CS 5330

Pattern Recognition and Computer Vision

Project 5: Recognition using Deep Networks

Team Members:
Member 1: Vishnu Vardhan Kolla
NUID: 002752854

Member 2: Vidya Ganesh
NUID: 002766414

Overview of the overall project:

In the completed project, we learned how to build, train, analyze, and modify a deep network for a recognition task. The MNIST digit recognition data set was used primarily because it was simple enough to build and train a network without a GPU, but also because it was challenging enough to provide a good example of what deep networks can do.
Throughout the project, we learned about the different types of neural network architectures, such as convolutional neural networks (CNNs) and deep feedforward neural networks, and how to train them using techniques like backpropagation and stochastic gradient descent. We also explored strategies for optimizing network performance, such as hyperparameter tuning and regularization techniques like dropout.
As part of the project's conclusion, we also attached the project proposal for the final project.

As part of **extensions**:

1. Used more Greek letters than alpha, beta, and gamma including theta, pie, and phi.
2. Explored different computer vision task with available data. In our case we built a smaller version of VGG model to classify Fashion MNIST.
3. Loaded and evaluated a pretrained model with its first couple of convolutional layers as in task 2.
4. Replaced the first layer of the MNIST network with a Gabor filter and retrained the rest of the network, holding the first layer constant and compared the results.
5. Built a live video digit recognition application using the trained network.

Required Images for Task 1: Build and train a network to recognize digits

A. **Get the MNIST digit data set -** *Include a plot of the first six example digits in your report.*

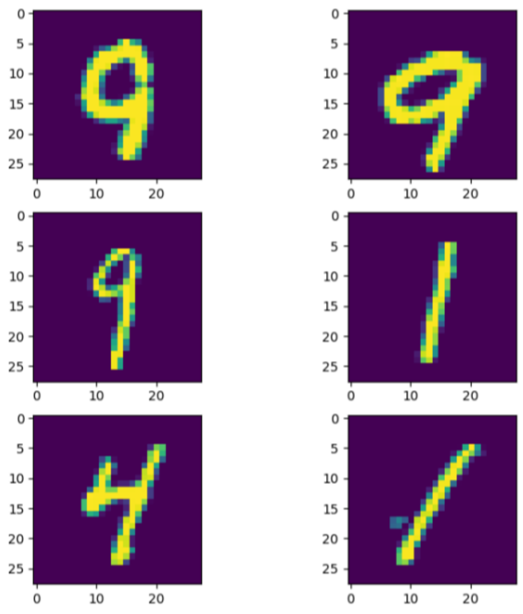


Fig for A: Downloaded the dataset and the first 6 images with the respective ground truth are as above.

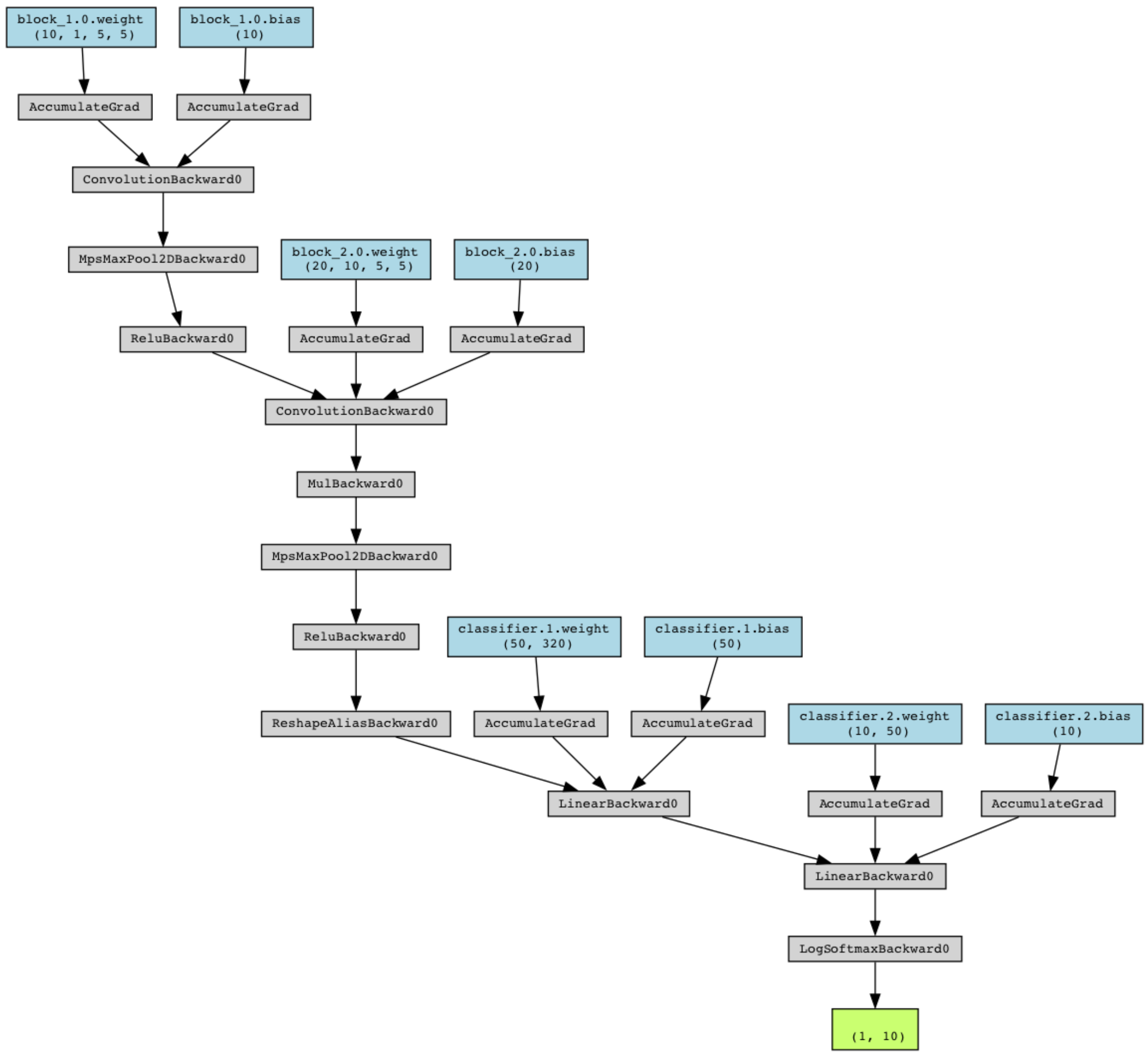
B. Make your network code repeatable

We set the random seed for the torch package, `torch.manual_seed(42)`, at the start of main function. To make processing truly repeatable, we didn't turn on MPS as we used Mac Silicon Chip for training the model.

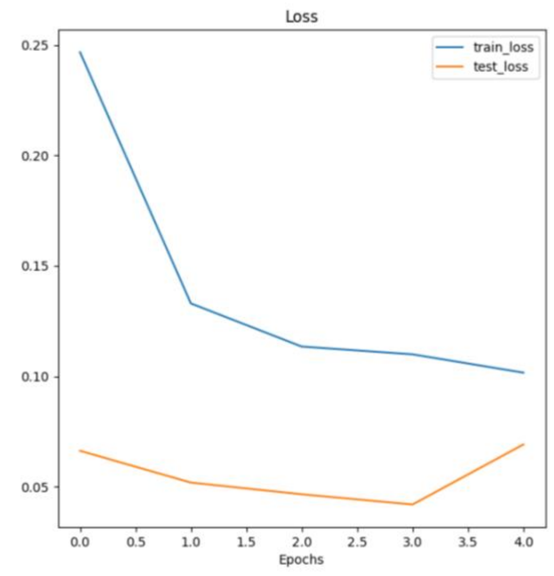
C. Build a network model – *Put a diagram of your network in your report.*

Input : 28*28 1 channel(Gray Scale) image.
A **convolution** layer with 10 5x5 filters. Produces 10*24*24 output.
A **dropout** layer with a 0.5 dropout rate (50%)
A **max-pooling** layer with a 2x2 window and a ReLU function applied. Produces 20*4*4 output.
A **max-pooling** layer with a 2x2 window and a ReLU function was applied. Produces 10*12*12 output.
A **fully connected Linear** layer with 10 nodes and the log_softmax function applied to the output. Produces 10 outputs.
A **convolution** layer with 20 5x5 filters. Produces 20*8*8 output.
A flattening operation followed by a **fully connected Linear** layer that takes 320 inputs and converts to 50 outputs.

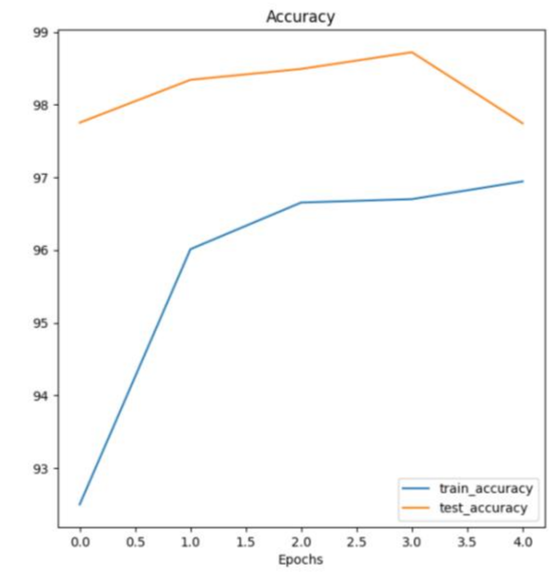
We used torchviz to visualize the neural network. Created a function, which when given a model and the data creates a .png file for the model.



