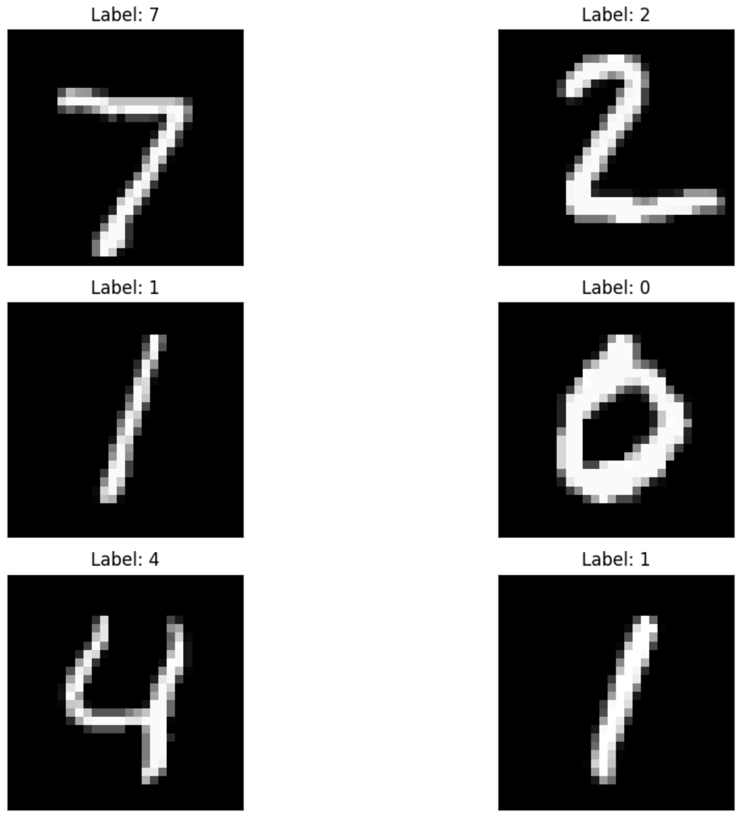
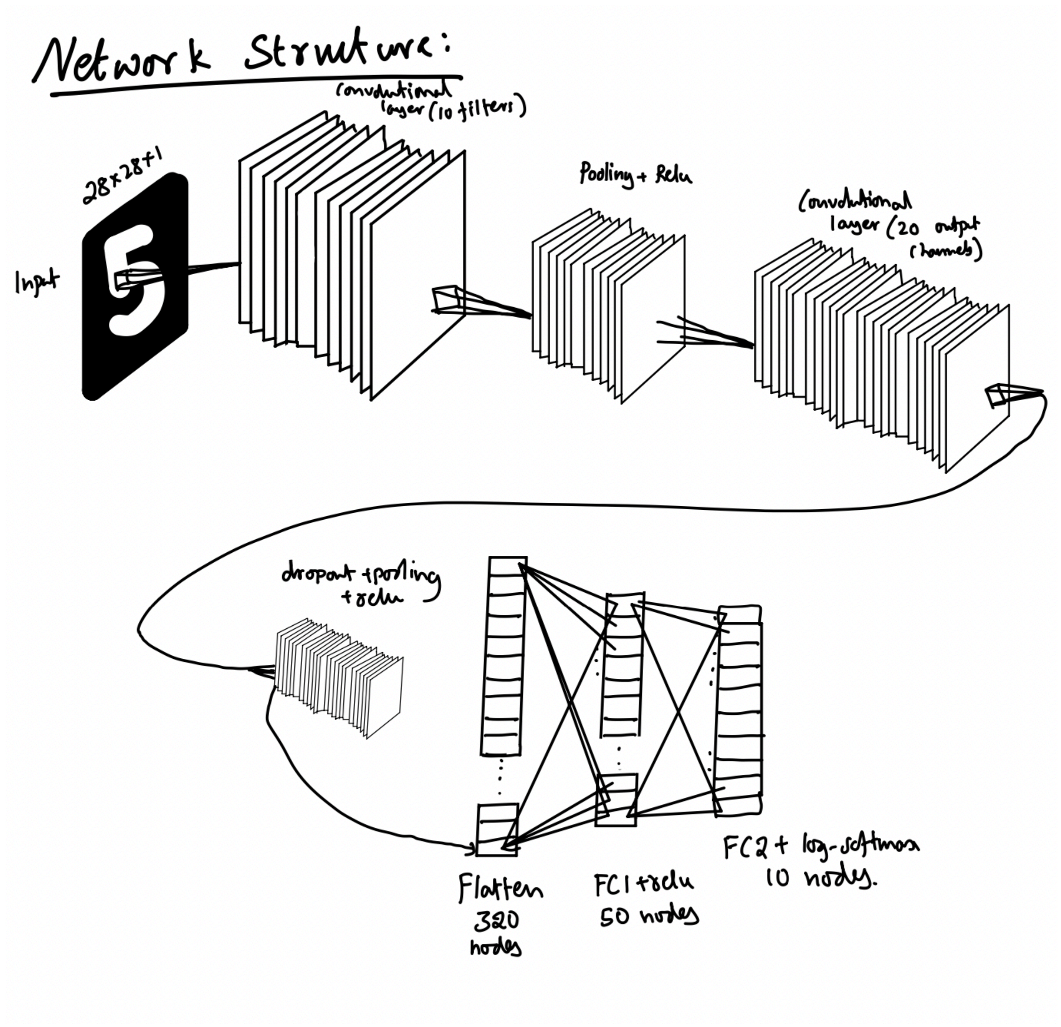
Recognition using Deep Networks

