Project 5: Recognition using Deep Networks

Name: Dev Vaibhav

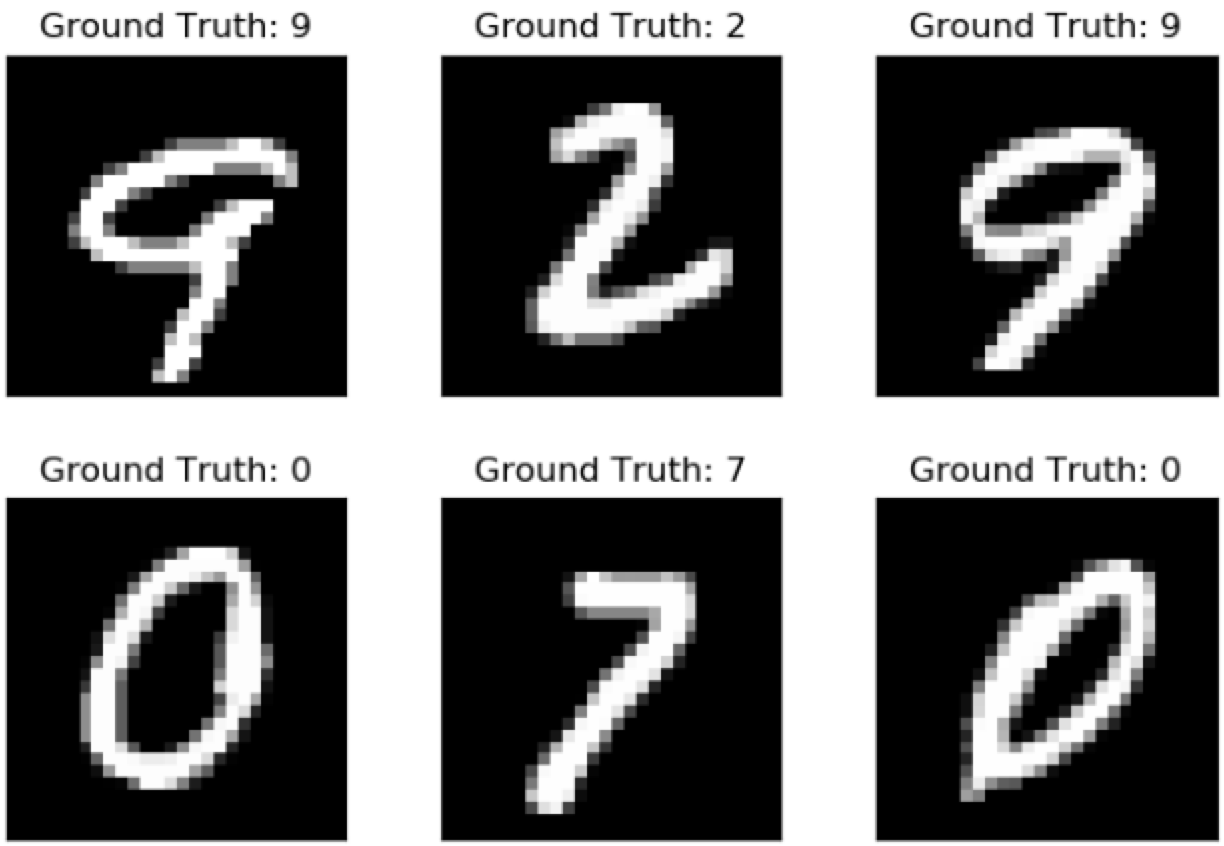
Email: vaibhav.d@northeastern.edu

Class: CS 5330 Pattern Recognition and Computer Vision

Description: This project's aim is to build, train, analyze, and modify a deep network for a recognition task (digit/Greek letters). MNIST digit recognition data set is used to build and train the network. The network model is successfully implemented in Python with the help of Pytorch library. It also provides the scope to examine and/or modify my own network using the constructed model. An experiment is also done to evaluate the effect of changing different aspects of the network.

Task 1) Build and train a network to recognize digits:

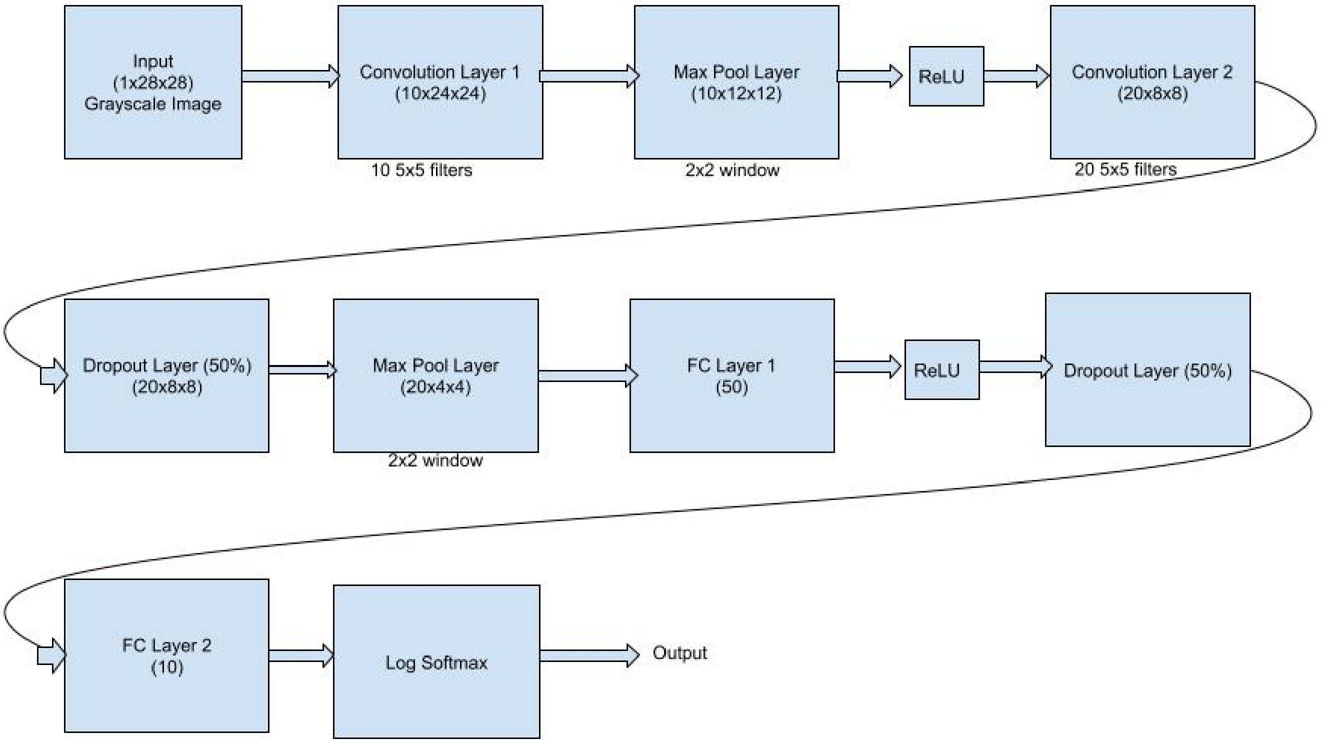
A. Get the MNIST digit data set: Downloaded the dataset and plot of the first six example digits using matplotlib pyplot package.



