Project 5 : Recognition using Deep Networks
Arun Madhsudhanan

Project Description:
In this project, I built and trained a network to recognize the MNIST digit. The network is tested on both test set and custom inputs. The first layers of the network were analyzed to understand how the layers process the data . Later, the network was modified and trained to detect Greek Letters. The final task was to experiment by changing the different aspects of the network and how it affects the performance.
As part of the extensions, a live digit recognition system based on the network is implemented. Additionally, the first layer of the network was changed to Gabor filter bank to understand how it would affect the performance. Another pre-trained network called alexnet is loaded and analyzes the first couple of layers to understand how it processes the data.

Task Performed

Task 1: Build and train a network to recognize digits

A : Get the MNIST digit data set

The matplotlib pyplot is used to look at the first six example digits.

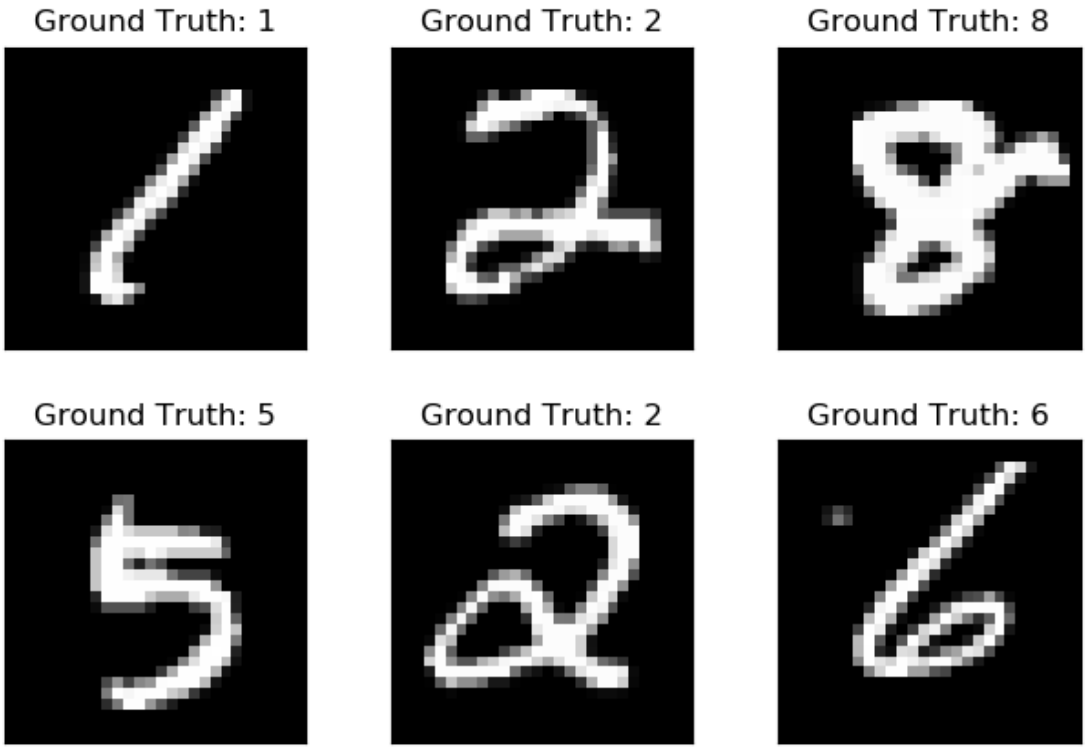


Figure 1 : first six example digits

C : Build a network model

The first layer is a convolutional layer with 10 filters of size 5x5, followed by a max pooling layer with a window size of 2x2 and a Rectified Linear Unit (ReLU) activation function applied. The second convolutional layer has 20 filters of size 5x5, followed by a dropout layer with a dropout rate of 0.5. Another max pooling layer with a window size of 2x2 and ReLU activation function is then applied. The output from this layer is flattened and fed into a fully connected Linear layer with 50 nodes and ReLU activation function. Finally, a fully connected Linear layer with 10 nodes and the log_softmax function applied to the output is used for classification.

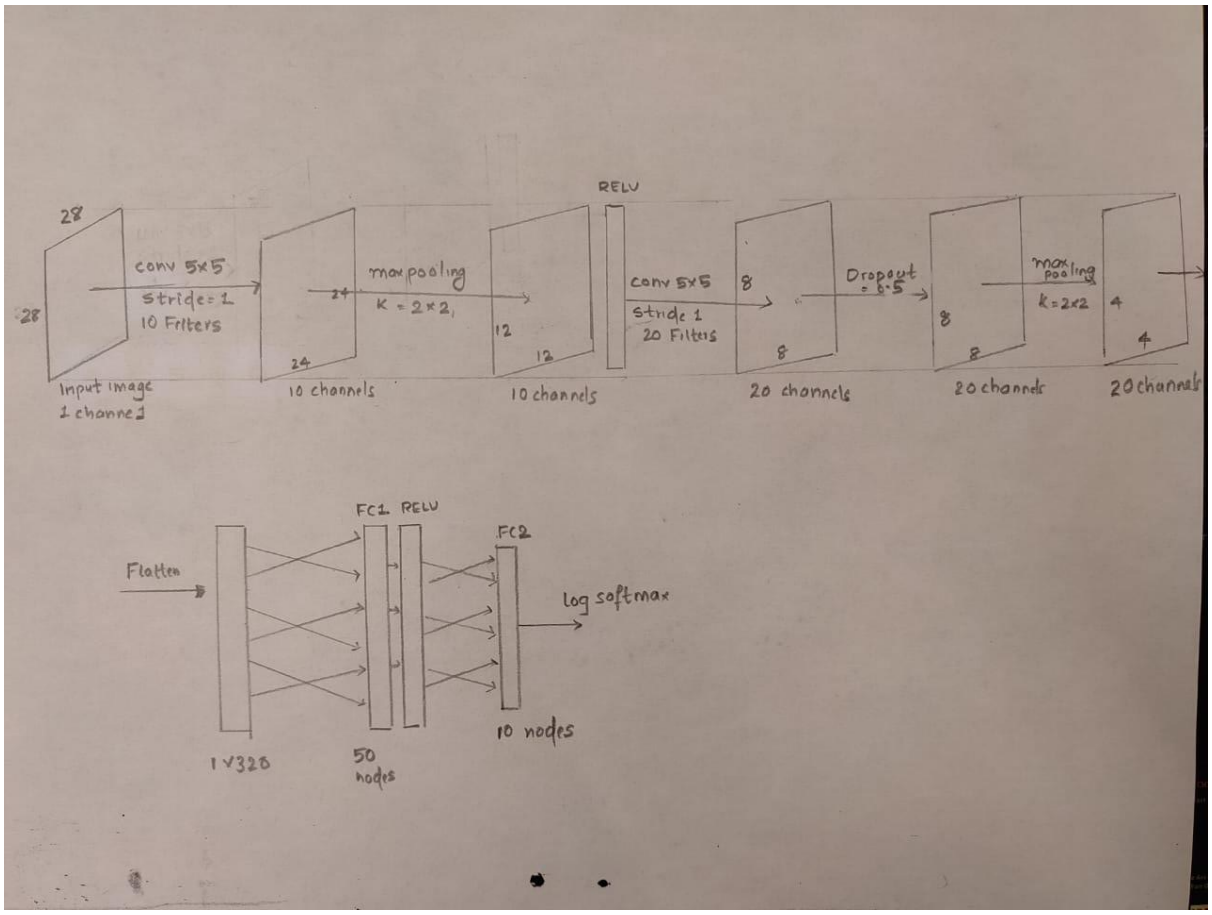


Figure 2: network diagram

D : Train the model

The training of the model occurs over five epochs, with one epoch being completed at a time. Following each epoch, the model's performance is assessed on both the training and test sets. During the training process, a batch size of 64 is utilized.

