Project 5 : Recognition using Deep Networks

Created by Ravina Lad, last modified on Apr 18, 2022

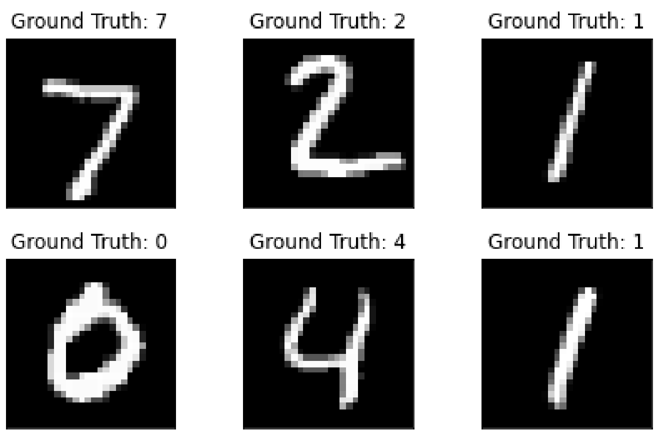
Summary

For this project I learnt about how to build, train, analyze and modify a deep network for a recognition task. MNIST digit recognition data set was used to build and train the network. The network model was successfully implemented in Python. I was also able to examine my own network using the constructed model. An experiment was undertaken to evaluate the effect of 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 display 6 images.



B) Make your network code repeatable

Made code repeatable by setting the random seed for the torch package, torch.manual_seed(42), at the start of your main function.

C) Build a network model

I then built a network which I will train on the MNIST training set. The network is structured as follows:

Inputs: 28x28 1 channel images

Convolutional Layer 1: 5x5 kernel with no image padding with 10 output channels → produces 10x24x24 output

Max Pool Layer: 2x2 kernel → produces 10x12x12 output

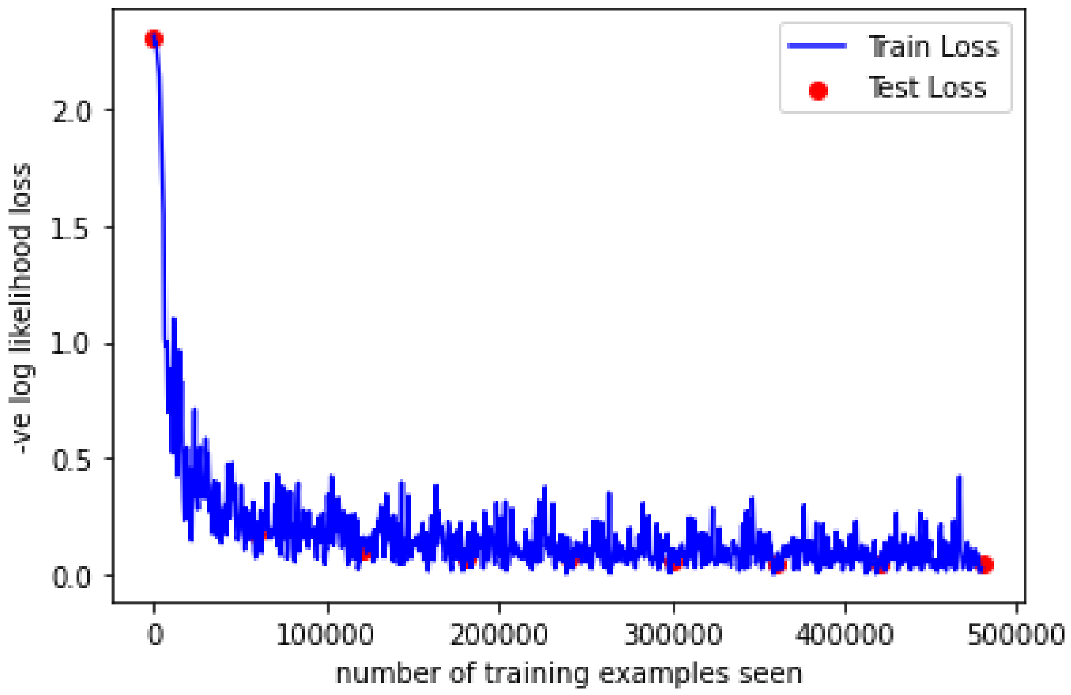
Convolutional Layer 2: 5x5 kernel with no image padding with 20 output channels → produces 20x8x8 output