D. **Train the model -** *Collect the accuracy scores and plot the training and testing accuracy in a graph. Include this plot in your report.*

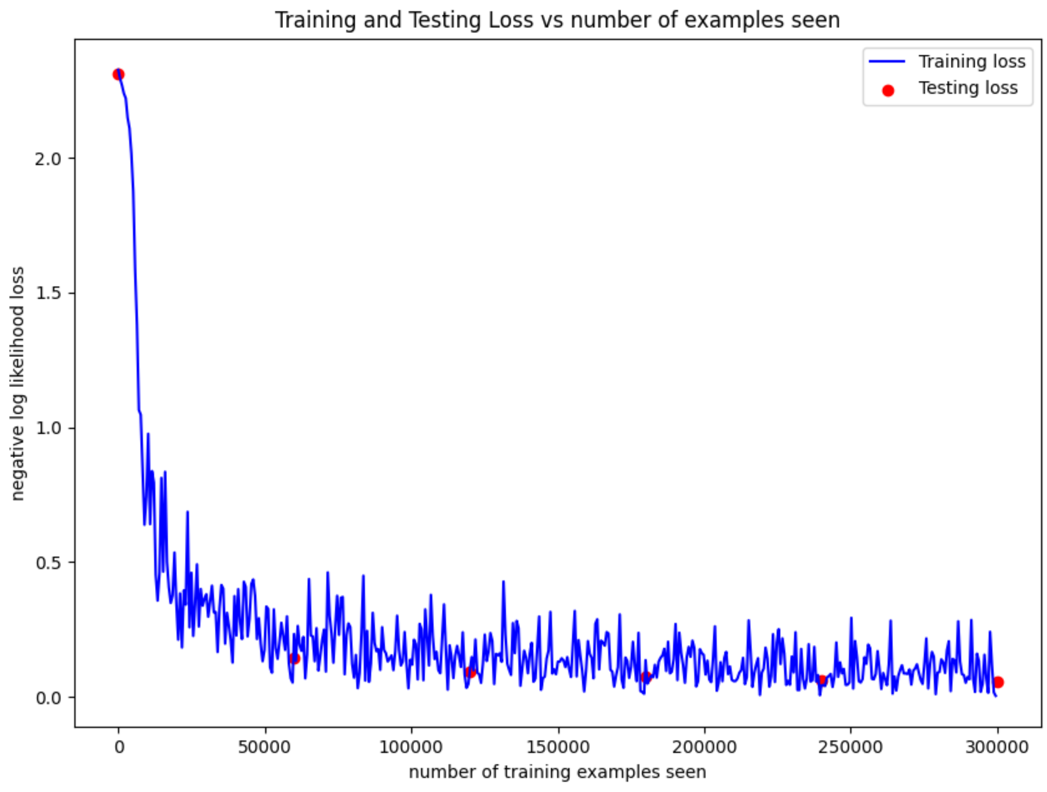


(a) Loss

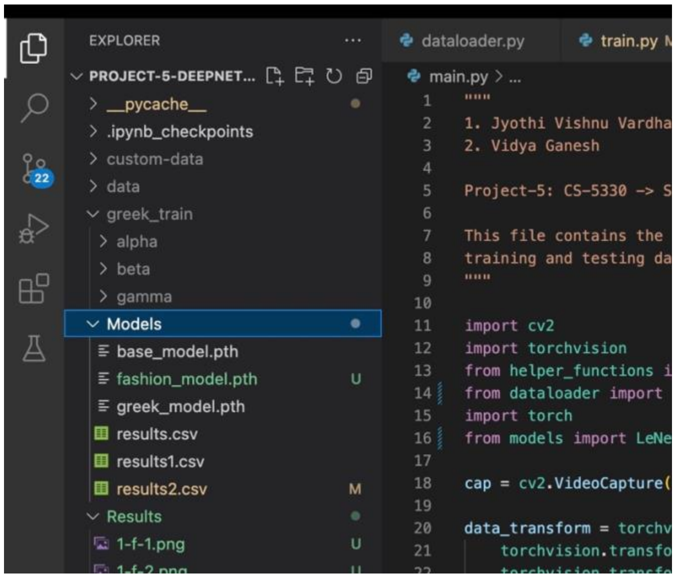


(b) Accuracy

The loss and accuracy graphs are shown above for every epoch.



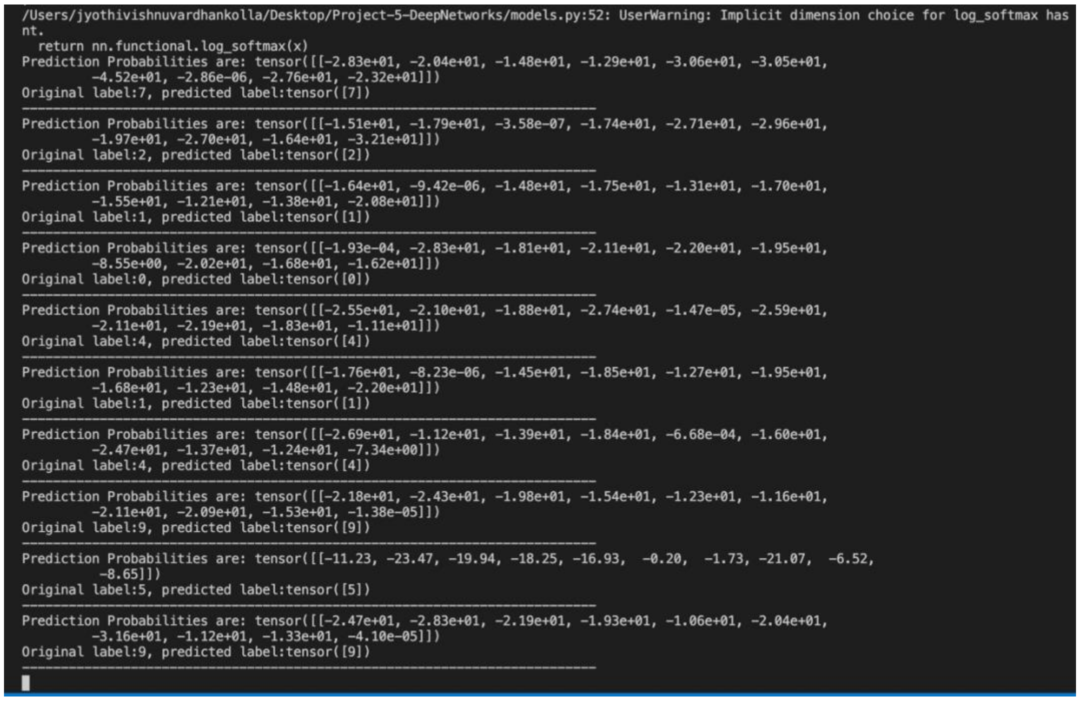
E. Save the network to a file



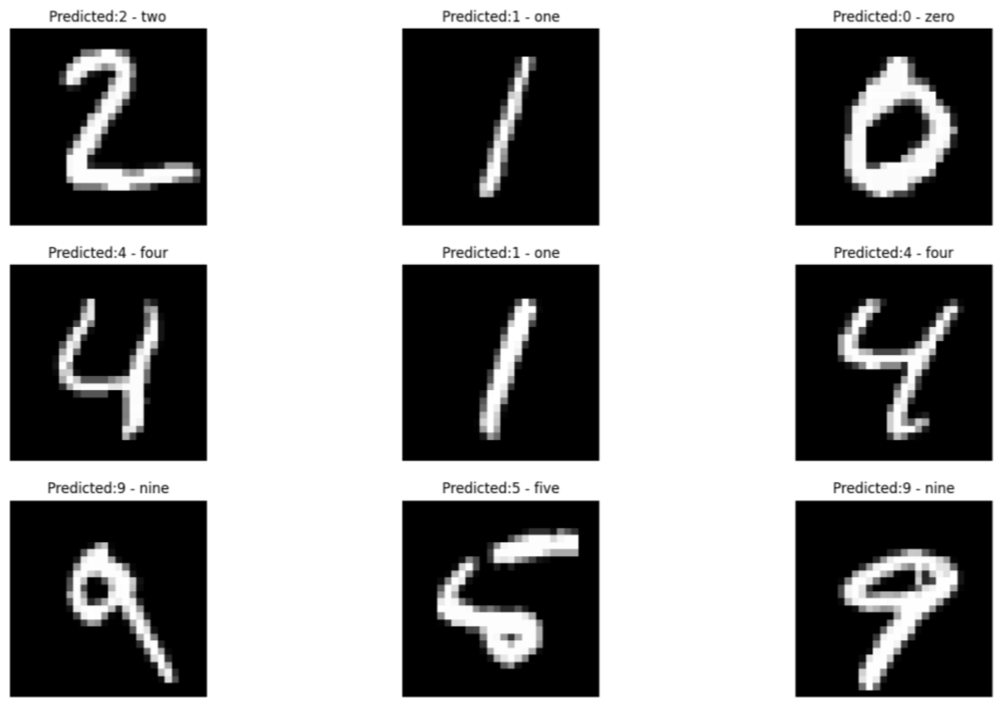
```
"""
1. Jyothi Vishnu Vardha
2. Vidya Ganesh

Project-5: CS-5330 -> S

This file contains the
training and testing da
"""

import cv2
import torchvision
from helper_functions i
from dataloader import
import torch
from models import LeNe

cap = cv2.VideoCapture(

data_transform = torchv
    torchvision.transfo
```

F. **Read the network and run it on the test set**
Include a table (or screen shot) of your printed values and the plot of the first 9 digits in your report.

Loaded the saved model and given 10 images from the MNIST test set. Showing the comparison results and the classification in the results.