An accuracy of 98.16% is obtained as shown below

```
Test set: Avg. loss: 0.0555, Accuracy: 9816/10000 (98.16%)
```

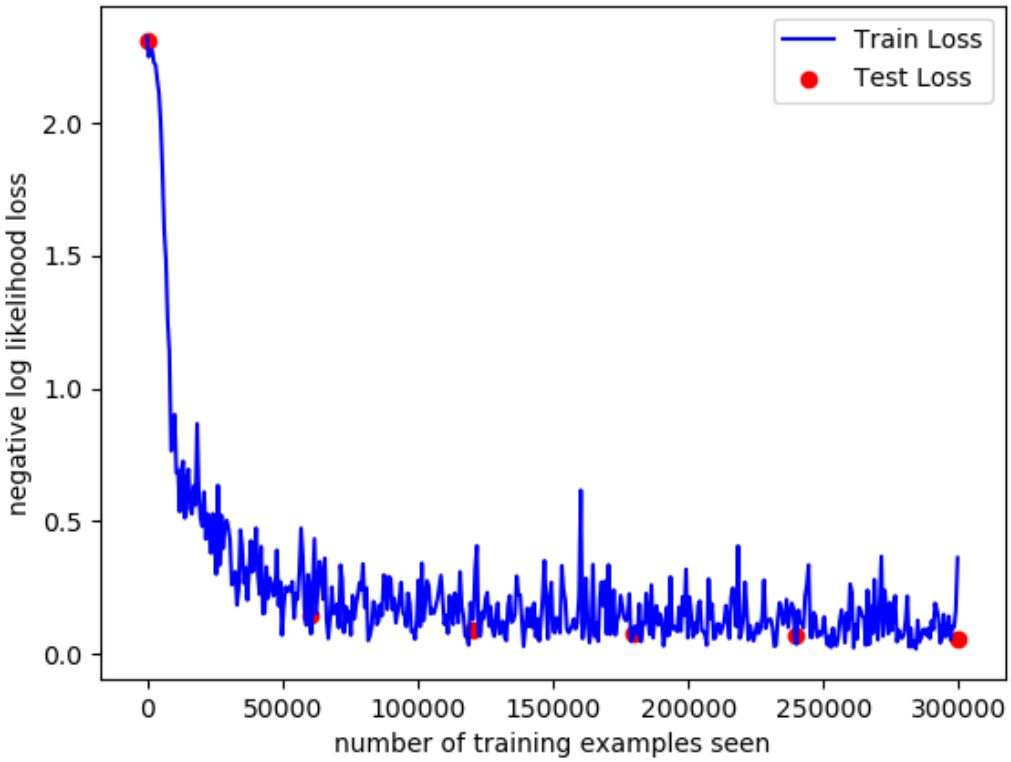


Figure 3: Train and test loss

F: Read the network and run it on the test set

Using a separate python file, the network was imported and ran on the first 10 examples in the test set. The program prints out the 10 output values , the index of the max output value, and the correct label of the digit as shown in figure 4.

```
Output value of 1th digit : tensor([[-2.21e+01, -2.32e+01, -2.15e+01, -1.26e+01, -1.22e+01, -1.48e+01,
         -2.84e+01, -1.04e+01, -1.30e+01, -4.27e-05]])
Index of max output value of 1th digit : 9
Correct label of 1th digit : 9
Output value of 2th digit : tensor([[-11.07, -11.17,  -3.96,  -0.08, -10.17,  -6.51, -17.24,  -3.34,  -4.94,
          -4.03]])
Index of max output value of 2th digit : 3
Correct label of 2th digit : 3
Output value of 3th digit : tensor([[-2.83e-04, -2.36e+01, -1.36e+01, -1.54e+01, -1.31e+01, -1.37e+01,
         -8.67e+00, -2.16e+01, -1.37e+01, -9.16e+00]])
Index of max output value of 3th digit : 0
Correct label of 3th digit : 0
Output value of 4th digit : tensor([[-1.81e+01, -1.43e+01, -1.75e+01, -9.53e+00, -4.83e+00, -1.30e+01,
         -2.10e+01, -9.58e+00, -1.12e+01, -8.21e-03]])
Index of max output value of 4th digit : 9
Correct label of 4th digit : 9
Output value of 5th digit : tensor([[-1.32e+01, -2.26e+01, -1.96e+01, -5.66e+00, -1.88e+01, -3.88e-03,
         -1.14e+01, -2.26e+01, -8.46e+00, -8.77e+00]])
Index of max output value of 5th digit : 5
Correct label of 5th digit : 5
Output value of 6th digit : tensor([[ -8.14,  -0.07,  -4.03,  -6.73,  -7.42,  -8.42, -10.73,  -3.49,  -4.97,
          -5.08]])
Index of max output value of 6th digit : 1
Correct label of 6th digit : 1
Output value of 7th digit : tensor([[-1.42e+01, -5.97e+00, -1.01e+01, -1.15e+01, -3.10e-03, -1.18e+01,
         -1.21e+01, -8.59e+00, -8.69e+00, -9.02e+00]])
Index of max output value of 7th digit : 4
Correct label of 7th digit : 4
Output value of 8th digit : tensor([[-1.69e+01, -1.45e+01, -1.18e+01, -7.02e+00, -1.70e+01, -1.20e+01,
         -2.02e+01, -1.36e+01, -1.04e-03, -8.96e+00]])
Index of max output value of 8th digit : 8
Correct label of 8th digit : 8
Output value of 9th digit : tensor([[-6.68e-06, -2.26e+01, -1.32e+01, -1.67e+01, -2.10e+01, -1.58e+01,
         -1.30e+01, -2.15e+01, -1.39e+01, -1.35e+01]])
Index of max output value of 9th digit : 0
Correct label of 9th digit : 0
Output value of 10th digit : tensor([[ -5.82,  -0.08,  -4.23,  -6.21,  -5.22, -10.45,  -9.81,  -7.01,  -3.04,
          -7.61]])
Index of max output value of 10th digit : 1
Correct label of 10th digit : 1
```

