Karan Shah
05/25/2023

Project 5 Report

Description of Project:

In this project, we worked on four separate tasks. In the first task, we built our own network, and then we trained and evaluated the network on Mnist data. We further delved into the task by saving/loading our model and then applying our trained model on a new set of digits. Task 2 complimented our understanding of what the first layer looked like as well as the effects of the filters on the digits. Task 3 was a deeper dive into transfer learning where we then applied our network on Greek letters that the network has never been trained on, and we evaluated that as well. Finally, in Task 4, we designed our own experiment to perform hyperparameter tuning by choosing potential dimension to evaluate our network on.

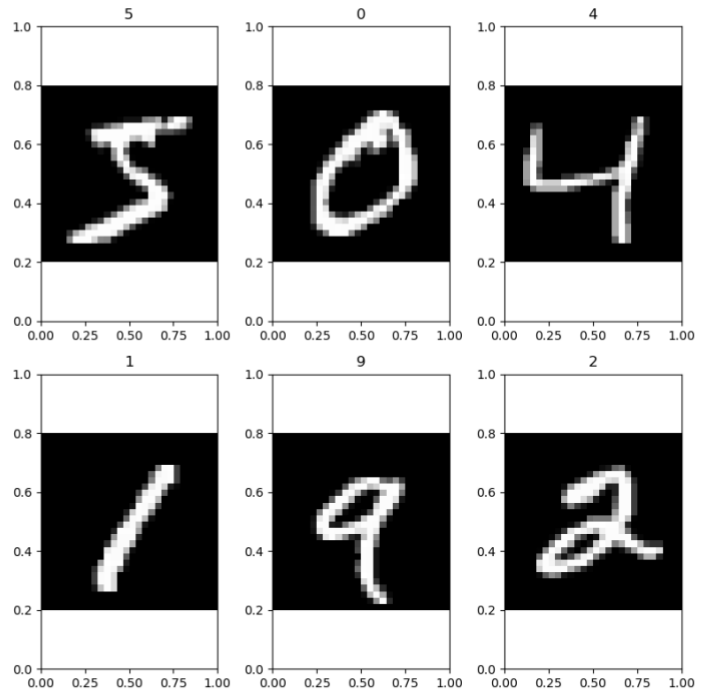
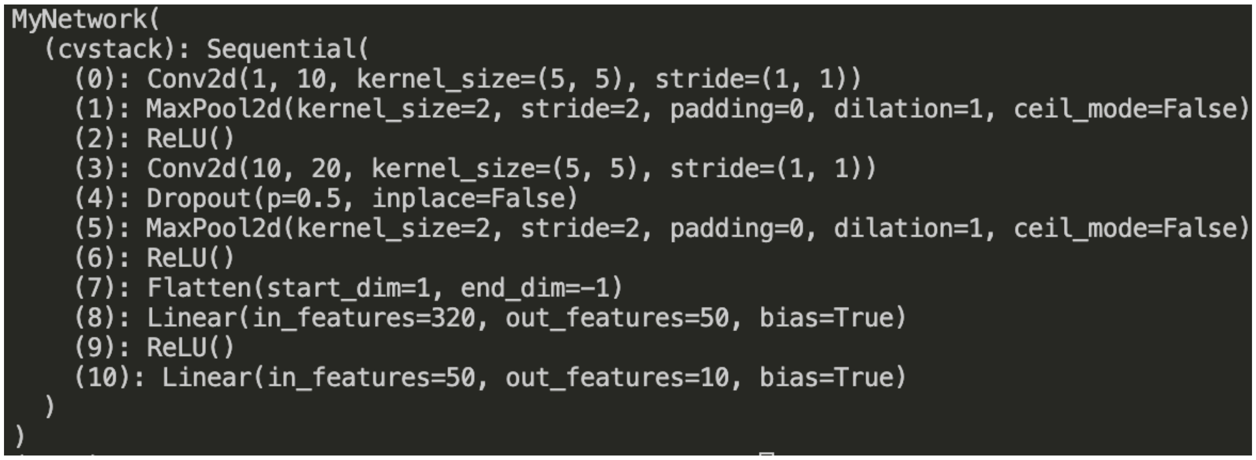


Figure 1. Mnist Examples

Figure 1 shows the first 6 examples from the Mnist training dataset.



```
MyNetwork(
  (cvstack): 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): Dropout(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)
  )
)
```

Figure 2. Network Architecture

Figure 2 shows a diagram of the network architecture. There are two convolution layers in this diagram. The input features of the fully connected linear layer (8th layer) are calculated accordingly:

- Input image = 1 x 28 x 28
- After first convolution = 10 x 24 x 24
- After first max pool = 10 x 12 x 12
- After second convolution = 20 x 8 x 8
- After second max pool = 20 x 4 x 4
- After flattening (input of linear layer) = 20 x 4 x 4 = 320

