CS 5330 - Pattern Recognition and Computer Vision

Project 5 : Recognition Using Deep Networks

Group Members : Shirish Kishore Kumar (002980081), Soorya Vijayaragavan (002921909)

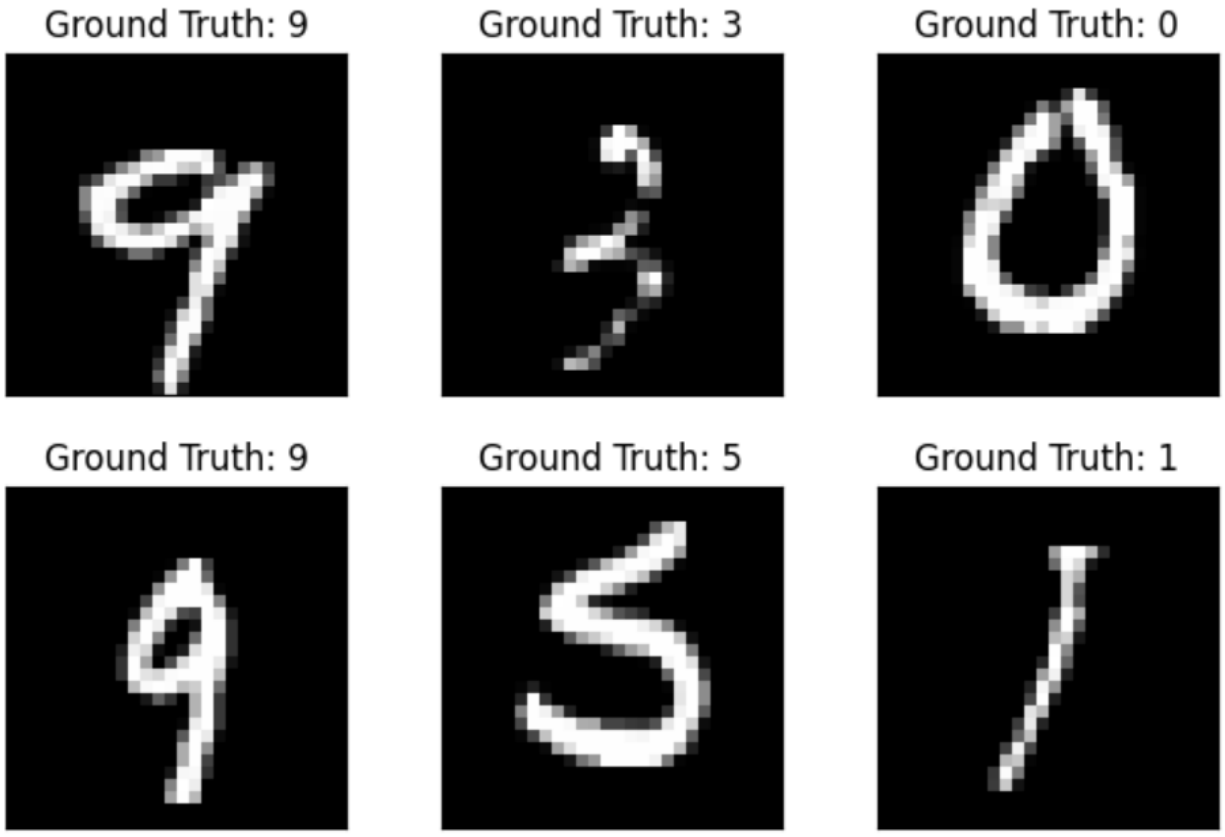
Introduction:

The main aim of the project is to build, train, analyze and modify deep networks for a recognition task. The dataset that we use to train and analyze the deep networks is the MNIST Digit recognition dataset. Then the built neural network underwent analysis by examining the impact of each filter on input images. The network was subsequently modified to recognize Greek letters as opposed to just digits. A series of experiments were also conducted on the MNIST Fashion dataset to evaluate how adjustments to the network would affect training outcomes.

1.Build and train a network to recognize digit

A. Get the MNIST digit dataset:

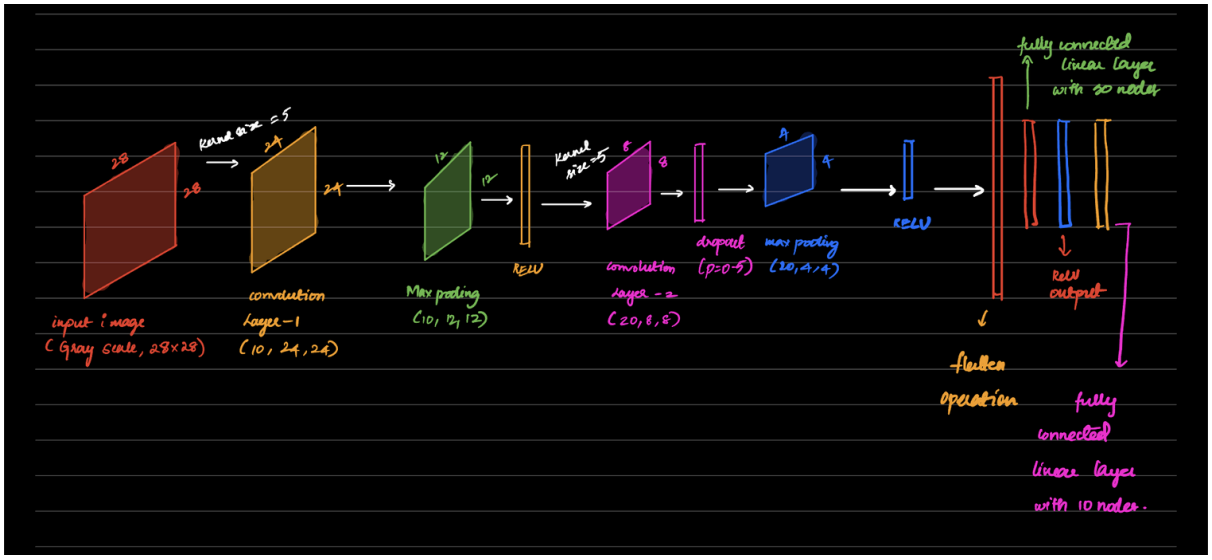
Below given is the plot of the first six examples of the MNIST digit dataset,



