EE6132: Advanced Topics in Signal Processing

Assignment # 2
Convolutional Neural Networks

Sourav Sahoo, EE17B040

September 24, 2019

Contents

1 Introduction

Deep learning methods have radically improved the state-of-the-art models in numerous tasks across various domains like image classification, segmentation, machine translation, speech recognition etc. Deep learning identifies the intricate patterns in very large datasets by backpropagation [1] algorithm due to which it has become a very successful idea in recent times [2]. In this assignment, we apply a convolutional neural network (CNN) [3] to classify the MNIST handwritten digit dataset [4].

2 MNIST Classification using CNN

2.1 Model Training

We use a CNN with two layers of convolution, each having 32 filters of kernel size 3×3, followed a single fully connected layer with 500 units. The fully connected layer is connected to the final layer that predicts the class for each image. The detailed architecture is presented in Table 1. ReLU [5] non-linearity after each layer is used and the network is trained in batches of 16 using Adam optimizer [6] for five epochs. All the experiments are conducted on Google Colab using Pytorch. The training and validation loss are plotted in Figure 1. Some examples of the test images

Table 1: Network Architecture

Layer	**Activation Size**
input	$1 \times 28 \times 28$
$32 \times 3 \times 3$ conv, stride 1	$32 \times 28 \times 28$
2×2 maxpool, stride 2	$32 \times 14 \times 14$
$32 \times 3 \times 3$ conv, stride 1	$32 \times 14 \times 14$
2×2 maxpool, stride 2	$32 \times 7 \times 7$
flatten	1×1568
fully connected I	1×500
output	1×10

where $C \times H \times W$ conv denotes a 2D convolutional layer with C filters of size $H \times W$. $H \times W$ maxpool denotes a max-pooling layer of pooling size $H \times W$.

are shown in Figure 2. The test accuracy reaches **98.92%** after the entire training.

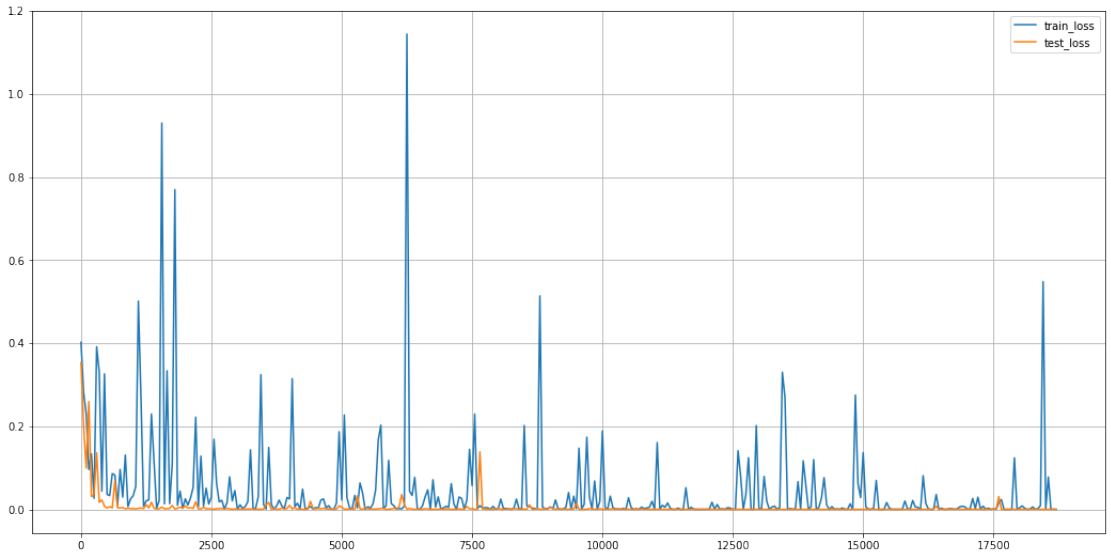

Figure 1: Variation of training and validation loss with number of iterations

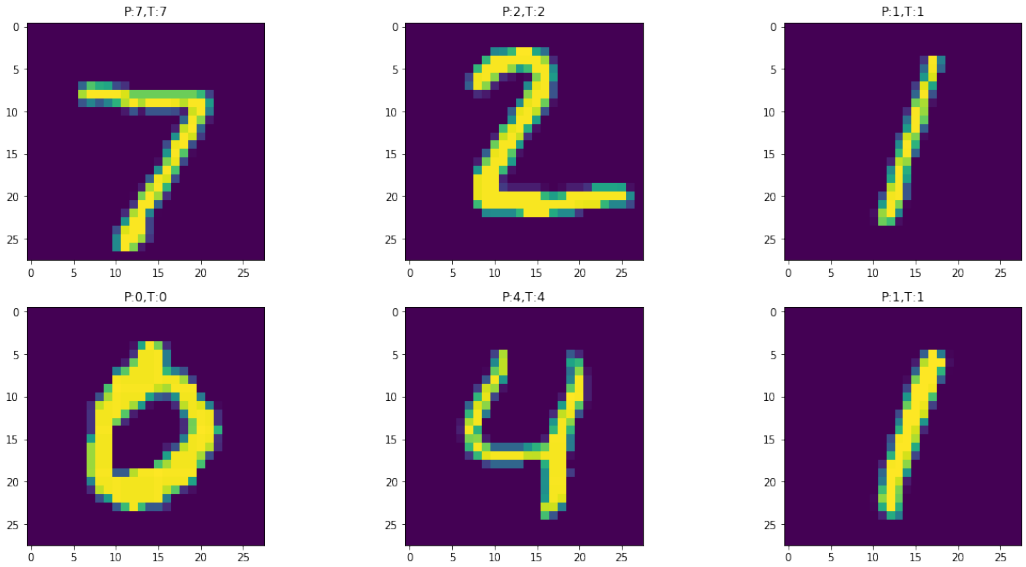

Figure 2: Some images from test set. P denotes the predicted class and T denotes the true class