In this project, I have built a CNN model to train handwritten digits and also have applied transfer learning to recognize Greek Letters. I have analyzed different layers weights and also the filters used to train the model. I have built a live digit recognition system using the model trained. I have implemented hyper tuning of different parameters to analyze and obtain the best model to predict the handwritten digits. Below is the description of each task in detail.

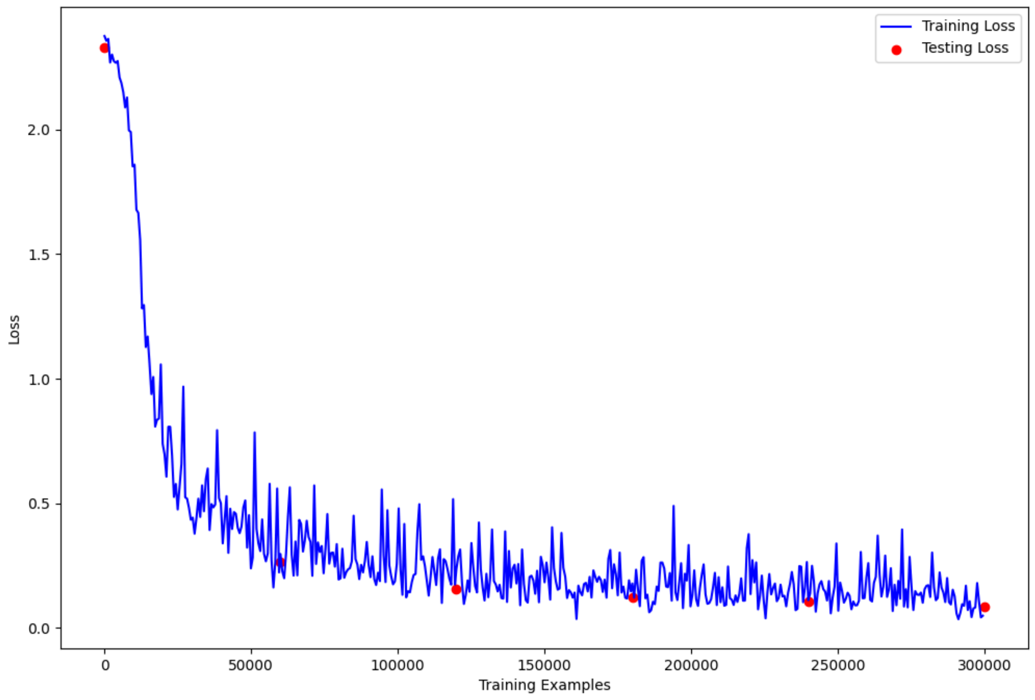
1. Task1: Build and train network to recognize digits:
    A.



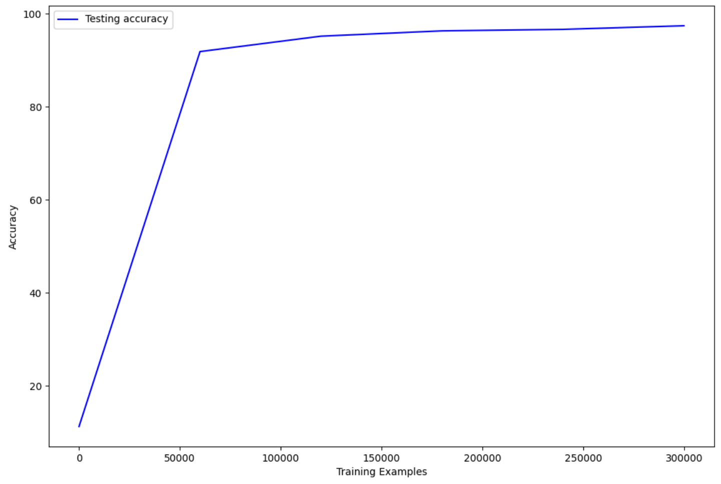
B.