Max Pool Layer: 2x2 kernel → produces 20x4x4 output

Fully Connected Layer 1: Takes 20x4x4 (320) inputs → produces 50 outputs

Fully Connected Layer 2: Takes 50 inputs → produces 10 outputs

D) Train the model

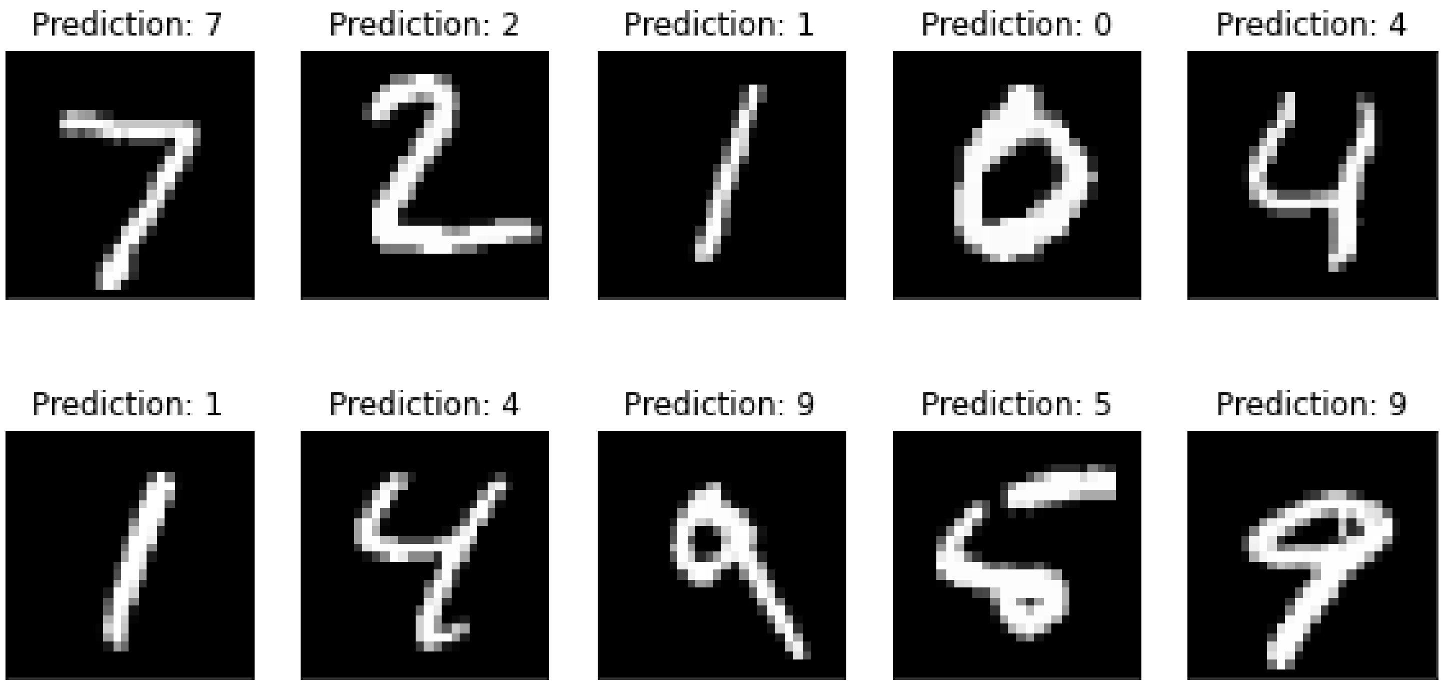


The network was trained for 5 epochs with a batch size of 64 for the training set and 1000 for the test set. Our final model accuracy on the test set was 98% and an average loss of 0.0785.

E) Save the network to a file

Saved the network as model.pth and optimizer.pth

F) Read the network and run it on the test set

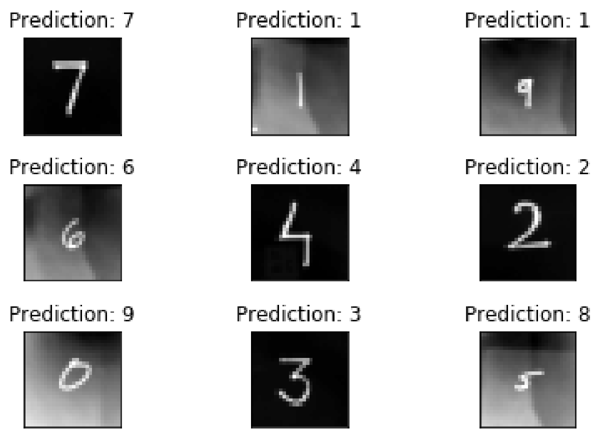


G) Test the network on new inputs

Evaluated the trained network on our hand-written digit images.