2.2 Parameter Count

From Table 1, we count the number of parameters and neurons.

Convolutional Layers: 288 + 288 = 576 neurons
Fully Connected Layers: 1568 + 500 + 10 = 2078 neurons
Total: 576 + 2078 = 2654 neurons

Convolutional Layers: 320 + 9248 = 9568 parameters
Fully Connected Layers: 784,500 + 5010 = 789,510 parameters
Total: 789,510 + 9568 = 799,078 parameters

It should be noted that the affine layers in a neural network are the primary contributors ($\sim$ 98.8%) in the high number of parameters. As the number of parameters reduces in CNN without hampering the performance, the common problems associated with multi-layered perceptions like overfitting, longer training time and memory issues are solved.

2.3 Batch Normalization

In case of deep neural networks, there is a high chance that the distribution changes after a number of forward and backward passes. This leads to delayed training process of the entire model. This phenomena is known as internal covariate shift. Batch Normalization was introduced by Ioffe and Szegedy [7] to counter this problem. Batch Normalization not only reduces covariate shift, it also speeds up the training process, acts as regularization and leads to higher accuracy during testing. We introduce batch normalization in our network and compare the performances. The training and test loss of the model is presented in Figure 3. The test accuracy of the model with batch normalization is **99.05%**. The variation of accuracy with number of iterations with and without batch normalization is presented in Figure 4.

3 Visualizing the Convolutional Neural Network

3.1 Visualizing Filters

We first plot the 32 filters of the first convolutional layer in Figure 5. The second layer uses box filters of size $32 \times 3 \times 3$. We visualize the random layers of the filters in Figure 6. The filters themselves do not convey much information on their own. So, activation maps are visualized in the next subsection.

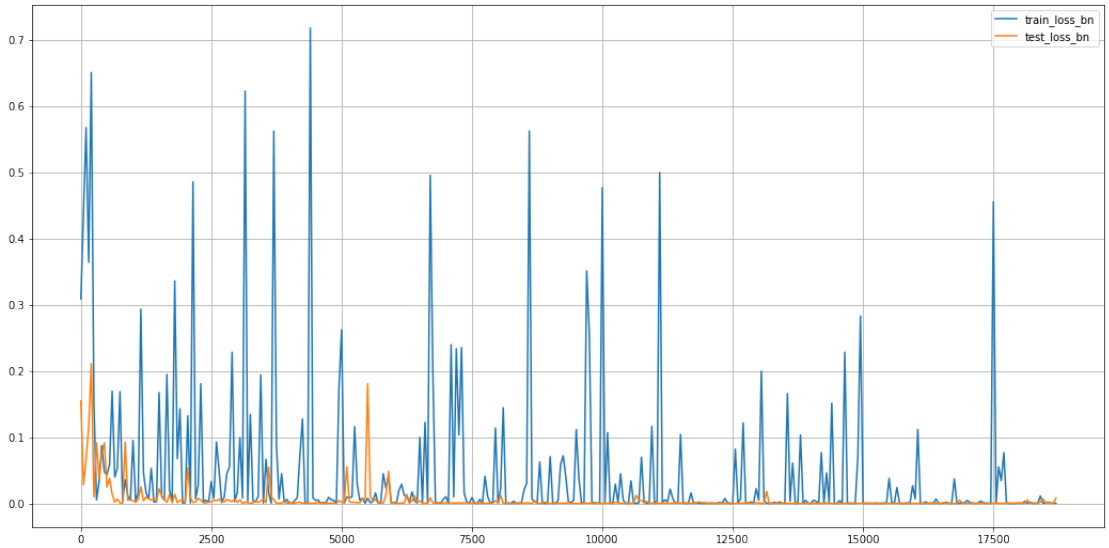

Figure 3: Variation of training and test loss with number of iterations when batch normalization is used. It is to be noticed that the training loss is much lower as compared to the case when the model did not use batch normalization.