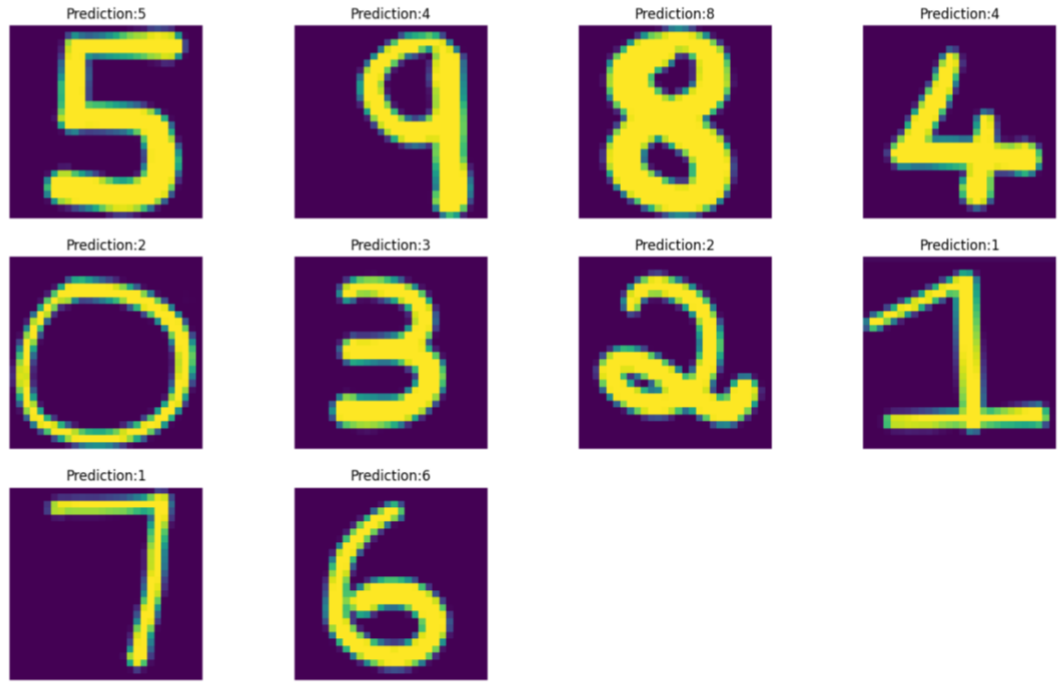


```
/Users/jyothivishnuvardhankolla/Desktop/Project-5-DeepNetworks/models.py:52: UserWarning: Implicit dimension choice for log_softmax has
nt.
  return nn.functional.log_softmax(x)
Prediction Probabilities are: tensor([[-2.83e+01, -2.04e+01, -1.48e+01, -1.29e+01, -3.06e+01, -3.05e+01,
         -4.52e+01, -2.86e-06, -2.76e+01, -2.32e+01]])
Original label:7, predicted label:tensor([7])
--------------------------------------------------------------------------------
Prediction Probabilities are: tensor([[-1.51e+01, -1.79e+01, -3.58e-07, -1.74e+01, -2.71e+01, -2.96e+01,
         -1.97e+01, -2.70e+01, -1.64e+01, -3.21e+01]])
Original label:2, predicted label:tensor([2])
--------------------------------------------------------------------------------
Prediction Probabilities are: tensor([[-1.64e+01, -9.42e-06, -1.48e+01, -1.75e+01, -1.31e+01, -1.70e+01,
         -1.55e+01, -1.21e+01, -1.38e+01, -2.08e+01]])
Original label:1, predicted label:tensor([1])
--------------------------------------------------------------------------------
Prediction Probabilities are: tensor([[-1.93e-04, -2.83e+01, -1.81e+01, -2.11e+01, -2.20e+01, -1.95e+01,
         -8.55e+00, -2.02e+01, -1.68e+01, -1.62e+01]])
Original label:0, predicted label:tensor([0])
--------------------------------------------------------------------------------
Prediction Probabilities are: tensor([[-2.55e+01, -2.10e+01, -1.88e+01, -2.74e+01, -1.47e-05, -2.59e+01,
         -2.11e+01, -2.19e+01, -1.83e+01, -1.11e+01]])
Original label:4, predicted label:tensor([4])
--------------------------------------------------------------------------------
Prediction Probabilities are: tensor([[-1.76e+01, -8.23e-06, -1.45e+01, -1.85e+01, -1.27e+01, -1.95e+01,
         -1.68e+01, -1.23e+01, -1.48e+01, -2.20e+01]])
Original label:1, predicted label:tensor([1])
--------------------------------------------------------------------------------
Prediction Probabilities are: tensor([[-2.69e+01, -1.12e+01, -1.39e+01, -1.84e+01, -6.68e-04, -1.60e+01,
         -2.47e+01, -1.37e+01, -1.24e+01, -7.34e+00]])
Original label:4, predicted label:tensor([4])
--------------------------------------------------------------------------------
Prediction Probabilities are: tensor([[-2.18e+01, -2.43e+01, -1.98e+01, -1.54e+01, -1.23e+01, -1.16e+01,
         -2.11e+01, -2.09e+01, -1.53e+01, -1.38e-05]])
Original label:9, predicted label:tensor([9])
--------------------------------------------------------------------------------
Prediction Probabilities are: tensor([[-11.23, -23.47, -19.94, -18.25, -16.93,  -0.20,  -1.73, -21.07,  -6.52,
          -8.65]])
Original label:5, predicted label:tensor([5])
--------------------------------------------------------------------------------
Prediction Probabilities are: tensor([[-2.47e+01, -2.83e+01, -2.19e+01, -1.93e+01, -1.06e+01, -2.04e+01,
         -3.16e+01, -1.12e+01, -1.33e+01, -4.10e-05]])
Original label:9, predicted label:tensor([9])
--------------------------------------------------------------------------------
█
```



G. Test the network on new inputs

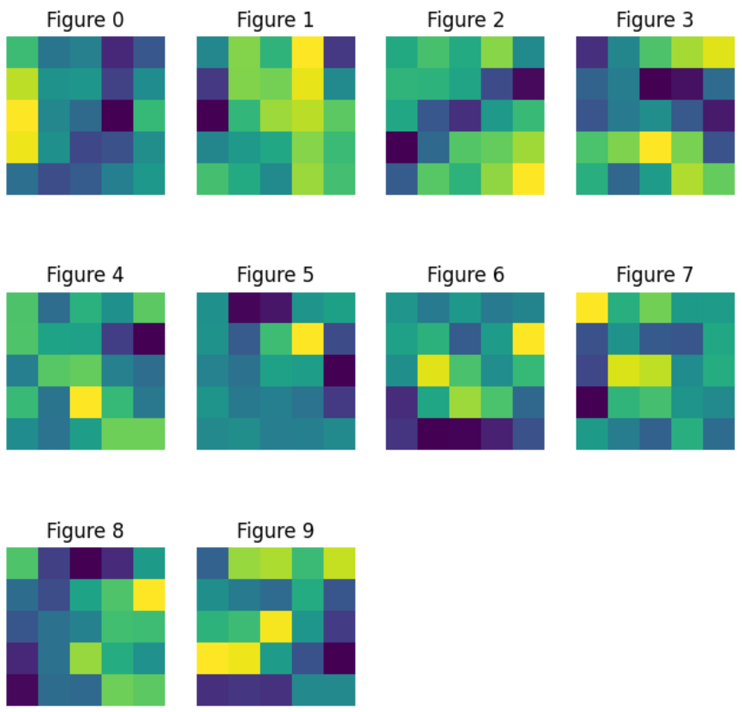
Display how well the network performed on this new input in your report. We wrote custom digits in our iPad, took screenshots, and processed the images and ran inference. The below are the predictions during inference.



Required images for Task 2: Examine your network

A. Analyze the first layer

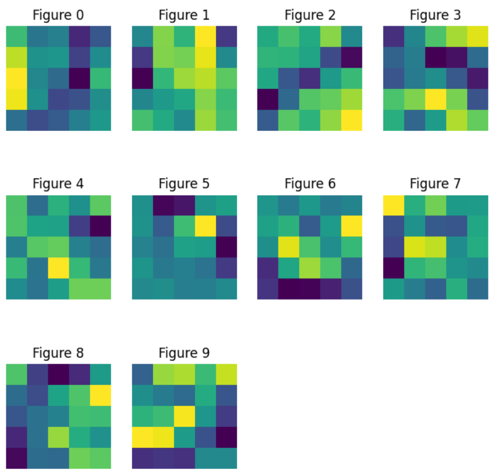
Visualized the ten filters using pyplot. Used the pyplot functions figure, subplot, and imshow to make a 3x4 grid of figures such as the image below. Set xticks and yticks to the empty list [],this gave a cleaner plot.

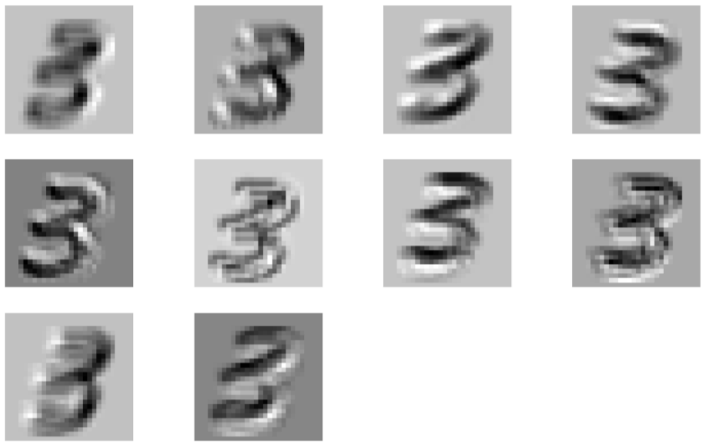


The first 10 filters in our first convolutional layer are shown above

B. Show the effect of the filters - *In your report, include the plot and note whether the results make sense given the filters.*

We used OpenCV's filter2D function to apply the 10 filters to the first training example image. We generated a plot of the 10 filtered images such as the one below. When working with the weights, we made sure pyTorch does not calculate gradients.

