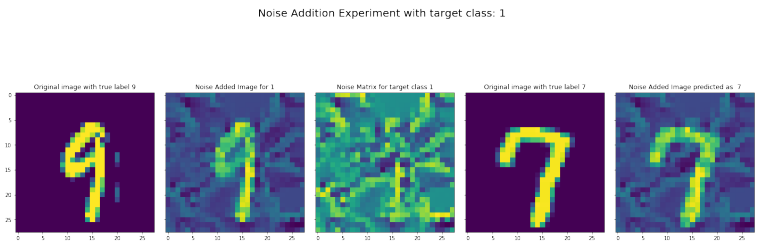

Figure 90: Adversarial Images generated from 9

Observations

- Here too, as in the previous case, we saw that with increasing learning rate more and more disjoint adversarial images were generated.
- As in the previous case, the adversarial images were either accepted or rejected with a very high confidence.
- We see that the generated first set of adversarial images has similar trends to that of the targeted attack seen before. As in, there is a 2 lurking in the background,etc. All the observations made there, seem to follow here as well.
- If we just look at the noise matrix, we see that there is a lot of similarities between it and the adversarial images obtained in the non-targeted case.
- Especially, in the cases where addition of noise matrix to another random test image leads to the desired target class, we see that, the noise matrix actually resembles the target digit. This can be seen in the cases of 1 where 2 is predicted as 7, the case of 6, where 8 is predicted as 5, the case of 7 where 2 is predicted as 1 and in the case of 8 where 2 is predicted as 3.
- We see that, if we increase the step size by a lot, the noise term will start overpowering and all the second adversarial images would've been dominated by the noise term and hence predicted to the desired target class. However, I chose the learning rate appropriately to ensure this didn't happen.
- Lastly, we see that, most of the second adversarial images are perceptible to the human eye and also to the CNN as it predicted the correct classes most of the time on repeating the experiment multiple times.
- These experiments finally establish the fact that the CNN did learn the correct features while training.