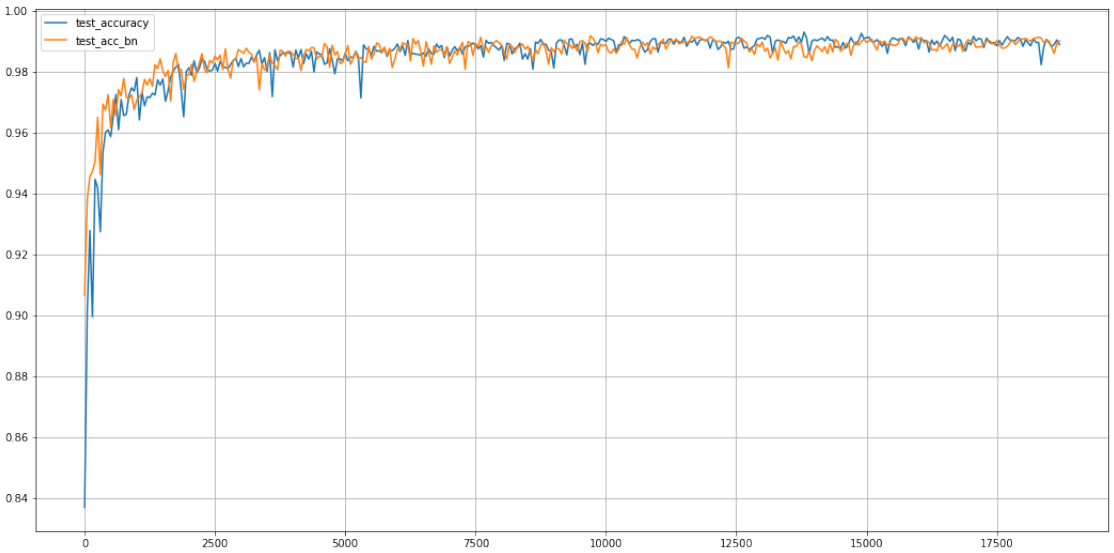

Figure 4: Variation of accuracy with number of iterations with and without using batch normalization. It is to be noticed that the initial rate of increase of accuracy is higher when batch normalization (orange) is used, although at the end the accuracy saturates to almost same value.

3.2 Visualizing Activation Maps

We visualize the activation maps of an image after it has passed through the first and second convolutional layer respectively. The activation maps are shown in Figure 7 and 8. It is observed that the activation map of the first convolutional layer detects the low level features like edges whereas the higher layers detect high-level features of the image as a whole which are mostly incomprehensible.

3.3 Occluding parts of the image

In this section we occlude the parts of an image using a grey patch and compute the corresponding output of the CNN. A gray patch of size 14×14 is slided over the 28×28 image with stride 7. So, we get a total of 9 occluded images for each image. We observe the effects while sliding over **7** and