B. Make your network code repeatable: The code is made repeatable by using torch.manual_seed(random_seed) with random_seed value 1. Additionally, turned CUDA off using torch.backends.cudnn.enabled = False. It is done because cuDNN uses some nondeterministic algorithms.

C. Build a network model:

MNIST architecture implemented from the tutorial

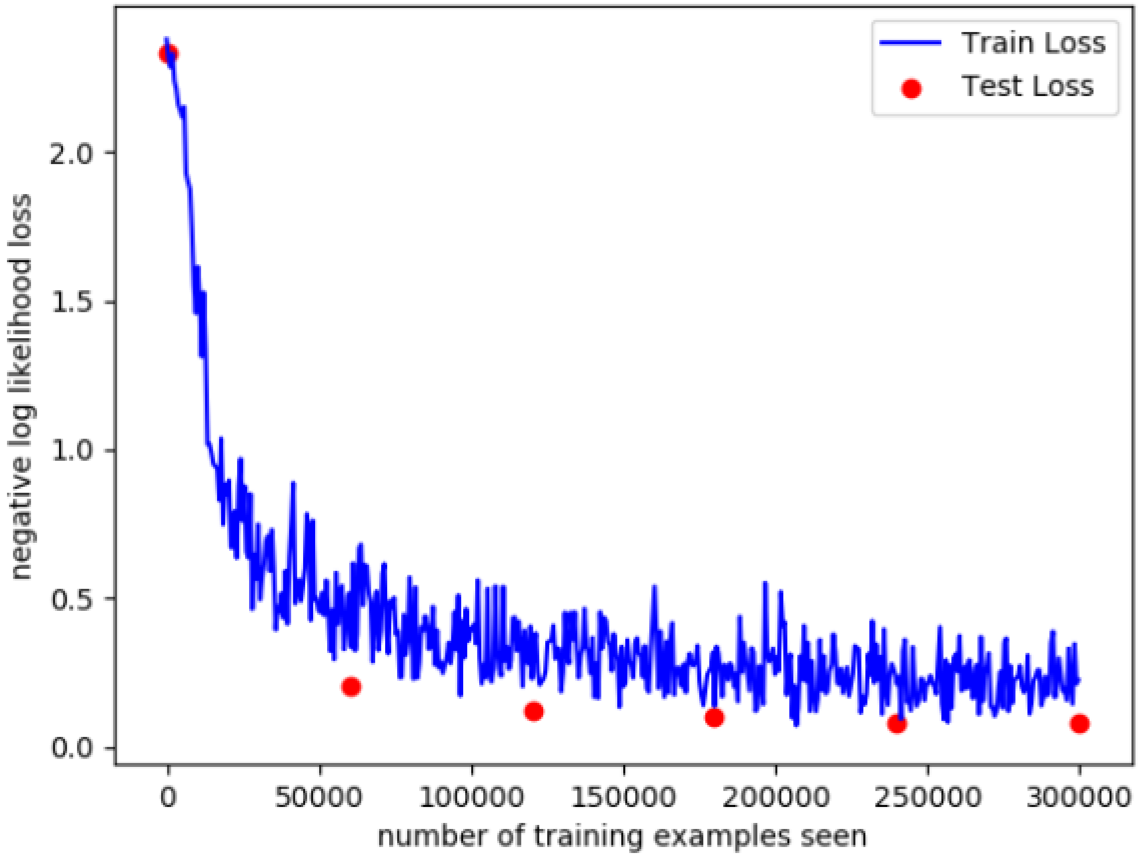


D. Train the model :

Training Loss: 0.224644

Validation/ Test Loss: 0.0778

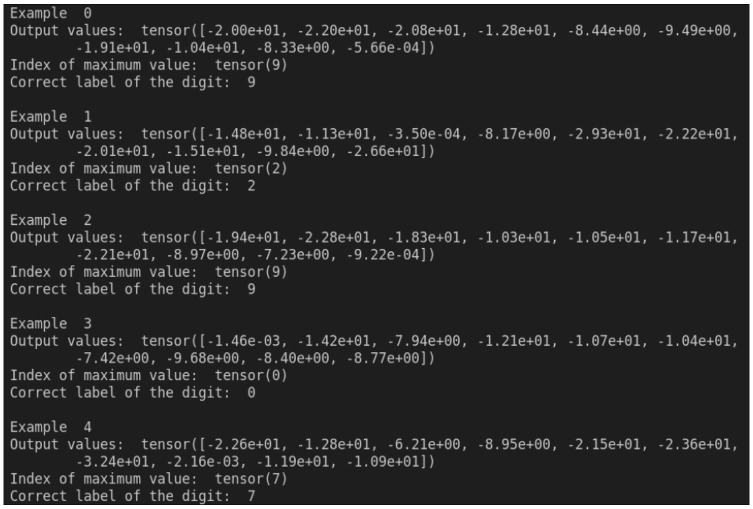
Accuracy = 9764/10000 (98%)