Figure 4: Screenshot of printed values

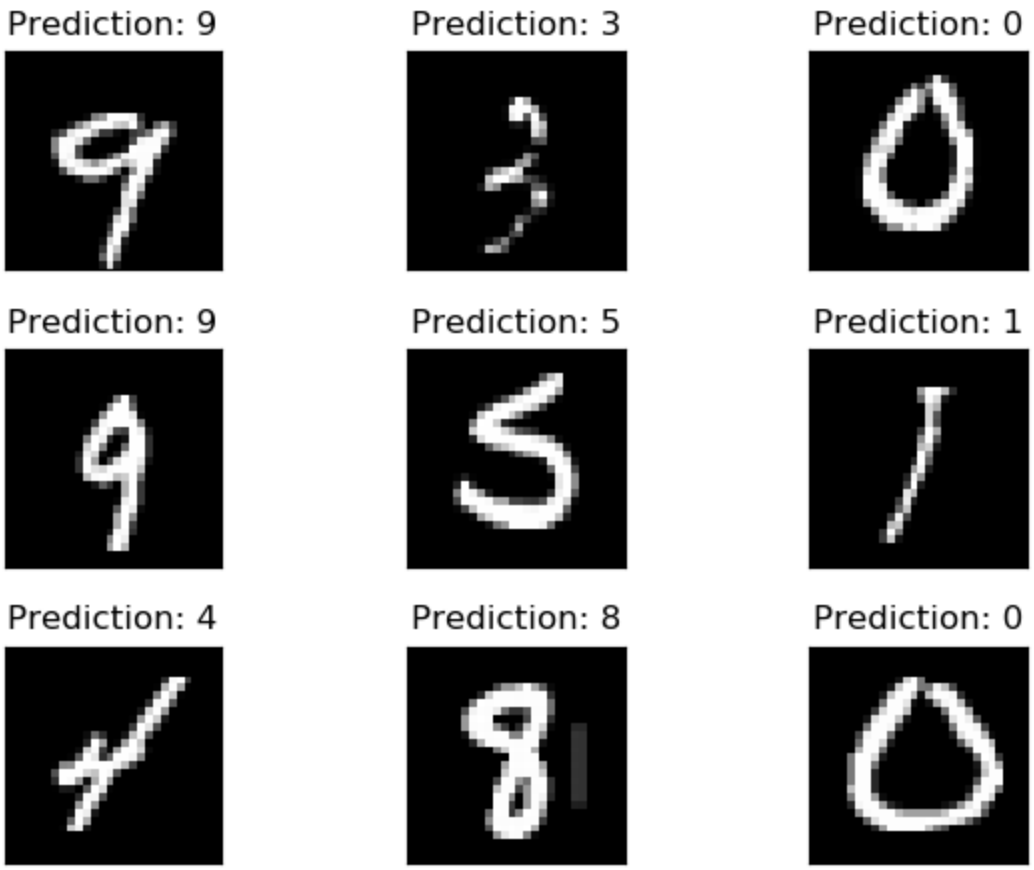


Figure 5: Screenshot of printed values and the plot of the first 9 digits.

The model was able to predict all the digits accurately.

G: Test the network on new inputs

In this task, I wrote 10 digits [0 -9] in a white paper and the network was tested to recognize these digits. All the images were pre-processed to resize the image to 28x28, to convert to grayscale and invert them

The program displays the first 9 images in the custom test and its predictions input as shown in figure 6.

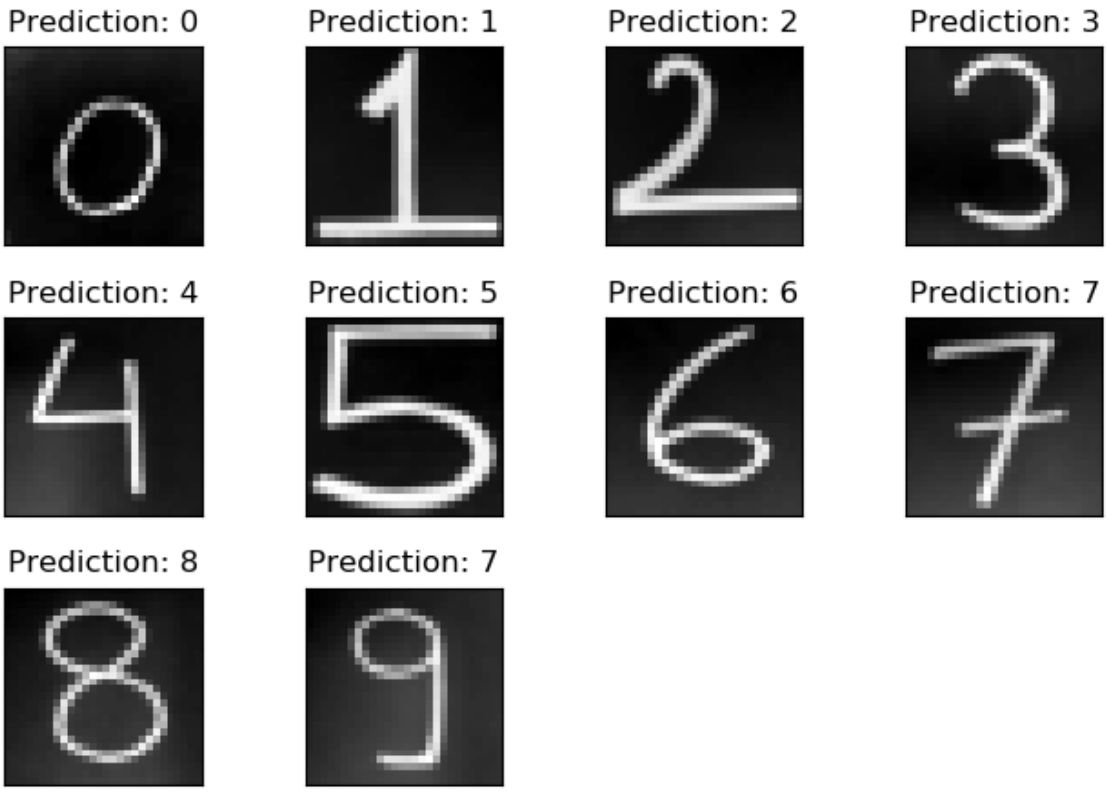


Figure 6 : Predictions for the first ten digits in custom dataset

The model was able to predict all the digits except 9 (90% accuracy). The digit 9 was predicted as 7 incorrectly. This might be due to the difference between intensity, clarity of the handwritten images and training set images. The model itself is not 100% accurate.

Task 2: Examine your network

A: Analyze the first layer

The ten filters of the first convolution filters are visualized as shown in figure 7.