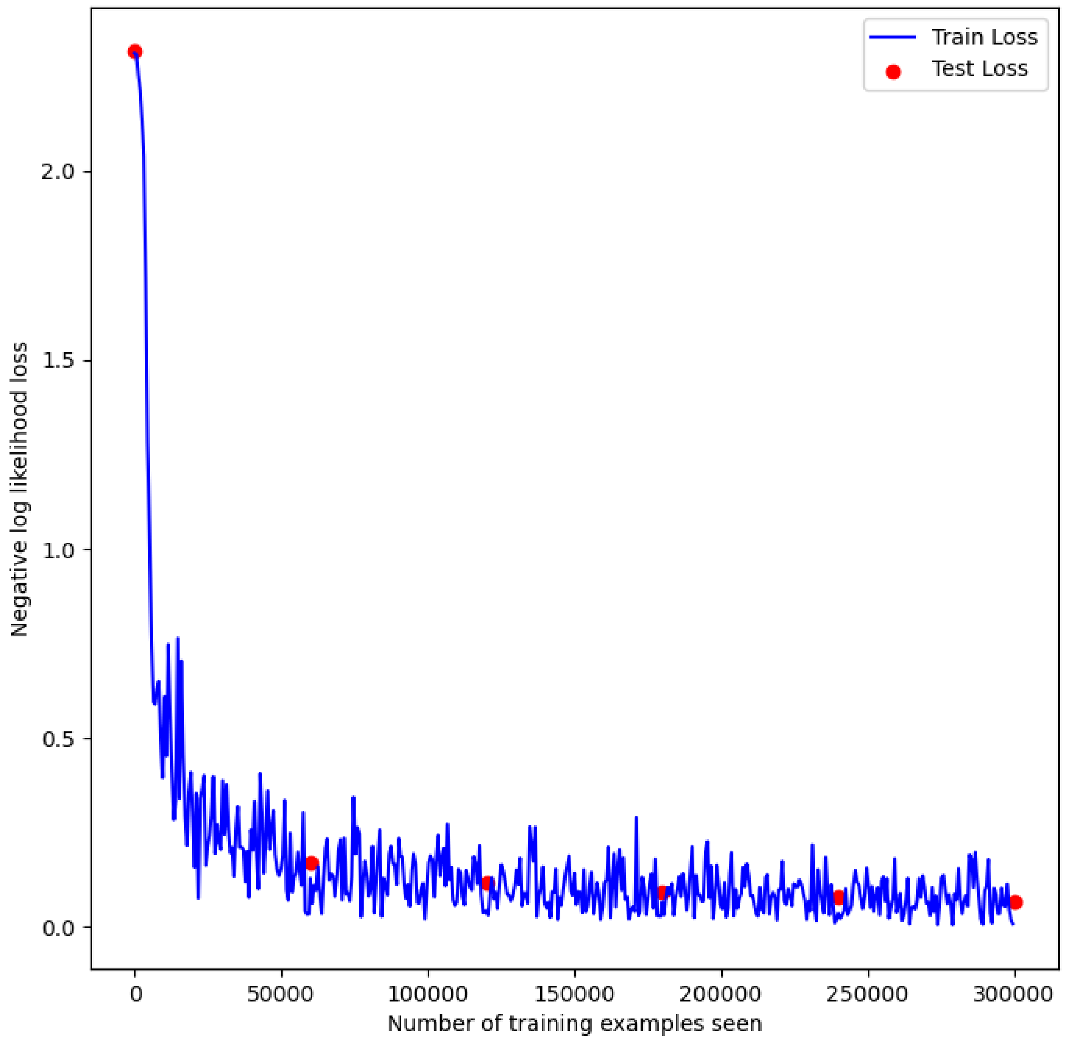


Figure 3. Training and Test Loss of Network

Figure 3 presents the training and test loss of the network from Figure 2 over 5 epochs. The first red dot signifies testing the model without training.

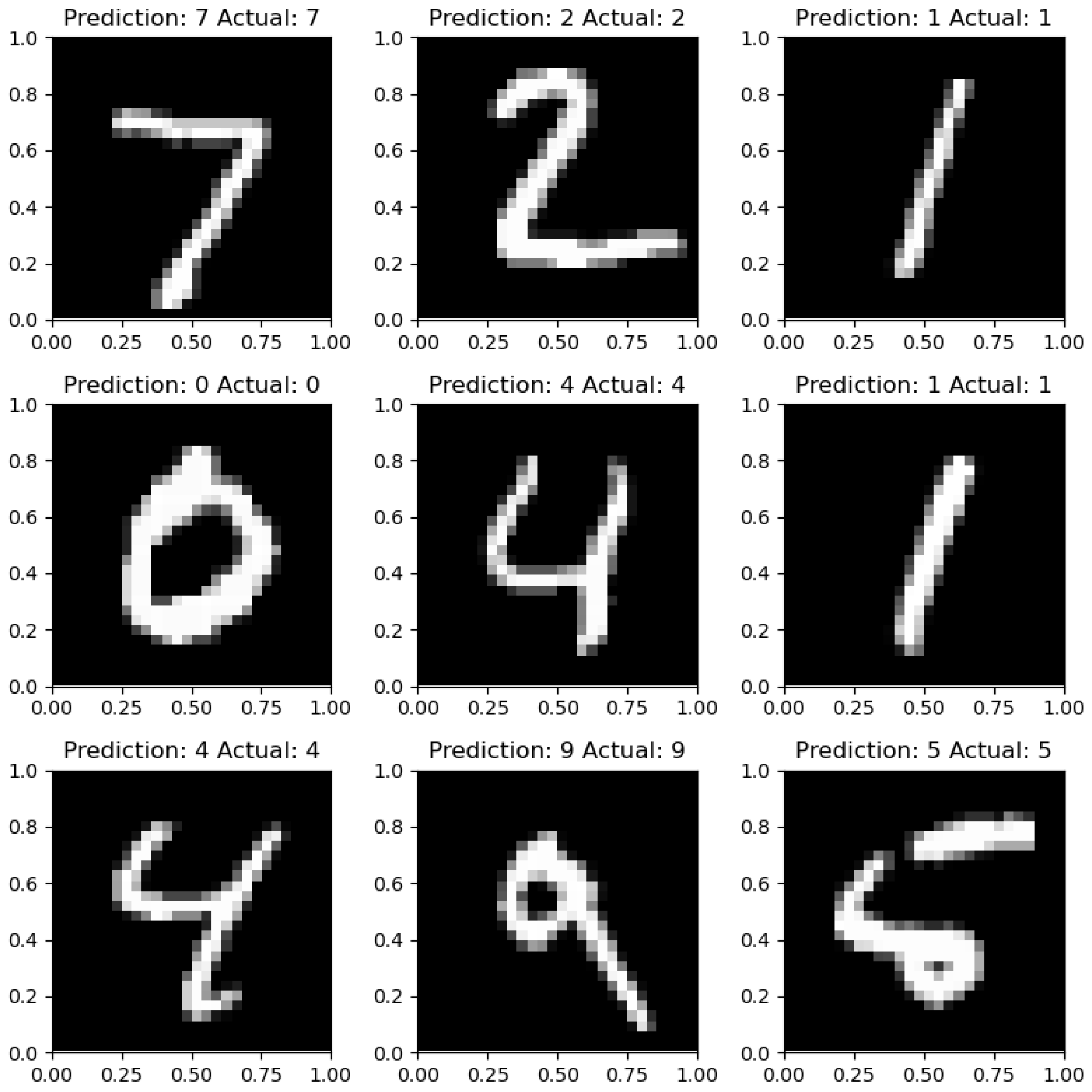
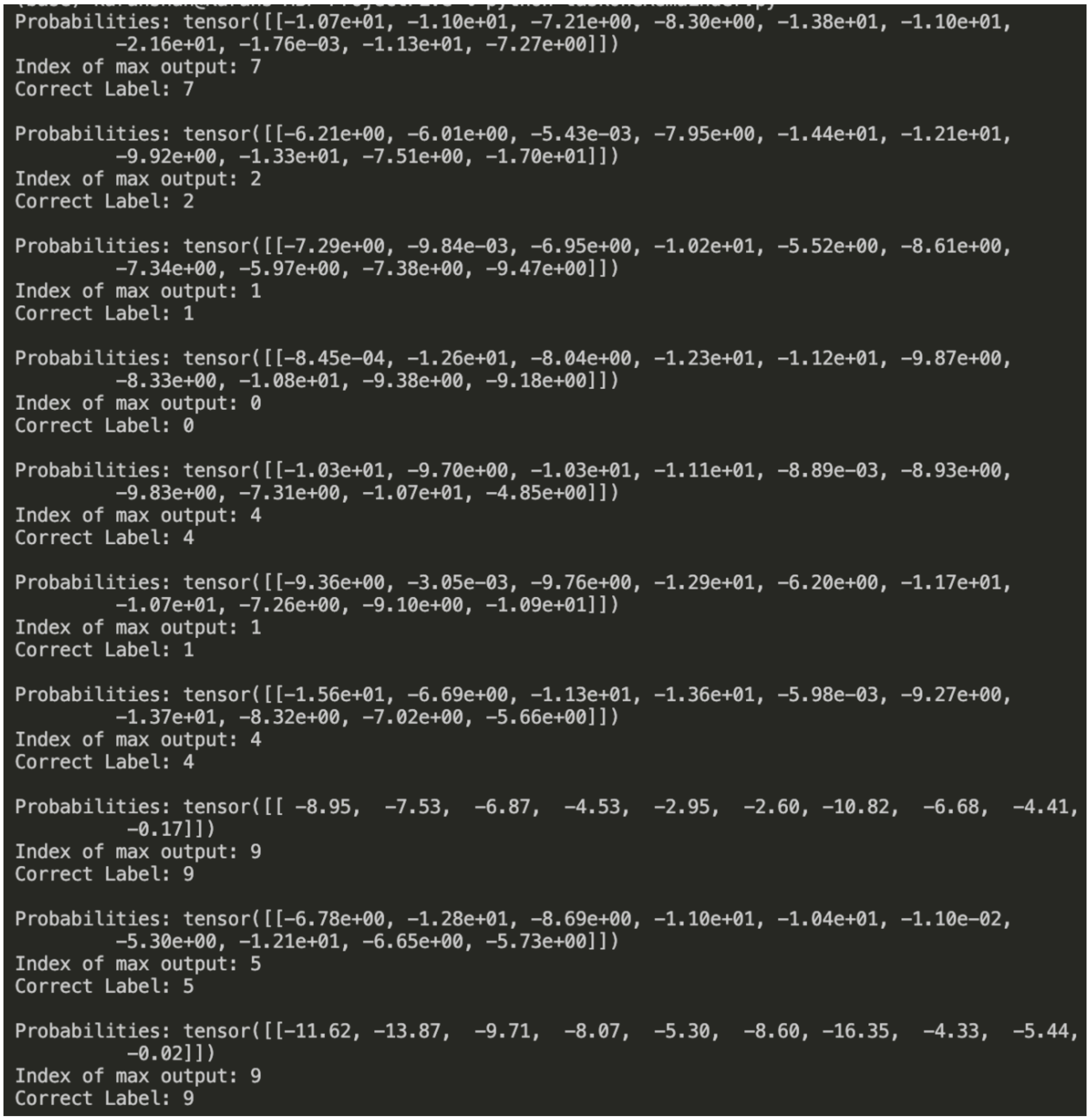


