The problem is formulated as a multiclass classification problem where each image will be assigned to one of the 10 classes.
The input image size was 28 X 28

Report on MNN:

For, MNN I flattened the image into 1 D array of size 784 X1 before feeding into the Network.
So, the input layer of my MNN consists of 784 nodes.
Input layer size=784
Model1: With 4 hidden layers (H1,H2,H3,H4 namely) and one output layer of size 10.
H1: has 192 nodes
H2: has 96 nodes
H3: has 48 nodes
H4: has 20 nodes
Output layer has 10 nodes as the number of classes = 10.
I tried with signle hidden layer, then 2 hidden layers and 3 hidden layers as well but found this architecture suits the best with same number of parameters (each hidden layer
Number of nodes I selected thinking about each of the layers will keep reducing the feature size into half of the size in the previous layer.
For, MNN we are feeding all the pixel values (784 pixels) as input features and each of the hidden layer reduces the feature vector size by 2 and in the end the output layer generates 10 scaler value which will further be used to find the class probability for each of the 10 classes. As I have used cross entropy function of pytorch, nn.CrossEntropyLoss(), we can send only the logits value, the loss function itself will compute the softmax probability to compute the loss.

Total number of parameters in the MNN=411625 (very high so computationally expensive), but the MNN fails to capture the spatial information of the pixels.

Training Detail:
I splitted the training set into 80:20 ratio for training and validation respectively.
BatchSize was 1000.
I did not use the test data for validation. I only have used the test data during final testing with the trained model (main.py).

I trained for 60 epochs with initial learning rate=.001 and weight decay=1e-4 using Adam optimizer. Then after 30 iteration as the validation loss was increasing, I stopped the training without saving the states and again ran for 30 iteration from the begining with same initial learning rate which gives 89% accuracy and at 30th iteration I decreased the learning rate to .00001 and

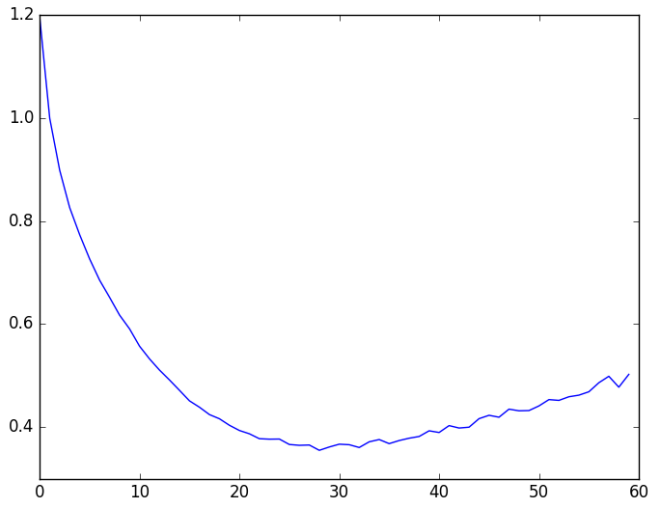

Fig1.1: Validation cross entropy vs nymber of epochs without early stopping

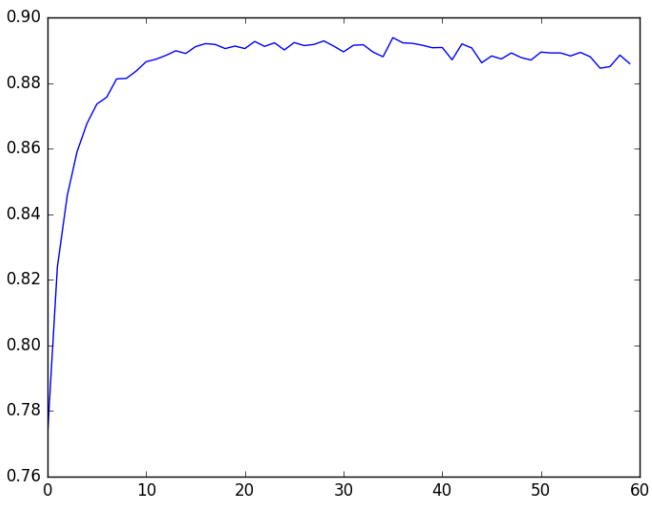

Fig1.2: Validation accuracy vs nymber of epochs without early stopping

ran for another 30 iterations which obtained validation accuracy of 92% and then again ran 10 more iterations with further decreasing the learning rate to .0000001 and validation through out these 10 iteration the validation accuracy was almost constant and the best accuracy obtained was 96.3% as we can see that after 8th iteration val loss tends to increase so, we stopped at the 10th iteration.