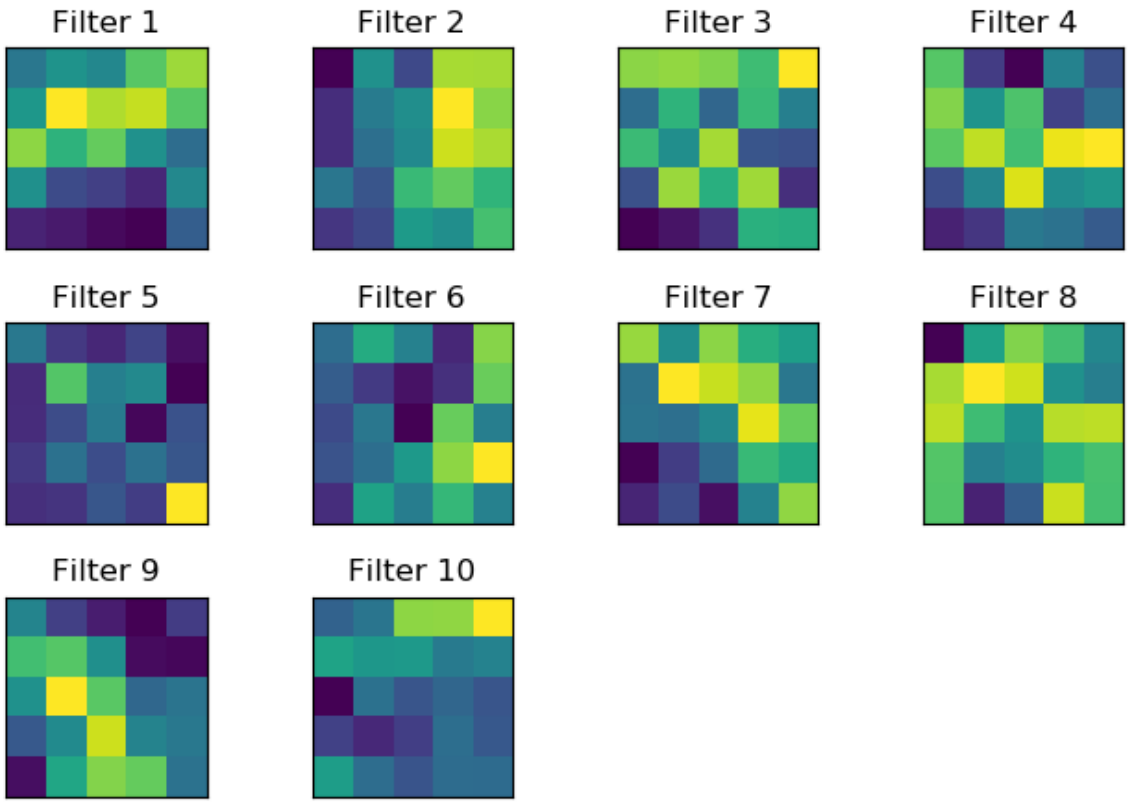


Figure 7: Visualization of first 10 filters.

B: Show the effect of the filters

OpenCV's filter2D function is used to apply the 10 filters to the first training example image.

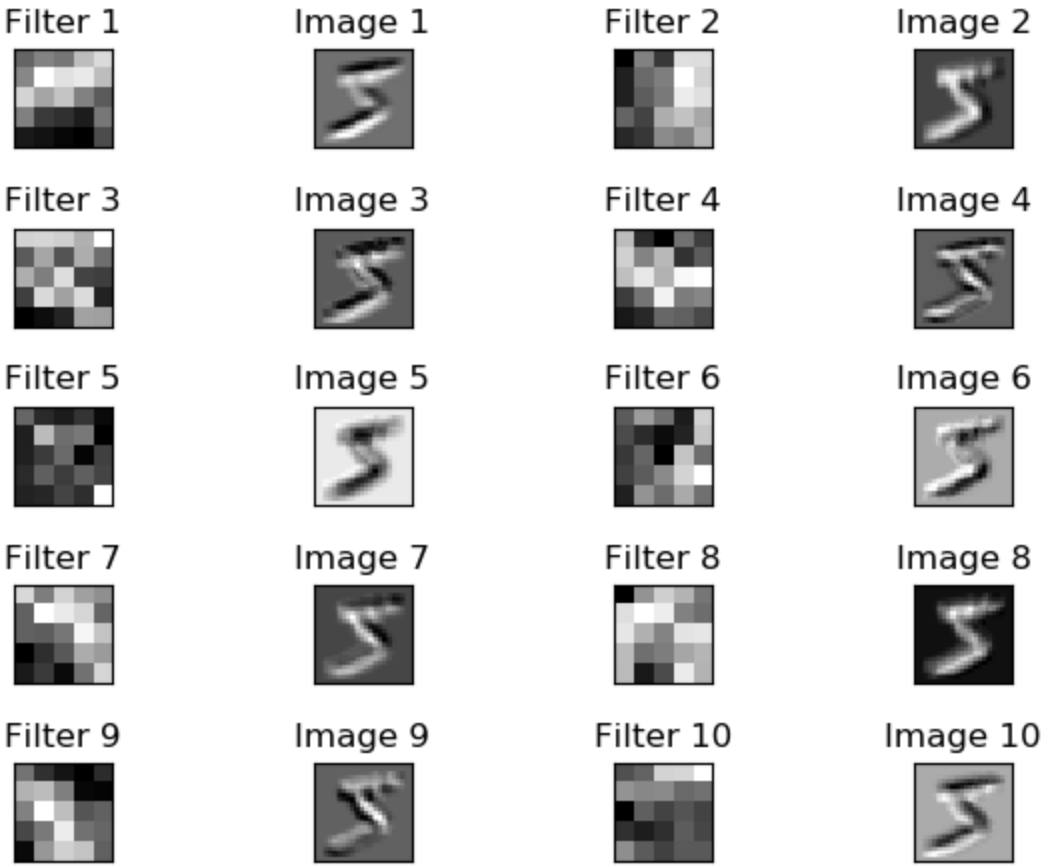


Figure 8: Visualization of first 10 filters.

Based on the observations, it appears that filters 1-4, 6-7, and 8-10 are capable of detecting edges within an image. Filter 5, on the other hand, seems to invert the intensity of the image. Additionally, filter 8 appears to segment digits from the background.It is logical that the majority of the filters are detecting edges that are oriented differently. For instance, the first filter in the first row appears to identify horizontal edges as the top side is brighter while the bottom side is darker. Similarly, filter 2 highlights vertical edges since the left side is darker and the right side is brighter.

Task 3: Transfer learning on Greek Letters

In this task, the MNIST digit recognition network is reused to detect three different greek letters: alpha, beta and gamma. The last layer was replaced with a new linear layer with three nodes. The modified network is shown in Figure 9.

```
Mynetwork(
  (conv1): Conv2d(1, 10, kernel_size=(5, 5), stride=(1, 1))
  (conv2): Conv2d(10, 20, kernel_size=(5, 5), stride=(1, 1))
  (conv2_drop): Dropout2d(p=0.5, inplace=False)
  (fc1): Linear(in_features=320, out_features=50, bias=True)
  (fc2): Linear(in_features=50, out_features=3, bias=True)
)
```