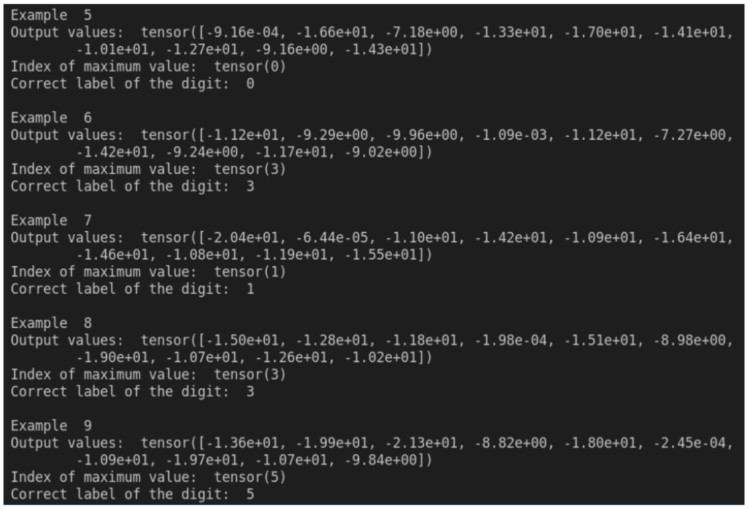
E. Save the network to a file: The network is saved to a file using: torch.save(network.state_dict(), 'model.pth')

F. Read the network and run it on the test set:Even after setting

torch.set_printoptions(sci_mode=False) and torch.set_printoptions(precision=2), the observed output is not printed upto 2 decimal places but it gets printed.



```
Example  0
Output values:  tensor([-2.00e+01, -2.20e+01, -2.08e+01, -1.28e+01, -8.44e+00, -9.49e+00,
        -1.91e+01, -1.04e+01, -8.33e+00, -5.66e-04])
Index of maximum value:  tensor(9)
Correct label of the digit:  9

Example  1
Output values:  tensor([-1.48e+01, -1.13e+01, -3.50e-04, -8.17e+00, -2.93e+01, -2.22e+01,
        -2.01e+01, -1.51e+01, -9.84e+00, -2.66e+01])
Index of maximum value:  tensor(2)
Correct label of the digit:  2

Example  2
Output values:  tensor([-1.94e+01, -2.28e+01, -1.83e+01, -1.03e+01, -1.05e+01, -1.17e+01,
        -2.21e+01, -8.97e+00, -7.23e+00, -9.22e-04])
Index of maximum value:  tensor(9)
Correct label of the digit:  9

Example  3
Output values:  tensor([-1.46e-03, -1.42e+01, -7.94e+00, -1.21e+01, -1.07e+01, -1.04e+01,
        -7.42e+00, -9.68e+00, -8.40e+00, -8.77e+00])
Index of maximum value:  tensor(0)
Correct label of the digit:  0

Example  4
Output values:  tensor([-2.26e+01, -1.28e+01, -6.21e+00, -8.95e+00, -2.15e+01, -2.36e+01,
        -3.24e+01, -2.16e-03, -1.19e+01, -1.09e+01])
Index of maximum value:  tensor(7)
Correct label of the digit:  7
```