Steps Followed :

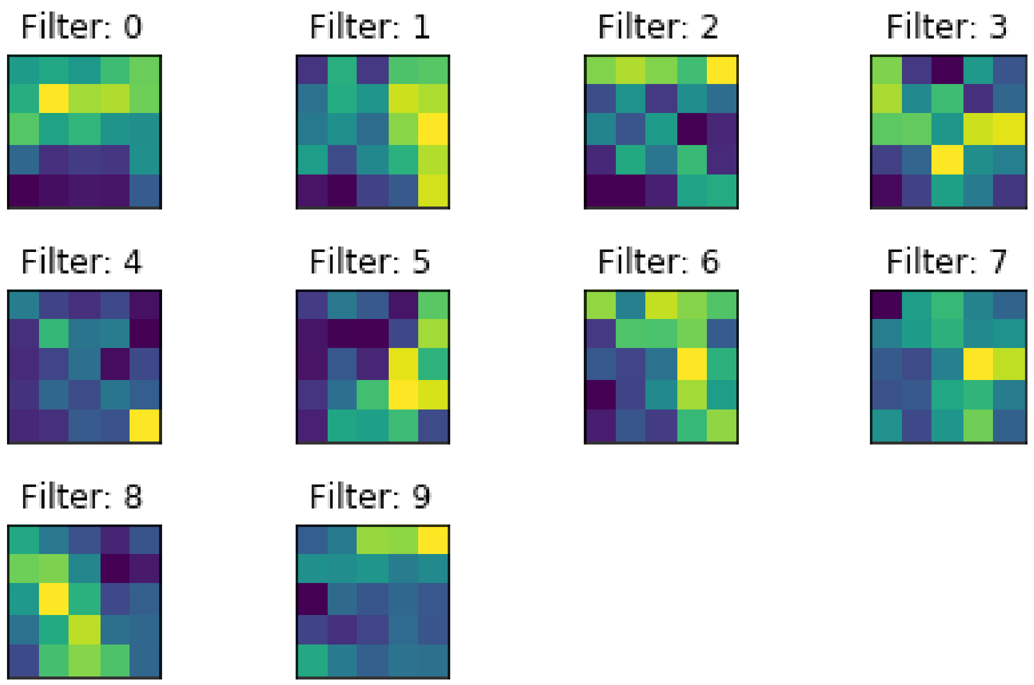
1. The images were scaled down to 28 x 28 to fit with the input size of the network and then the intensities of the images were inverted.
2. Converted to gray scale.
3. Also inverted the images from black hand written digits to white hand written digits.



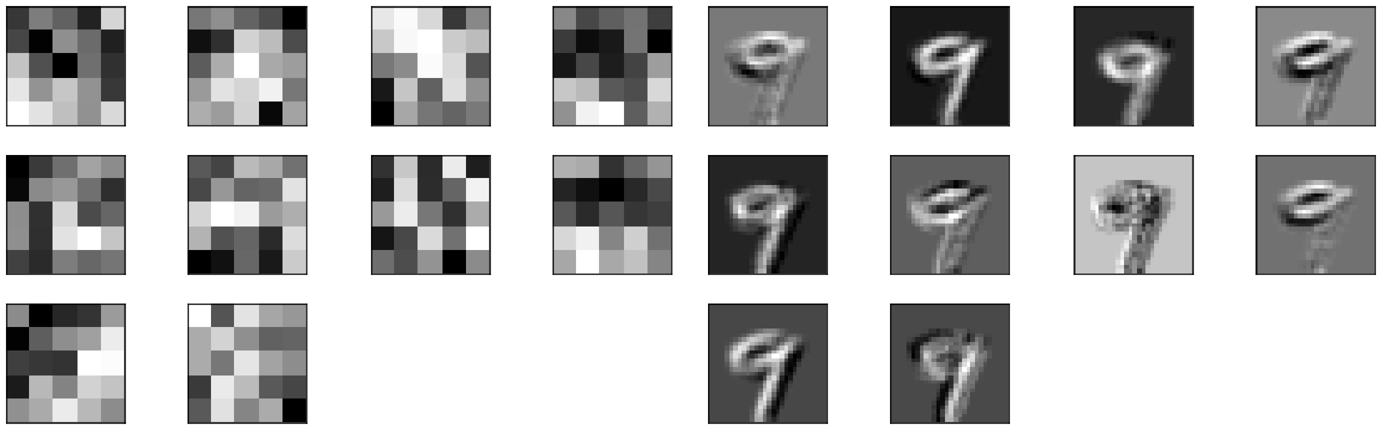
Task 2: Examine your network

A) Analyze the first layer

Extracted the learned weights from the first layer of the network and displayed them.