Network Structure:
Convolutional layer (10 filters)
28x28x1
Input
Pooling + Relu
Convolutional layer (20 output channels)
dropout + pooling + relu
Flatten 320 nodes
FC1 + relu 50 nodes
FC2 + log-softmax 10 nodes.

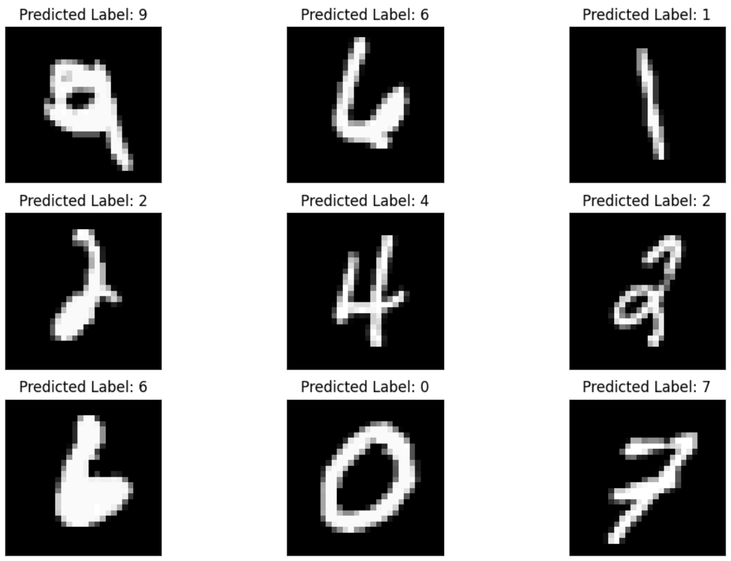
C. Accuracy Plot:



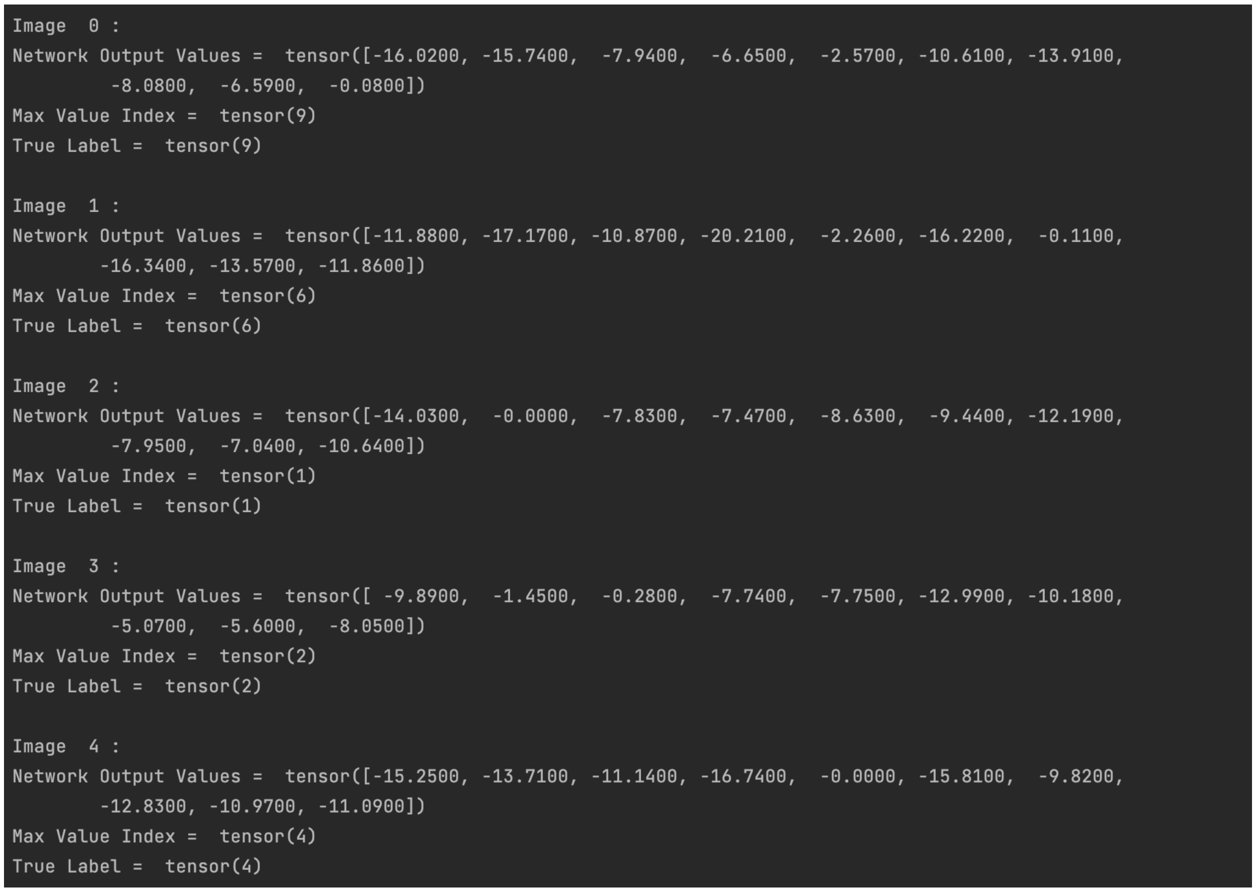
Tes Accuracy Plot:



D. I have successfully saved the model to a file once the training was completed

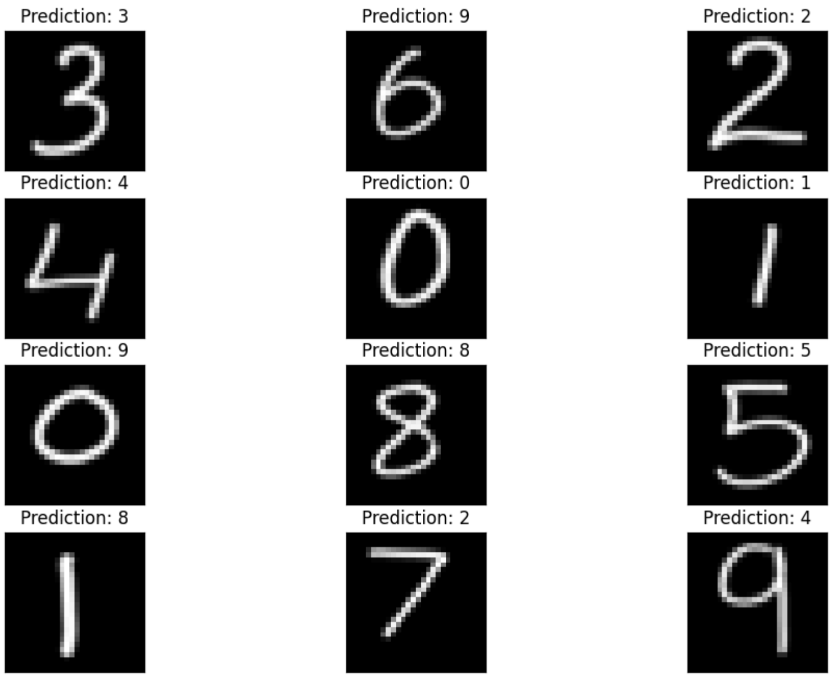


E.



```
Image  0 :
Network Output Values =  tensor([-16.0200, -15.7400,  -7.9400,  -6.6500,  -2.5700, -10.6100, -13.9100,
         -8.0800,  -6.5900,  -0.0800])
Max Value Index =  tensor(9)
True Label =  tensor(9)

Image  1 :
Network Output Values =  tensor([-11.8800, -17.1700, -10.8700, -20.2100,  -2.2600, -16.2200,  -0.1100,
        -16.3400, -13.5700, -11.8600])
Max Value Index =  tensor(6)
True Label =  tensor(6)

Image  2 :
Network Output Values =  tensor([-14.0300,  -0.0000,  -7.8300,  -7.4700,  -8.6300,  -9.4400, -12.1900,
         -7.9500,  -7.0400, -10.6400])
Max Value Index =  tensor(1)
True Label =  tensor(1)

Image  3 :
Network Output Values =  tensor([ -9.8900,  -1.4500,  -0.2800,  -7.7400,  -7.7500, -12.9900, -10.1800,
         -5.0700,  -5.6000,  -8.0500])
Max Value Index =  tensor(2)
True Label =  tensor(2)

Image  4 :
Network Output Values =  tensor([-15.2500, -13.7100, -11.1400, -16.7400,  -0.0000, -15.8100,  -9.8200,
        -12.8300, -10.9700, -11.0900])
Max Value Index =  tensor(4)
True Label =  tensor(4)
```

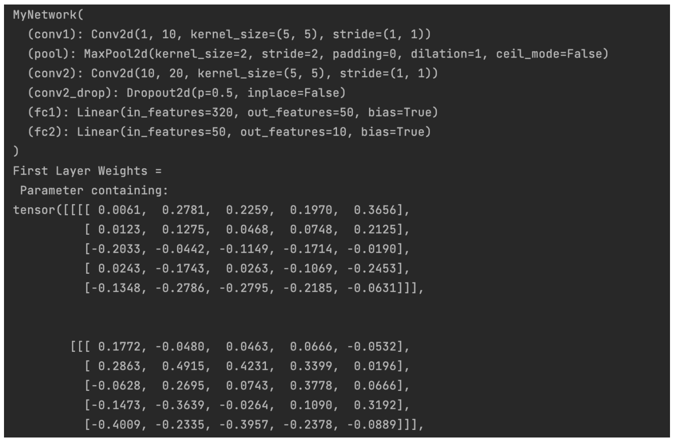
F.