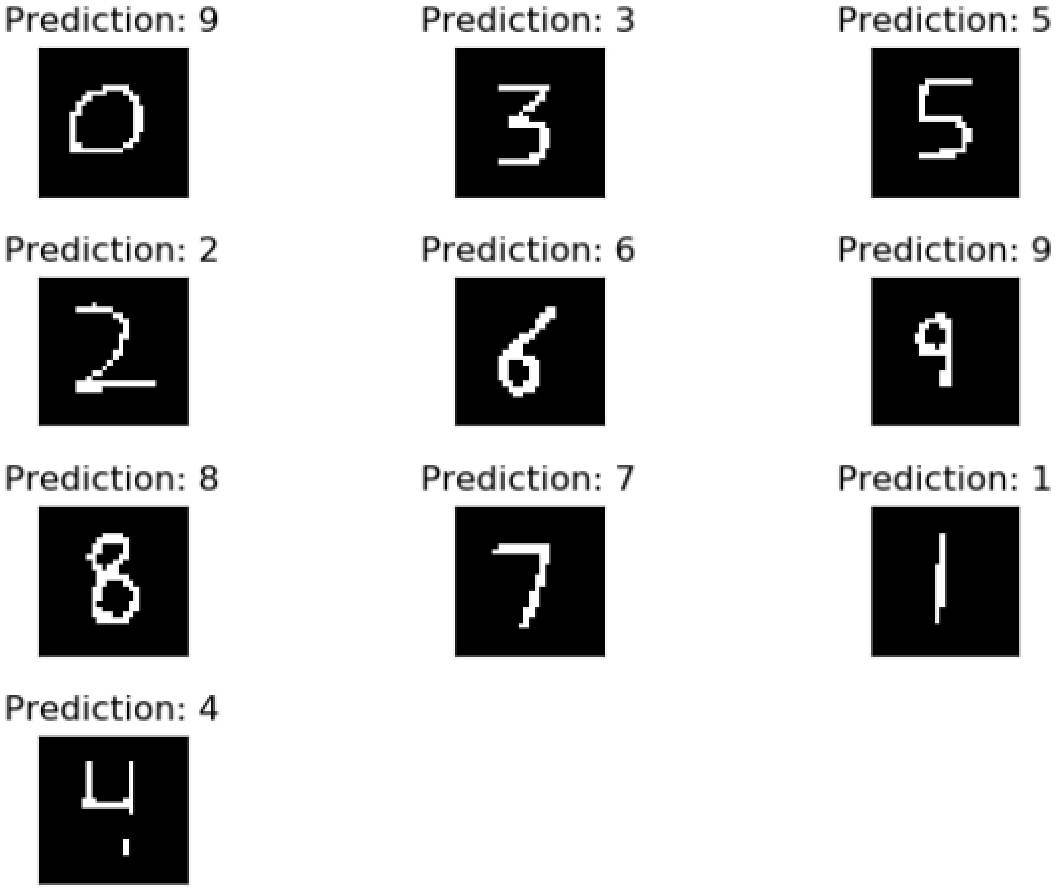


```
Example  5
Output values:  tensor([-9.16e-04, -1.66e+01, -7.18e+00, -1.33e+01, -1.70e+01, -1.41e+01,
        -1.01e+01, -1.27e+01, -9.16e+00, -1.43e+01])
Index of maximum value:  tensor(0)
Correct label of the digit:  0

Example  6
Output values:  tensor([-1.12e+01, -9.29e+00, -9.96e+00, -1.09e-03, -1.12e+01, -7.27e+00,
        -1.42e+01, -9.24e+00, -1.17e+01, -9.02e+00])
Index of maximum value:  tensor(3)
Correct label of the digit:  3

Example  7
Output values:  tensor([-2.04e+01, -6.44e-05, -1.10e+01, -1.42e+01, -1.09e+01, -1.64e+01,
        -1.46e+01, -1.08e+01, -1.19e+01, -1.55e+01])
Index of maximum value:  tensor(1)
Correct label of the digit:  1

Example  8
Output values:  tensor([-1.50e+01, -1.28e+01, -1.18e+01, -1.98e-04, -1.51e+01, -8.98e+00,
        -1.90e+01, -1.07e+01, -1.26e+01, -1.02e+01])
Index of maximum value:  tensor(3)
Correct label of the digit:  3

Example  9
Output values:  tensor([-1.36e+01, -1.99e+01, -2.13e+01, -8.82e+00, -1.80e+01, -2.45e-04,
        -1.09e+01, -1.97e+01, -1.07e+01, -9.84e+00])
Index of maximum value:  tensor(5)
Correct label of the digit:  5
```

G. Test the network on new inputs: The model is tested out on 10 handwritten digits (0 to 9). The digits are written in black ink on white background. After reading them from the disk, they are converted to a tensor, converted to grayscale, resized to 28x28, color inverted and thresholded (> 180 (found experimentally) is set to 255/ 1 and 0 otherwise) to get a binary image. The colors are inverted in the handwritten images because the images in the MNIST dataset are digits written in white ink on a black background. So, both images need to match this way. The following image shows the prediction of the network for the 10 digits:

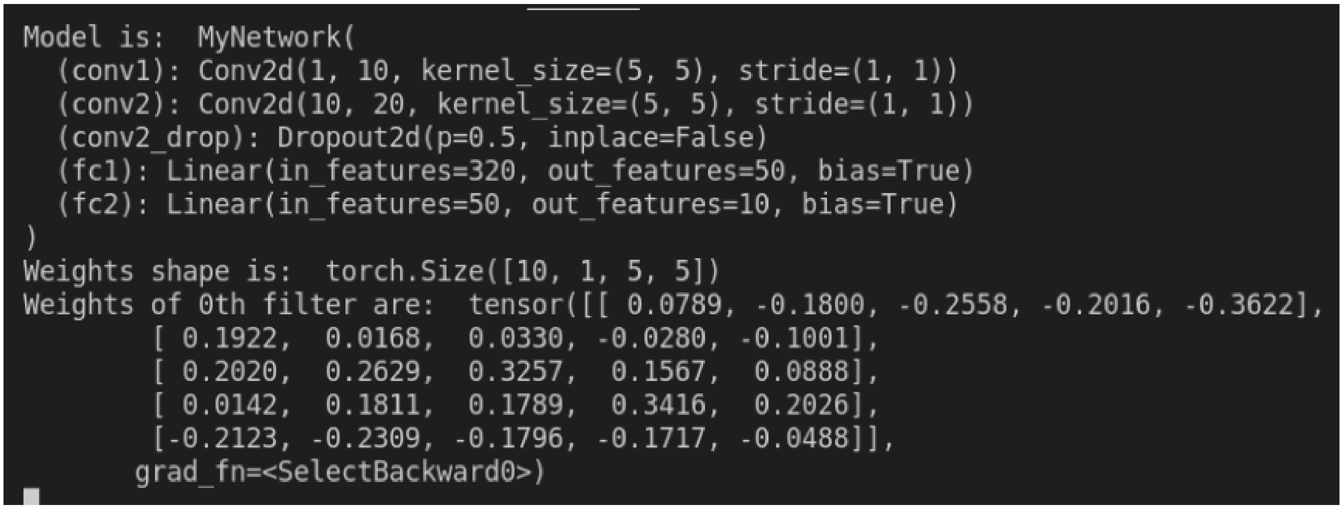
The network is able to predict 9/10 handwritten digits successfully. There could be various reasons for the same ranging from the input images varying in size, intensities, the thickness of lines, etc from the training images. This problem can be rectified by including more images in the dataset.

Initially, my digits were thin. So, the network was not giving good results. After making the digits thick, it gave much better results.