C. Build a Network:

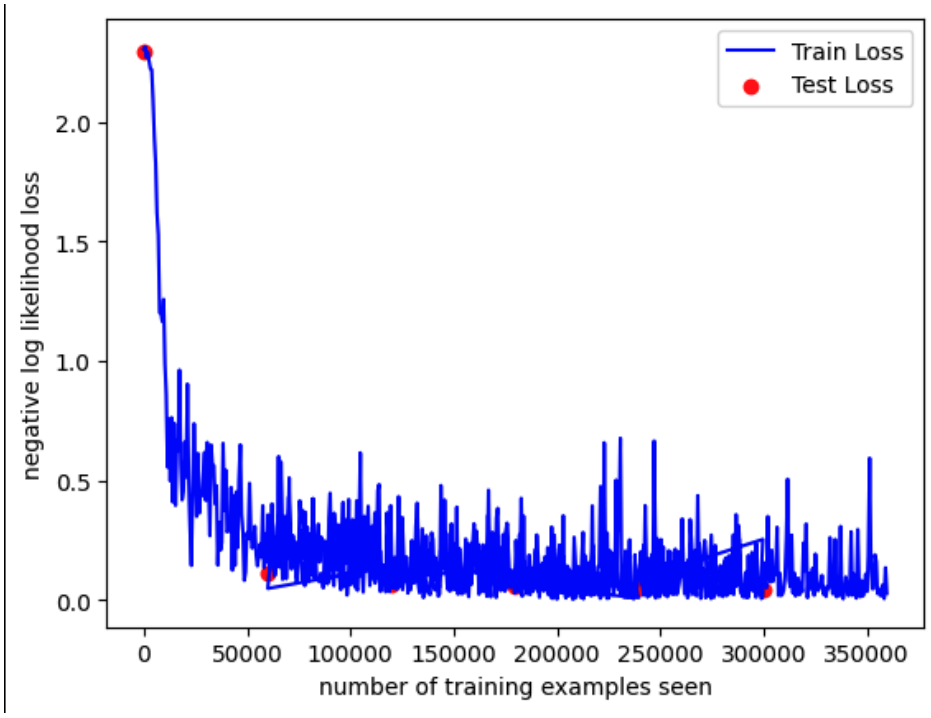
Model:

Below is the diagram of the network built,



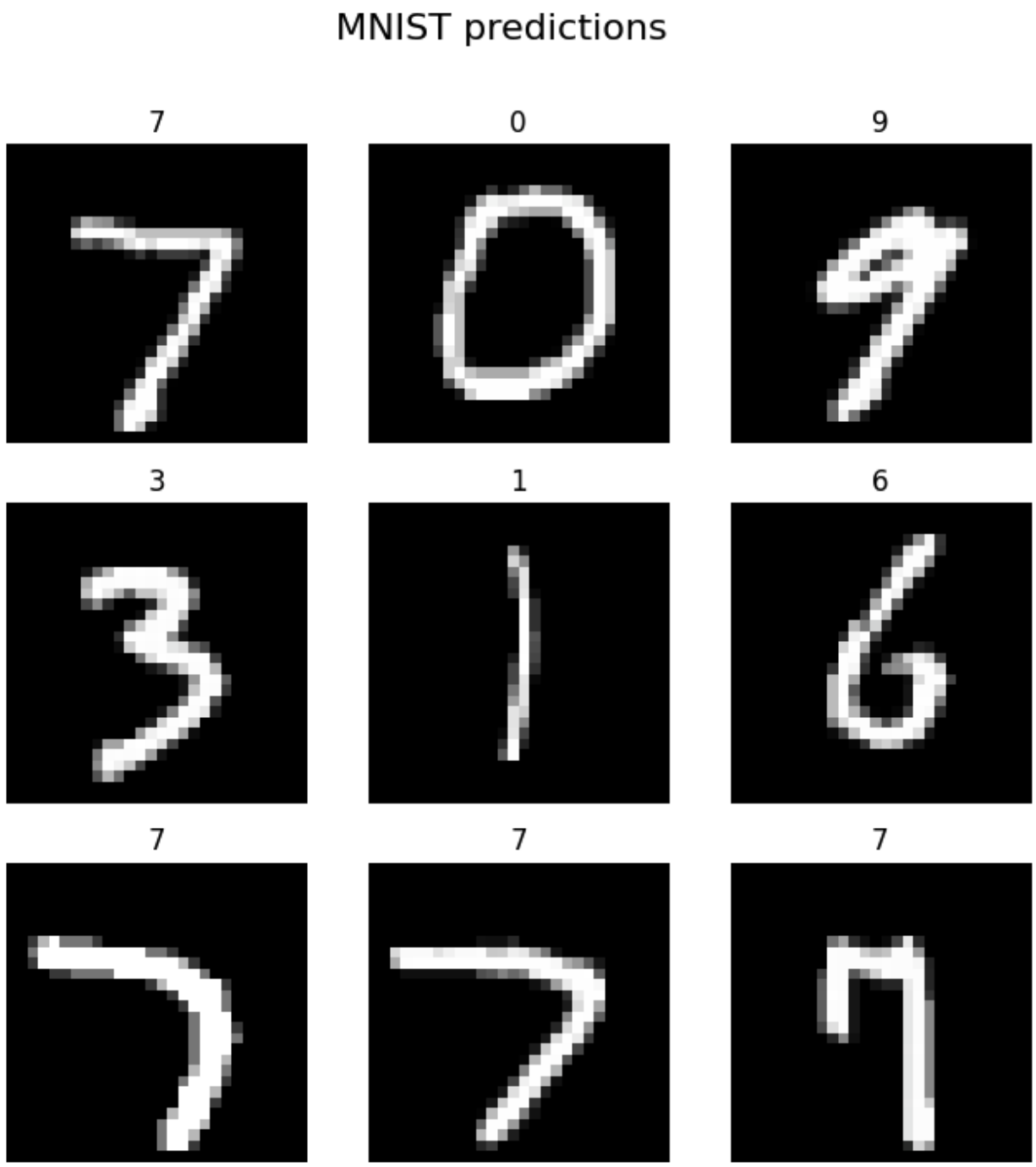
D.Train the model:

The model was trained for five epochs, one epoch at a time, evaluating the model on both the training and test sets after each epoch. Batch size was taken as 32.Below given is the training and testing accuracy.