Figure 4. Test Set Examples

Figure 4 shows the result of loading the saved model and evaluating on the first 10 examples of the test set. Model evaluation has 100 percent accuracy.



```
[illegible]
Probabilities: tensor([[-1.07e+01, -1.10e+01, -7.21e+00, -8.30e+00, -1.38e+01, -1.10e+01,
         -2.16e+01, -1.76e-03, -1.13e+01, -7.27e+00]])
Index of max output: 7
Correct Label: 7

Probabilities: tensor([[-6.21e+00, -6.01e+00, -5.43e-03, -7.95e+00, -1.44e+01, -1.21e+01,
         -9.92e+00, -1.33e+01, -7.51e+00, -1.70e+01]])
Index of max output: 2
Correct Label: 2

Probabilities: tensor([[-7.29e+00, -9.84e-03, -6.95e+00, -1.02e+01, -5.52e+00, -8.61e+00,
         -7.34e+00, -5.97e+00, -7.38e+00, -9.47e+00]])
Index of max output: 1
Correct Label: 1

Probabilities: tensor([[-8.45e-04, -1.26e+01, -8.04e+00, -1.23e+01, -1.12e+01, -9.87e+00,
         -8.33e+00, -1.08e+01, -9.38e+00, -9.18e+00]])
Index of max output: 0
Correct Label: 0

Probabilities: tensor([[-1.03e+01, -9.70e+00, -1.03e+01, -1.11e+01, -8.89e-03, -8.93e+00,
         -9.83e+00, -7.31e+00, -1.07e+01, -4.85e+00]])
Index of max output: 4
Correct Label: 4

Probabilities: tensor([[-9.36e+00, -3.05e-03, -9.76e+00, -1.29e+01, -6.20e+00, -1.17e+01,
         -1.07e+01, -7.26e+00, -9.10e+00, -1.09e+01]])
Index of max output: 1
Correct Label: 1

Probabilities: tensor([[-1.56e+01, -6.69e+00, -1.13e+01, -1.36e+01, -5.98e-03, -9.27e+00,
         -1.37e+01, -8.32e+00, -7.02e+00, -5.66e+00]])
Index of max output: 4
Correct Label: 4

Probabilities: tensor([[ -8.95,  -7.53,  -6.87,  -4.53,  -2.95,  -2.60, -10.82,  -6.68,  -4.41,
          -0.17]])
Index of max output: 9
Correct Label: 9

Probabilities: tensor([[-6.78e+00, -1.28e+01, -8.69e+00, -1.10e+01, -1.04e+01, -1.10e-02,
         -5.30e+00, -1.21e+01, -6.65e+00, -5.73e+00]])
Index of max output: 5
Correct Label: 5

Probabilities: tensor([[-11.62, -13.87,  -9.71,  -8.07,  -5.30,  -8.60, -16.35,  -4.33,  -5.44,
          -0.02]])
Index of max output: 9
Correct Label: 9
```

The above screenshot shows the probabilities, index of max output, and correct label for the 10 examples in the test set.