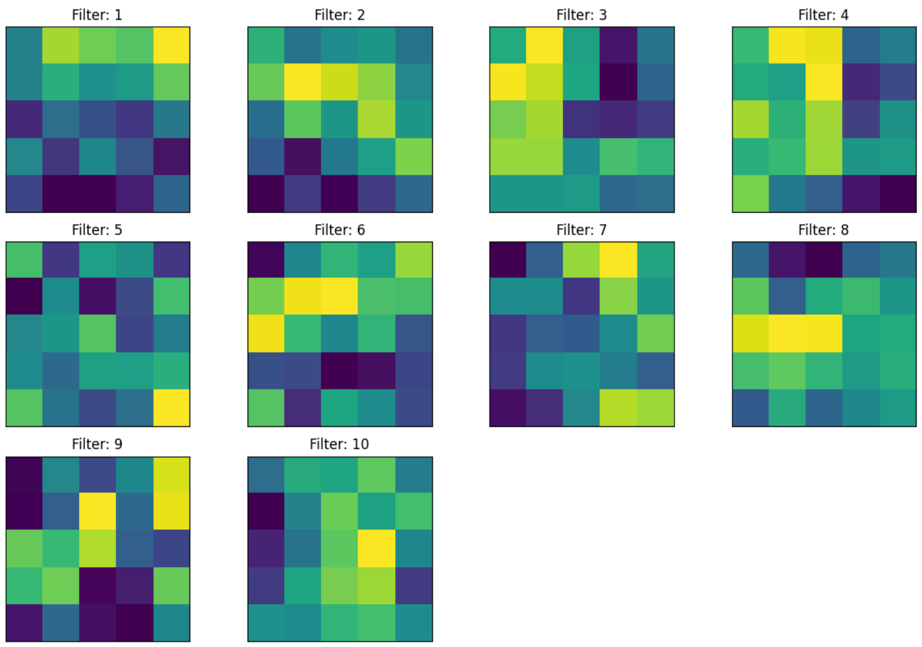
Above are the predictions obtained for the new digits, however, the predictions are not very accurate, maybe because the way the digits are written by me is different than it is trained on. I can observe that the digits written in the style which are similar to the MNIST dataset predicts correctly and the digits which are written little differently don't predict that well. Increasing the epochs or fine tuning the parameters might help in predicting well.

2. Examine your Network:

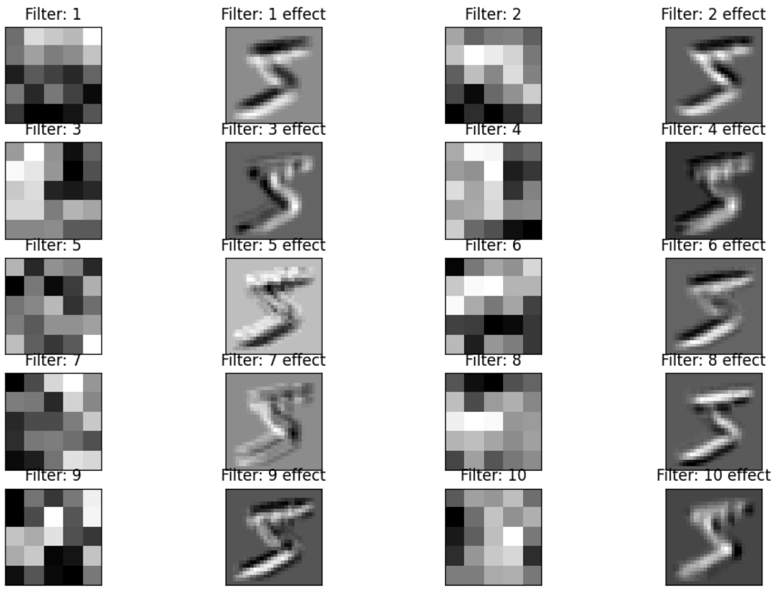
A. Analyze the First Layer:



```
MyNetwork(
  (conv1): Conv2d(1, 10, kernel_size=(5, 5), stride=(1, 1))
  (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (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)
)
First Layer Weights =
 Parameter containing:
tensor([[[[ 0.0061,  0.2781,  0.2259,  0.1970,  0.3656],
          [ 0.0123,  0.1275,  0.0468,  0.0748,  0.2125],
          [-0.2033, -0.0442, -0.1149, -0.1714, -0.0190],
          [ 0.0243, -0.1743,  0.0263, -0.1069, -0.2453],
          [-0.1348, -0.2786, -0.2795, -0.2185, -0.0631]]],


        [[[ 0.1772, -0.0480,  0.0463,  0.0666, -0.0532],
          [ 0.2863,  0.4915,  0.4231,  0.3399,  0.0196],
          [-0.0628,  0.2695,  0.0743,  0.3778,  0.0666],
          [-0.1473, -0.3639, -0.0264,  0.1090,  0.3192],
          [-0.4009, -0.2335, -0.3957, -0.2378, -0.0889]]],
```