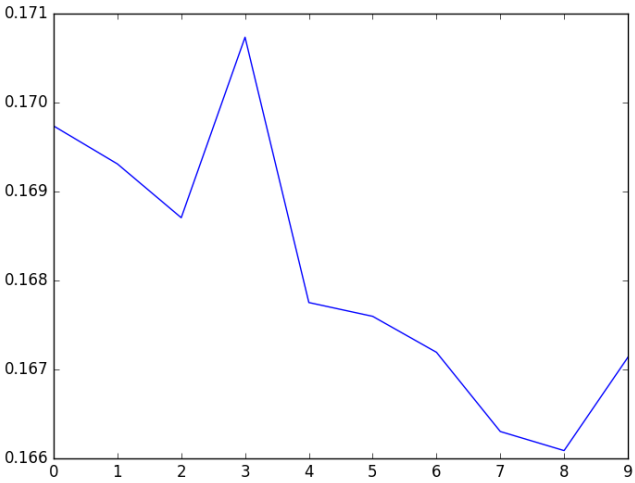

Fig2.1: Validation Loss vs number of epochs with early stopping (the loss shown after Ist 30 iterations were performed with Initial learning rate= .0001in the Ist phase)in the 2nd phase of the training

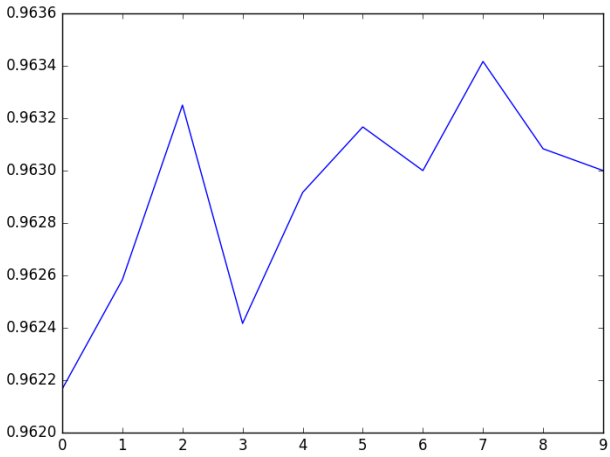

Fig2.2: Validation Accuracy vs number of epochs with early stopping (the loss shown after Ist 30 iterations were performed with Initial learning rate= .0001in the Ist phase)in the 2nd phase of the training

Now before the last hidden layers I added dropout layer with probability=.5 and with initial learning rate =.0001 and using Adam and with weight decay 1e-4 the accuracy obtained was 81.85% which was 89% without dropout. So, adding dropout I did not see any improvement.

Model2:
Now another architecture(H1=200, H2= 100, H3=50, H4=50) gives with 184945 number of parameters give the similar accuracy of 89% after 30 epochs with initial learning rate =.001 and using Adam optimizer. Further, running 30 iteration after decreasing the learniung rate to .00001 we obtain 90% validation accuracy.

Model2:
Now, using only 2 hidden layers (H1=400,H2=100) gives 93% validation accuracy even the total number of parameters = 356110. But, as the number of hidden layers are less, less non linearity was captured.

Model4:
Now another architecture(H1=200, H2= 100, H3=50) gives with 183360 parameters best validation accuracy was obtained 90% (Ist 10 iterations with initial learning rate =.001 adam optimizer and weight decay=1e-4 were used and next 20 iteration with initial learning rate =.00001 adam optimizer and weight decay=1e-4 were used, after a total of 10+20=30 iterations

So considering the trade-off between best obtained validation accuracy and number of parameters used I finally selected the model4 which consists of 3 hidden layers (H1=200, H2= 100, H3=50)

Below are the graphs of validation loss and validation accuracy obtained from final MNN model:

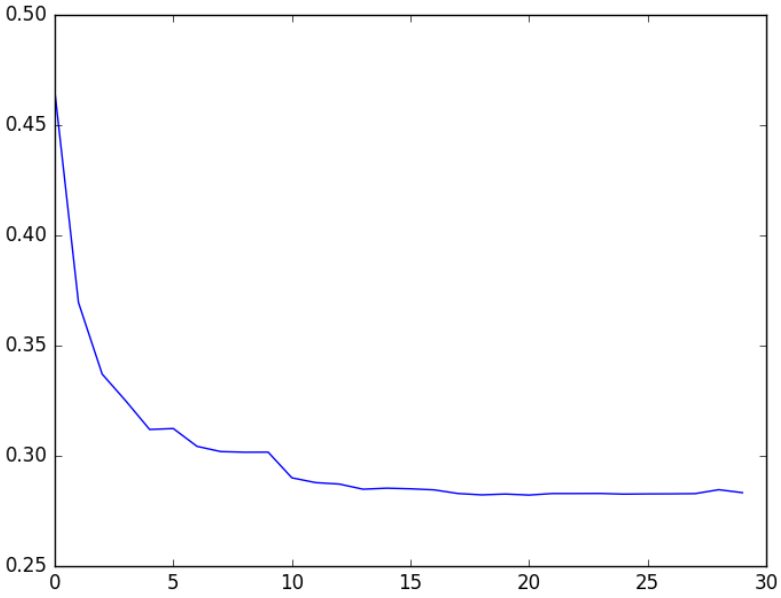

Fig 3.1: ValCrossEntropy vs number of epochs

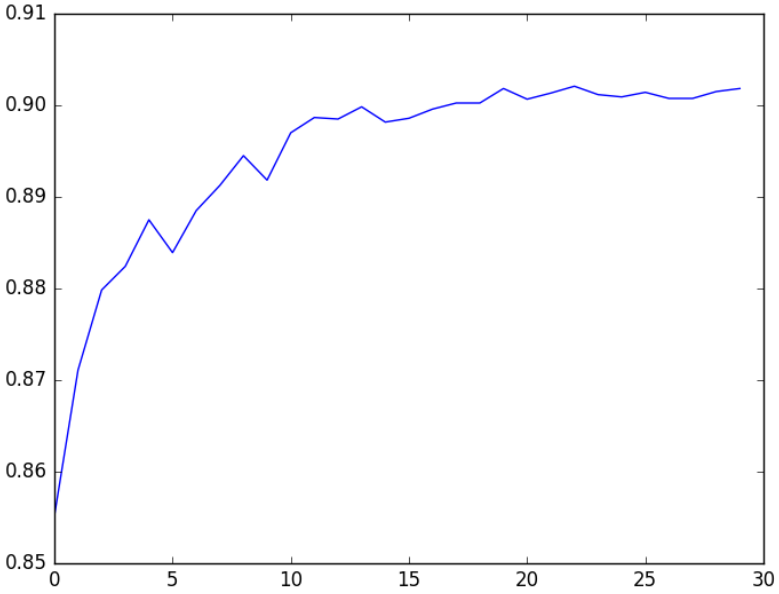

Fig 3.2: Val Acc vs number of epochs

Report on CNN:

For CNN, the architecture I first proposed is illustrated below.
A Convolution layers with 14, 3X3 filters, then another Conv layer with 14, 3X3 filters followed by a Batch Normalization layer, ReLU as an activation function and a 2X2 maxpool layer, then again 8, 3X3 filters followed by a ReLU, another 4, 3X3 filter followed by a ReLU, and again another 3, 3X3 filters followed by a 2X2 maxpool layer, ReLU as an activation function and a final convolution with 10, 3X3 filter gives 10 layers of output , where each layer is of dimension 1X1, so these 10 values can be considered as logits for the individual classes for the input image. As mentioned earlier, we can send the logit directly to the CrossEntropy Function which itself compute the class probability using softmax internally.

First, I ran for 50 iterations with initial learning rate as .0001(default for Adam in Pytorch) and weight decay as 1e-4 the following result I get in the validation set. Accuracy obtained was 85%.