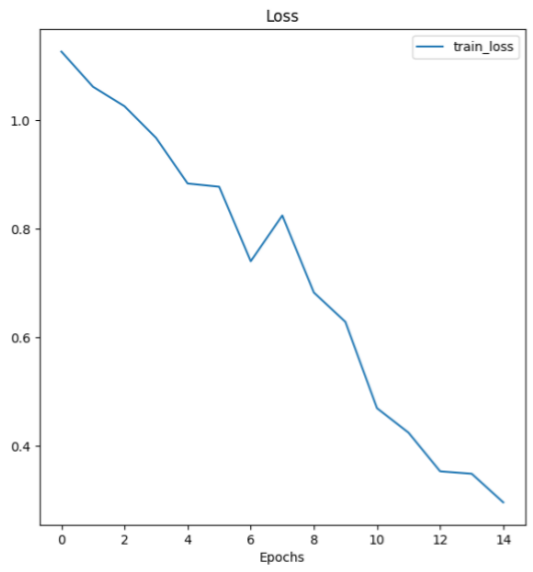


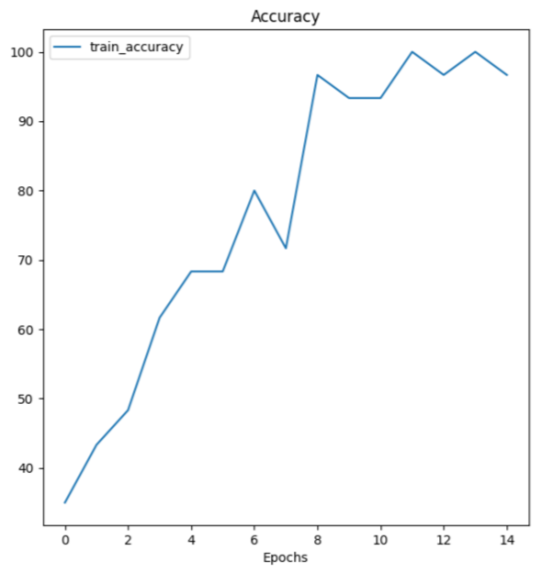


From the above effect of the filter analyzed, we can see how digit 3 features are extracted. All the filters identify the edges of the digit 3. The learning is random apart from which, we can predict what can be learned. Aside from the random guess than one of the features would help in identifying edges.

Required Image for Task 3: Transfer Learning on Greek Letters

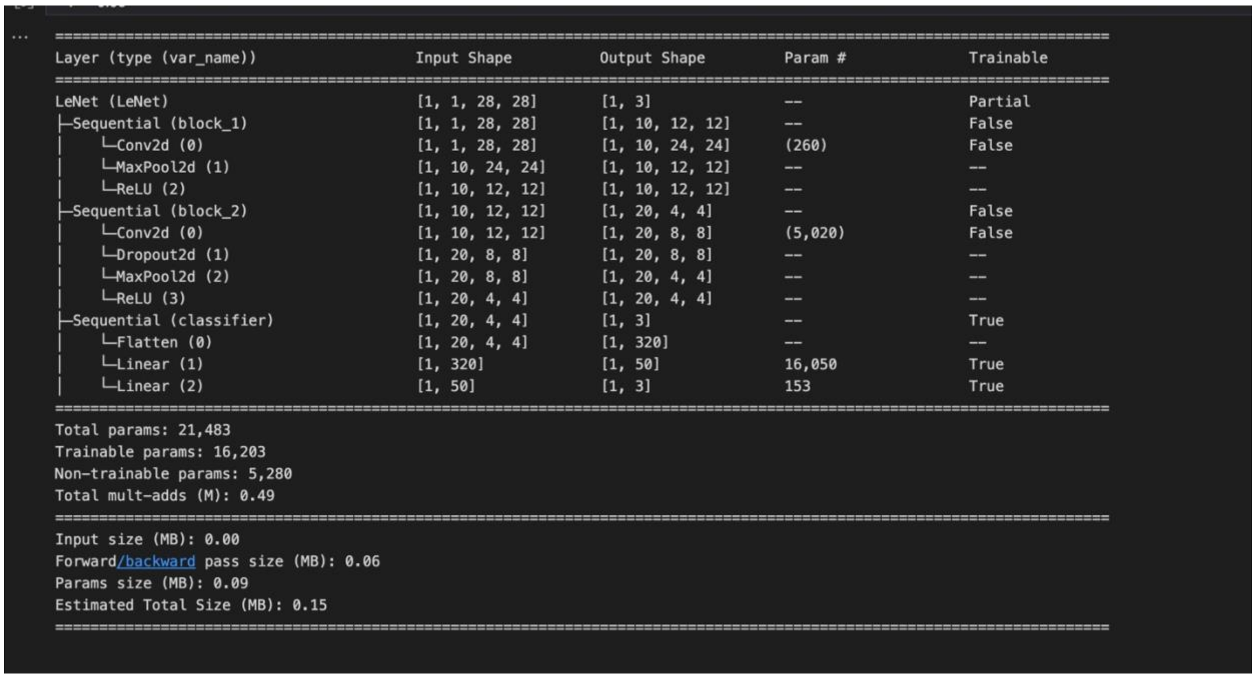
A plot of the training error, a printout of your modified network, and the results on the additional data. Please include a zip file with your additional examples in your submission.



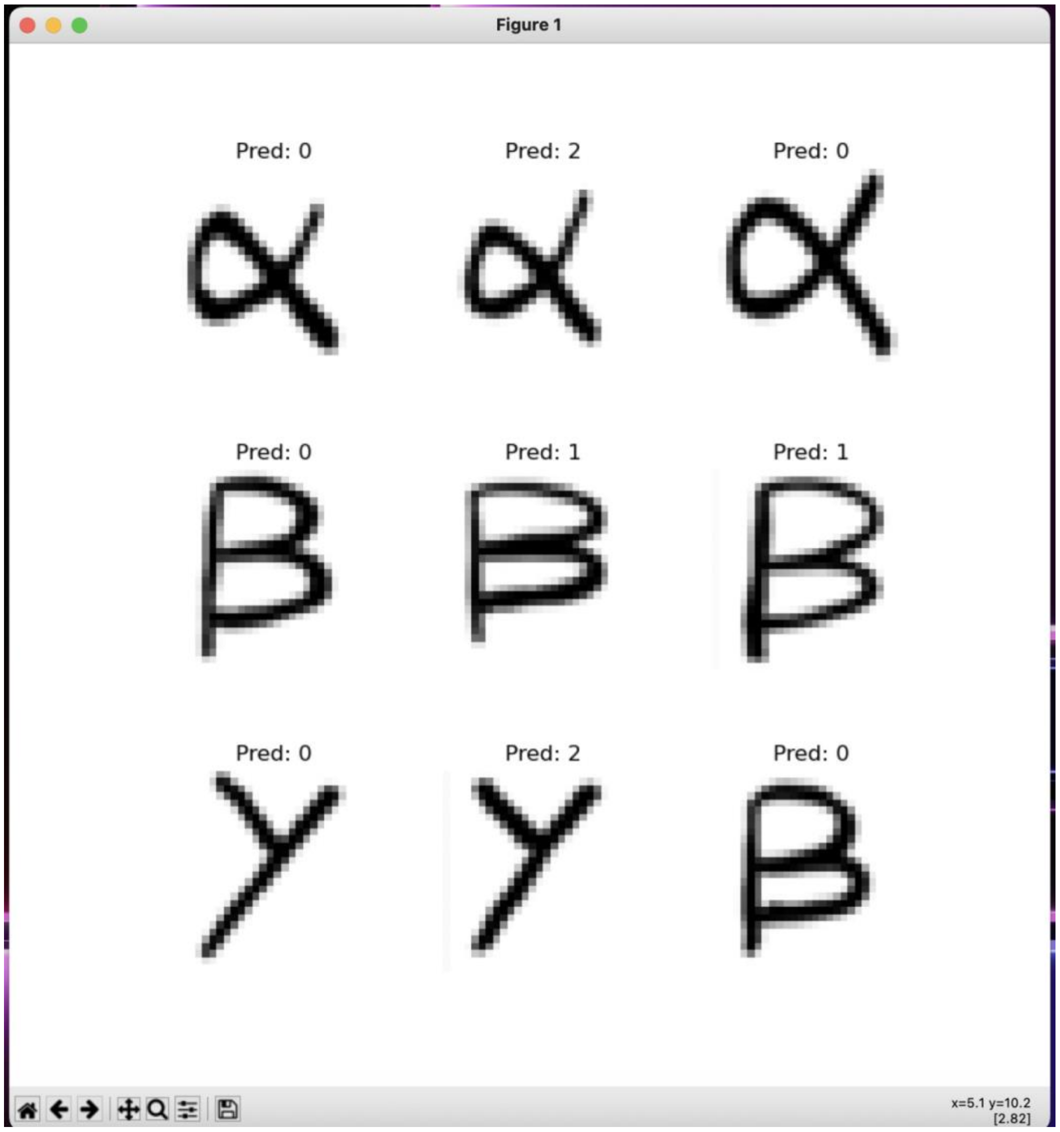


How many epochs does it take using the 27 examples to perfectly identify them?