B. Filter Effects



Here, we can see that different filters capture different features of the image. Some features have highlighted the specific regions. We can see that, filter 5 and 7 is more blurred and feature 9 is trying to detect edges as we can see that the filter 9 has black values corresponding to the lower region of the digit and hence highlights it well. Filter 6 identifies horizontal pixel values which can be clearly seen in the accept of filter 6

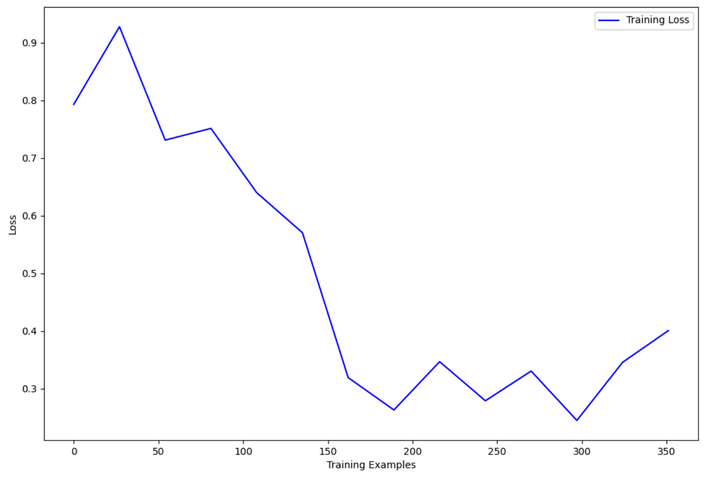
3. Transfer Learning with Greek Letters

Using the 27 samples,training the model with 10-13 epochs will predict 26 samples accurately giving 96% accuracy. The average loss with 10 epochs was 0.41, for 11 epochs it was 0.38.

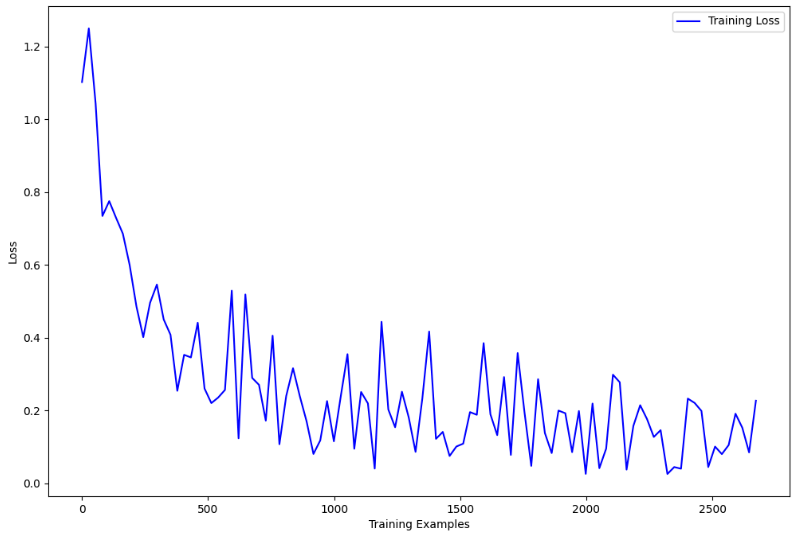
With 14 epochs, got 100% accuracy with average loss of 0.299

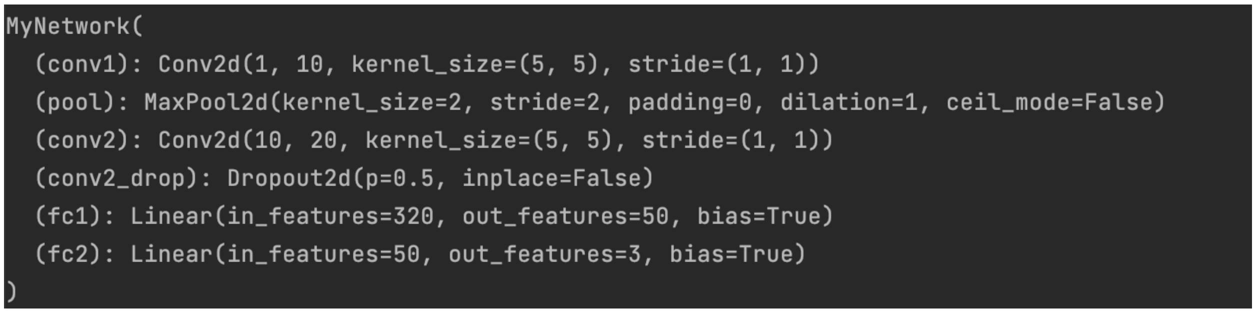
100 epochs will give average loss of around 0.06 and accurately predicted all 27 samples(100% accuracy)