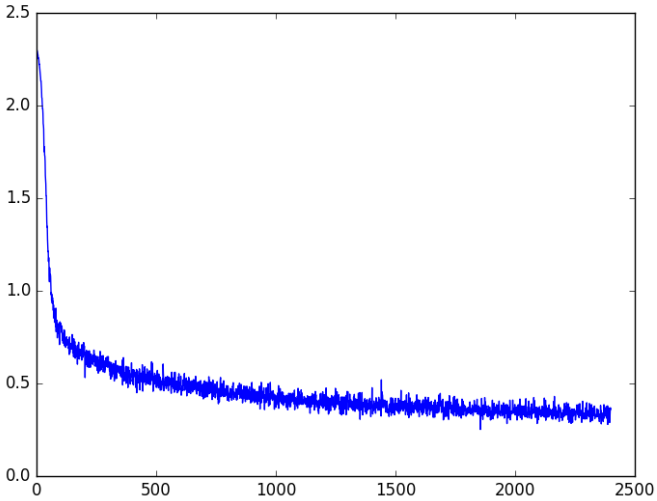

Fig 4.1 TrainLoss VS numberOfEpoch

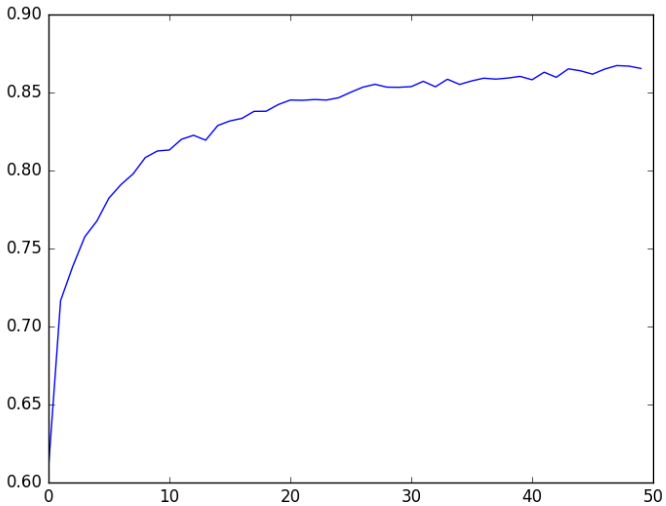

Fig 4.2 ValLoss VS numberOfEpoch

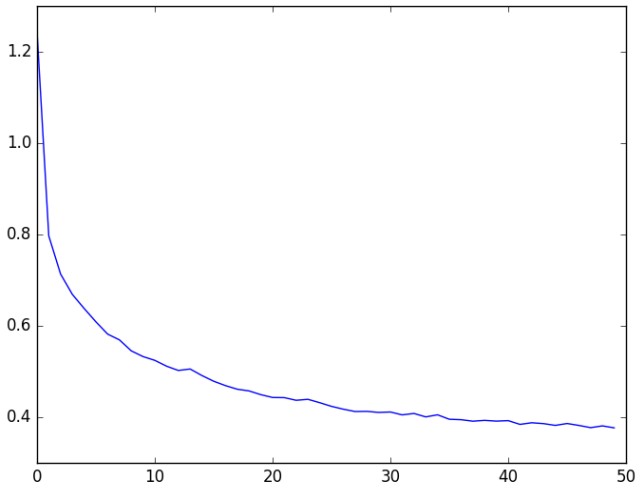

Fig 4.3 ValAcc VS numberOfEpoch

Then another 10 iterations with lr=.00001 gives 86.7% accuracy, hence. So, I again re ran from the begining with .001 for first 50 and next 30 iterations (instead of 10 this time) with .00001, which also gave no significant imprivement in the accuracy. So, again I re-ran for 80 iterations where after 50^{th} iteration I decreased learning rate to .00001 and then after 60^{th} iteration I again decreased to 0.0000001 and finally in the end of 80 iterations I obtain 88% validation accuracy. ***Below are the final plots for CNN model:***