12 epochs



```
==========================================================================================================================
Layer (type (var_name))                  Input Shape          Output Shape         Param #              Trainable
==========================================================================================================================
LeNet (LeNet)                            [1, 1, 28, 28]       [1, 3]               --                   Partial
├─Sequential (block_1)                   [1, 1, 28, 28]       [1, 10, 12, 12]      --                   False
│    └─Conv2d (0)                        [1, 1, 28, 28]       [1, 10, 24, 24]      (260)                False
│    └─MaxPool2d (1)                     [1, 10, 24, 24]      [1, 10, 12, 12]      --                   --
│    └─ReLU (2)                          [1, 10, 12, 12]      [1, 10, 12, 12]      --                   --
├─Sequential (block_2)                   [1, 10, 12, 12]      [1, 20, 4, 4]        --                   False
│    └─Conv2d (0)                        [1, 10, 12, 12]      [1, 20, 8, 8]        (5,020)              False
│    └─Dropout2d (1)                     [1, 20, 8, 8]        [1, 20, 8, 8]        --                   --
│    └─MaxPool2d (2)                     [1, 20, 8, 8]        [1, 20, 4, 4]        --                   --
│    └─ReLU (3)                          [1, 20, 4, 4]        [1, 20, 4, 4]        --                   --
├─Sequential (classifier)                [1, 20, 4, 4]        [1, 3]               --                   True
│    └─Flatten (0)                       [1, 20, 4, 4]        [1, 320]             --                   --
│    └─Linear (1)                        [1, 320]             [1, 50]              16,050               True
│    └─Linear (2)                        [1, 50]              [1, 3]               153                  True
==========================================================================================================================
Total params: 21,483
Trainable params: 16,203
Non-trainable params: 5,280
Total mult-adds (M): 0.49
==========================================================================================================================
Input size (MB): 0.00
Forward/backward pass size (MB): 0.06
Params size (MB): 0.09
Estimated Total Size (MB): 0.15
==========================================================================================================================
```

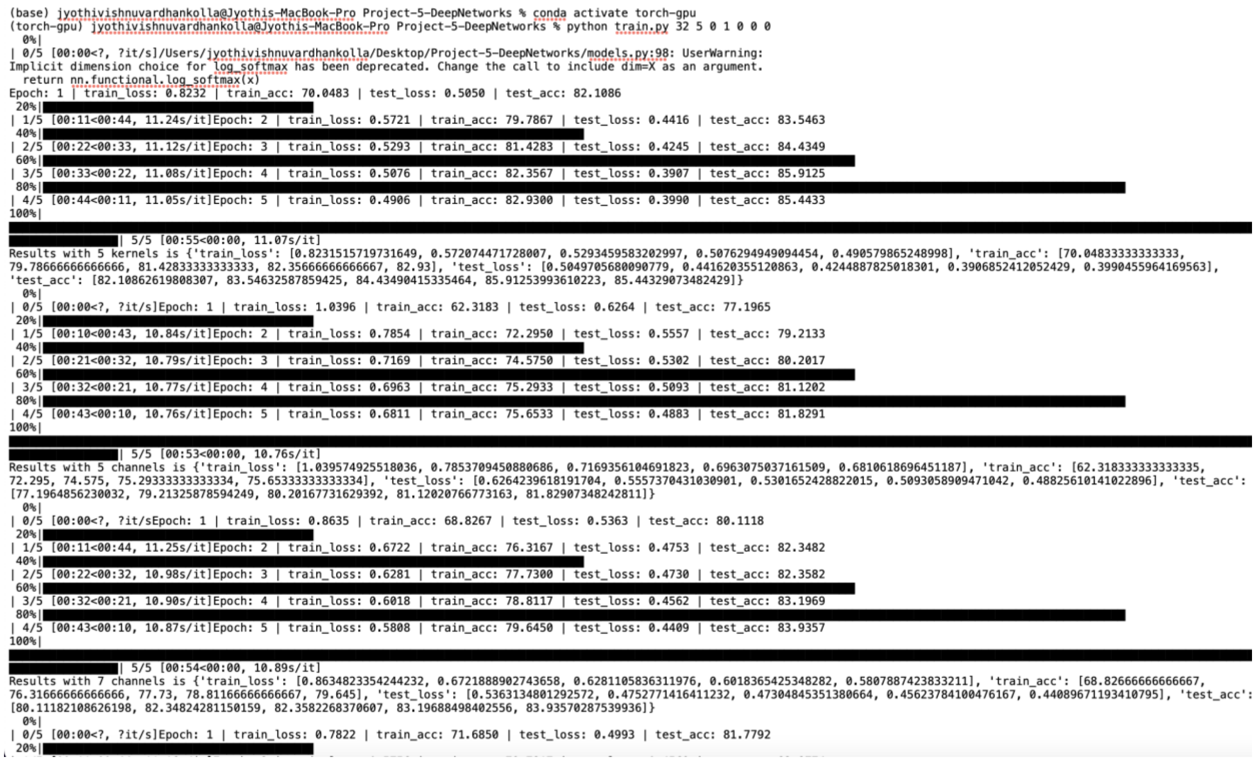


Required Image for Task 4: Design your own experiment

Link: https://drive.google.com/file/d/1-UeblVRuz9Vl2rkY5Eyg7VLR6Xv-QI9W/view?usp=share_link

We tested with the following different parameters :

1. Kernel : 5
2. channels : [5, 7, 9, 11, 13, 15, 17, 19, 21, 24, 25, 28, 31, 33, 40]
3. dropouts : [0.2, 0.25, 0.3, 0.4, 0.45, 0.5, 0.55, 0.6, 0.65, 0.7, 0.75]
4. hidden_neurons: [20, 30, 40, 50, 60, 70, 80, 90, 100, 110, 120, 130, 140, 150, 160, 170, 190, 200, 210, 220]
   The results for all the variations are persisted in the link provided above.



```
(base) jyothivishnuvardhankolla@Jyothis-MacBook-Pro Project-5-DeepNetworks % conda activate torch-gpu
(torch-gpu) jyothivishnuvardhankolla@Jyothis-MacBook-Pro Project-5-DeepNetworks % python train.py 32 5 0 1 0 0 0
  0%|
| 0/5 [00:00<?, ?it/s]/Users/jyothivishnuvardhankolla/Desktop/Project-5-DeepNetworks/models.py:98: UserWarning:
Implicit dimension choice for log_softmax has been deprecated. Change the call to include dim=X as an argument.
  return nn.functional.log_softmax(x)
Epoch: 1 | train_loss: 0.8232 | train_acc: 70.0483 | test_loss: 0.5050 | test_acc: 82.1086
 20%|████████
| 1/5 [00:11<00:44, 11.24s/it]Epoch: 2 | train_loss: 0.5721 | train_acc: 79.7867 | test_loss: 0.4416 | test_acc: 83.5463
 40%|████████████████
| 2/5 [00:22<00:33, 11.12s/it]Epoch: 3 | train_loss: 0.5293 | train_acc: 81.4283 | test_loss: 0.4245 | test_acc: 84.4349
 60%|████████████████████████
| 3/5 [00:33<00:22, 11.08s/it]Epoch: 4 | train_loss: 0.5076 | train_acc: 82.3567 | test_loss: 0.3907 | test_acc: 85.9125
 80%|████████████████████████████████
| 4/5 [00:44<00:11, 11.05s/it]Epoch: 5 | train_loss: 0.4906 | train_acc: 82.9300 | test_loss: 0.3990 | test_acc: 85.4433
100%|████████████████████████████████████████
█████| 5/5 [00:55<00:00, 11.07s/it]
Results with 5 kernels is {'train_loss': [0.8231515719731649, 0.572074471728007, 0.5293459583202997, 0.5076294949094454, 0.490579865248998], 'train_acc': [70.04833333333333,
79.78666666666666, 81.42833333333333, 82.35666666666667, 82.93], 'test_loss': [0.5049705680090779, 0.441620355120863, 0.4244887825018301, 0.3906852412052429, 0.3990455964169563],
'test_acc': [82.10862619808307, 83.54632587859425, 84.43490415335464, 85.91253993610223, 85.44329073482429]}
  0%|
| 0/5 [00:00<?, ?it/s]Epoch: 1 | train_loss: 1.0396 | train_acc: 62.3183 | test_loss: 0.6264 | test_acc: 77.1965
 20%|████████
| 1/5 [00:10<00:43, 10.84s/it]Epoch: 2 | train_loss: 0.7854 | train_acc: 72.2950 | test_loss: 0.5557 | test_acc: 79.2133
 40%|████████████████
| 2/5 [00:21<00:32, 10.79s/it]Epoch: 3 | train_loss: 0.7169 | train_acc: 74.5750 | test_loss: 0.5302 | test_acc: 80.2017
 60%|████████████████████████
| 3/5 [00:32<00:21, 10.77s/it]Epoch: 4 | train_loss: 0.6963 | train_acc: 75.2933 | test_loss: 0.5093 | test_acc: 81.1202
 80%|████████████████████████████████
| 4/5 [00:43<00:10, 10.76s/it]Epoch: 5 | train_loss: 0.6811 | train_acc: 75.6533 | test_loss: 0.4883 | test_acc: 81.8291
100%|████████████████████████████████████████
█████| 5/5 [00:53<00:00, 10.76s/it]
Results with 5 channels is {'train_loss': [1.039574925518036, 0.7853709450880686, 0.7169356104691823, 0.6963075037161509, 0.6810618696451187], 'train_acc': [62.318333333333335,
72.295, 74.575, 75.29333333333334, 75.65333333333334], 'test_loss': [0.6264239618191704, 0.5557370431030901, 0.5301652428822015, 0.5093058909471042, 0.48825610141022896], 'test_acc':
[77.1964856230032, 79.21325878594249, 80.20167731629392, 81.12020766773163, 81.82907348242811]}
  0%|
| 0/5 [00:00<?, ?it/sEpoch: 1 | train_loss: 0.8635 | train_acc: 68.8267 | test_loss: 0.5363 | test_acc: 80.1118
 20%|████████
| 1/5 [00:11<00:44, 11.25s/it]Epoch: 2 | train_loss: 0.6722 | train_acc: 76.3167 | test_loss: 0.4753 | test_acc: 82.3482
 40%|████████████████
| 2/5 [00:22<00:32, 10.98s/it]Epoch: 3 | train_loss: 0.6281 | train_acc: 77.7300 | test_loss: 0.4730 | test_acc: 82.3582
 60%|████████████████████████
| 3/5 [00:32<00:21, 10.90s/it]Epoch: 4 | train_loss: 0.6018 | train_acc: 78.8117 | test_loss: 0.4562 | test_acc: 83.1969
 80%|████████████████████████████████
| 4/5 [00:43<00:10, 10.87s/it]Epoch: 5 | train_loss: 0.5808 | train_acc: 79.6450 | test_loss: 0.4409 | test_acc: 83.9357
100%|████████████████████████████████████████
█████| 5/5 [00:54<00:00, 10.89s/it]
Results with 7 channels is {'train_loss': [0.8634823354244232, 0.6721888902743658, 0.6281105836311976, 0.6018365425348282, 0.5807887423833211], 'train_acc': [68.82666666666667,
76.31666666666666, 77.73, 78.81166666666667, 79.645], 'test_loss': [0.5363134801292572, 0.4752771416411232, 0.47304845351380664, 0.45623784100476167, 0.44089671193410795], 'test_acc':
[80.11182108626198, 82.34824281150159, 82.3582268370607, 83.19688498402556, 83.93570287539936]}
  0%|
| 0/5 [00:00<?, ?it/s]Epoch: 1 | train_loss: 0.7822 | train_acc: 71.6850 | test_loss: 0.4993 | test_acc: 81.7792
 20%|████████
```

Extensions

Extension 1: Experimented with more Greek letters than alpha, beta, and gamma including theta, pie, and phi.