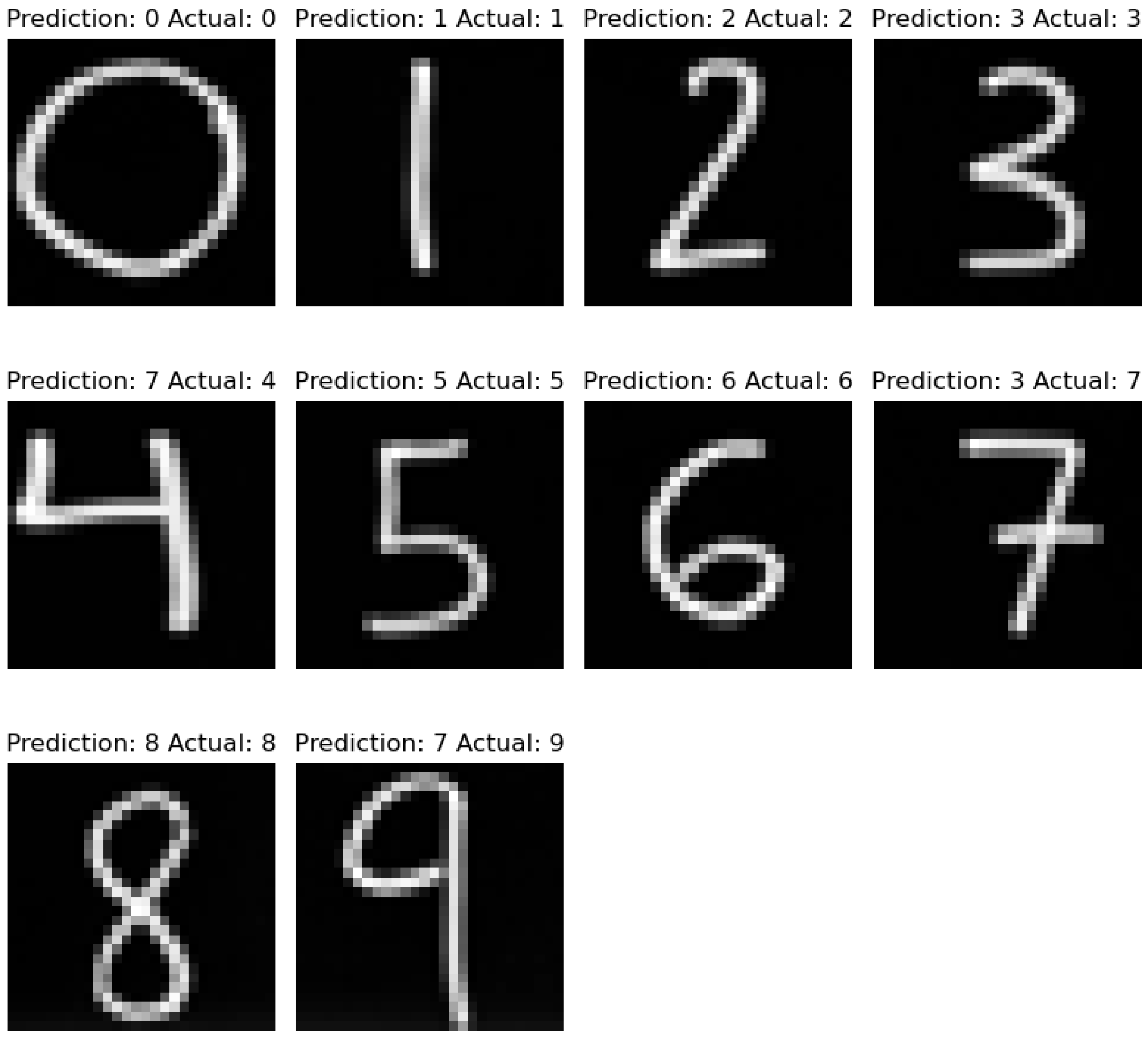


Figure 5. Network Perform on New Digits

Figure 5 presents how well the network performed on new inputs consisting of hand-written digits. In this case, the model was incorrect 3 times resulting in accuracy of 70%.

```
Shapes: torch.Size([5, 5])
-------
-
Weights: tensor([[ 0.1616,  0.2504,  0.0608,  0.2793, -0.0440],
        [ 0.0040,  0.0460,  0.4031,  0.3468, -0.1713],
        [ 0.1558,  0.1921,  0.3914,  0.1453,
0.1387],
        [-0.0516,  0.2035,  0.1210, -0.0160,  0.1311],
        [-0.1822, -0.0778, -0.0548,  0.2170, -0.0722]],
       grad_fn=)

Shapes: torch.Size([5, 5])
--------
Weights: tensor([[-0.1686, -0.1743, -0.2189, -0.0527, -0.2522],
        [ 0.1787, -0.1907,  0.1369, -0.0090, -0.1190],
        [ 0.2110,  0.1598,  0.3148,  0.1333, -0.0286],
        [ 0.1372,  0.0951,  0.2856,  0.3308,  0.2133],
        [-0.1254,  0.1687,  0.1072,  0.1549, -0.0383]],
       grad_fn=)

Shapes: torch.Size([5,
5])
--------
Weights: tensor([[-0.1894,  0.0627,  0.0370,  0.2356,  0.0025],
        [ 0.0944,  0.2383,  0.1763,  0.1535, -
0.2840],
        [ 0.1344,  0.1540,  0.2616, -0.1390, -0.1444],
        [ 0.0990,  0.3819, -0.0341, -0.2343, -0.2847],
        [ 0.3022,  0.2018,  0.1649, -0.3087, -
0.3641]],
       grad_fn=)

Shapes: torch.Size([5, 5])
--------
Weights: tensor([[-0.2152, -0.1754,  0.0837,  0.1142,  0.2230],
        [-0.1604, -0.1440,  0.1542, -0.0072,
0.1945],
        [-0.1046,  0.1370, -0.0387,  0.0135,  0.1068],
        [-0.0030, -0.0216,  0.2098,  0.1972, -0.1676],
        [-0.0634,  0.2571,  0.0101, -0.0648, -
0.2318]],
       grad_fn=)

Shapes: torch.Size([5,
5])
--------
Weights: tensor([[-0.1554,  0.0030, -0.0589, -0.1699, -0.0309],
        [-0.1792, -0.2231, -0.1491, -0.2089, -
0.1628],
        [ 0.1229, -0.0278,  0.0165, -0.2246, -0.1663],
        [-0.1092, -0.0056, -0.2101, -0.1218, -0.1573],
        [-0.0026, -0.0828, -0.0295, -0.1656, -0.1815]],
       grad_fn=)

Shapes: torch.Size([5, 5])
--------
```

```
Weights: tensor([[-0.1198, -0.0827, -0.1606,  0.1578, -
0.0008],
        [ 0.1696,  0.0076, -0.2492,  0.0725, -0.0534],
        [-0.0325, -0.1798, -0.2780, -0.0288, -0.0770],
        [ 0.2072, -0.0603,  0.1689, -0.1827, -
0.0554],
        [ 0.1623,  0.2802,  0.3042,  0.2424, -0.1407]],
       grad_fn=)

Shapes: torch.Size([5, 5])
--------
Weights: tensor([[ 2.5733e-02, -1.3359e-01, -7.4895e-02, -1.5894e-01,
2.3559e-01],
        [-1.7952e-01,  1.2993e-01,  1.1646e-01,  1.2940e-01, -5.7565e-02],
        [ 2.6500e-01,  1.5399e-01,  9.3087e-02, -1.5755e-01,  1.2126e-01],
        [ 8.4289e-05, -7.2551e-02,  2.0996e-01,  3.9947e-02, -1.2152e-
01],
        [-1.4738e-01, -2.1755e-02,  1.4468e-01, -6.2009e-02, -1.3552e-01]],
       grad_fn=)

Shapes: torch.Size([5, 5])
--------
Weights: tensor([[-0.1576, -0.1234,  0.2101,  0.3014, -0.1431],
        [-0.0357,  0.1694,  0.1811, -0.0005,  0.0720],
        [ 0.0720,  0.1135,  0.0677,  0.0213,
0.0183],
        [-0.0246, -0.0206,  0.2492,  0.3203,  0.0786],
        [-0.0441,  0.0130,  0.0424,  0.1595,  0.1933]],

grad_fn=)

Shapes: torch.Size([5, 5])
-------
-
Weights: tensor([[-0.1965, -0.1344,  0.0947,  0.0482, -0.0211],
        [-0.1369, -0.2640, -0.2924, -0.1179,  0.0099],
        [ 0.2029,  0.0312, -0.2462, -0.1953, -
0.1195],
        [ 0.3188,  0.1424, -0.0247,  0.0408, -0.1230],
        [-0.0481,  0.2936,  0.0267,  0.0235, -0.1875]],
       grad_fn=)

Shapes: torch.Size([5, 5])
-------
-
Weights: tensor([[ 0.1504,  0.2640,  0.2989,  0.1411,  0.0748],
        [-0.0117,  0.3948,  0.3794,  0.4278,  0.2963],
        [ 0.1562, -0.0037, -0.2191, -0.2171, -
0.1948],
        [-0.0052, -0.3355, -0.3139, -0.2593, -0.1304],
        [-0.3224, -0.3312, -0.0685, -0.0671,  0.0469]],
       grad_fn=)
```