B) Show the effect of the filters



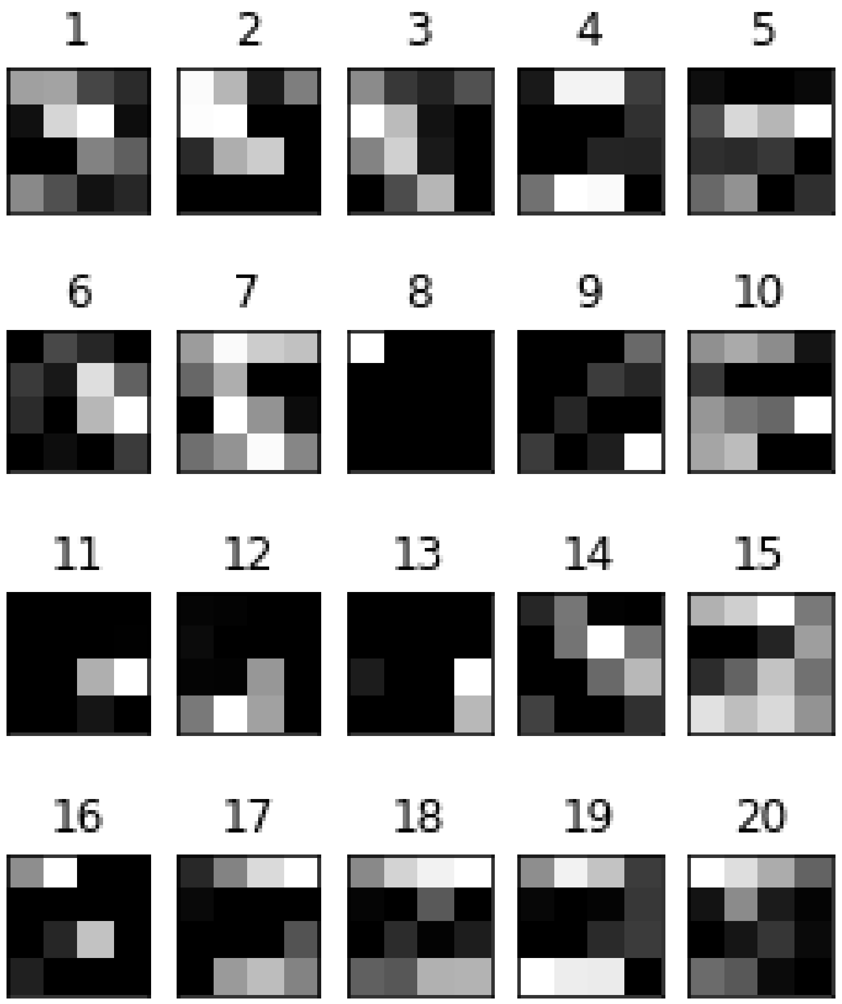
C) Build a truncated model

Created a sub-model from the original network.

Steps Followed :

1.The submodel had only the first two convolution layers of the original model. We then classified a test example on this truncated model and analyzed the results.

The 20 output channels of size 4x4 each are shown below.



Task 3: Create a digit embedding space

A) Create a greek symbol data set

1. Downloaded the greek alphabet dataset
2. Generated a data loader using Pytorch's ImageFolder method
3. Generated a dataset with transforms like resize, grayscale, and invert.

B) Create a truncated model

1. Loaded the model created in task 1 .
2. generated a sub-model that terminates at the dense layer, and removed the last softmax layer.

Model Summary:

```
Layer (type:depth-idx)          Output Shape          Param #
==========================================================================================
├─Conv2d: 1-1                  [-1, 10, 24, 24]        260
├─Conv2d: 1-2                  [-1, 20, 8, 8]          5,020
├─Dropout2d: 1-3               [-1, 20, 8, 8]           --
├─Linear: 1-4                  [-1, 50]                16,050
```

C) Project the greek symbols into the embedding space

1. Applied the truncated network to the greek symbols to get a set of 27 50 element vectors. The results areas below :

- **SSD Between an Alpha Output and Everything Else**
-
```
tensor([[ 0.0000,  0.0000],
        [ 0.6539,  0.0000],
        [ 1.2356,  0.0000],
        [ 1.7802,  0.0000],
        [ 2.0005,  0.0000],
        [ 2.3747,  1.0000],
        [ 2.4242,  1.0000],
        [ 2.4663,  0.0000],
        [ 2.4766,  0.0000],
        [ 2.7706,  1.0000],
        [ 3.8343,  0.0000],
        [ 3.8917,  0.0000],
        [ 3.9279,  1.0000],
        [ 3.9358,  1.0000],
        [ 3.9756,  1.0000],
        [ 4.0335,  1.0000],
        [ 4.4910,  1.0000],
        [ 4.6236,  1.0000],
        [ 6.6639,  2.0000],
        [ 7.9156,  2.0000],
        [ 8.5211,  2.0000],
        [ 8.8232,  2.0000],
        [ 8.9148,  2.0000],
        [ 9.1582,  2.0000],
        [10.1931,  2.0000],
        [10.3499,  2.0000],
        [10.4383,  2.0000]])
```