Figure 9: Printout of modified network

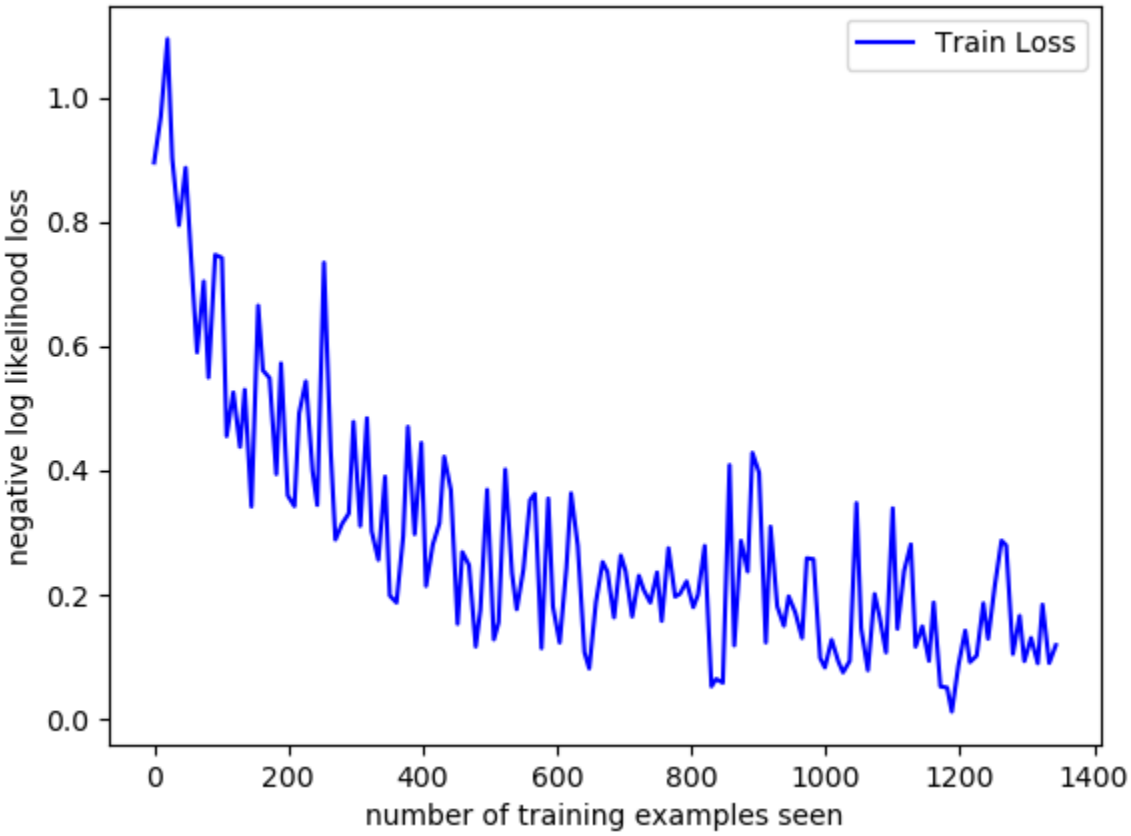


Figure 10: Training Loss

The training is done for an epoch of 50. But according to the terminal logs, the network attained 100% accuracy after only 11 epochs. As a result, the number of epochs was decreased from 50 to 18 to prevent the risk of overfitting.

The model is tested on several handwritten alpha, beta and gamma letters. The predictions are shown in figure 11. The test images were captured using a smartphone and resized directly to 28 x 28 by using *transforms.Resize((28, 28)* function (app

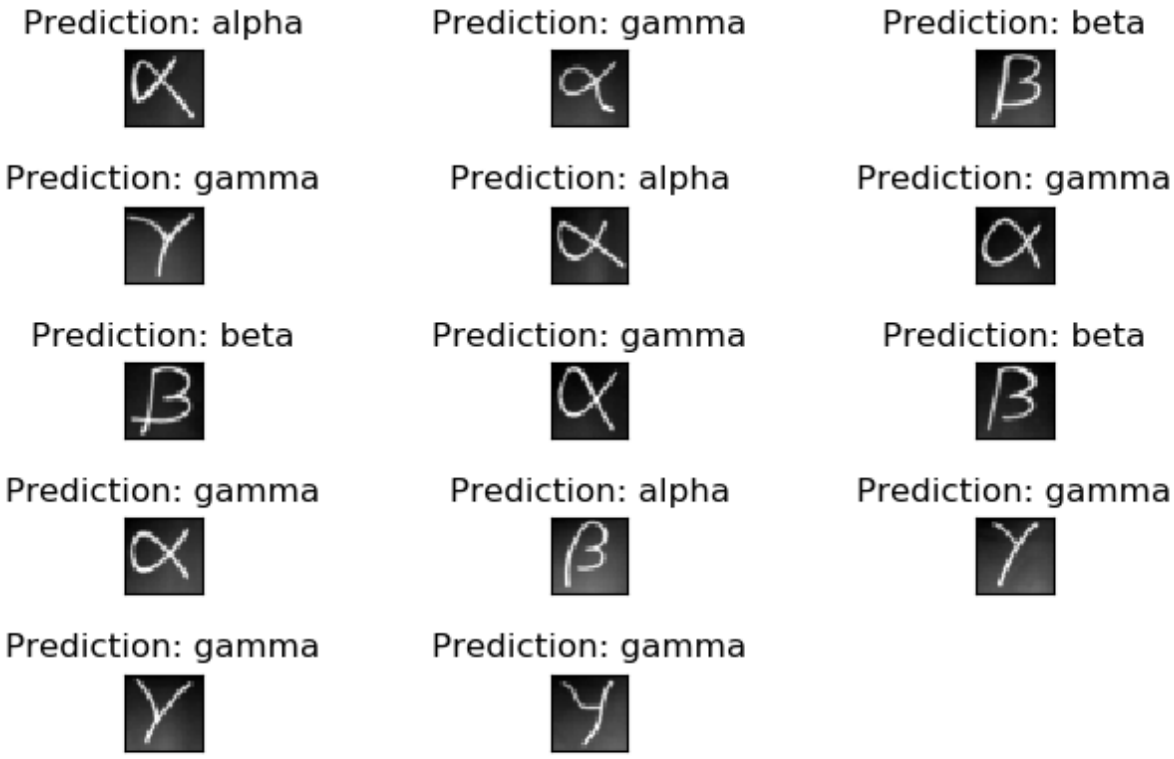


Fig 11 :Recognition on hand drawn greek digits - Approach 1

As shown in the figure below, alpha letters were detected correctly 2 out of 6 times. Beta letters were detected correctly 3 out of 4 times and Gamma letters were detected correctly 4 out of 4 times. The total accuracy is 64%. I think this is satisfactory considering the fact that the number of images in the training dataset were less.

Another approach used for generating the input for testing was first cropping images to 128 x 128 to use Greektransform() for preprocessing. However the accuracy dropped to 50% as shown in figure 12. Hence I retained the first approach for preprocessing images.