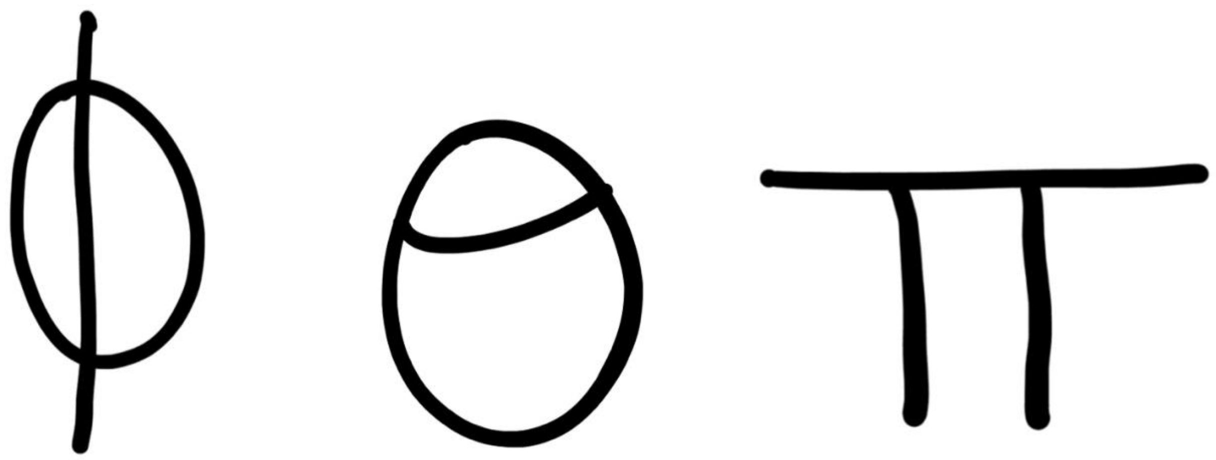
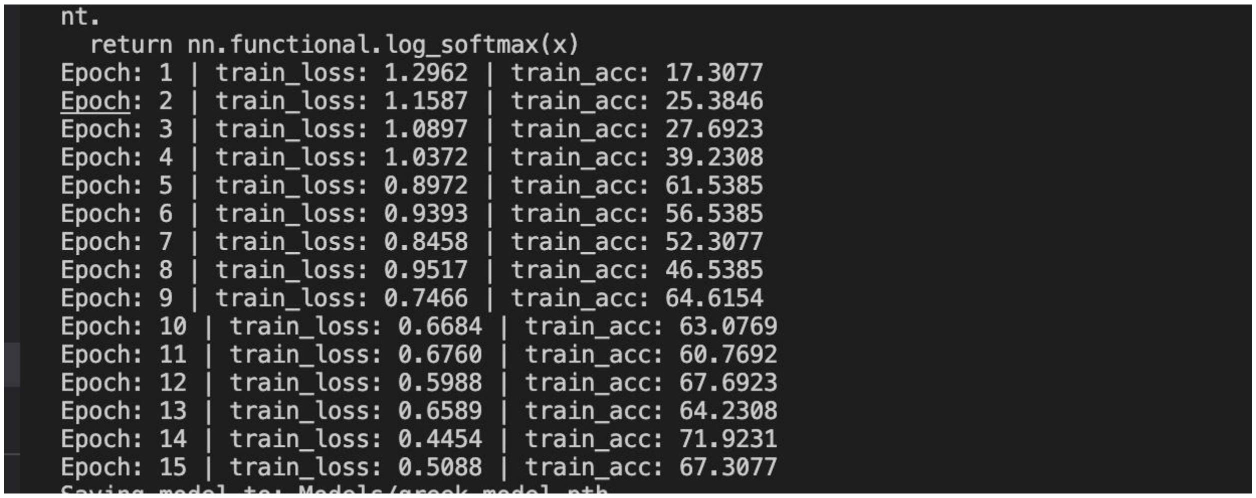


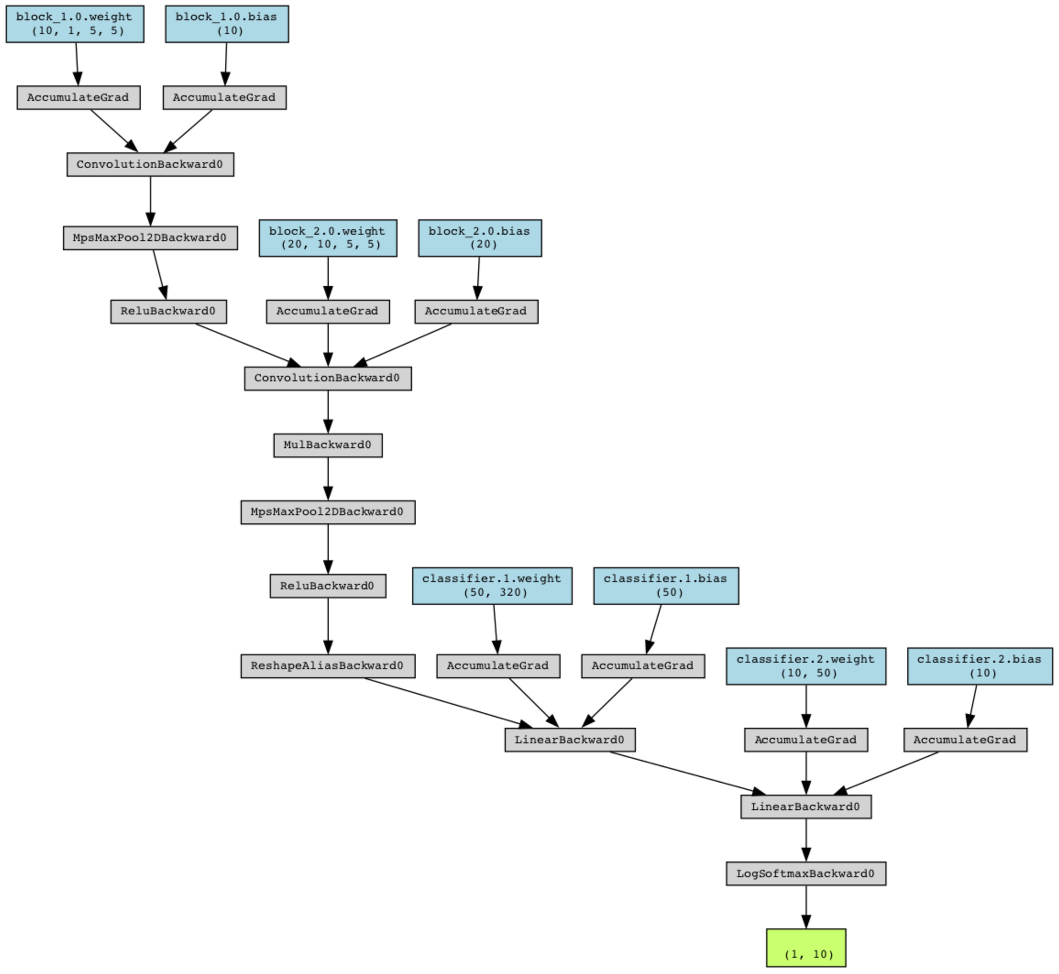
Fig Set Ext 1: Above are the three other Greek examples we experimented on. We used phi, theta and pie for the extension.