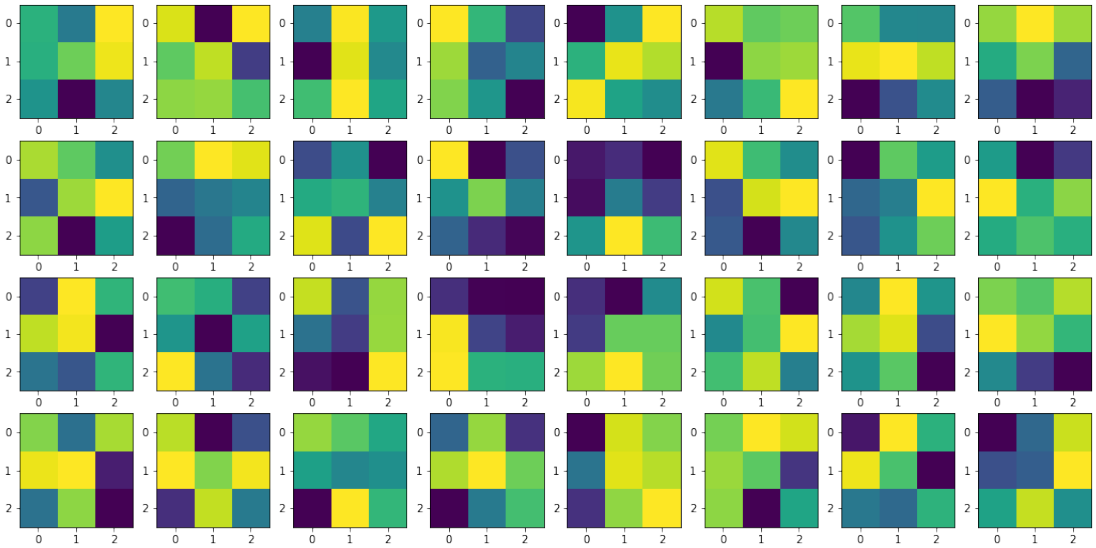

Figure 5: Filters in the first convolutional layer

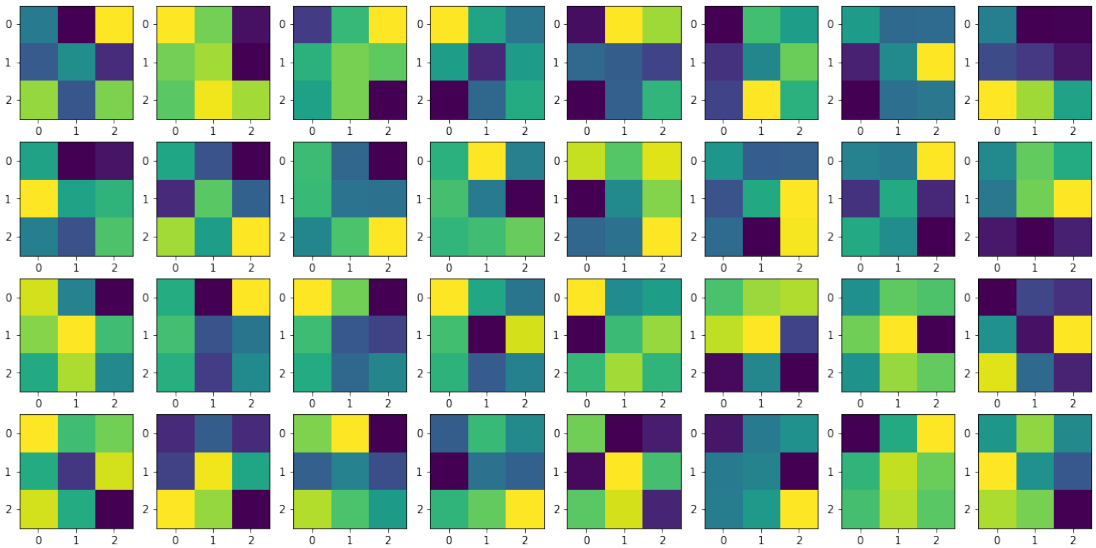

Figure 6: Filters in the layer 23 of the second convolutional layer

9. The occluded images are presented in Figure 9 and 10. The corresponding model output are presented in Table 2 and 3. In the images. we observe that as the main distinguishing features are occluded, the accuracy also decreases substantially. For example, when the top part of 7 is occluded the model mistakenly predicts it to be 9 because the 9 also has a similar longitudinal part. In case of 9, also when the top round part is occluded, the model predicts it to be 4 or 7 as rest of the features are common to both 4 and 7.

4 Adversarial Attacks

Adversarial examples are those examples used to fool a neural network and make them predict incorrectly. Here, we try three different types of adversarial attacks as mentioned in the subsequent subsections.