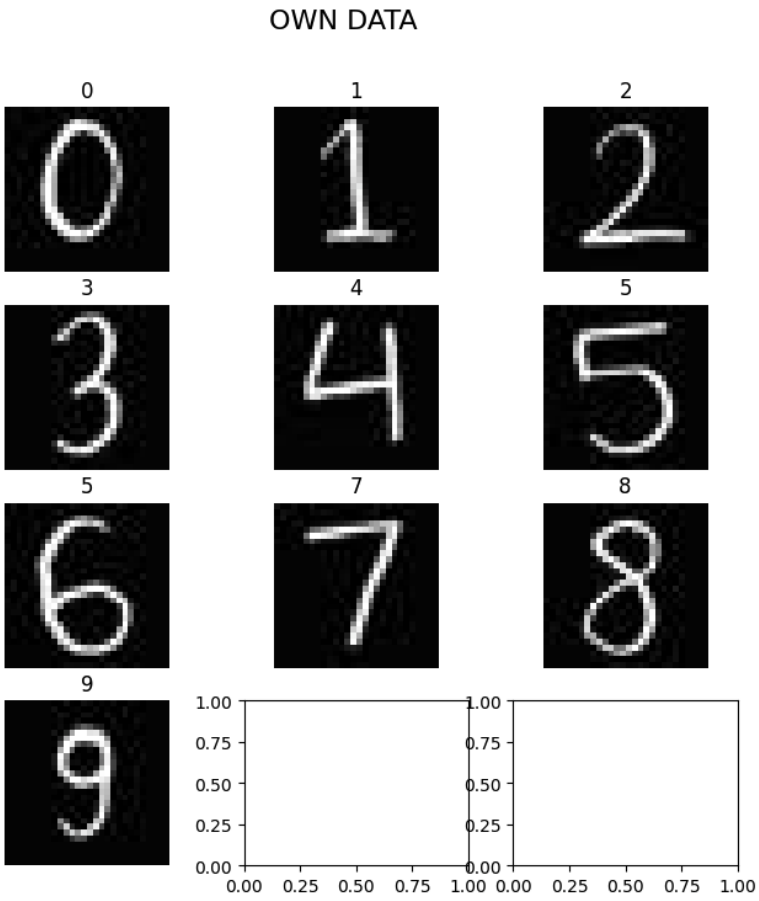
F.Read the network and run it on test set:

Below given are the test results obtained using the MNIST digit test dataset, the model predicted the numbers accurately.



G.Test the network on new inputs(own input):

Below is the test result for the dataset that was created by us which had numbers from 0 to 9, the model predicted every numbers correctly,



2. Examine your network

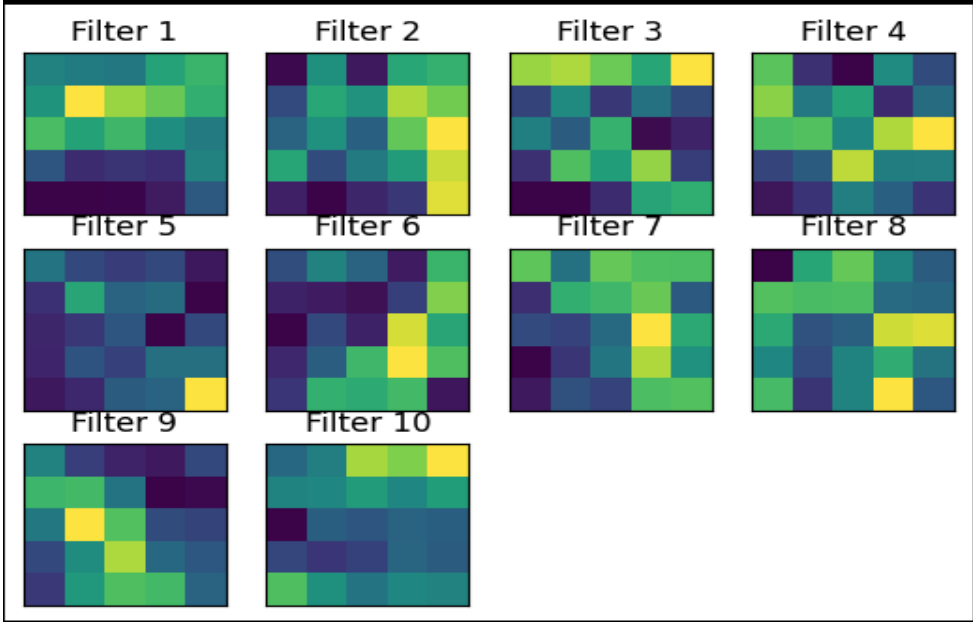
Model:

```
Net(
  (mnist_net): Sequential(
    (0): Conv2d(1, 10, kernel_size=(5, 5), stride=(1, 1))
    (1): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (2): ReLU()
    (3): Conv2d(10, 20, kernel_size=(5, 5), stride=(1, 1))
    (4): Dropout2d(p=0.5, inplace=False)
    (5): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (6): ReLU()
    (7): Flatten(start_dim=1, end_dim=-1)
    (8): Linear(in_features=320, out_features=50, bias=True)
    (9): ReLU()
    (10): Linear(in_features=50, out_features=10, bias=True)
  )
)
```