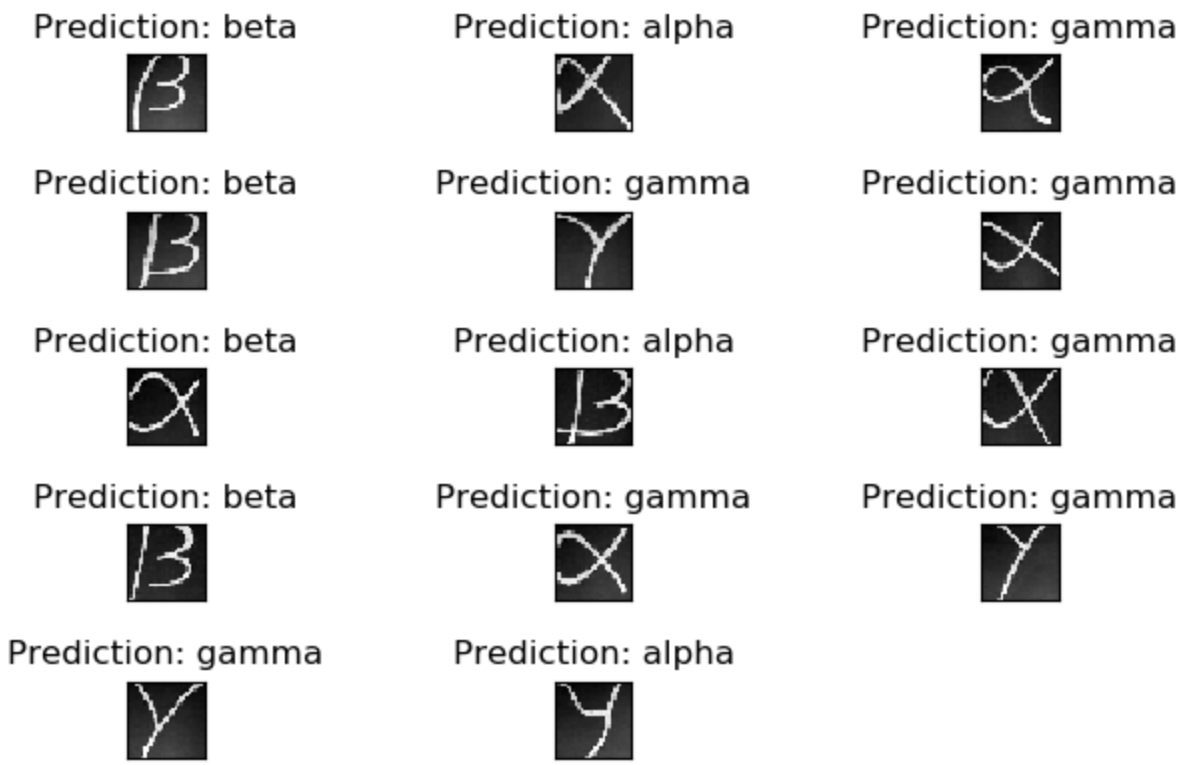


Fig 12 :Recognition on hand drawn greek digits - Approach 2

Task 4: Design your own experiment

In this task, few parameters of the deep network for the MNIST digit recognition were changed inorder to evaluate the effect of changing different aspects of the network. Following are the parameters changed.

1. Number of epochs, values to try: 5,10,15
2. Batch size, values to try: 32,64,128
3. Dropout rate, values to try: 0.3 ,0.4, 0.5
4. Filter size, values to try: 3,5 and 7

Hypothesis:

Number of epochs: Increasing the number of epochs is expected to enhance the model's accuracy by allowing it to train for longer and update its weights more frequently. However, too many epochs might cause overfitting to occur, which could reduce the model's performance on the test set.

Batch size:
Increasing the batch size is predicted to speed up the model's training process since it can process more examples simultaneously before updating its weights. However, larger batches may require more memory. Smaller batch sizes may produce more noisy gradients, leading to slower convergence.

Dropout rate: Increasing the dropout rate is expected to create a more regularized model that generalizes better to new data, but it may also decrease the model's capacity to learn complex patterns from the data.

Filter size: Increasing the filter size is anticipated to enable the model to capture more intricate features from the input data, potentially improving its accuracy. However, larger filter sizes will result in more model parameters, increasing the risk of overfitting if there isn't enough training data. Smaller filter sizes may be able to capture simpler patterns in the data, but they may not be able to capture more complex patterns.

Observations:

As expected, increasing the number of epochs increased training time. Increasing the batch size decreased the training time. The experiment provided results for about 80 combinations of different parameters considered. The loss curve and accuracy details were saved for training on each combination. The code was automated to print out the best and worst accuracy after all the runs.

As shown in figure 13, following combination gave highest accuracy (99.28%)
Number of epochs : 15 , Batch Size : 32, Dropout rate : 0.3 Filter size 5.
This might be due to the fact that increasing the filter size and controlling batch size helped the model to capture more intricate features from the input data.

As shown in figure 13, following combination gave lowest accuracy (96.5%)
Number of epochs : 5 , Batch Size : 128, Dropout rate : 0.5 Filter size 3.
Increase in Dropout rate, higher batch size, lower filter size and low number of epochs might have been insufficient for the model to learn complex features, hence the accuracy might have reduced.

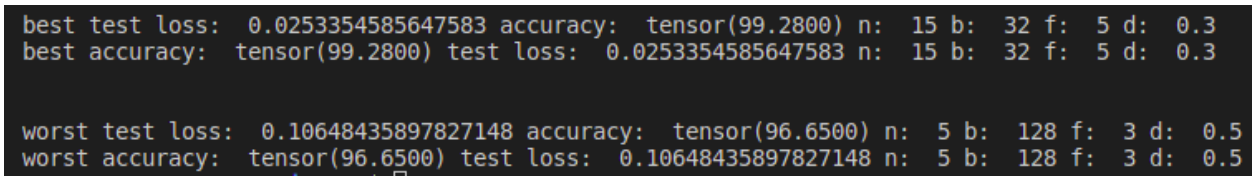


Fig 13 : Best and Lowest accuracy

Extensions

Extension 1: Total 4 dimensions were considered in task 4 along which we can change the architecture

Extension 2:
Loaded pre-trained AlexNet from PyTorch Package and analyzed its first couple of convolution layers. AlexNet is pre-trained on Imagenet dataset to classify images into 1000 different classes which consists of over 1 million labeled images belonging to 1000 different object categories . The architecture of AlexNet is shown in figure 14

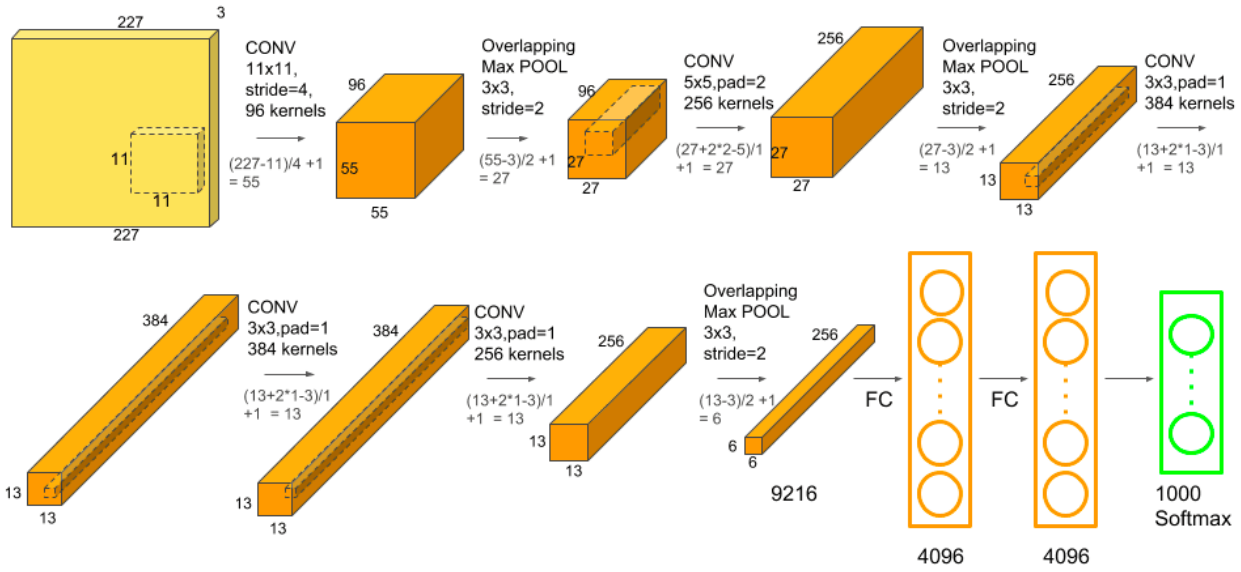
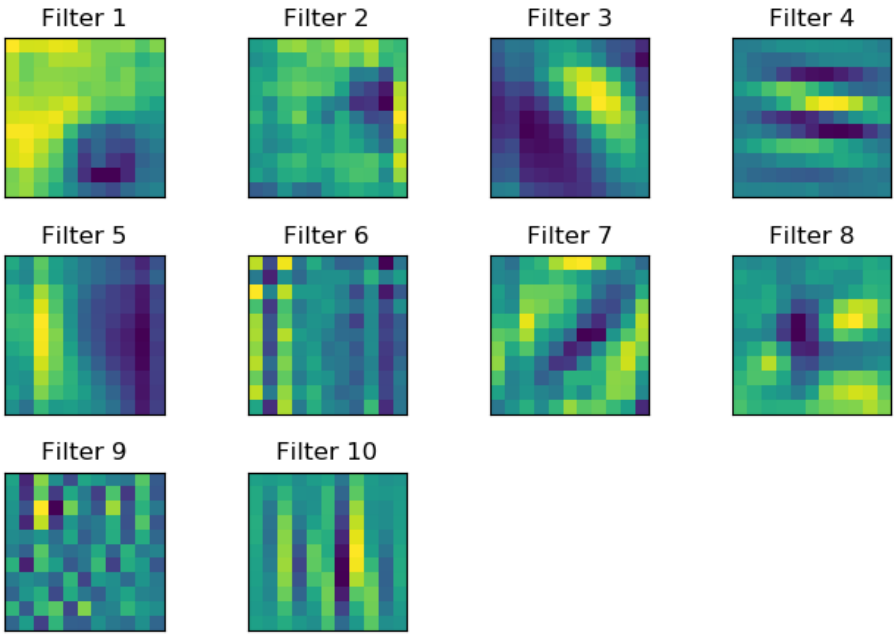