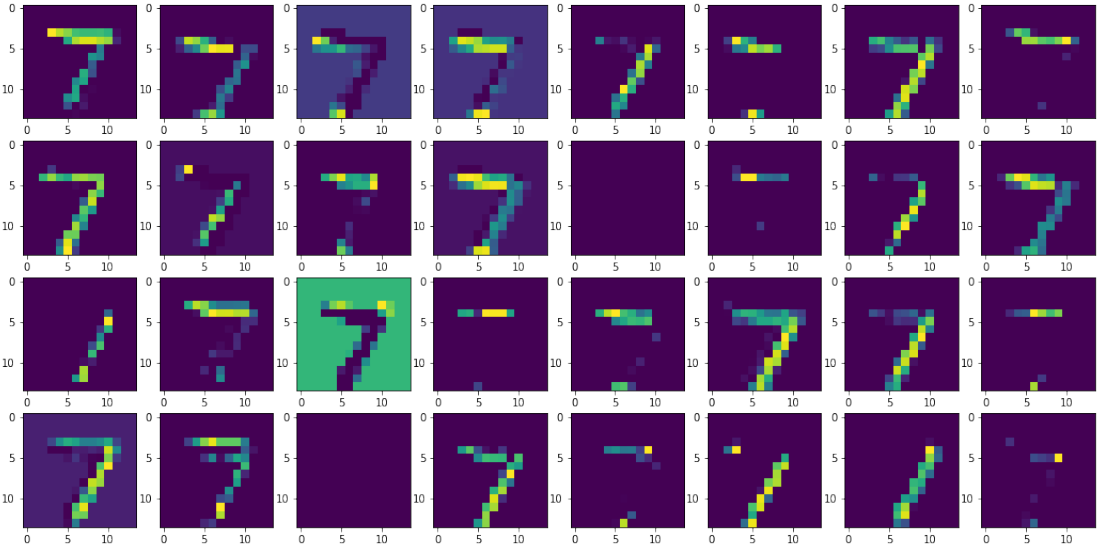

Figure 7: Activation maps of the first convolutional layer

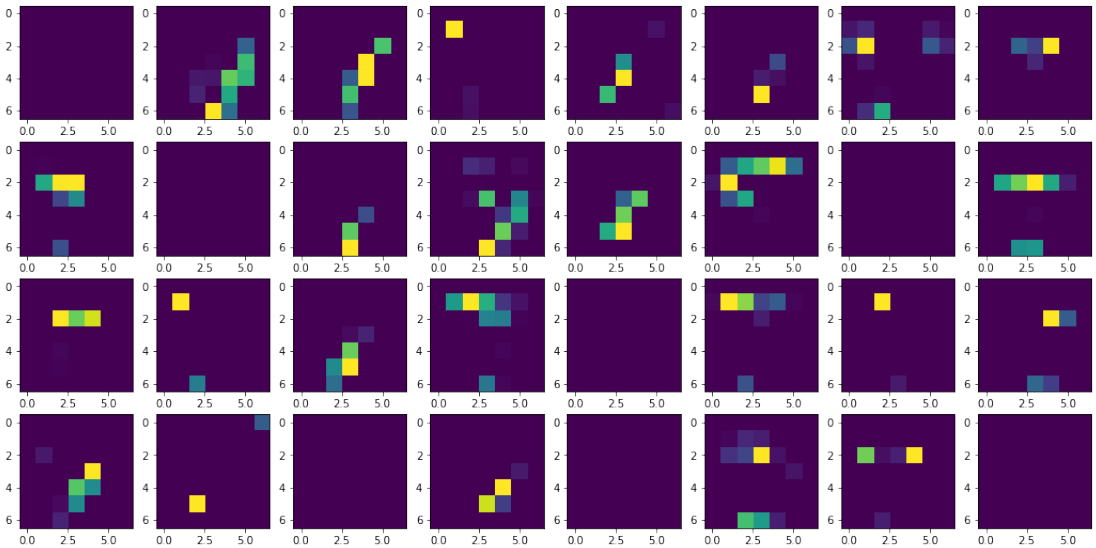

Figure 8: Activation maps of the second convolutional layer

4.1 Non Targeted Attack

In non-targeted attack, we started a matrix with Gaussian noise and try to modify the noise in each iteration so that the model predicts the generated image as some image class. The generated images are shown in Figure 11. As it can be seen that except in some cases like 6 and 8, the other generated images are not comprehensible. Even in case of 6 and 8, only the high level curves are comprehended.

4.2 Targeted Attack

In targeted attack, we initialized a matrix with Gaussian noise and try to modify the noise in each iteration so that the model generates an image. However, we add an additional MSE loss term with respect to some 'target class' with it so that the image looks like the target class but model predicts the generated image as something else. We try to present two cases where the images look like 7 and 2, but the predictions are quite different. The images are presented in Figure 12 and 13. We want our generated image to look as close to the target class, so the MSE Loss should be more penalized

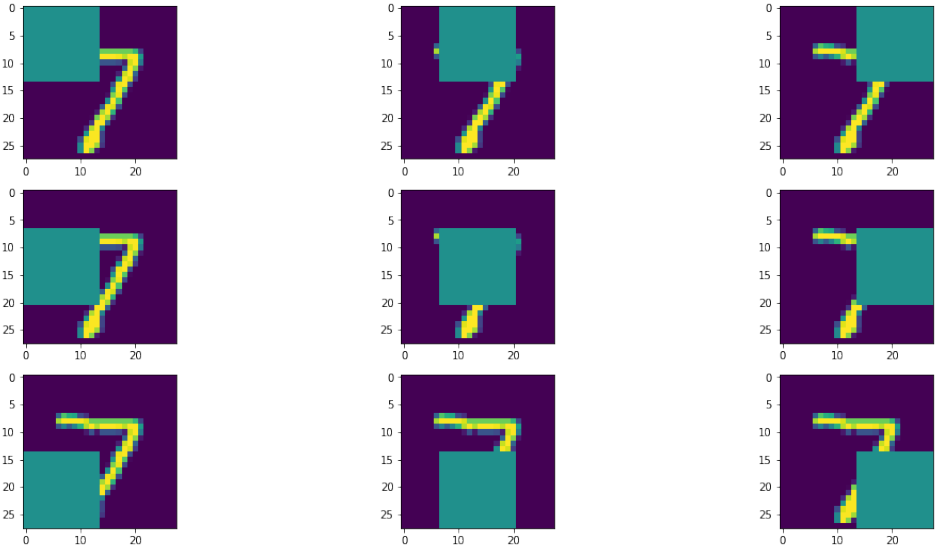

Figure 9: Occlusion in a test image depicting 7 by a gray patch for the nine positions. The figure numbers in Table 2 is calculated from left to right.

Table 2: Model predictions when 7 is occluded.

Figure	**Probability of true class (7)**	**Predicted**
1	0.9999	7
2	0.1734	9
3	0.9956	7
4	0.9859	7
5	0.8804	7
6	0.9999	7
7	0.9982	7
8	0.9853	7
9	0.9847	7

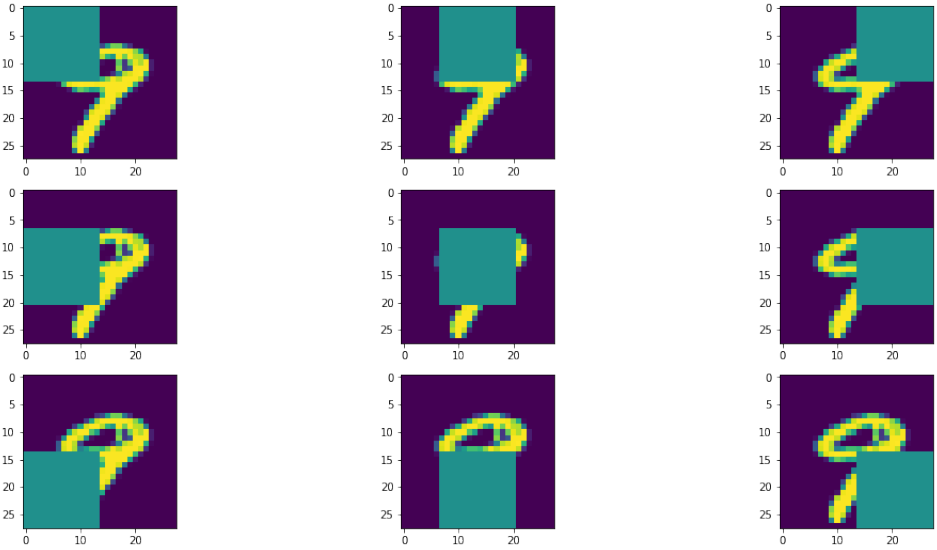

Figure 10: Occlusion in a test image depicting 9 by a gray patch for the nine positions. The figure numbers in Table 3 is calculated from left to right.