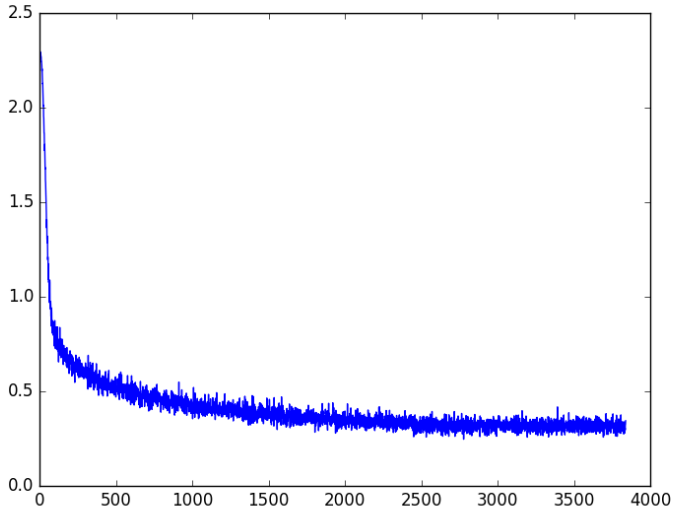

Fig 5.1 TrainLoss VS numberOfEpoch

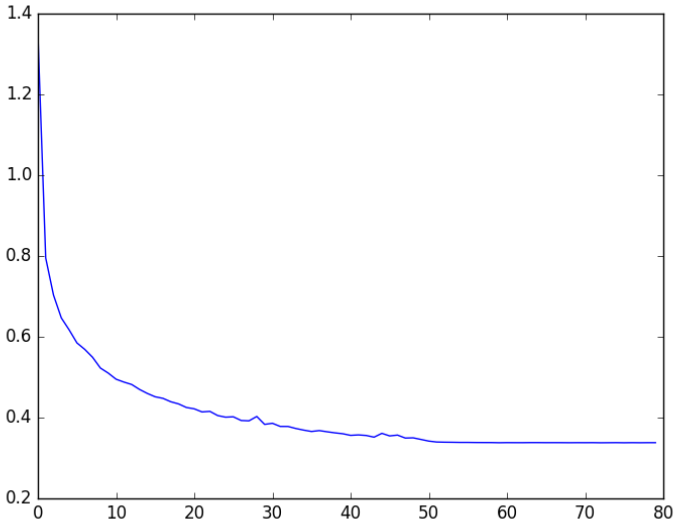

Fig 5.2 ValLoss VS numberOfEpoch

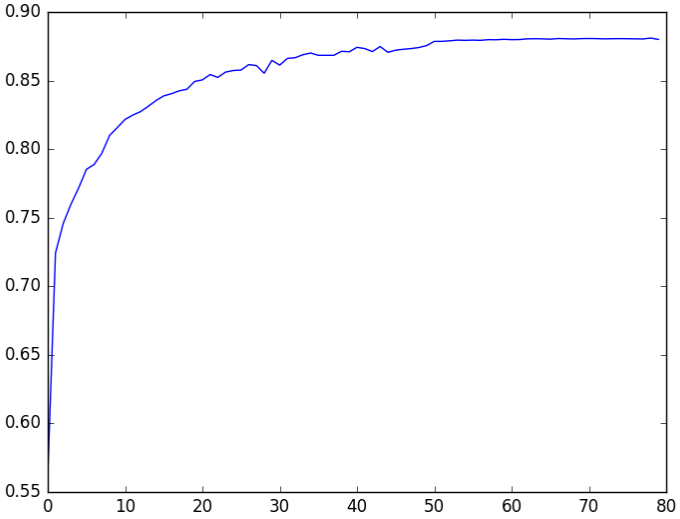

Fig 5.3 ValAcc VS numberOfEpoch

Total number of parameters used **in the above architecture = 3772** which is **far less than the number of parameters used in the final MNN model(Model 4) which is 183360** . Still the CNN

architecture produces a very close accuracy. **This happens because CNN architecture can extract better features as it can exploit the spatial correlation between pixels of an image.**
As, the image size is 28X28 I could easily reduce the size to 1X1 with Conv and MaxPool layers without using any fully connected layer.

Also, for larger images this is not feasible to come to the scaler value with series of only Conv and MaxPool. We add one or two fully connected layer in the end for a deep network.

I, also checked with using 2 linear layers in the end as I removed the last 3 conv layer from the above architecture and used only 1 filter in the 5^{th} ConvLayer (rest architecture is same as the above CNN model). But, although the **number of features became 4147 (>3772 as was in previous architecture) I could not obtain better accuracy(obtained only 86%). So, the previous architecture I used as final architecture for CNN.**