Figure 14 : AlexNet architecture

It seems like the first 10 filters of the first convolution layer of AlexNet would be detecting different types of edges and orientations in the image. For example, some filters might be detecting vertical edges, while others might be detecting horizontal edges or diagonal edges. The first 10 filters of the first couple of convolution layers of the AlexNet are shown in figure 15 and 16.



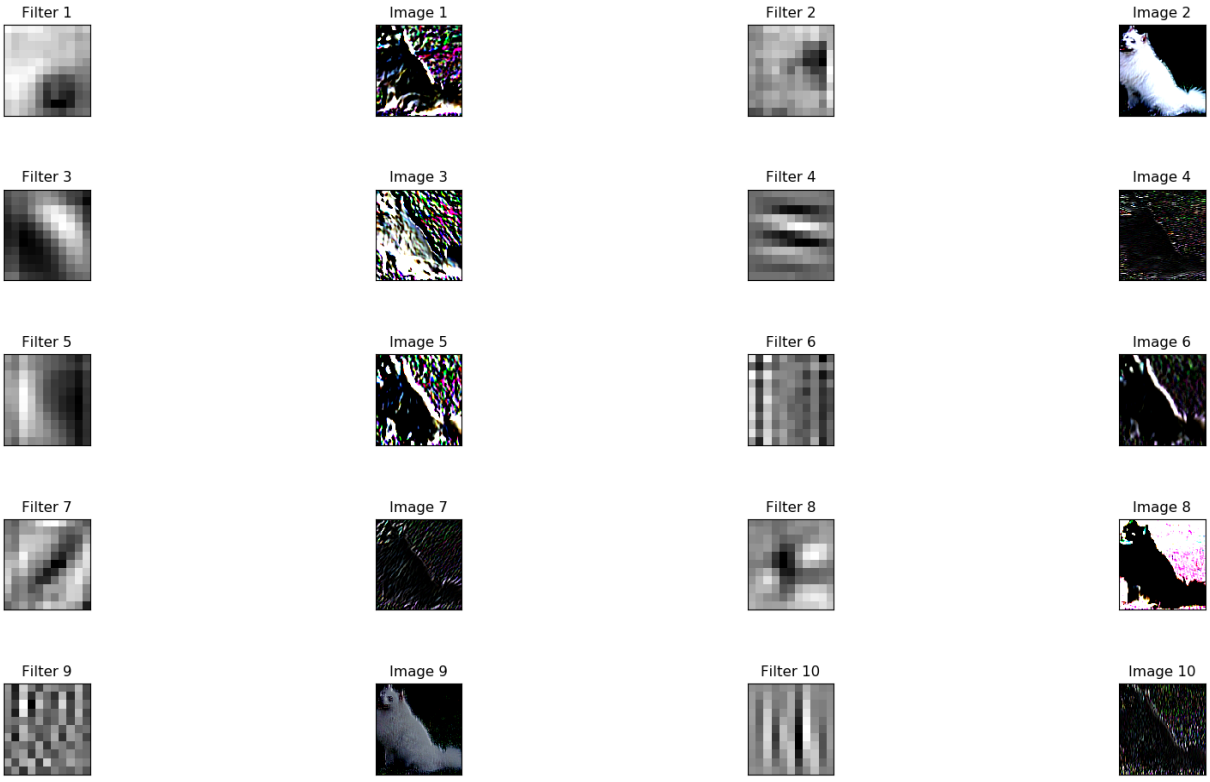
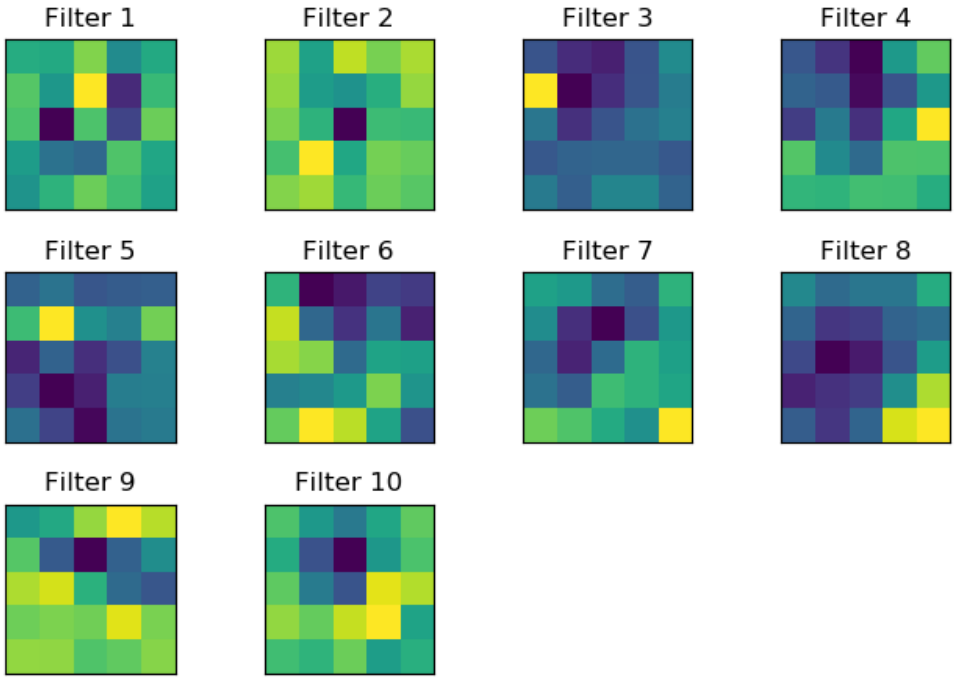