Table 3: Model predictions when 9 is occluded.

Figure	Probability of true class (9)	Predicted
1	0.9975	9
2	0.0097	4
3	0.7969	9
4	0.4620	7
5	0.0209	7
6	0.9913	9
7	0.9888	9
8	0.4722	9
9	1.0000	9

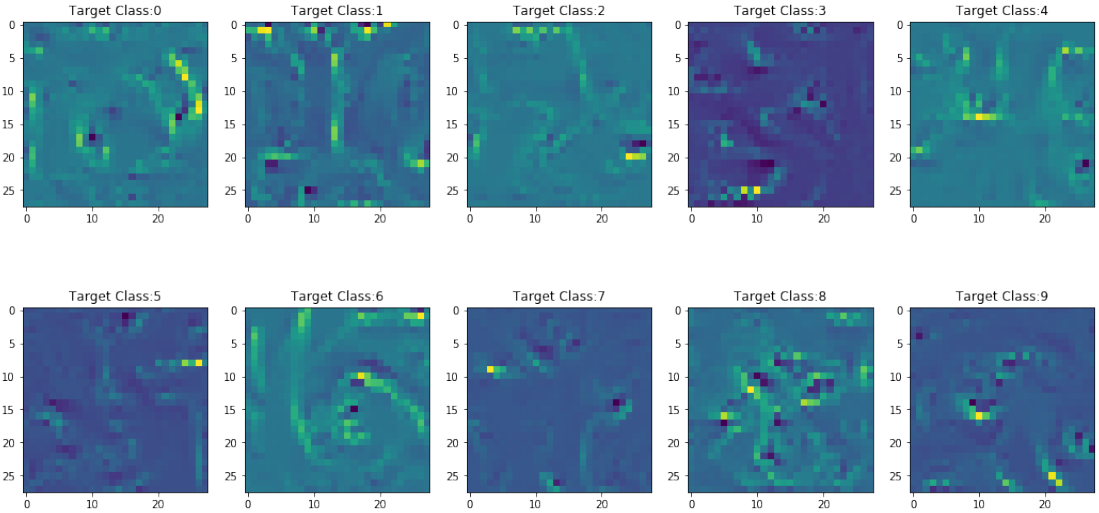

Figure 11: Images generated using non-targeted attack

than the logit loss. We try to modify the image such a way that the image looks like the target class but the model predicts from 0 to 9. The model outputs are obtained at $\beta = 1000$. As seen in Figure 12, we see that even if the images look like 7 in all the cases the model predicts differently in classes like 0,2,3 etc.

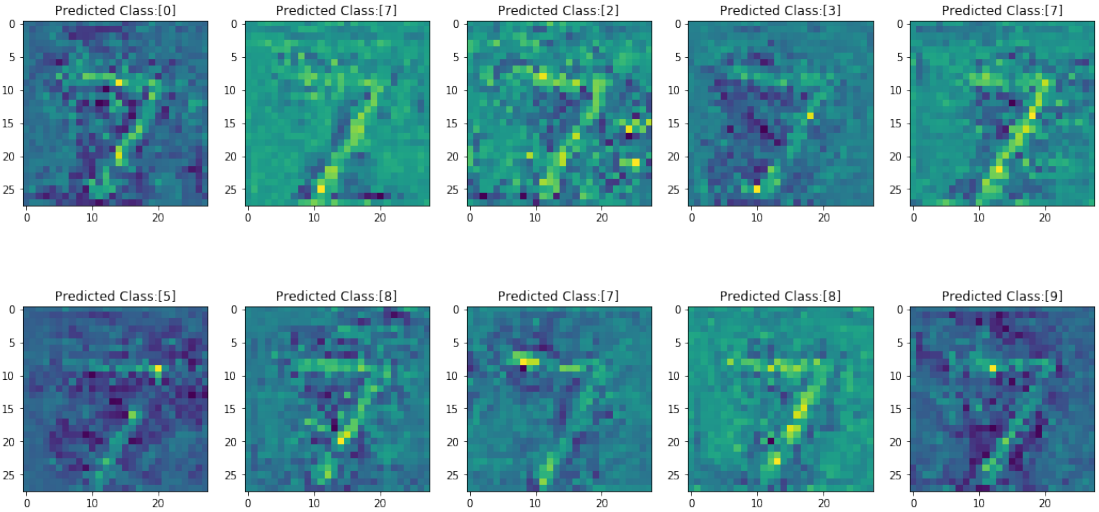

Figure 12: Images generated using targeted attack when the target class is 7. The title of the subplots denote the predicted class for the particular image.