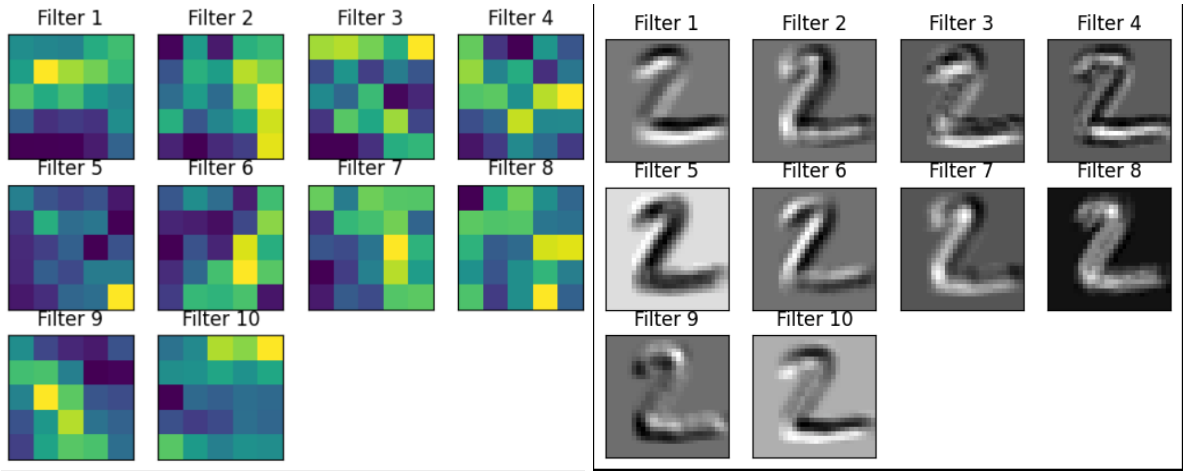
A.Analyze the first layer:

Below given are the weights of each filter and visualization of the filters and shape of the layer would be [10,1,5,5], i.e, it has 10, 5x5 filters.

```
Filter 1
[[ 0.03472518  0.01124013 -0.00469662  0.15362175  0.21928568]
 [ 0.0996647   0.49821374  0.37198848  0.30603734  0.19414051]
 [ 0.2505096   0.14894116  0.22651133  0.07526287  0.00782622]
 [-0.13315193 -0.28327435 -0.2532577  -0.27947462  0.03405384]
 [-0.41104227 -0.40527102 -0.40147945 -0.34183672 -0.11986975]]
Filter 2
[[-0.32634345  0.02118316 -0.2898769   0.07836542  0.10722324]
 [-0.16703992  0.07788645  0.02711583  0.24640393  0.1881506 ]
 [-0.10339779  0.02430805 -0.10827802  0.17058957  0.31764382]
 [ 0.07519466 -0.16427377 -0.03742868  0.04859957  0.271443  ]
 [-0.27100116 -0.336745   -0.25777295 -0.21647379  0.29056153]]
Filter 3
[[ 0.22272997  0.23823833  0.18746522  0.10504019  0.29531902]
 [-0.10119015  0.04643345 -0.12585895 -0.00650491 -0.08537879]
 [ 0.02404448 -0.0493151   0.13117804 -0.21086724 -0.16450708]
 [-0.14194353  0.16029905  0.08557826  0.21957347 -0.11422151]
 [-0.2217759  -0.22189438 -0.14880505  0.10018204  0.12112013]]
Filter 4
[[ 0.14498025 -0.16275354 -0.24103664  0.02559336 -0.10741442]
 [ 0.18423161 -0.01177632  0.07229292 -0.17230076 -0.04189159]
 [ 0.12823063  0.13662295  0.01450535  0.20999251  0.26510274]
 [-0.11846448 -0.07308317  0.22085153 -0.00523361 -0.00279375]
 [-0.2114003  -0.15422815 -0.00203455 -0.06399599 -0.15538496]]
...
 [-0.0369997  -0.02399899  0.05275748 -0.02369523  0.05947785]
 [-0.50985986 -0.1820497  -0.22011916 -0.16167542 -0.18313777]
 [-0.2761427  -0.3396831  -0.2944342  -0.15951698 -0.19449224]
 [ 0.19842547  0.01412915 -0.09424742 -0.02621324 -0.03802652]]
```



B.Show the effects of the filter:



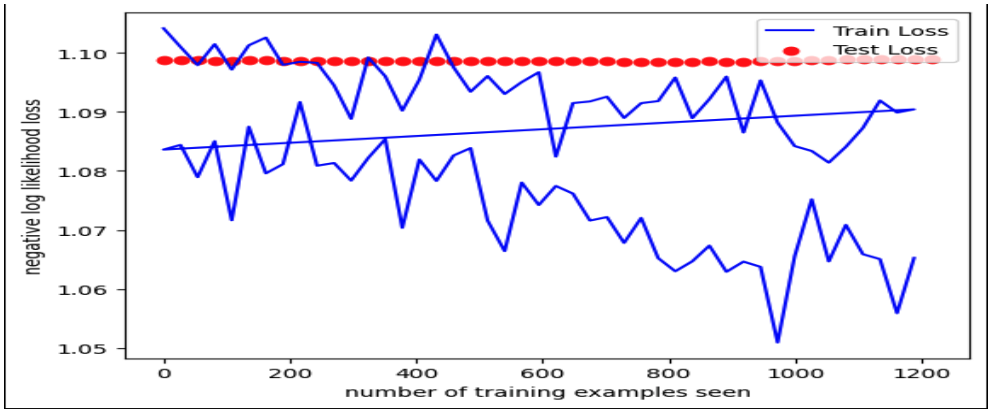
The filters here makes sense because if we look at the filter number 8 it performs similar to thresholding although it is not perfect but still close enough to replicate a thresholding effect, if we look at filter number 1,10,7 they perform similar to the blur filter, the filters 2,3,4,6 are similar to the edge filters like sobel x,y cause they emphasize the edges.