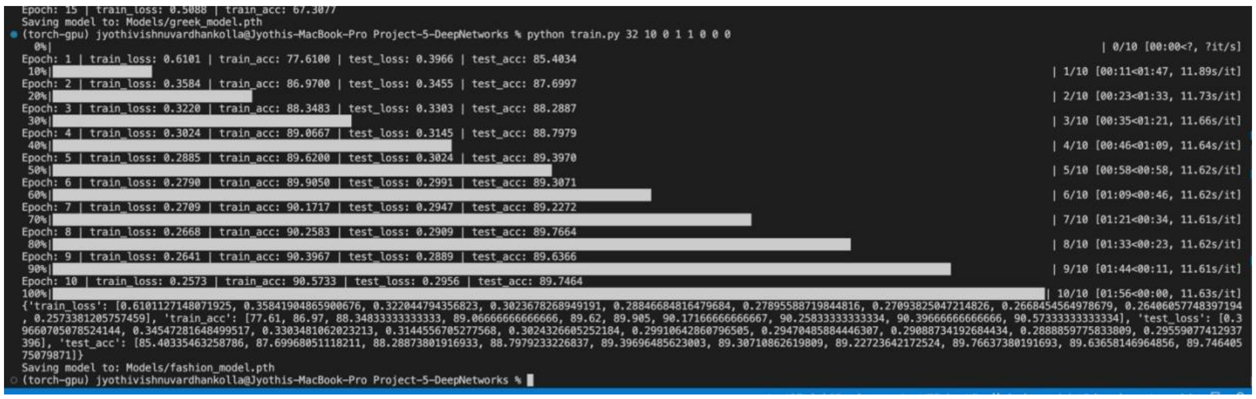


```
nt.
  return nn.functional.log_softmax(x)
Epoch: 1 | train_loss: 1.2962 | train_acc: 17.3077
Epoch: 2 | train_loss: 1.1587 | train_acc: 25.3846
Epoch: 3 | train_loss: 1.0897 | train_acc: 27.6923
Epoch: 4 | train_loss: 1.0372 | train_acc: 39.2308
Epoch: 5 | train_loss: 0.8972 | train_acc: 61.5385
Epoch: 6 | train_loss: 0.9393 | train_acc: 56.5385
Epoch: 7 | train_loss: 0.8458 | train_acc: 52.3077
Epoch: 8 | train_loss: 0.9517 | train_acc: 46.5385
Epoch: 9 | train_loss: 0.7466 | train_acc: 64.6154
Epoch: 10 | train_loss: 0.6684 | train_acc: 63.0769
Epoch: 11 | train_loss: 0.6760 | train_acc: 60.7692
Epoch: 12 | train_loss: 0.5988 | train_acc: 67.6923
Epoch: 13 | train_loss: 0.6589 | train_acc: 64.2308
Epoch: 14 | train_loss: 0.4454 | train_acc: 71.9231
Epoch: 15 | train_loss: 0.5088 | train_acc: 67.3077
Saving model to: Models/greek_model.pth
```

We obtained the above accuracies for the samples we ran inference on.

Extension 2: Explored different computer vision task with available data. In our case we built a smaller version of VGG model to classify Fashion MNIST.

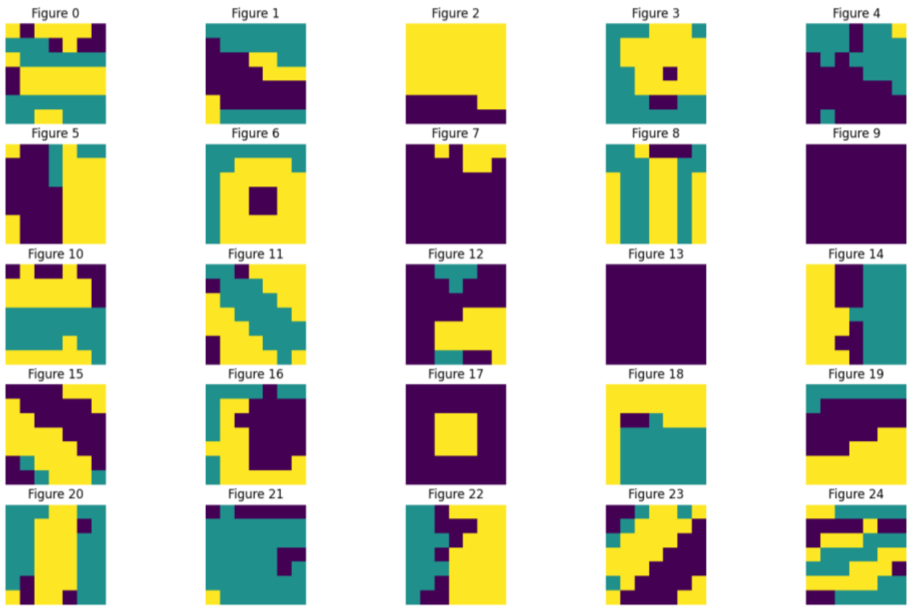


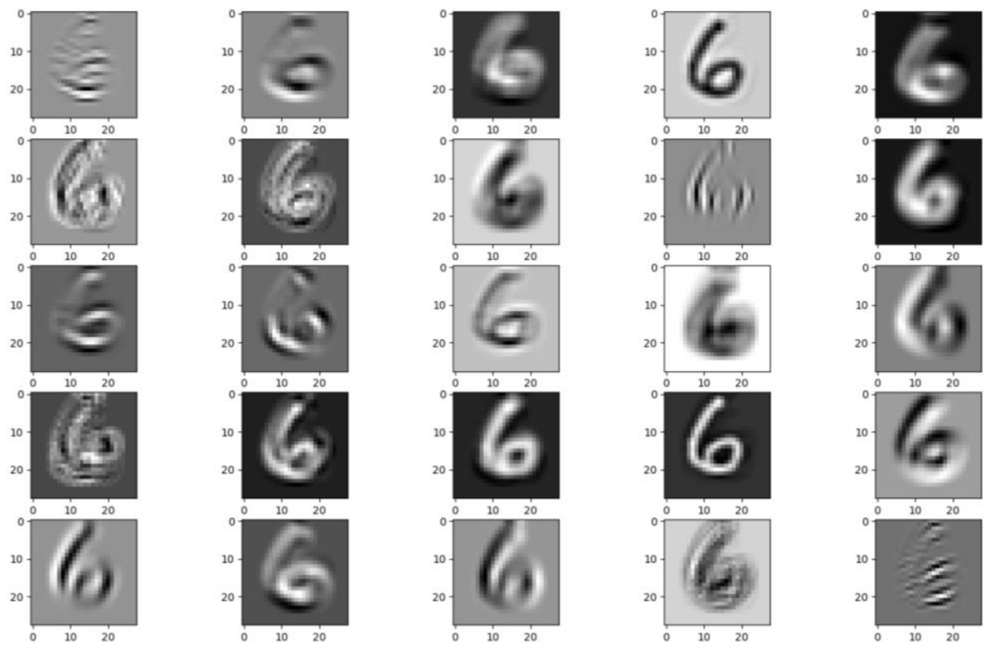


```
Epoch: 15 | train_loss: 0.5088 | train_acc: 67.3077
Saving model to: Models/greek_model.pth
(torch-gpu) jyothivishnuvardhankolla@Jyothis-MacBook-Pro Project-5-DeepNetworks % python train.py 32 10 0 1 1 0 0 0
  0%|                                                                                 | 0/10 [00:00<?, ?it/s]
Epoch: 1 | train_loss: 0.6101 | train_acc: 77.6100 | test_loss: 0.3966 | test_acc: 85.4034
 10%|                                                                              | 1/10 [00:11<01:47, 11.89s/it]
Epoch: 2 | train_loss: 0.3584 | train_acc: 86.9700 | test_loss: 0.3455 | test_acc: 87.6997
 20%|                                                                              | 2/10 [00:23<01:33, 11.73s/it]
Epoch: 3 | train_loss: 0.3220 | train_acc: 88.3483 | test_loss: 0.3303 | test_acc: 88.2887
 30%|                                                                              | 3/10 [00:35<01:21, 11.66s/it]
Epoch: 4 | train_loss: 0.3024 | train_acc: 89.0667 | test_loss: 0.3145 | test_acc: 88.7979
 40%|                                                                              | 4/10 [00:46<01:09, 11.64s/it]
Epoch: 5 | train_loss: 0.2885 | train_acc: 89.6200 | test_loss: 0.3024 | test_acc: 89.3970
 50%|                                                                              | 5/10 [00:58<00:58, 11.62s/it]
Epoch: 6 | train_loss: 0.2790 | train_acc: 89.9050 | test_loss: 0.2991 | test_acc: 89.3071
 60%|                                                                              | 6/10 [01:09<00:46, 11.62s/it]
Epoch: 7 | train_loss: 0.2709 | train_acc: 90.1717 | test_loss: 0.2947 | test_acc: 89.2272
 70%|                                                                              | 7/10 [01:21<00:34, 11.61s/it]
Epoch: 8 | train_loss: 0.2668 | train_acc: 90.2583 | test_loss: 0.2909 | test_acc: 89.7664
 80%|                                                                              | 8/10 [01:33<00:23, 11.62s/it]
Epoch: 9 | train_loss: 0.2641 | train_acc: 90.3967 | test_loss: 0.2889 | test_acc: 89.6366
 90%|                                                                              | 9/10 [01:44<00:11, 11.61s/it]
Epoch: 10 | train_loss: 0.2573 | train_acc: 90.5733 | test_loss: 0.2956 | test_acc: 89.7464
100%|                                                                             | 10/10 [01:56<00:00, 11.63s/it]
{'train_loss': [0.6101127148071925, 0.35841904865900676, 0.322044794356823, 0.3023678268949191, 0.28846684816479684, 0.27895588719844816, 0.27093825047214826, 0.2668454564978679, 0.26406057748397194
, 0.2573381205757459], 'train_acc': [77.61, 86.97, 88.34833333333333, 89.06666666666666, 89.62, 89.905, 90.17166666666667, 90.25833333333334, 90.39666666666666, 90.57333333333334], 'test_loss': [0.3
9660705078524144, 0.34547281648499517, 0.3303481062023213, 0.3144556705277568, 0.3024326605252184, 0.29910642860796505, 0.29470485884446307, 0.29088734192684434, 0.2888859775833809, 0.29559077412937
396], 'test_acc': [85.40335463258786, 87.69968051118211, 88.28873801916933, 88.7979233226837, 89.39696485623003, 89.30710862619809, 89.22723642172524, 89.76637380191693, 89.63658146964856, 89.746405
75079871]}
Saving model to: Models/fashion_model.pth
(torch-gpu) jyothivishnuvardhankolla@Jyothis-MacBook-Pro Project-5-DeepNetworks %
```