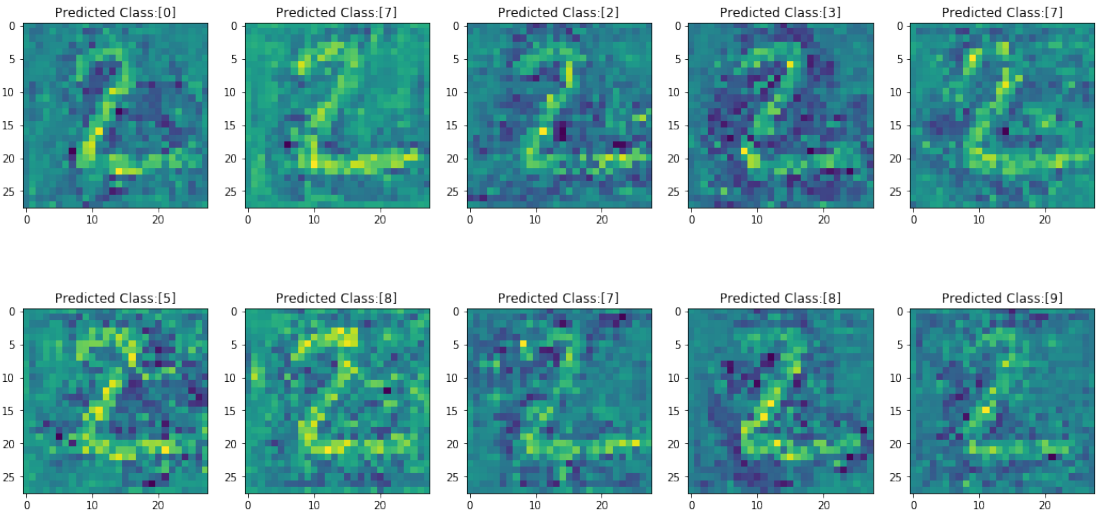

Figure 13: Images generated using targeted attack when the target class is 2. The title of the subplots denote the predicted class for the particular image.

4.3 Adding Noise

In this experiment, we iteratively add noise to a given image and try to classify it as an image from a different class. We have fixed target as 7 in Figure 14 and target as 4 in Figure 15. It is to be noted that even if the noisy image still looks like the original image, the added noise actually makes the model predict a different class as described in Goodfellow et. al [8]. We then pick a final noise matrix and add it to random images and predict the output of the noisy input. This is shown in Figure 16. The noise added to each image is the noise that was added to 1 to misclassiffy as 4. We see that even if rows 1,4 and 6 in Figure 16 look no where like 4, the model still classifies them as 4. In rows 2,3 and 5, the model predicts the true class even for the noisy images as 0 and 6 are harder to "modify" into 4 whereas in the last row the model predicts a different class altogether.

5 Conclusion

In this assignment, we implement a convolutional neural network (CNN) using Pytorch. We trained models with and without using batch normalization. The model using batch normalization takes lesser time to train and there is a slight improvement in accuracy too. We further visualize the filters and activation maps of the CNN and conclude that the lower layers capture basic features like edges whereas the high level features are captured in the higher layers. By occluding the image, we observe that when the main portions of the images are hidden, the model fails to predict the correct class of the image. In the final section, we use adversarial examples to fool the model using non-targeted attack, targeted attack and by adding noise. In case of non-targeted attack, the generated images are not very clear. In case of targeted attack, we observe that even if the image looks like a specific number, the model still classifies it incorrectly. In the final subsection, we try to modify a given image by adding suitable amount of noise to make the model predict incorrect class for the same image.

References

[1] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533âĂŞ536, 1986.

[2] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436, 2015.

[3] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. ImageNet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, 2012.

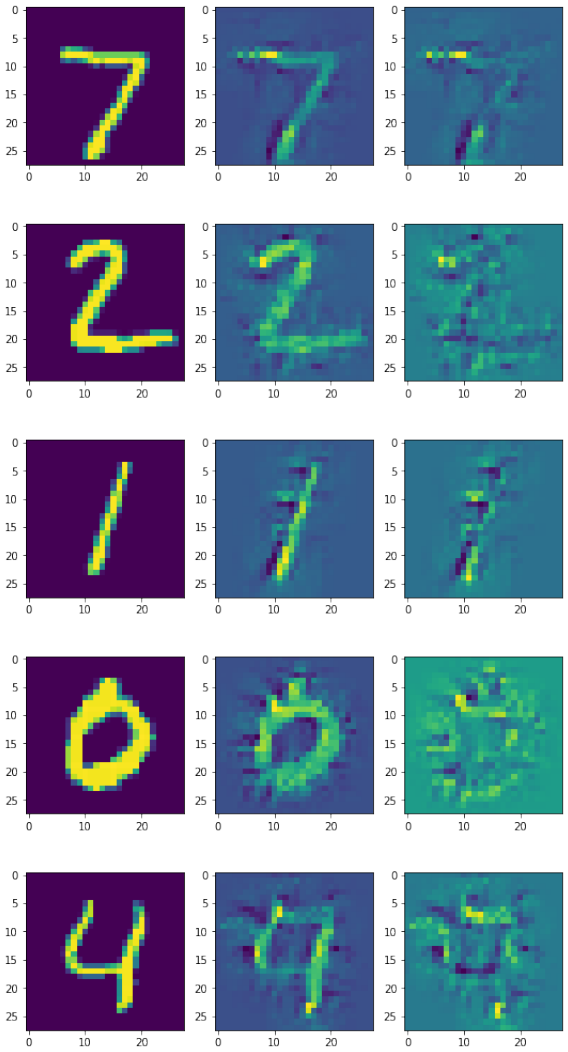

Figure 14: Images generated using adding noise when target class is 7. The first column represents the true image, the second column shows the image after the noise is added and third column depicts the noise. The model predicts 7 for each of the noisy images with very high confidence.