3.Transfer Learning on Greek Letter:

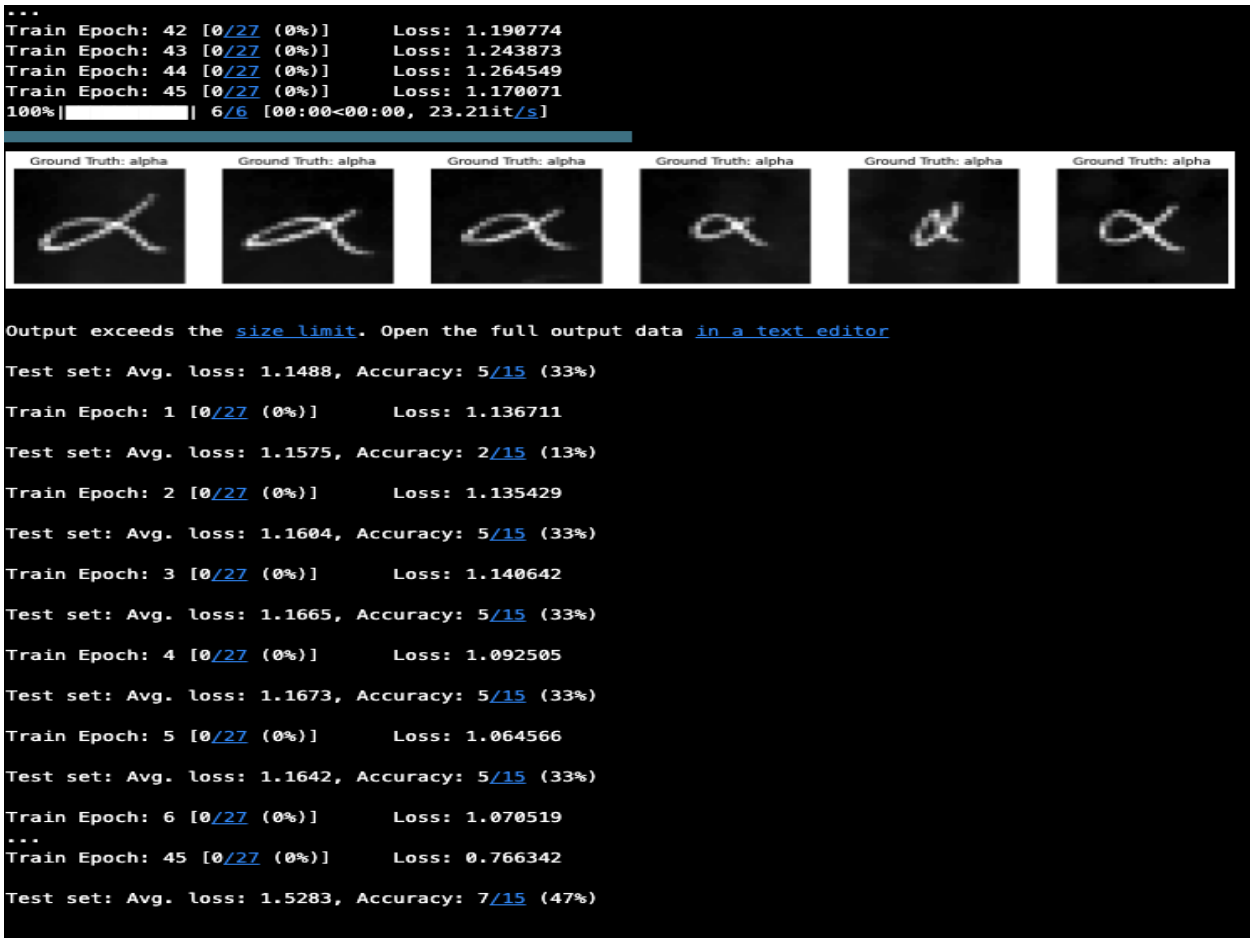
According to this given task, the network is to be modified and trained again to identify the greek letters- alpha, beta and gamma. Below given the modified network,

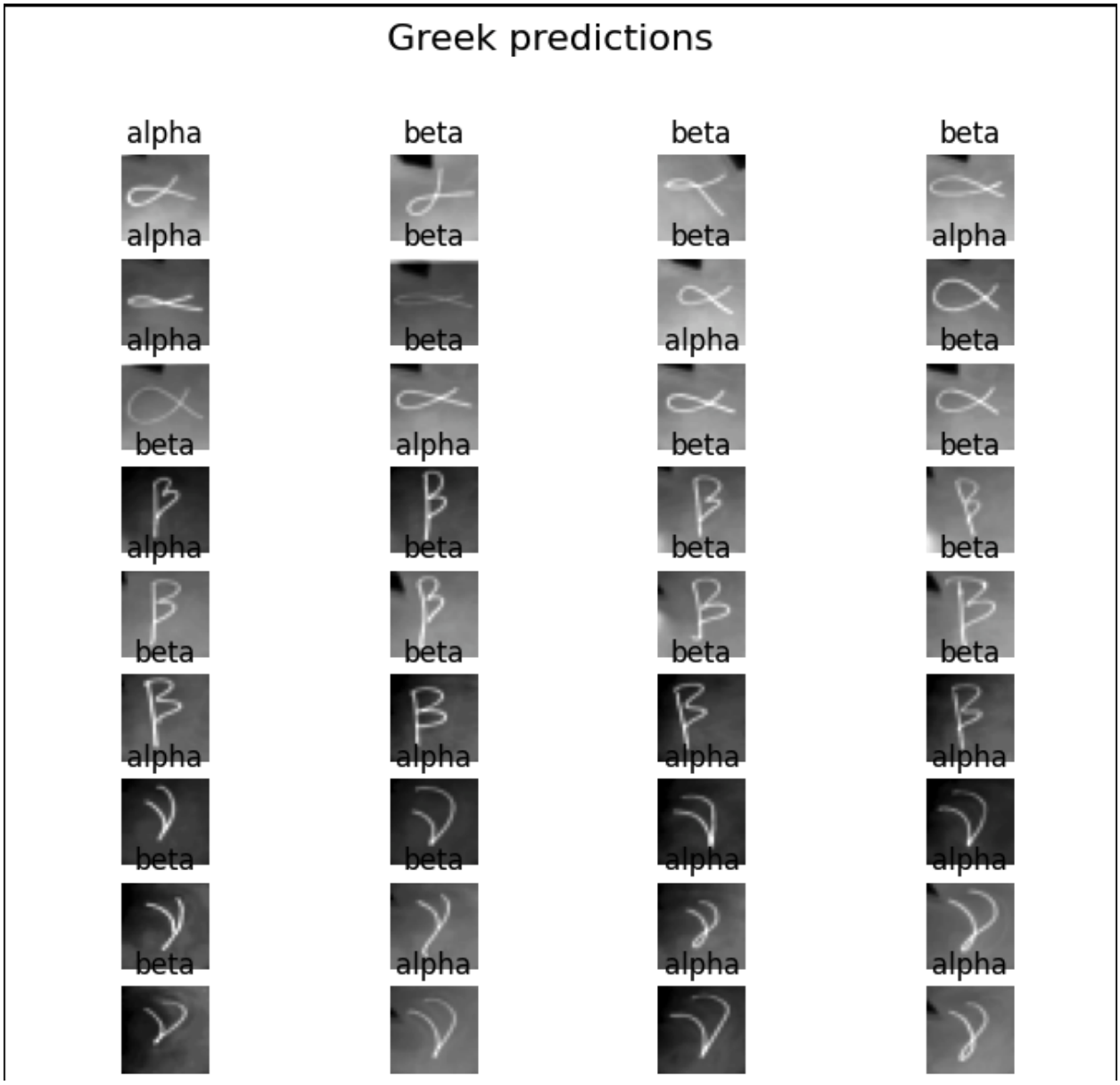
```
Net(
  (mnist_net): Sequential(
    (0): Conv2d(1, 10, kernel_size=(5, 5), stride=(1, 1))
    (1): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (2): ReLU()
    (3): Conv2d(10, 20, kernel_size=(5, 5), stride=(1, 1))
    (4): Dropout2d(p=0.5, inplace=False)
    (5): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (6): ReLU()
    (7): Flatten(start_dim=1, end_dim=-1)
    (8): Linear(in_features=320, out_features=50, bias=True)
    (9): ReLU()
    (10): Linear(in_features=50, out_features=3, bias=True)
  )
)
```