SSD Between a Beta Output and Everything Else

```
tensor([[0.0000, 1.0000],
        [0.5550, 1.0000],
        [1.0720, 1.0000],
        [1.1593, 1.0000],
        [1.4891, 1.0000],
        [1.6539, 1.0000],
        [2.0289, 1.0000],
        [2.0933, 1.0000],
        [2.3008, 0.0000],
        [2.3747, 0.0000],
        [2.3957, 1.0000],
        [2.5130, 0.0000],
        [2.7092, 0.0000],
        [2.8045, 2.0000],
        [3.0667, 0.0000],
        [3.1879, 0.0000],
        [3.2670, 0.0000],
        [3.4352, 0.0000],
        [4.7274, 2.0000],
        [5.2066, 0.0000],
        [5.4850, 2.0000],
        [6.0698, 2.0000],
        [6.2458, 2.0000],
        [6.4551, 2.0000],
        [6.5179, 2.0000],
        [7.1887, 2.0000],
        [7.4561, 2.0000]])
```

SSD Between a Gamma Output and Everything Else

```
tensor([[0.0000, 2.0000],
        [0.6989, 2.0000],
        [0.8808, 2.0000],
        [0.9785, 2.0000],
        [1.0642, 2.0000],
        [1.4129, 2.0000],
        [1.6941, 0.0000],
        [2.7337, 2.0000],
        [3.4392, 2.0000],
        [4.7337, 0.0000],
        [4.7642, 2.0000],
        [5.2808, 1.0000],
        [5.3887, 1.0000],
        [5.5720, 1.0000],
        [5.6197, 1.0000],
        [5.6981, 1.0000],
        [5.8332, 1.0000],
        [5.9034, 1.0000],
        [6.2458, 1.0000],
        [6.5384, 1.0000],
        [6.8275, 0.0000],
        [7.5142, 0.0000],
        [7.7369, 0.0000],
        [7.9338, 0.0000],
        [8.3272, 0.0000],
        [8.5211, 0.0000],
        [8.5926, 0.0000]])
```

`A reasonably good model for classification could be the K Nearest Neighbor classifier.`

```
KNN Model score =  1.0
```

E) Create your own greek symbol data

- **SSD Between an Alpha Output and Everything Else**

-
  ```
  tensor([[ 0.6521,  0.0000],
          [ 0.6539,  0.0000],
          [ 1.2356,  0.0000],
          [ 1.7802,  0.0000],
          [ 2.0005,  0.0000],
          [ 2.3747,  1.0000],
          [ 2.4242,  1.0000],
          [ 2.4663,  0.0000],
          [ 2.4766,  0.0000],
          [ 2.7706,  1.0000],
          [ 3.8343,  0.0000],
          [ 3.8917,  0.0000],
          [ 3.9279,  1.0000],
  ```

```
        [ 3.9358,  1.0000],
        [ 3.9756,  1.0000],
        [ 4.0335,  1.0000],
        [ 4.4910,  1.0000],
        [ 4.6236,  1.0000],
        [ 6.6639,  2.0000],
        [ 7.9156,  2.0000],
        [ 8.5211,  2.0000],
        [ 8.8232,  2.0000],
        [ 8.9148,  2.0000],
        [ 9.1582,  2.0000],
        [10.1931,  2.0000],
        [10.3499,  2.0000],
        [10.4383,  2.0000]])
```

SSD Between a Beta Output and Everything Else

```
tensor([[0.5234, 1.0000],
        [0.5550, 1.0000],
        [1.0720, 1.0000],
        [1.1593, 1.0000],
        [1.4891, 1.0000],
        [1.6539, 1.0000],
        [2.0289, 1.0000],
        [2.0933, 1.0000],
        [2.3008, 0.0000],
        [2.3747, 0.0000],
        [2.3957, 1.0000],
        [2.5130, 0.0000],
        [2.7092, 0.0000],
        [2.8045, 2.0000],
        [3.0667, 0.0000],
        [3.1879, 0.0000],
        [3.2670, 0.0000],
        [3.4352, 0.0000],
        [4.7274, 2.0000],
        [5.2066, 0.0000],
        [5.4850, 2.0000],
        [6.0698, 2.0000],
        [6.2458, 2.0000],
        [6.4551, 2.0000],
        [6.5179, 2.0000],
        [7.1887, 2.0000],
        [7.4561, 2.0000]])
```