The weights and shapes of the ten filters of the first convolution layer are presented above.

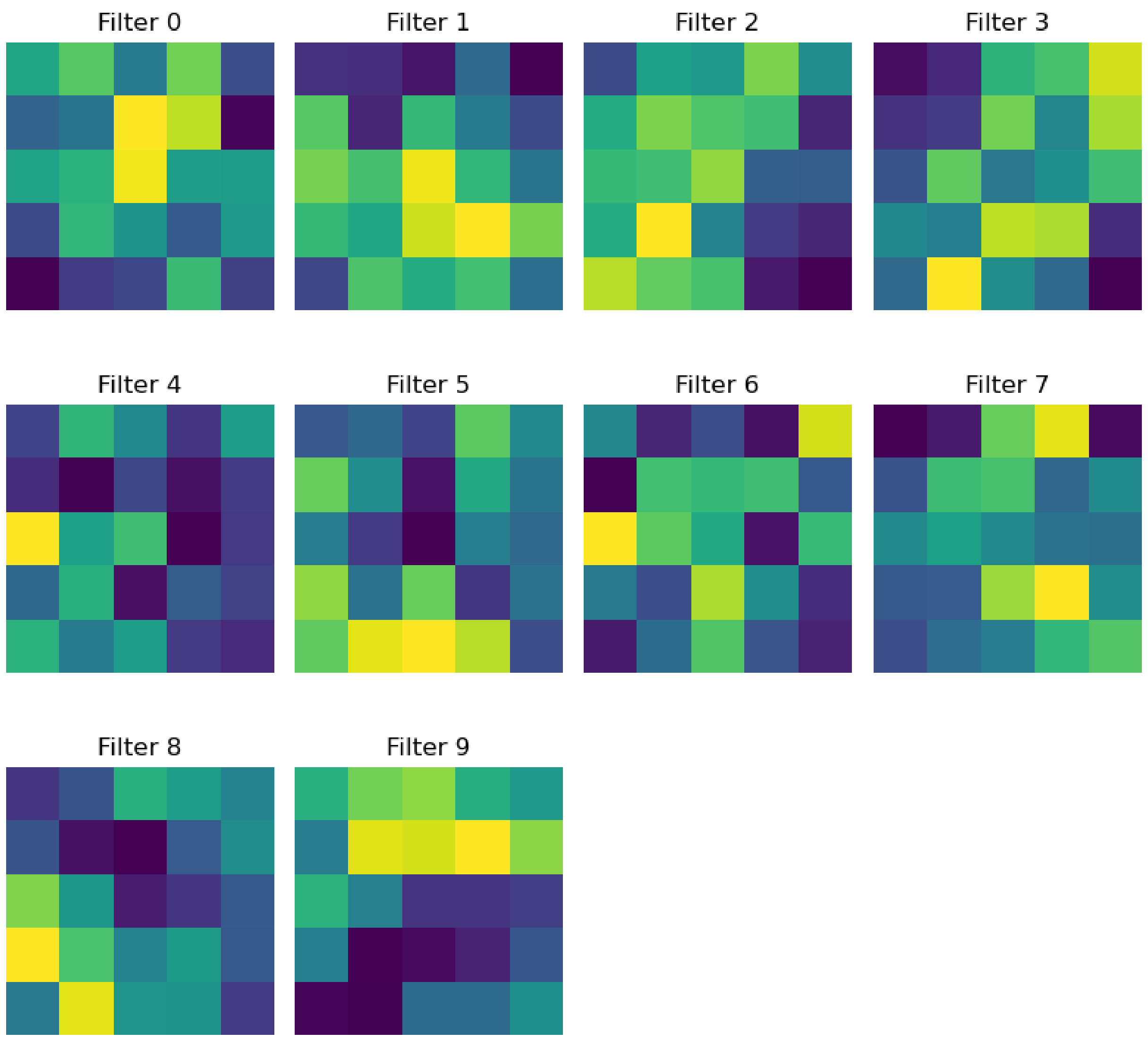


Figure 6. Visual representation of first convolution layer

Figure 6 presents the 10 5 x 5 filters of the first convolution layer of the network presented in Figure 2.