The below image shows training and test loss,



The below image shows the given data set,



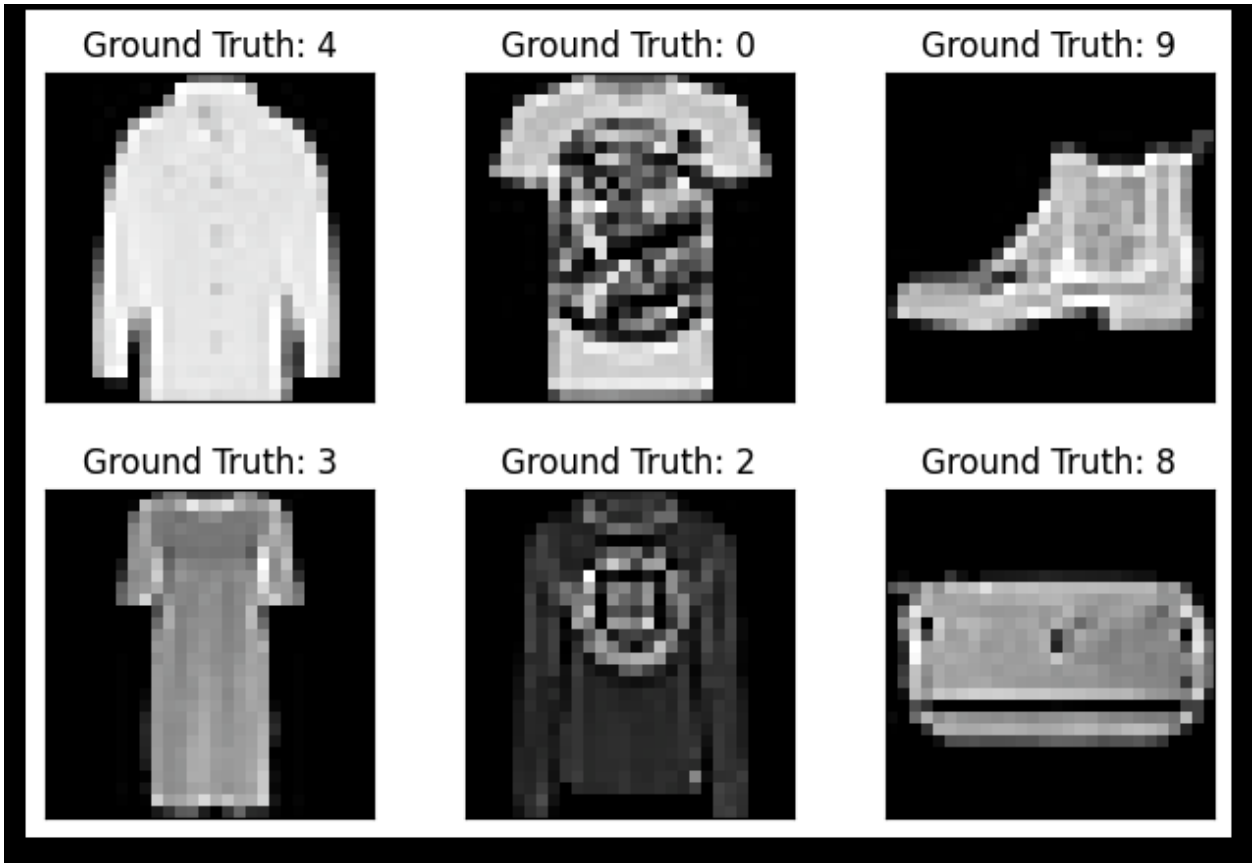


As we can see from the above image the overall accuracy was around 42%

4. Design your own experiment:

A. Develop a plan:

We have decided to use 4 dimensions and we will be performing the 4th dimension as part of the extension.