SSD Between a Gamma Output and Everything Else

```
tensor([[0.6754, 2.0000],
        [0.6989, 2.0000],
        [0.8808, 2.0000],
        [0.9785, 2.0000],
        [1.0642, 2.0000],
        [1.4129, 2.0000],
        [1.6941, 0.0000],
        [2.7337, 2.0000],
        [3.4392, 2.0000],
        [4.7337, 0.0000],
        [4.7642, 2.0000],
        [5.2808, 1.0000],
        [5.3887, 1.0000],
        [5.5720, 1.0000],
        [5.6197, 1.0000],
        [5.6981, 1.0000],
        [5.8332, 1.0000],
        [5.9034, 1.0000],
        [6.2458, 1.0000],
        [6.5384, 1.0000],
        [6.8275, 0.0000],
        [7.5142, 0.0000],
        [7.7369, 0.0000],
        [7.9338, 0.0000],
        [8.3272, 0.0000],
        [8.5211, 0.0000],
        [8.5926, 0.0000]])
```

```
A reasonably good model for classification could be the K Nearest Neighbor classifier.

KNN Model score =  0.85
```

Task 4: Design your own experiment

A) Develop a plan

- I then built a network which I will train on the MNIST training set. The network is structured as follows:

  Inputs: 38x38 1 channel images

  Convolutional Layer 1: 7x7 kernel with no image padding with 10 output channels → produces 15x32x32 output

  Max Pool Layer: 2x2 kernel → produces 15x16x16 output

  Convolutional Layer 2: 7x7 kernel with no image padding with 30 output channels → produces 30x10x10 output

  Max Pool Layer: 2x2 kernel → produces 30x5x5 output

  Fully Connected Layer 1: Takes 30x4x4 (750) inputs → produces 50 outputs

  Fully Connected Layer 2: Takes 50 inputs → produces 10 outputs

B) Predict the results

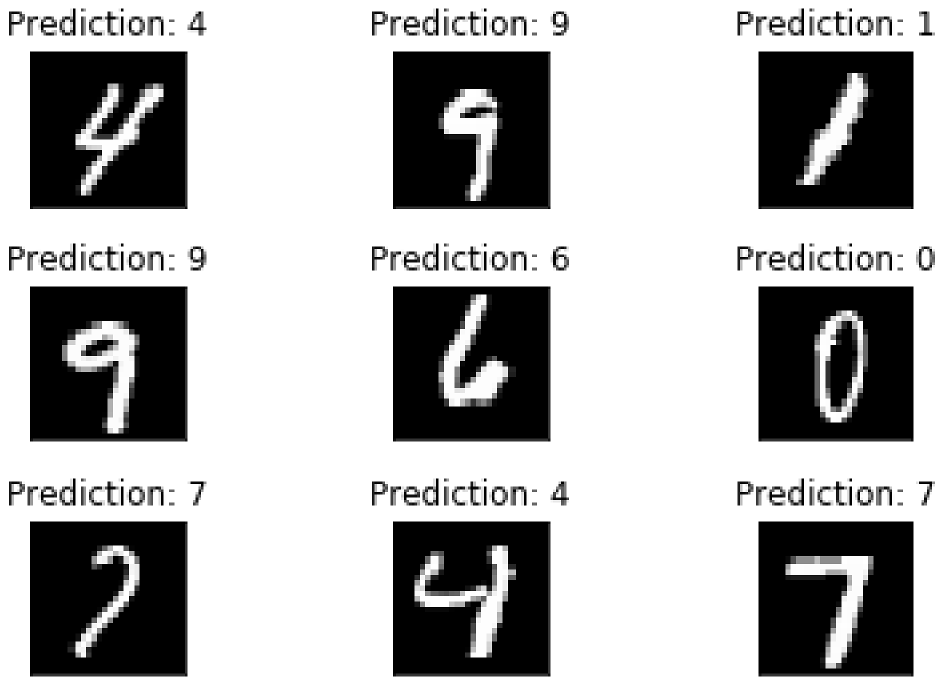
Hypothesis:

1. In regards to the number of convolutional filters,I expected that fewer filters would take less time to train but would be less accurate, whereas a greater number of filters would take more time but would be far more precise.
2. In regards to kernel size, I thought a smaller kernel size would be less accurate and take less time to train, while a bigger kernel size would be more accurate and take more time to train.
3. In regards to epoch size,I anticipated that training with fewer epochs would take less time but be slightly less accurate, whereas training with more epochs would take longer but be more accurate.