[4] Yann LeCun and Corinna Cortes. MNIST handwritten digit database. `http://yann.lecun.com/exdb/mnist/`, 2010.

[5] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on Machine Learning (ICML)*, pages 807–814, 2010.

[6] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[7] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *32nd International Conference on Machine Learning, ICML 2015*, 2015.

[8] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

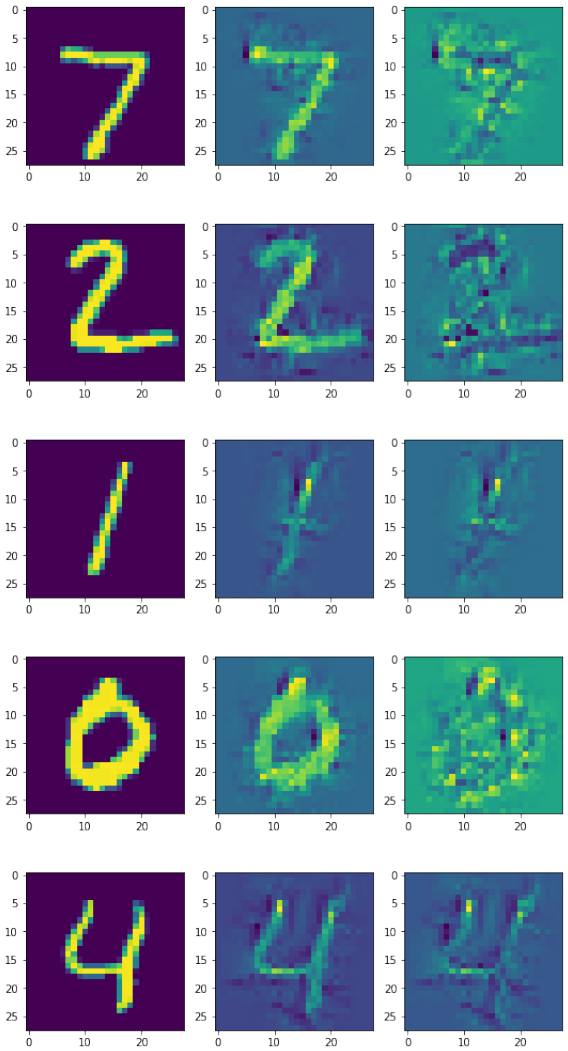

Figure 15: Images generated using adding noise when target class is 4. The first column represents the true image, the second column shows the image after the noise is added and third column depicts the noise. The model predicts 4 for each of the noisy images with very high confidence.

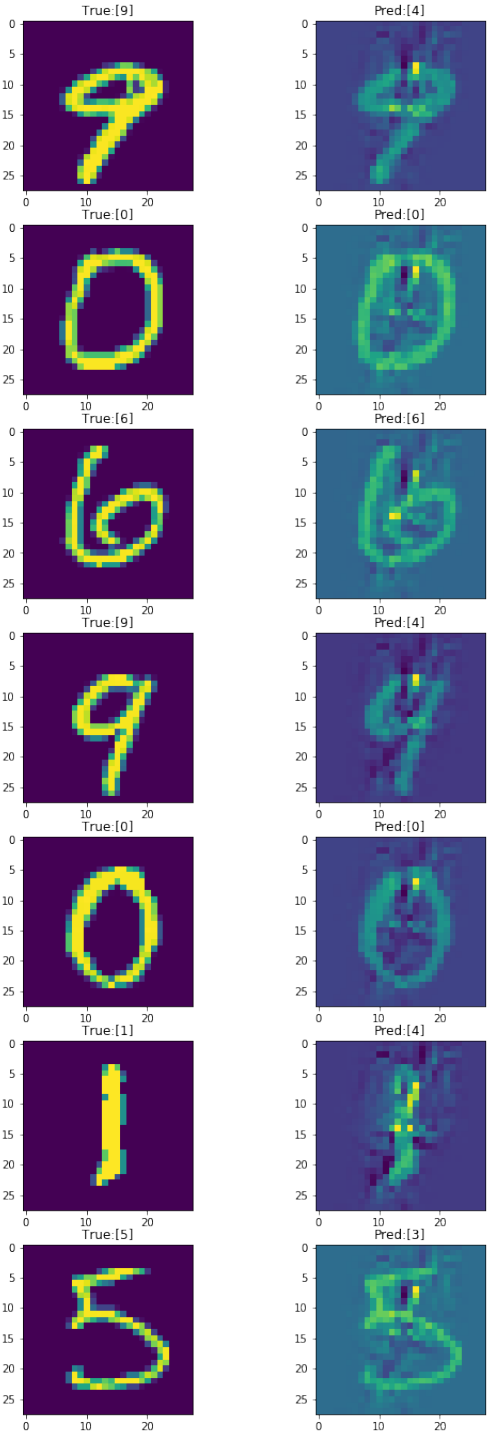

Figure 16: The title of each image on the left are predictions of the model when no noise is added and the title of the images on the right are the predictions when the noise is added.