We will be performing the evaluation task on the fashion MNIST dataset, our goal here is to improve the accuracy by modify 4 parameters Dropout rate, batch size, number of epochs, size of the convolution filters, here are the variations in which we performed the test

1- Batch size (32,64,128,256,516)
2- Number of epochs (3,4,5)
3- Dropout Rates (0.1,0.2,0.3,0.4)
4- Size of convolution layers (3,5)

Intotal we have **5*3*4*2 = 120**, therefore we have created **120 network variations**.

In order to automate this task we used the linear search strategy where we held 3 parameters constant and varied one variable in the for loop so in total we used 4 for loops since it is a dimension 4 evaluation process.

B. Predict the results:

Hypothesis 1: High batch number leads to better results since in high batch numbers we have less noise, which will lead to faster generalization and better performance
Hypothesis 2: High kernel size will lead to bad performance in our case since it can capture more details/features from the input but our input data is very small so it may lead to overfitting
Hypothesis 3: high dropout rates will increase the performance. In our case it should perform better with the high sized kernel because it helps with reducing the over fitting.
Hypothesis 4: Higher epochs should lead to better performance, until the model overfits.

C. Execute your plan

Hypothesis 1 results: As we see from the data it is clear that it did not have any huge impact on the overall results, this might be due to the fact that the total number of epochs were too low.
Hypothesis 2 results: As we can see from the data most of the time the kernel size 3 out performed the kernel size 5 with the same parameters therefore our second hypothesis is true.
Hypothesis 3 results: Our hypothesis failed here as we can see from our data most of the time the difference between those two are very less therefore it didn't make much difference with our dropout rates, we might have to increase the dropout rates therefore our hypothesis failed.
Hypothesis 4 results: Here our hypothesis worked because the performance increased as we increase the epoch number, even though the change in the results are less compared to the previous epoch but still the results were consistent through the entire set of epochs therefore our hypothesis worked, if we keep more epoch numbers for example (5,10,15) instead of (3,4,5) the one we have used , we might get to see some drastic changes in the overall results**.**

The results are displayed below,

No. of Epochs	Batch Size	Dropout	Kernel Size	Accuracy(Epoch 1)	Accuracy(Epoch 2)	Accuracy(Epoch 3)	Accuracy(Epoch 4)	Accuracy(Epoch 5)
3	32	0.1	3	81.71	84.95	85.23	-	-
3	32	0.1	5	80.85	83.4	85.59	-	-
3	32	0.2	3	82.7	85.19	84.54	-	-
3	32	0.2	5	76.92	81.91	85.51	-	-
3	32	0.3	3	81.3	84.38	86.02	-	-
3	32	0.3	5	79.69	83.28	85.36	-	-
3	32	0.4	3	79.57	81.51	85.09	-	-
3	32	0.4	5	79.7	82.13	83.45	-	-
3	64	0.1	3	80.9	85.55	86.05	-	-
3	64	0.1	5	81.19	83.85	84.43	-	-
3	64	0.2	3	82.1	81.71	85.48	-	-
3	64	0.2	5	80.75	83.76	84.49	-	-
3	64	0.3	3	81.47	84.17	83.92	-	-
3	64	0.3	5	80.55	84.18	85.42	-	-
3	64	0.4	3	80.09	84.31	84.58	-	-
3	64	0.4	5	78.39	82.88	83.88	-	-
3	128	0.1	3	82.75	82.08	85.28	-	-
3	128	0.1	5	76.84	81.65	83.98	-	-
3	128	0.2	3	80.57	85.08	84.83	-	-
3	128	0.2	5	80.33	83.81	85.32	-	-
3	128	0.3	3	79.41	82.48	85.64	-	-
3	128	0.3	5	79.79	81.98	84.9	-	-
3	128	0.4	3	82.02	83.48	84.29	-	-
3	128	0.4	5	81.18	84.14	83.79	-	-
3	256	0.1	3	80.52	85.41	86.4	-	-
3	256	0.1	5	81.01	84.65	85.72	-	-
3	256	0.2	3	79.93	84.2	84.15	-	-
3	256	0.2	5	79.8	83.95	85.01	-	-
3	256	0.3	3	81.39	83.79	84.8	-	-
3	256	0.3	5	79	81.65	84.96	-	-
3	256	0.4	3	81.45	82.9	85.22	-	-
3	256	0.4	5	78.71	81.93	83.34	-	-
3	512	0.1	3	81.39	84.24	85.44	-	-

No. of Epochs	Batch Size	Dropout	Kernel Size	Accuracy(Epoch 1)	Accuracy(Epoch 2)	Accuracy(Epoch 3)	Accuracy(Epoch 4)	Accuracy(Epoch 5)
3	512	0.1	5	80.29	82.71	84.49	-	-
3	512	0.2	3	79.27	84.46	85.6	-	-
3	512	0.2	5	77.82	82.82	85.07	-	-
3	512	0.3	3	80.67	82.19	84.95	-	-
3	512	0.3	5	81.52	84.52	83.95	-	-
3	512	0.4	3	79.14	82.15	83.91	-	-
3	512	0.4	5	80.41	82.26	84.47	-	-
4	32	0.1	3	83.63	86.33	87.28	87.57	-
4	32	0.1	5	80.25	82.24	85.18	85.94	-
4	32	0.2	3	83.08	84.78	86.42	87.45	-
4	32	0.2	5	80.21	83.33	83.5	85.62	-
4	32	0.3	3	81.05	84.77	85.69	86.1	-
4	32	0.3	5	77.49	83.72	84.7	86.34	-
4	32	0.4	3	81.29	83.55	84.99	84.55	-
4	32	0.4	5	75.51	79.88	83.1	83.99	-
4	64	0.1	3	82.19	85.13	86.05	86.83	-
4	64	0.1	5	80.94	84.65	85.42	86.47	-
4	64	0.2	3	80.49	82.92	85.14	85.27	-
4	64	0.2	5	81.25	84.47	85.1	86.25	-
4	64	0.3	3	80.65	83.05	85.86	86.31	-
4	64	0.3	5	80.34	83.98	83.53	85.16	-
4	64	0.4	3	78.59	84.71	85.7	86.52	-
4	64	0.4	5	80.62	84.19	84.58	84.6	-
4	128	0.1	3	81.69	86.11	86.67	87.15	-
4	128	0.1	5	78.76	81.82	83.71	86.52	-
4	128	0.2	3	82.18	84.95	86.11	86.89	-
4	128	0.2	5	80.24	83.74	84.47	85.32	-
4	128	0.3	3	79.09	84.79	85.61	86.33	-
4	128	0.3	5	78.73	84.5	85.55	85.84	-
4	128	0.4	3	77.49	82.31	84.63	83.23	-
4	128	0.4	5	78.6	82.22	84.15	85	-
4	256	0.1	3	81.9	85.37	85.7	86.19	-
4	256	0.1	5	79.96	81.89	86.01	86.03	-