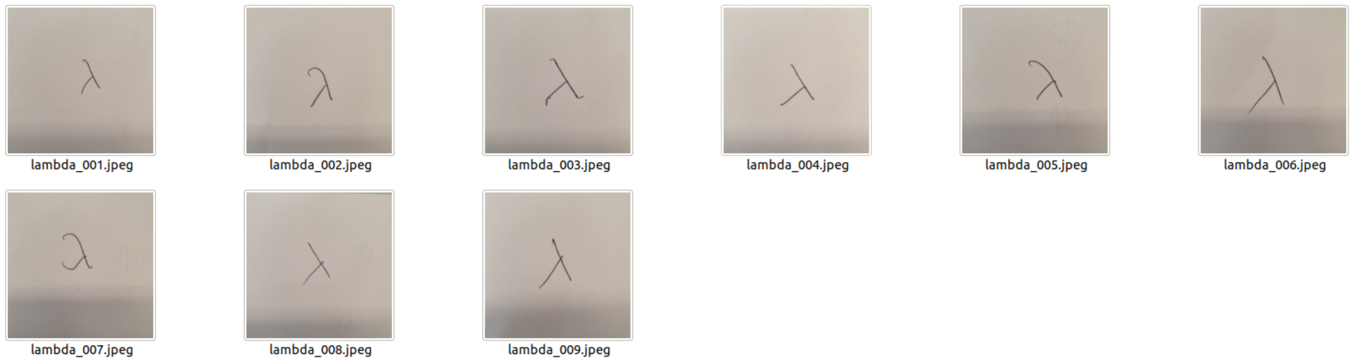
C) Execute your plan

- Results:

  The accuracy for the Control Model after each epoch is: ['95', '96', '97', '97', '98']

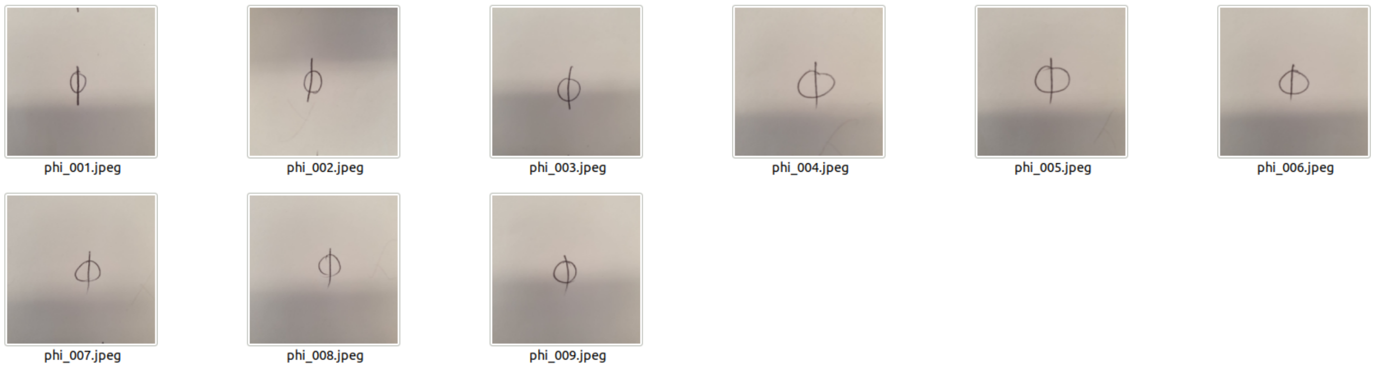


Extension 1: Implement KNN Classifier to predict the handwritten Greek symbols



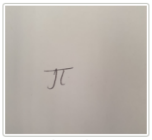
SSD Between a Lambda Output and Everything Else

```
tensor([[0.0000, 0.0000],
        [0.0169, 0.0000],
        [0.0521, 2.0000],
        [0.0542, 0.0000],
        [0.1141, 0.0000],
        [0.1267, 0.0000],
        [0.1428, 2.0000],
        [0.1842, 2.0000],
        [0.2555, 2.0000],
        [0.2717, 2.0000],
        [0.3161, 0.0000],
        [0.3679, 2.0000],
        [0.4121, 2.0000],
        [0.4479, 2.0000],
        [0.5002, 0.0000],
        [0.5799, 0.0000],
        [0.5947, 0.0000],
        [0.6466, 1.0000],
        [0.7182, 1.0000],
        [0.7664, 1.0000],
        [0.7672, 1.0000],
        [0.8312, 1.0000],
        [0.9758, 2.0000],
        [1.0084, 1.0000],
        [1.1153, 1.0000],
        [1.4234, 1.0000],
        [1.6173, 1.0000]])
```

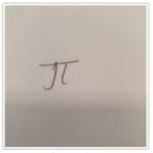


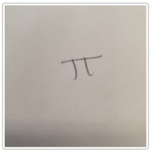
SSD Between a Phi Output and Everything Else

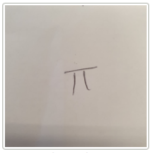
```
tensor([[0.0000, 1.0000],
        [0.1113, 1.0000],
        [0.6755, 1.0000],
        [0.7592, 1.0000],
        [0.9028, 1.0000],
        [0.9767, 2.0000],
        [0.9876, 1.0000],
        [1.0236, 1.0000],
        [1.0688, 0.0000],
        [1.0847, 2.0000],
        [1.1231, 1.0000],
        [1.1809, 2.0000],
        [1.1848, 0.0000],
        [1.2382, 2.0000],
        [1.2472, 0.0000],
        [1.2532, 0.0000],
        [1.2658, 0.0000],
        [1.3101, 2.0000],
        [1.3735, 0.0000],
        [1.4174, 0.0000],
        [1.4234, 0.0000],
        [1.4314, 2.0000],
        [1.4600, 2.0000],
        [1.5277, 0.0000],
        [1.5582, 2.0000],
        [1.6679, 2.0000],
        [2.5507, 1.0000]])
```