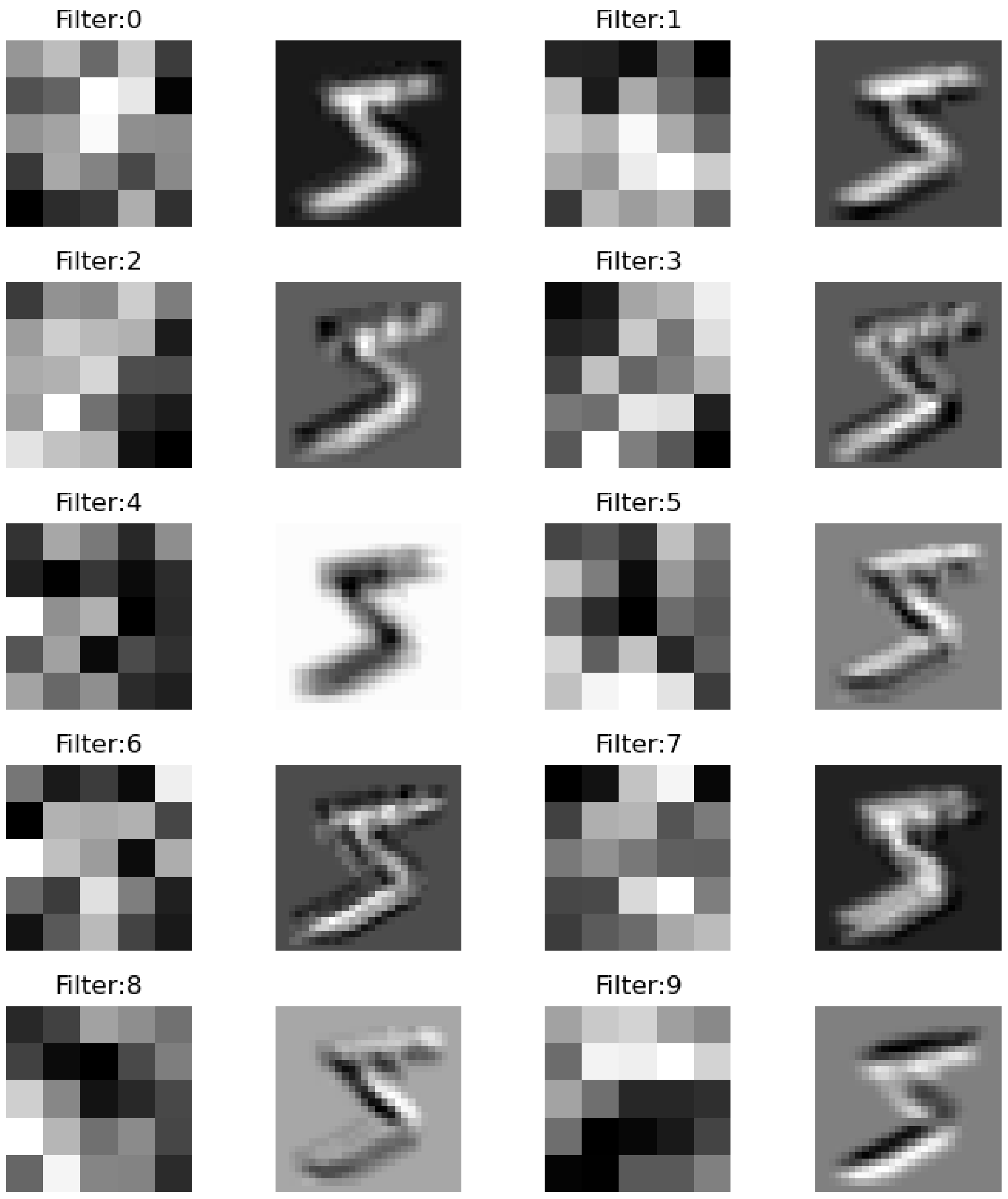
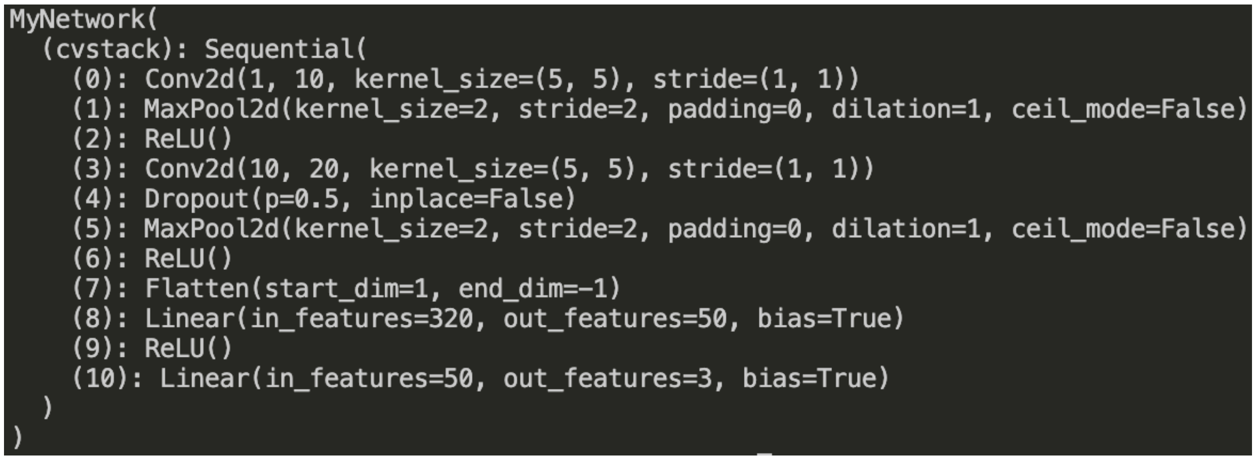


Figure 7. Filter effects

Figure 7 shows the effect of the 10 filters in the first convolution layer on the first training image. The results make sense given the filters. For example, if you look at filter 9, you can see the direct effect of the horizontal Sobel filter clearly defined in the image. For the number 5, you can see the change in gradient effect the edges. Filter 8 has a diagonal filter targeting the diagonal edge of the number 5.



```
MyNetwork(
  (cvstack): 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): Dropout(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)
  )
)
```

Figure 8. Updated network

Figure 8 presents the network from Figure 2 but with the last layer updated to be a new fully connected layer with 50 inputs and 3 outputs. The 3 outputs are for the Greek letters' alpha, beta, and gamma.