Training loss for 14 epochs:



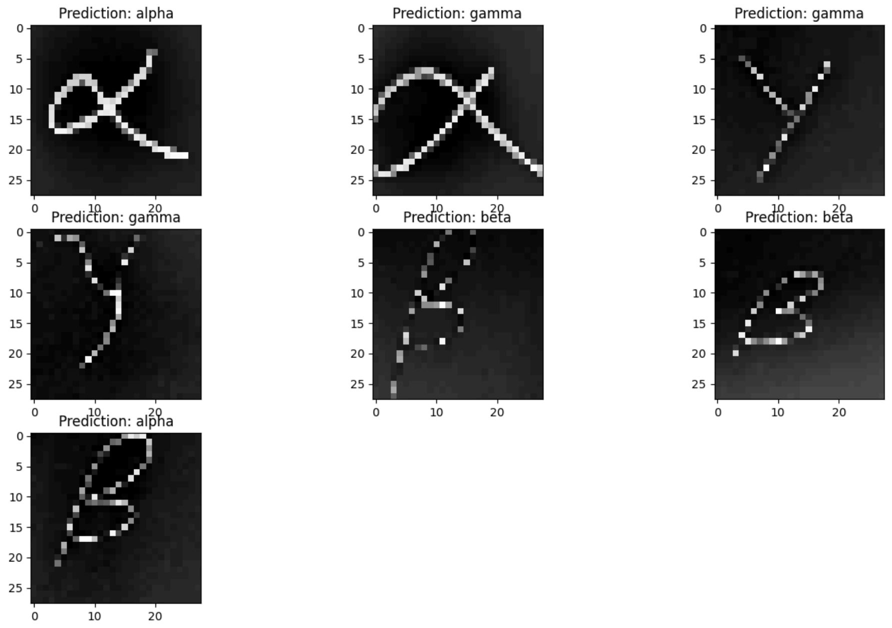
Training Loss for 100 epochs:





```
MyNetwork(
  (conv1): Conv2d(1, 10, kernel_size=(5, 5), stride=(1, 1))
  (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (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)
)
```

Above is the screenshot of the modified network to train the greek letters. Here, the final layer has only 3 output nodes to classify the 3 target greek letters.



This is trained with 100 epochs and we can see that it predicts correctly 5 out of 7. With better samples and training with better parameter values, we might get better results.

4. Design your own experiment:

I have used MNIST digits dataset to analyze the models.

For analyzing the designs of the model, I have considered hyper tuning parameters dropout probability, second convolution layers size and kernel size. The dropout probability is used to reduce overfitting by dropping some neurons randomly. Hence, higher the dropout probabilities, the model will take more time to converge. Similarly, having higher convolution layer size will be able to capture complex patterns and hence will be able to predict well. However, it comes with a disadvantage of having higher computations time because with more filters, the parameters increase and hence the training time increases drastically. Having lesser convolutional layer size will train faster but might miss capturing some complex patterns thereby resulting in giving bad accuracy. Having higher kernel size will allow it to capture some patterns

which are present in bigger areas of the images and smaller kernel size will focus on capturing local features. Larger kernel size will increase the computational time because of higher parameters requirements and smaller kernel size will take lesser time to train. Smaller kernel sizes are better to identify local features like edges and higher kernel size will be able to detect larger region features. Here, I feel, the larger region features are important in identifying images and hence higher kernel size might give better results.
Analysis from the results: We can observe from the results that, for lower values dropout values with other parameters value lower, the training time is lesser compared to when the dropout values are higher. Similarly for the convolutional layers size and kernel size. With lower values of these parameters, the training time is lesser compared to higher values. The training loss values are fluctuating when the dropout values vary. For some its higher than lower values and for 0.5, the loss is highest compared to other dropout values. We can see that increasing the convolutional layer is giving better loss results compared to lower size of the convolutional layer. What I observed from the results is that the kernel size of 5 is giving test and train results compared to other kernel sizes.
From evaluating 96 models, best training models parameters are:
Dropout_probability: 0.0
2nd Convolution Layer Size: 60
Kernel Size: 5

From evaluating 96 models, best testing models parameters are:
Dropout_probability: 0.1
2nd Convolution Layer Size: 80
Kernel Size: 5.
I have attached the excel of the results of all the 96 models being analyzed.