No. of Epochs	Batch Size	Dropout	Kernel Size	Accuracy(Epoch 1)	Accuracy(Epoch 2)	Accuracy(Epoch 3)	Accuracy(Epoch 4)	Accuracy(Epoch 5)
4	256	0.2	3	82.38	83.93	85.57	86.75	-
4	256	0.2	5	80.57	83.5	84.42	86.32	-
4	256	0.3	3	79.34	84.89	85.81	86.37	-
4	256	0.3	5	80.41	83.46	84.68	85.15	-
4	256	0.4	3	81.66	84.4	86.21	85.87	-
4	256	0.4	5	80.09	84.25	83.96	84.98	-
4	512	0.1	3	80.3	84.56	86.46	87.18	-
4	512	0.1	5	79.49	84.04	84.08	86.54	-
4	512	0.2	3	81.06	84.96	83.93	86.09	-
4	512	0.2	5	80.65	83.92	85.46	85.7	-
4	512	0.3	3	79.91	82.77	85.04	86.1	-
4	512	0.3	5	81.13	83.39	85.57	85.25	-
4	512	0.4	3	79.21	81.38	83.74	84.66	-
4	512	0.4	5	78.08	81.16	84.92	86.06	-
5	32	0.1	3	81.34	84.92	85.78	86.95	87.86
5	32	0.1	5	81.27	84.28	85.35	85.5	86.52
5	32	0.2	3	80.5	84.75	85.67	85.36	86.87
5	32	0.2	5	79.92	83.83	85.71	85.85	86.64
5	32	0.3	3	78.82	83.08	86.09	86.16	86.84
5	32	0.3	5	79.08	82.01	82.71	84.44	85.96
5	32	0.4	3	79.66	82.91	85.51	86.44	86.42
5	32	0.4	5	77.45	78.84	83.83	84.97	85.92
5	64	0.1	3	79.97	82.51	85.66	86.98	87
5	64	0.1	5	79.08	84.15	83.26	85.97	85.49
5	64	0.2	3	82.6	83.64	86.38	86.1	85.63
5	64	0.2	5	78.13	81.94	84.34	86.35	86.53
5	64	0.3	3	81.53	84.87	85.85	86.03	86.19
5	64	0.3	5	79.28	81.79	84.7	85.3	85.72
5	64	0.4	3	81.89	85.44	84.53	86.75	86.69
5	64	0.4	5	80.32	81.59	85.16	85.37	84.56
5	128	0.1	3	82.02	86.04	86.56	87.26	87.98
5	128	0.1	5	80.25	84.62	84.19	85.98	85.95
5	128	0.2	3	81.74	84.63	86.05	85.49	87.4

No. of Epochs	Batch Size	Dropout	Kernel Size	Accuracy(Epoch 1)	Accuracy(Epoch 2)	Accuracy(Epoch 3)	Accuracy(Epoch 4)	Accuracy(Epoch 5)
5	128	0.2	5	80.89	81.83	84.85	86.01	86.16
5	128	0.3	3	80.1	84.1	85.59	85.59	86.83
5	128	0.3	5	78.47	83.18	84.9	85.93	85.45
5	128	0.4	3	78.09	82.96	84.34	85.32	86.7
5	128	0.4	5	80.74	83.07	84.63	85.49	86.13
5	256	0.1	3	81.18	83.26	85.21	86.26	86.55
5	256	0.1	5	79.97	84.46	85.57	86.53	87.09
5	256	0.2	3	81.15	82.41	85.67	82.72	85.09
5	256	0.2	5	80.02	83.72	84.93	85.64	85.25
5	256	0.3	3	81.95	85.45	86.22	86.96	86.5
5	256	0.3	5	80.12	83.39	84.13	85.77	85.7
5	256	0.4	3	80.48	82.02	84.47	85.78	86.39
5	256	0.4	5	79.62	81.8	84.03	85.22	86.03
5	512	0.1	3	80.78	84.91	86.19	86.61	86.47
5	512	0.1	5	81.41	83.02	84.54	86.55	85.02
5	512	0.2	3	80.76	84.88	84.13	86.49	87.14
5	512	0.2	5	81.61	83.66	85.29	86.45	86.06
5	512	0.3	3	78.97	84.09	85.74	85.85	86.23
5	512	0.3	5	79.83	83.57	85.21	84.99	86.11
5	512	0.4	3	80.71	83.64	85.41	86.33	86.19
5	512	0.4	5	80.76	84.39	84.25	85.82	86.59

Extension:

1. Evaluate more than 3 dimensions:

We have evaluated 4 dimensions in task 4

1- Batch size (32,64,128,256,516)
2- Number of epochs (3,4,5)
3- Dropout Rates (0.1,0.2,0.3,0.4)
4- Size of convolution layers (3,5)

Intotal we have 5*3*4*2 = 120, therefore we have created 120 network variations.

Conclusion:

In this project, we are able to understand and use PYTORCH. We learned to build a deep learning network, train and test it, analyze the different layers of the model, we understood how the convolution layers, dropout layers, max-pooling layers, and fully connected layers work. We also got to know and modify the various features of the network and understand the changes happening.

References:

- https://pytorch.org/tutorials/beginner/basics/intro.html
- https://nextjournal.com/gkoehler/pytorch-mnist
- https://pytorch.org/tutorials/beginner/basics/data_tutorial.html
- https://discuss.pytorch.org/t/loading-mnist-from-pytorch/137456/5