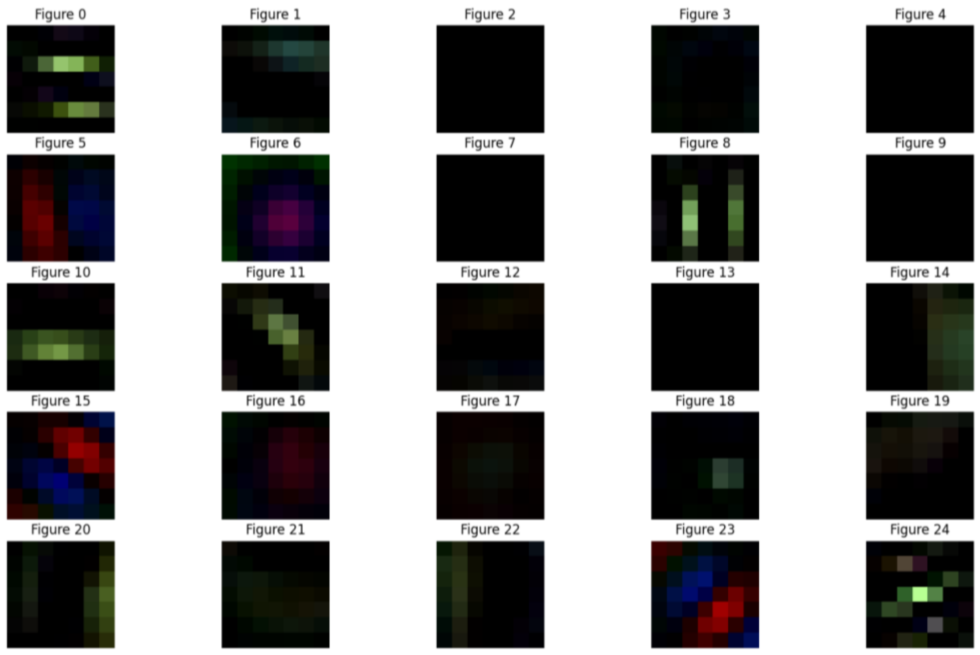
Fig Set Ext 2: *The performance as we can see is 89% accurate at the end of 10 epochs.*

Extension 3: Loaded and evaluated a pretrained model with its first couple of convolutional layers as in task 2.





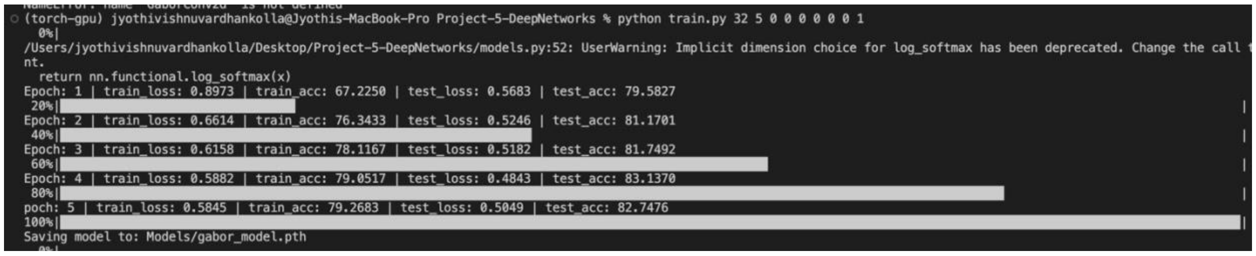
As seen from the layer results, as the layer increases the original image is almost not similar to the resultant of after that filter. Layer1 we can figure alpha more easily than Layer2.



Extension 4: Replaced the first layer of the MNIST network with a Gabor filter and retrained the rest of the network, holding the first layer constant.

```
class GaborConv2d(nn.Module):
  def __init__(self, in_channels, out_channels, kernel_size, stride=1, padding=0):
    super(GaborConv2d, self).__init__()
    self.in_channels = in_channels
    self.out_channels = out_channels
    self.kernel_size = kernel_size
    self.stride = stride
    self.padding = padding

    # Define Gabor filter bank parameters
    self.sigma = nn.Parameter(torch.randn(out_channels, in_channels))
    self.theta = nn.Parameter(torch.randn(out_channels, in_channels))
    self.lambda_ = nn.Parameter(torch.randn(out_channels, in_channels))
    self.psi = nn.Parameter(torch.randn(out_channels, in_channels))
    self.gamma = nn.Parameter(torch.randn(out_channels, in_channels))

    # Initialize filter bank weights
    self.weight = nn.Parameter(torch.randn(out_channels, in_channels, kernel_size, kernel_size))

  def forward(self, x):
    # Apply Gabor filter bank to input image
    filters = []
    for i in range(self.out_channels):
      real, imag = F.gabor_filter(x, self.kernel_size, self.sigma[i], self.theta[i],
                     self.lambda_[i], self.psi[i], self.gamma[i])
      filters.append(torch.stack([real, imag], dim=1))
    filters = torch.cat(filters, dim=0)

    # Apply convolution operation to filtered image
    output = F.conv2d(filters, self.weight, stride=self.stride, padding=self.padding)
    return output
```

The first layer was replaced with Gabor filter and the remaining layers where trained.
The performance was not as good as the previous model as we made the first layer nonlearnable which is usually the one that learns most of the feature.

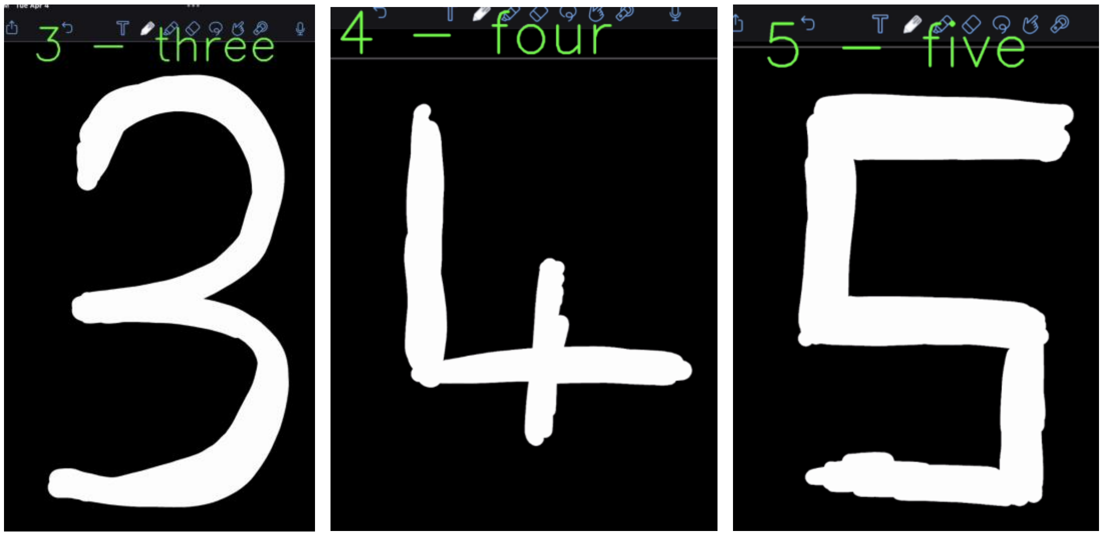


```
○ (torch-gpu) jyothivishnuvardhankolla@Jyothis-MacBook-Pro Project-5-DeepNetworks % python train.py 32 5 0 0 0 0 0 0 1
    0%|
  /Users/jyothivishnuvardhankolla/Desktop/Project-5-DeepNetworks/models.py:52: UserWarning: Implicit dimension choice for log_softmax has been deprecated. Change the call
  nt.
    return nn.functional.log_softmax(x)
  Epoch: 1 | train_loss: 0.8973 | train_acc: 67.2250 | test_loss: 0.5683 | test_acc: 79.5827
   20%|
  Epoch: 2 | train_loss: 0.6614 | train_acc: 76.3433 | test_loss: 0.5246 | test_acc: 81.1701
   40%|
  Epoch: 3 | train_loss: 0.6158 | train_acc: 78.1167 | test_loss: 0.5182 | test_acc: 81.7492
   60%|
  Epoch: 4 | train_loss: 0.5882 | train_acc: 79.0517 | test_loss: 0.4843 | test_acc: 83.1370
   80%|
  poch: 5 | train_loss: 0.5845 | train_acc: 79.2683 | test_loss: 0.5049 | test_acc: 82.7476
  100%|
  Saving model to: Models/gabor_model.pth
```

Extension 5: Explored some of the object recognition tools in OpenCV using cascade filters

Video Link:

https://drive.google.com/file/d/1wtTHRDWCpPVUB9NVRJhtN6aPufkAwoGN/view?usp=share_link



Live video digit recognition application was implemented using the trained network. The above was captured when the model was used to run inference using the alphabets that were written live using an iPad.

Reflections:

Through our exploration of PyTorch, we gained knowledge on the development, training, and evaluation of deep neural networks. By closely examining the various layers of the network and examining the outputs at each stage, we gained valuable insights into the inner workings of the network. This project has provided us with a strong foundation for learning, as well as the opportunity to explore and expand upon the concepts of recognition using deep networks.

Acknowledgements:

Professor Bruce, and his lectures were helpful to understand concepts of deep neural networks, and the PyTorch resources provided helped us with the implementation.

Piazza discussions also helped us better understand certain details necessary for this project.

Online Resource References:

1. https://stackoverflow.com/questions/51041128/pytorch-predict-single-example
2. https://stackoverflow.com/questions/42479902/how-does-the-view-method-work-in-pytorch
3. https://stackoverflow.com/questions/13317753/convert-rgb-to-grayscale-in-imagemagick-command-line
4. How to convert a tensor to numpy type: https://discuss.pytorch.org/t/converting-tensors-to-images/99482
5. https://www.geeksforgeeks.org/numpy-ndarray-flatten-function-python/#:~:text=flatten()%20function%20return%20a,array%20collapsed%20into%20one%20dimension.&text=Parameters%20%3A,(Fortran%2D%20style)%20order.
6. https://stackoverflow.com/questions/57727372/how-do-i-get-the-value-of-a-tensor-in-pytorch