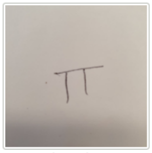

pi_001.jpeg

pi_002.jpeg

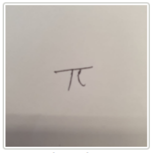

pi_003.jpeg


pi_004.jpeg


pi_005.jpeg


pi_006.jpeg

pi_007.jpeg


pi_008.jpeg

pi_009.jpeg

SSD Between a Pi Output and Everything Else

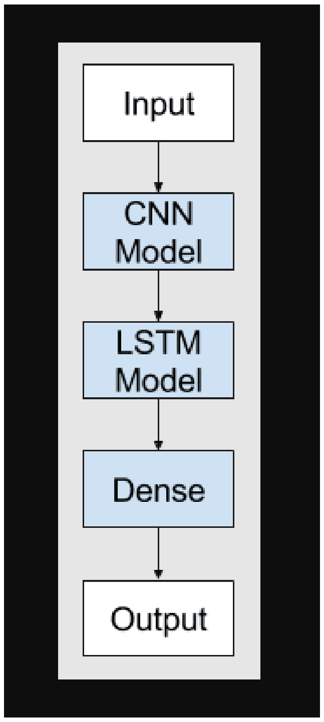
```
tensor([[0.0000, 2.0000],
        [0.0208, 2.0000],
        [0.2748, 2.0000],
        [0.3247, 0.0000],
        [0.3570, 0.0000],
        [0.3591, 2.0000],
        [0.3601, 2.0000],
        [0.3651, 0.0000],
        [0.3929, 2.0000],
        [0.4121, 0.0000],
        [0.4139, 0.0000],
        [0.4364, 0.0000],
        [0.5079, 2.0000],
        [0.5414, 1.0000],
        [0.5527, 2.0000],
        [0.5697, 0.0000],
        [0.5764, 2.0000],
        [0.5859, 1.0000],
        [0.6278, 0.0000],
        [0.7612, 1.0000],
        [0.8152, 0.0000],
        [0.8216, 1.0000],
        [0.8585, 1.0000],
        [0.9798, 1.0000],
        [1.0792, 1.0000],
        [1.0847, 1.0000],
        [1.6281, 1.0000]])
```

KNN score =

```
score =  0.8518518518518519
```

Extension 2: Concatenate time distributed CNN with LSTM

1. The CNN LSTM architecture involves using Convolutional Neural Network (CNN) layers for feature extraction on input data combined with LSTMs to support sequence prediction.
2. The CNN Long Short-Term Memory Network or CNN LSTM for short is an LSTM architecture specifically designed for sequence prediction problems with spatial inputs, like images or videos.



```
Test set: Average loss: 0.0905, Accuracy: 9747/10000 (97%)
```

Extension 3: Design your own CNN and test the accuracy of model

- I then built a network which I will train on the MNIST training set. The network is structured as follows:

  Inputs: 48x48 1 channel images

  Convolutional Layer 1: 3x3 kernel with no image padding with 10 output channels → produces 20x44x44 output

  Max Pool Layer: 2x2 kernel → produces 20x22x22 output

  Convolutional Layer 2: 7x7 kernel with no image padding with 30 output channels → produces 40x16x16 output

  Max Pool Layer: 2x2 kernel → produces 40x8x8 output

  Fully Connected Layer 1: Takes 40x8x8 (2560) inputs → produces 50 outputs

  Fully Connected Layer 2: Takes 50 inputs → produces 10 outputs

I hypothesized less epochs would take less time to train but be slightly less accurate, whereas more epochs would be longer training and more accurate

- Number of Epochs used for training
  - Model A: 5 Epochs
  - Model B: 10 Epochs

Results :

The accuracy for the ModelA: [40,80]- Conv Filters after each epoch is: ['95', '96' ,'97', '97', '97']
The accuracy for the Model3B:4 epochs after each epoch is: ['94','95','95', '96', '96', '97', '97', '97','98', '98']

Reflection

Through each of the tasks in the assignment my learning outcome was as follows:

- Learned about Pytorch library.
- Learned implementing the deep neural networks.
- Hypothesized how changing hyper-parameters would work.
- Understood how each layer filters out features from an input image,

Acknowledgements:

- Pytorch documentation
- Python tutorials
- OpenCV python documentation
- https://discuss.pytorch.org/t/solved-concatenate-time-distributed-cnn-with-lstm/15435

No labels