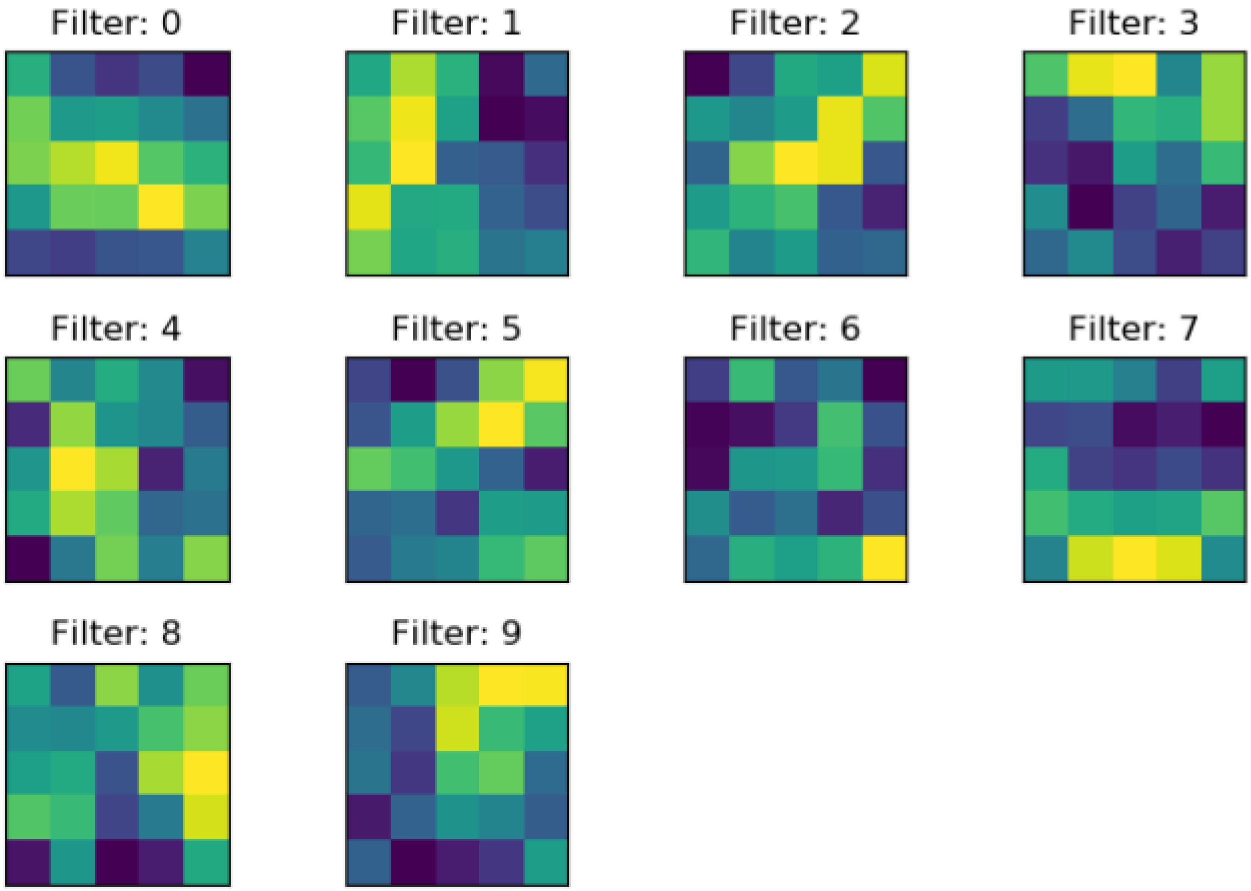


2. Examine your network

A. Analyze the first layer :

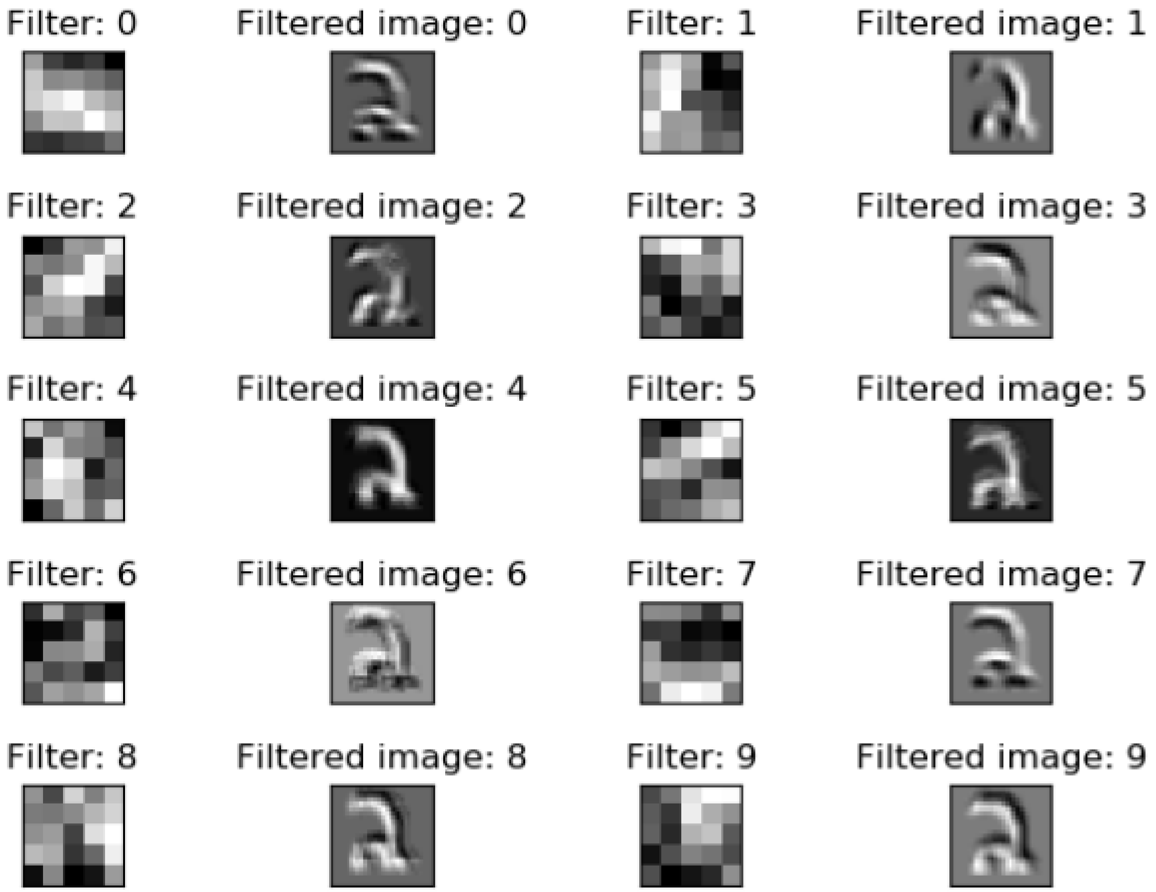


```
Model is:  MyNetwork(
  (conv1): Conv2d(1, 10, kernel_size=(5, 5), stride=(1, 1))
  (conv2): Conv2d(10, 20, kernel_size=(5, 5), stride=(1, 1))
  (conv2_drop): Dropout2d(p=0.5, inplace=False)
  (fc1): Linear(in_features=320, out_features=50, bias=True)
  (fc2): Linear(in_features=50, out_features=10, bias=True)
)
Weights shape is:  torch.Size([10, 1, 5, 5])
Weights of 0th filter are:  tensor([[ 0.0789, -0.1800, -0.2558, -0.2016, -0.3622],
        [ 0.1922,  0.0168,  0.0330, -0.0280, -0.1001],
        [ 0.2020,  0.2629,  0.3257,  0.1567,  0.0888],
        [ 0.0142,  0.1811,  0.1789,  0.3416,  0.2026],
        [-0.2123, -0.2309, -0.1796, -0.1717, -0.0488]],
       grad_fn=<SelectBackward0>)
```