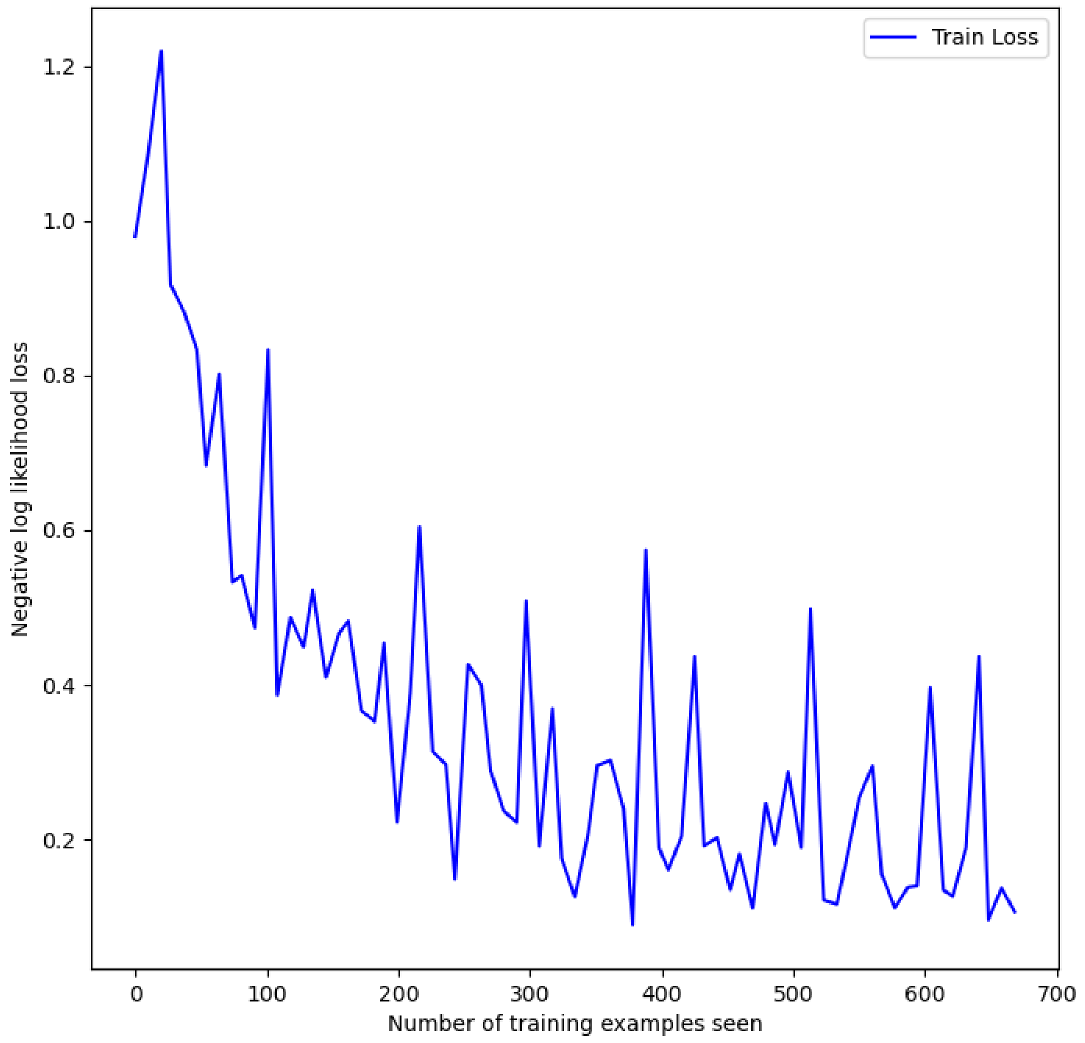


Figure 9. Train loss on Greek letters

Figure 9 shows the result of training loss over 25 epochs on identifying the 27 Greek letters. After 25 epochs, all the letters have been identified correct with the loss being close to 0. However, you can see the loss be even lower than after 25 epochs at around 14 epochs. I was able to see this by printing out the loss values manually and at 14 epochs, it was the closest to 0.

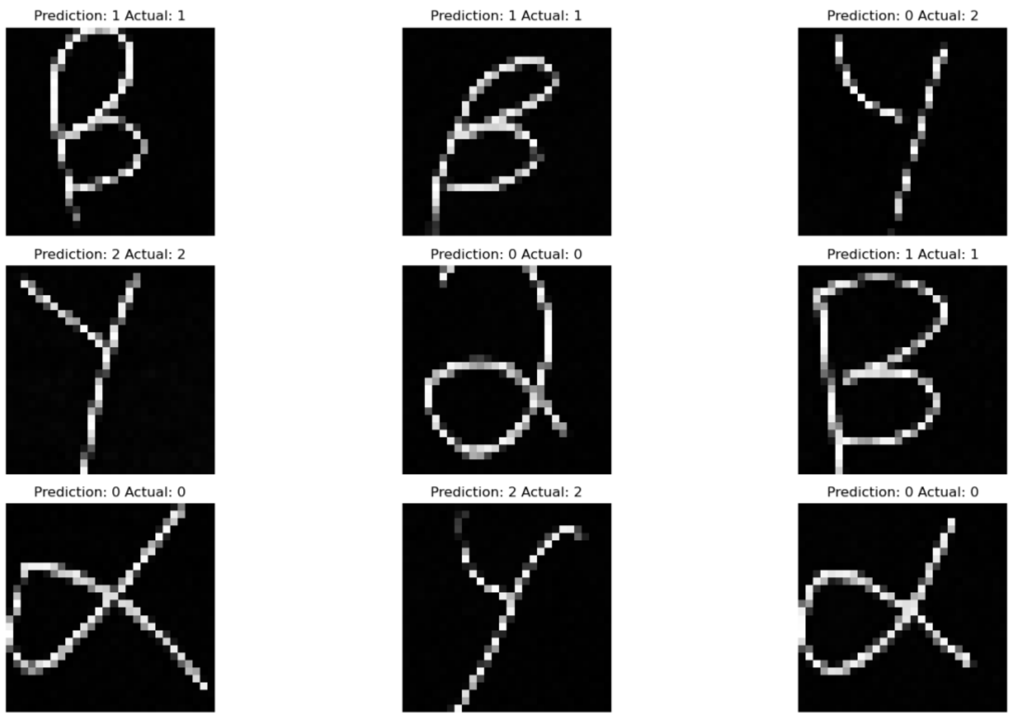


Figure 10. Personal alpha, gamma, and beta predictions

Figure 10 shows the prediction of the model on 9 handwritten Greek letters cropped to 128 x 128. Result shows 88% accuracy for this set of 9 letters. This accuracy is rather promising.

Task 4: Experimentation

The dimensions that I decided to evaluate with the deep network from Mnist are the size of the convolution filters, number of convolution filters in a layer, and the number of epochs. The design of the network for the filter size, filter number, and number of epochs is [2, 3, 5, 7], [5, 10, 15, 20], and [2, 5, 7], respectively. This leads to 4 * 4 * 3 = 48 variations (roughly 50). Before starting my evaluation, my hypothesis is the network will train better with smaller filter sizes, a greater number of filters, and larger number of epochs. I believe smaller filter sizes will train better particularly because smaller filter sizes can help obtain more granular/defined feature maps.