Figure 15 : AlexNet first convolution layer filter and its effect



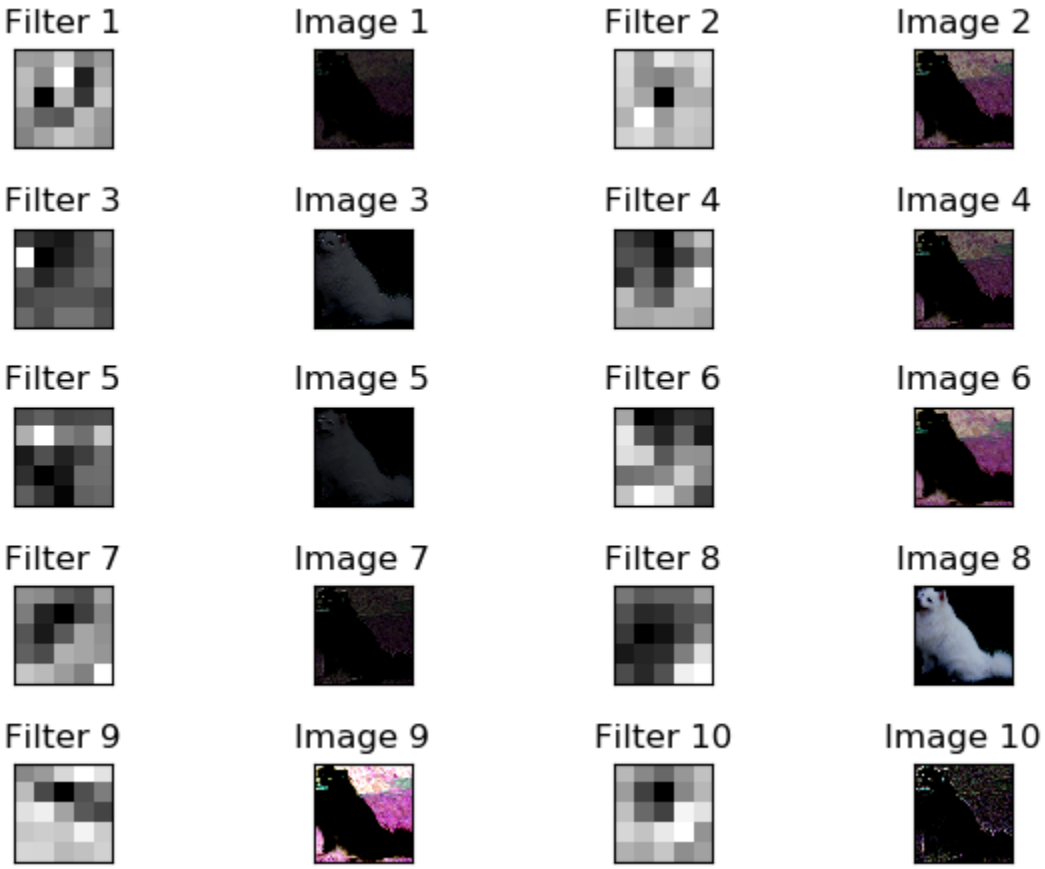


Figure 16: AlexNet second convolution layer filter and its effect

Extension 3: Used Gabor filter instead of first convolution layer

Replaced the first layer of convolution layer of MNIST digit recognition network with a bank of Gabor filters. OpenCV inbuilt gabor filter functions are used to generate 10 gabor filters by changing the orientation. Rest of the network is trained.

The accuracy obtained is reduced to 96.39% when compared to actual network 98.16%. However the new model is still comparable to actual architecture. This might be due to the reason that the 10 filters of the first convolutional layer are capable of detecting edges within an image similar to Gabor filters.

```
Test set: Avg. loss: 0.1147, Accuracy: 9639/10000 (96.39%)
```

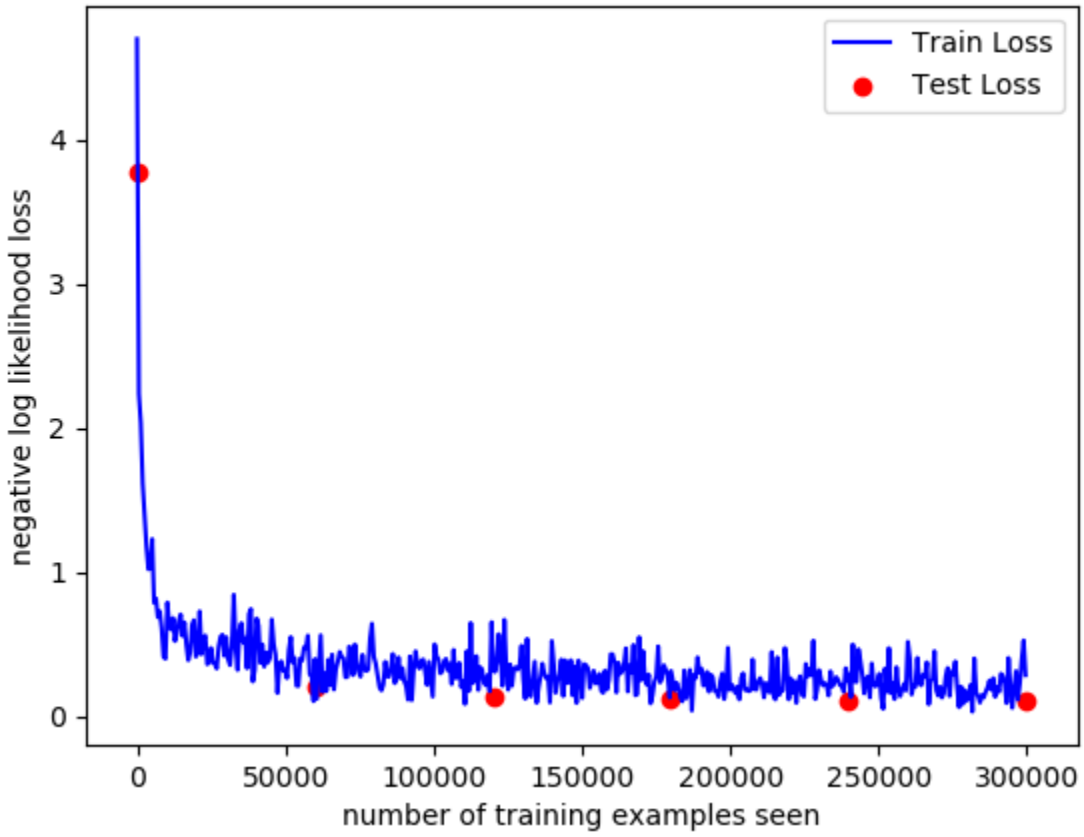


Figure 17 : Train and test loss (gabor filter implementation)

Extension 4: Live digit recognition system

I attempted to develop a real-time system for recognizing digits, but the results I received were inaccurate. This could be because the clarity of the images is compromised when larger frames are resized to 28 x 28 during execution. Another potential factor is the difference in intensity between the frames captured during live recognition and the intensity of the test data. However, due to time constraints, I chose not to address this issue.

Short Reflection:

I learnt about how to build, train and test a deep learning network for performing a certain task. I got to know more about max pooling, Activation layers, fully connected layers etc. I have also learnt about how to do transfer learning. The experiment part helped me understand how each parameter affects model performance. The extension part helped me to understand and compare performance of trained filters vs filter banks like gabor filters.

Acknowledgements

[1] MNIST digit recognition network implementation :
https://www.hackersrealm.net/post/mnist-handwritten-digits-recognition-using-python

[2] AlexNet implementation: https://pytorch.org/hub/pytorch_vision_alexnet/

[3] Resizing images: https://picresize.com/