B. Show the effect of the filters:

The following plot shows the 10 filters applied to the first training example image using OpenCV filter2D function. We can see from the results that in all the different filters, a gradient along a different direction is highlighted, hence helping us recognize edges along various different directions, and eventually shapes which help us recognize digits.



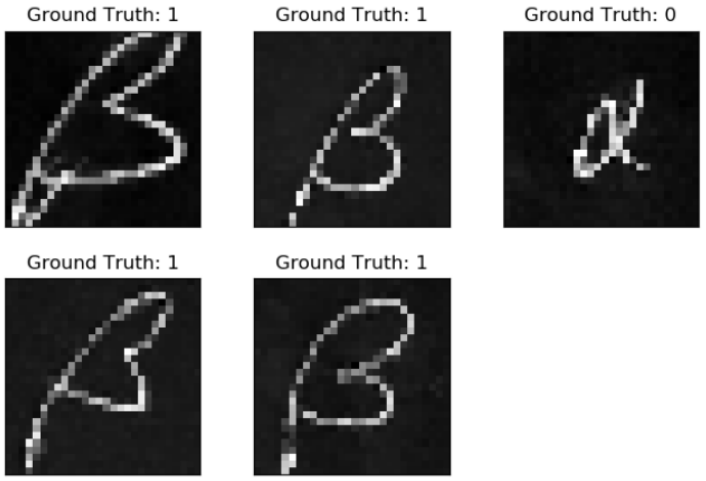
3. Transfer Learning on Greek Letters: 21/27 images are set aside for the training set whereas 2 from each class i.e. 6 images are taken for the test set. It required a minimum of **72 epochs** to achieve 100% accuracy.

Training loss: 0.002270

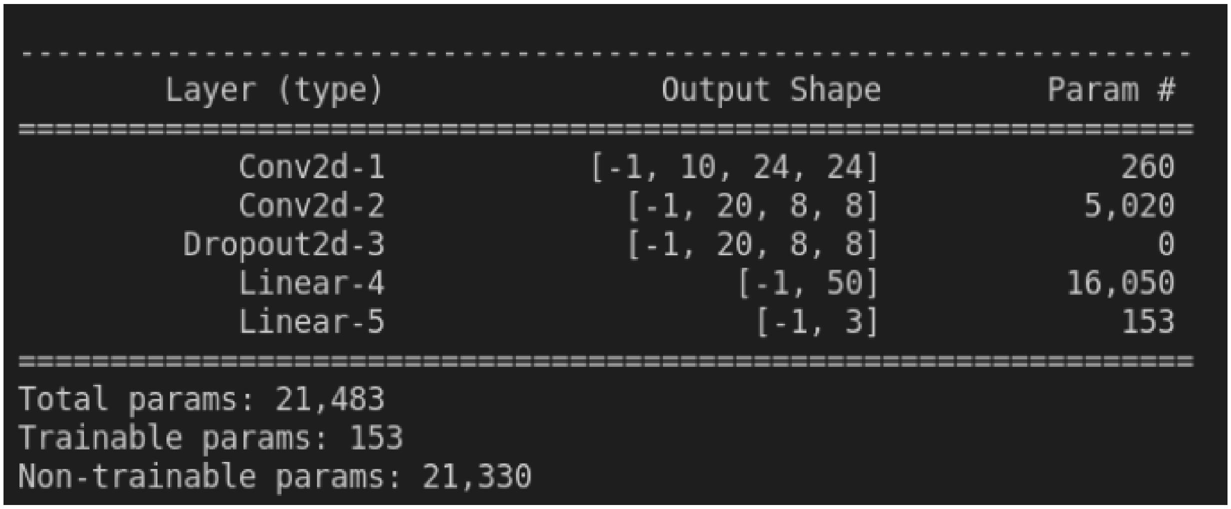
Test Loss: 0.4312

Accuracy: 100%

Ground truth data: alpha means 0 | beta means 1 | gamma means 2



Modified Network:



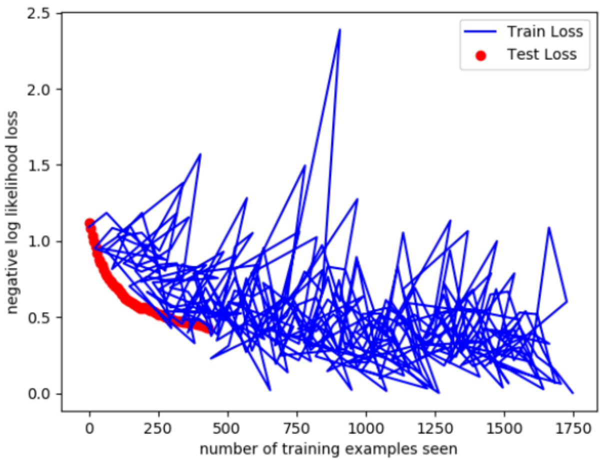
```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1           [-1, 10, 24, 24]             260
            Conv2d-2             [-1, 20, 8, 8]           5,020
         Dropout2d-3             [-1, 20, 8, 8]               0
            Linear-4                   [-1, 50]          16,050
            Linear-5                    [-1, 3]             153
================================================================
Total params: 21,483
Trainable params: 153
Non-trainable params: 21,330
```

Train and test loss curves: It is very noisy possibly because of the small data for the Greek letters and the batch size for train/ test.

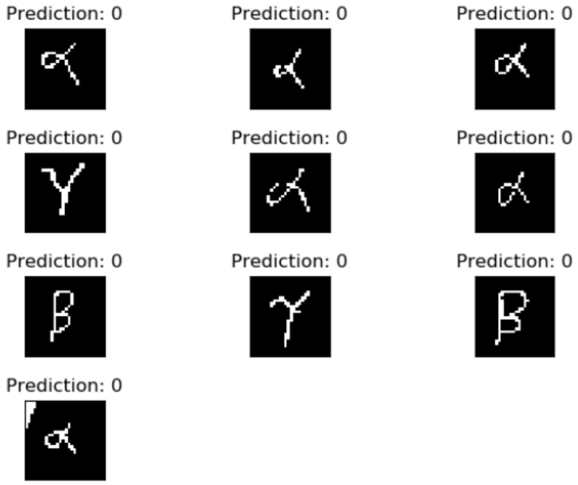
n_epochs = 72

batch_size_train = 7

batch_size_test = 2



Test result on handwritten alpha, beta, and gamma. The network is able to detect alpha correctly but has difficult detecting beta and gamma correctly. It can be resolved by training on more data.



4. Design your own experiment:

A. Develop a plan

The dimensions I chose to vary were the following:

n_epoch_list = [3,4,5] #3

batch_size_train_list = [16, 32, 64, 128] #4

dropout_rate_list = [0.2,0.4, 0.5, 0.7, 0.8] #5

learning_rate_list = [0.001, 0.01, 0.1] #3

momentum_list = [0.4, 0.5, 0.6] #3

The network is trained for a total of 3*4*5*3*3 = 540 combinations which took ~14-15 hours. Data for all of them are captured in network_combinations (CSV/ ods file) but is not shown fully here.

B. Predict the results

I believe that as the number of epochs increases, the performance of the network should go up to a certain extent, as the batch size for training increases the performance will get worse, and as the dropout rate increases the performance will get worse as we might get rid of a lot of useful data that would have helped the network to perform better. Upon increasing the learning rate/ momentum, the network might skip the minima/ reach the minima. It all depends on the loss function.

C. Execute your plan

The following table shows the accuracy and losses observed for the different combinations of the three dimensions. The accuracy always increased by increasing the number of epochs, hence I have only shown the accuracy of the model after the maximum epoch value.

Batch Size Train	Dropout Rate	Learning Rate	Momentum	Train Loss	Validation Loss	Epoch	Accuracy
16	0.2	0.001	0.4	0.7056	0.617	3	76.25
16	0.2	0.001	0.4	0.7277	0.5937	4	77.74
16	0.2	0.001	0.4	0.7025	0.5378	5	78.72
32	0.2	0.001	0.4	0.7236	0.7118	3	72.92
64	0.2	0.001	0.4	1.0807	0.8581	3	69.88
128	0.2	0.001	0.4	1.5793	1.4192	3	62.03
16	0.4	0.001	0.4	0.262	0.6229	3	75.84
16	0.5	0.001	0.4	0.9405	0.7211	3	72.57
16	0.7	0.001	0.4	0.725	0.6804	3	74.44
16	0.8	0.001	0.4	1.2994	0.7235	3	72.94
16	0.2	0.01	0.4	0.2865	0.3946	3	85.3
16	0.2	0.1	0.4	0.5278	0.483	3	81.85
16	0.2	0.001	0.5	0.9171	0.627	3	75.27
16	0.2	0.001	0.6	0.5368	0.5718	3	76.99

We can see that, as predicted, the model's performance got worse as we increased the batch size and the dropout rate (on average). Upon increasing the learning rate, accuracy is increased and then decreased again. Increasing momentum reduced the accuracy slightly and then increased.

EXTENSIONS:

1) Live video digit recognition application using the trained network.

I used classical computer vision and trained network to detect digits on a live video. Below/ here: https://youtu.be/z7YPsGEKxis is its video.