Extensions:

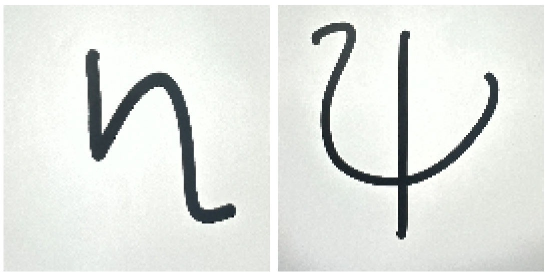
1. **Analyzed more dimensions for obtaining better model:**

Since, number of epochs, training_batch_size, leraning_rate will also play a huge factor in obtaining a better model and also efficiently managing the memory, I have hypertuned these parameters to analyze how the model performs with these parameters. Having a higher number of epochs will produce better results, however it will take more time as it needs to train for a longer time repeatedly on the given data. Having higher batch size will train the given model faster without compromising on the accuracy but will take up more space as it will load more data into the memory at once. Learning rate determines the size of the step optimizer takes during gradient descent and hence plays a major role in training results. Lower learning rate will increase training time and higher learning rate will decrease training time however higher learning rate model might have not converged to its best values compared to lower learning rates.
We can see from the results that having a higher number of epochs will give better test and training results compared to models trained on lower epochs but the training times are considerably higher compared with models trained on the same other parameters. Similarly, the models trained with higher batch size have lesser time compared models trained on lower batch

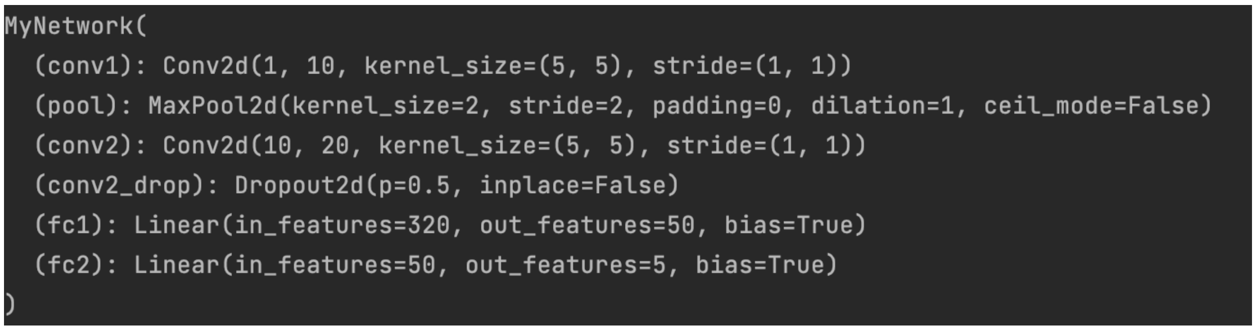
size. We can observe that the loss values won't change much with change in batch sizes. However, the impact of learning rates taken is not much as the time taken is almost similar and is fluctuating and there is no considerable difference. I can also see that the loss values are better for higher learning rates considered and hence it is important to correctly finetune the hyperparameters to get the best results.
I have attached the excel of the results in the submission.

2. **Live Digit Recognition:**
   I have considered the best parameters obtained from the above analysis and have used it to train the model to perform live digit recognition.
   I have attached the video in the below link:
   https://drive.google.com/drive/folders/1bRL-U9pm-u-TjYuCC0vmIEMkmbLLUv4q?usp=drive_link

3. **Trained the transfer learning model for more greek characters:**
   I have added two more characters eta and psi and retrained the model by changing the last layer to have 5 output nodes instead of 3 and retrained the model.
   Letters considered:

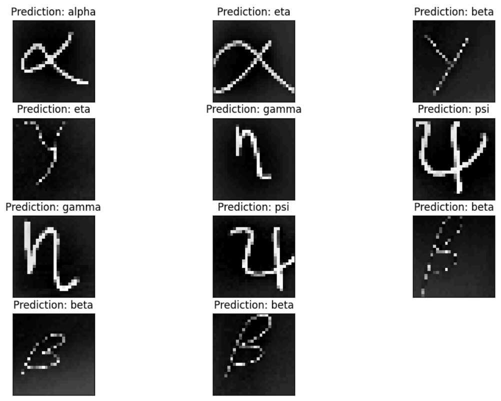


Model Architecture:



```
MyNetwork(
  (conv1): Conv2d(1, 10, kernel_size=(5, 5), stride=(1, 1))
  (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (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=5, bias=True)
)
```

With 100 epochs, the model gives training accuracy of 100%, might have given for lesser epochs as well and the average loss is o=0.08.
Below is the prediction for test dataset:



Adding the new greek letters is predicting few correctly and few incorrectly. Having more data might help in producing better results.