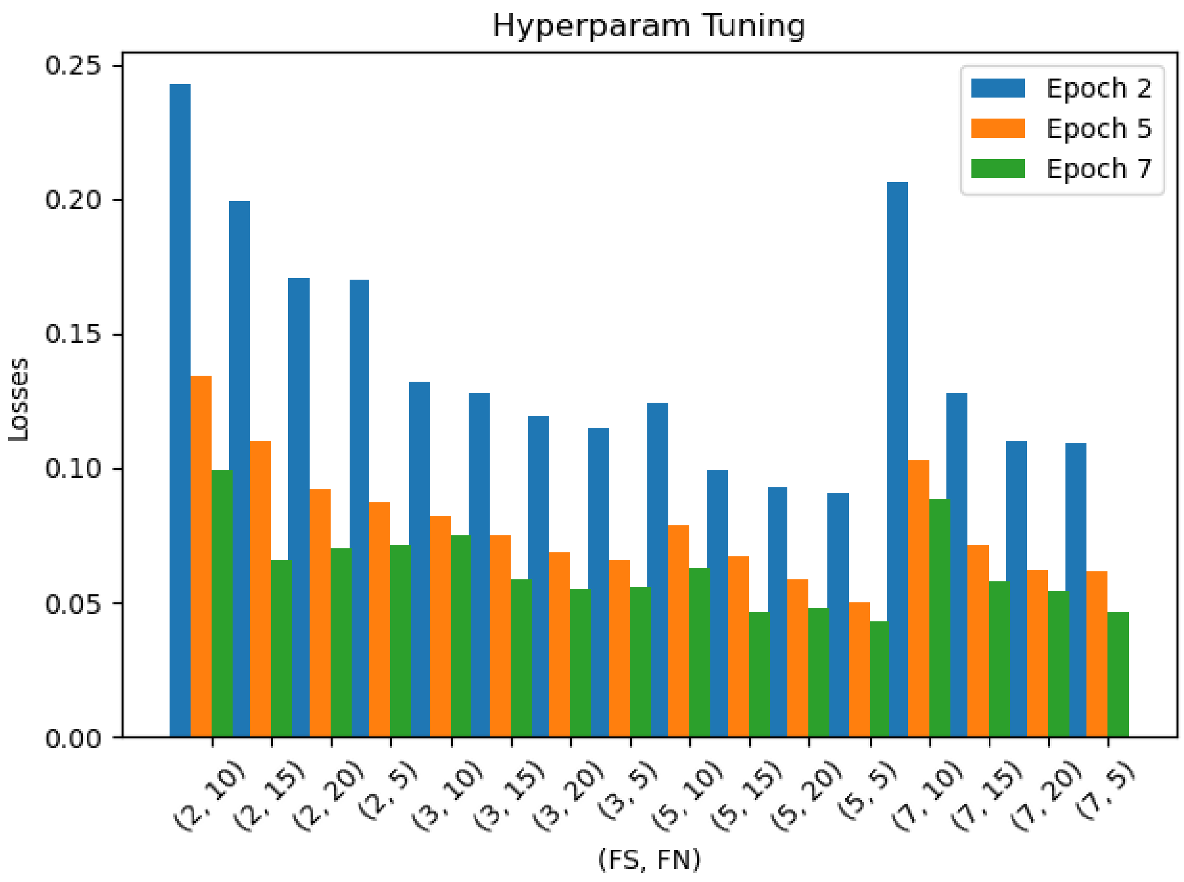


Figure 11. Hyperparameter Tuning

Figure 11 shows the results of hyperparameter tuning of the deep network. Specifically, the hyperparameters that are being tuned are the filter size (FS), filter number (FN), and number of epochs. After observing Figure 11, it is the case that increasing the number of epochs will ultimately train the network better. In addition, if you keep the filter size constant (FS), and observe the change in channels (FN), you will notice roughly an increase in channels (FN), 15-20, helps decrease the loss. On the other hand, the “sweet spot” for the filter size is 5. More experimentation could perhaps help better deduce the effect of the filter size on the network.

Extensions

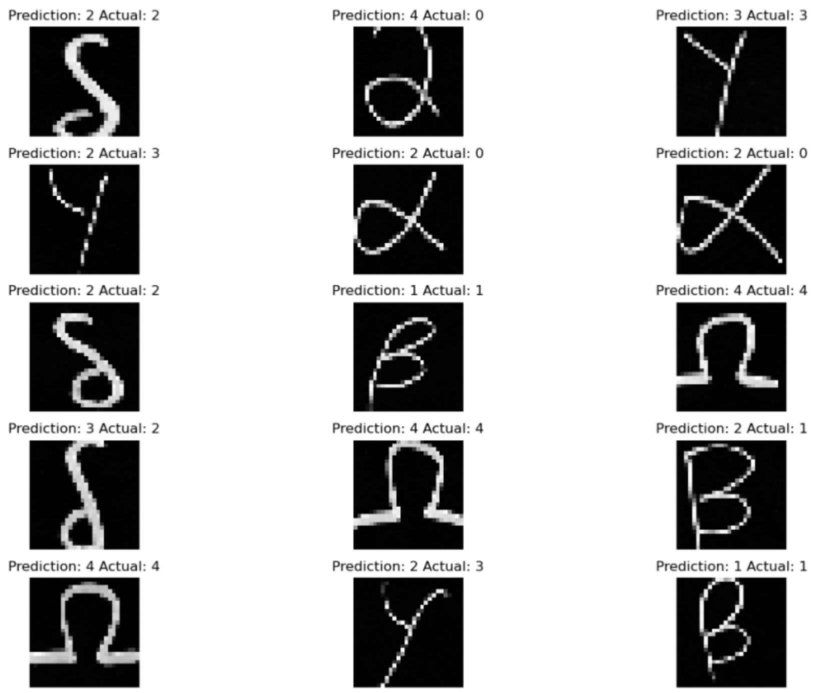


Figure 12. Additional Greek Letters

I decided to further extend experimenting with Greek letters. So, I created an additional set of omega and delta letters. I created an additional 9 omega and delta training letters while providing 3 additional omega and delta test letters. In addition, when I created the omega and delta letters, I tried to draw them in a manner where their orientation was changed to a certain degree. Furthermore, unlike before, I only trained to 14 epochs based on the loss values from Figure 9. Figure 12 shows the result for 15 examples. Out of 3 deltas, the model was able to get 2 correct, so an accuracy of 2/3 = 0.67%. On the other hand, the model was able to correctly classify all 3 omegas. Again, these results were surprising. Overall accuracy, 8/15 = 0.53%. This is rather poor performance, and in this set of 15, the model is performing worse for the handwritten Greek letters (alpha, beta, gamma).

Reflections

In this project, I learned how to utilize Pytorch to build my own network. Specifically, I am much more familiar with convolution layers, dropout layers, fully connected layers, etc. and how to manipulate them to construct my network. In addition, I learned how to train and evaluate my models on provided data as well as my own data. The experimentation showed me how to apply my models in different contexts especially since transfer learning was new and impressive for me. I learned how to use ImageMagick to resize my images, and overall, I learned how it's a command line tool that can be used for multipurpose image preprocessing tasks.

Acknowledgements

The MNIST digit recognition tutorial: https://nextjournal.com/gkoehler/pytorch-mnist