Approach:

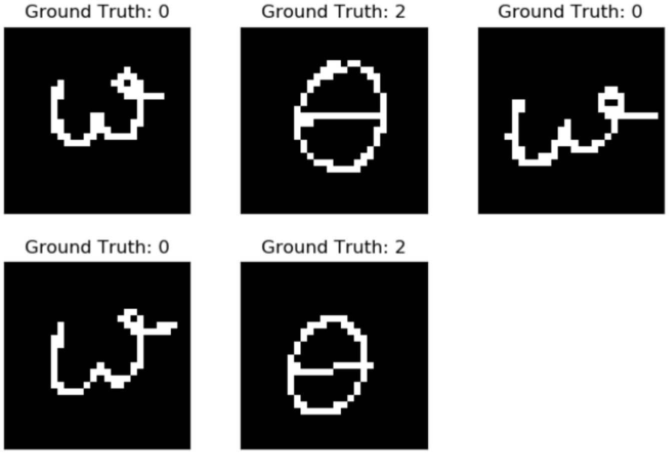
1. Capture Live color video frame from webcam
2. Convert to grayscale, apply gaussian blur 5x5 filter, canny edge detection
3. Find contours in the image and choose contours in the area range (300, 3000) (Found experimentally)
4. Find the Axis Aligned Bounding Box (AABB) of the contour and store that region of interest (roi) into an image
5. Pass these colored regions of interest for further processing.
6. Apply grayscale conversion, scaling to 28x28, binary image conversion and normalization to pass the right image to network.
7. Prediction is obtained and shown on the live feed

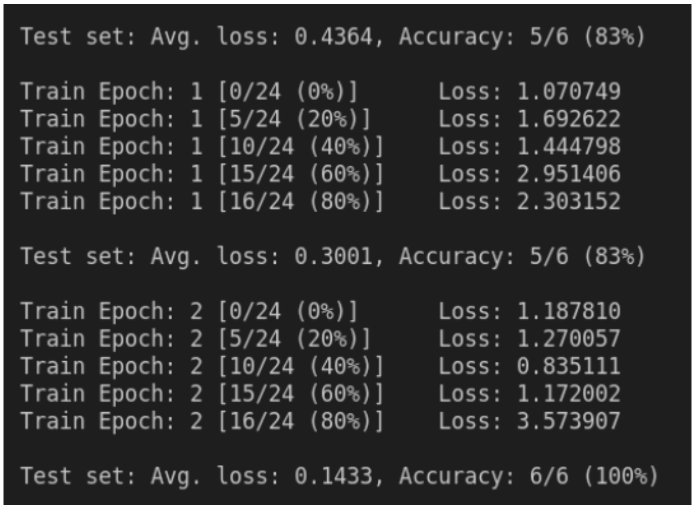
ext1_live_digit-...-06_18.54.06.mkv

2) Evaluate more dimensions on task 4: It is already done by exploring 2 additional dimensions (total 5) in task 4. The script also logs the results to a CSV file so that it is easy to do the analysis.

3) Try more greek letters than alpha, beta, and gamma: I captured a set of three more greek letters (10 each): pi, theta, and omega. 8 images from each class, 24 at random are selected for training set and the rest 6 (2 from each class) are taken for validation set. After 2 epochs, 100 % accuracy is achieved.

omega is 0 | pi is 1 (not shown) | theta is 2





```
Test set: Avg. loss: 0.4364, Accuracy: 5/6 (83%)

Train Epoch: 1 [0/24 (0%)]      Loss: 1.070749
Train Epoch: 1 [5/24 (20%)]     Loss: 1.692622
Train Epoch: 1 [10/24 (40%)]    Loss: 1.444798
Train Epoch: 1 [15/24 (60%)]    Loss: 2.951406
Train Epoch: 1 [16/24 (80%)]    Loss: 2.303152

Test set: Avg. loss: 0.3001, Accuracy: 5/6 (83%)

Train Epoch: 2 [0/24 (0%)]      Loss: 1.187810
Train Epoch: 2 [5/24 (20%)]     Loss: 1.270057
Train Epoch: 2 [10/24 (40%)]    Loss: 0.835111
Train Epoch: 2 [15/24 (60%)]    Loss: 1.172002
Train Epoch: 2 [16/24 (80%)]    Loss: 3.573907

Test set: Avg. loss: 0.1433, Accuracy: 6/6 (100%)
```

Reflections:

This project gave me an insight into how deep networks work and helped put the lectures into perspective. I got a better understanding of convolutional layers, the various dimensions and how varying them impacts the performance of the network. Additionally, it also familiarised me with Pytorch and its various functions. The project seemed daunting at first, just because I am new to the world of neural networks but the lectures and various online resources helped me pursue the project and clarify many doubts I had about deep networks.

Acknowledgments:

Heavily relied on Pytorch documentation and the tutorials linked in the project description. https://nextjournal.com/gkoehler/pytorch-mnist

https://redirect.cs.umbc.edu/courses/331/fall11/notes/python/python3.ppt.pdf

https://stackoverflow.com/questions/30230592/loading-all-images-using-imread-from-a-given-folder;
https://www.projectpro.io/recipes/convert-image-tensor-pytorch;
https://www.analyticsvidhya.com/blog/2021/04/10-pytorch-transformations-you-need-to-know/

https://discuss.pytorch.org/t/access-weights-of-a-specific-module-in-nn-sequential/3627

https://stackoverflow.com/questions/34097281/convert-a-tensor-to-numpy-array-in-tensorflow
https://stackoverflow.com/questions/63582590/why-do-we-call-detach-before-calling-numpy-on-a-pytorch-tensor

https://medium.com/@nutanbhogendrasharma/pytorch-convolutional-neural-network-with-mnist-dataset-4e8a4265e118

https://www.geeksforgeeks.org/python-opencv-capture-video-from-camera/

https://pyimagesearch.com/2017/02/13/recognizing-digits-with-opencv-and-python/

https://pillow.readthedocs.io/en/stable/handbook/concepts.html#concept-modes
https://www.geeksforgeeks.org/convert-opencv-image-to-pil-image-in-python/

https://discuss.pytorch.org/t/grayscale-image-plotted-to-have-colours/135860