Project 5 Report: Recognition using Deep Networks

Created by Yao Zhong zhong.yao@northeastern.edu

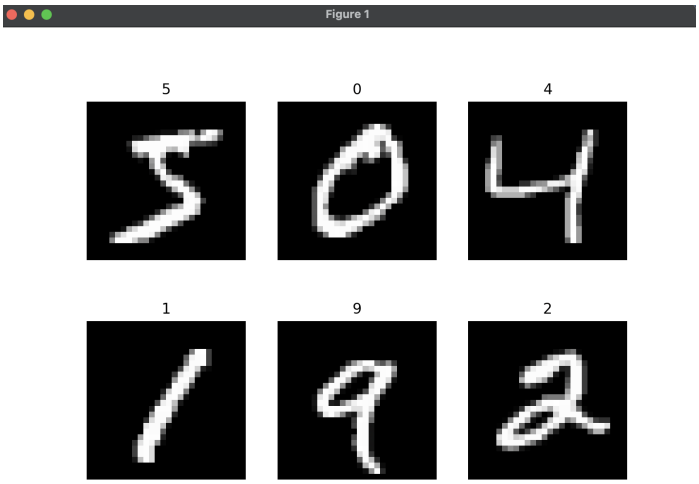
Overview

In this project, we learned how to build, train, test, analyze, and modify a deep network for a recognition task. This project started with building and training with the MNIST digit dataset and then added new handwriting of digits as new tests. After that, the network was analyzed by looking at each filter of the convolution layer and seeing their effects of them on the input images. More deeply, the network was modified and trained to recognize greek letters rather than just digits. At last, we conducted several experiments with MNIST Fashion dataset on the network, to see how different aspects of changes in the network will affect the training result.

Task 1: Build and train a network to recognize digits

A. Get the MNIST digit data set

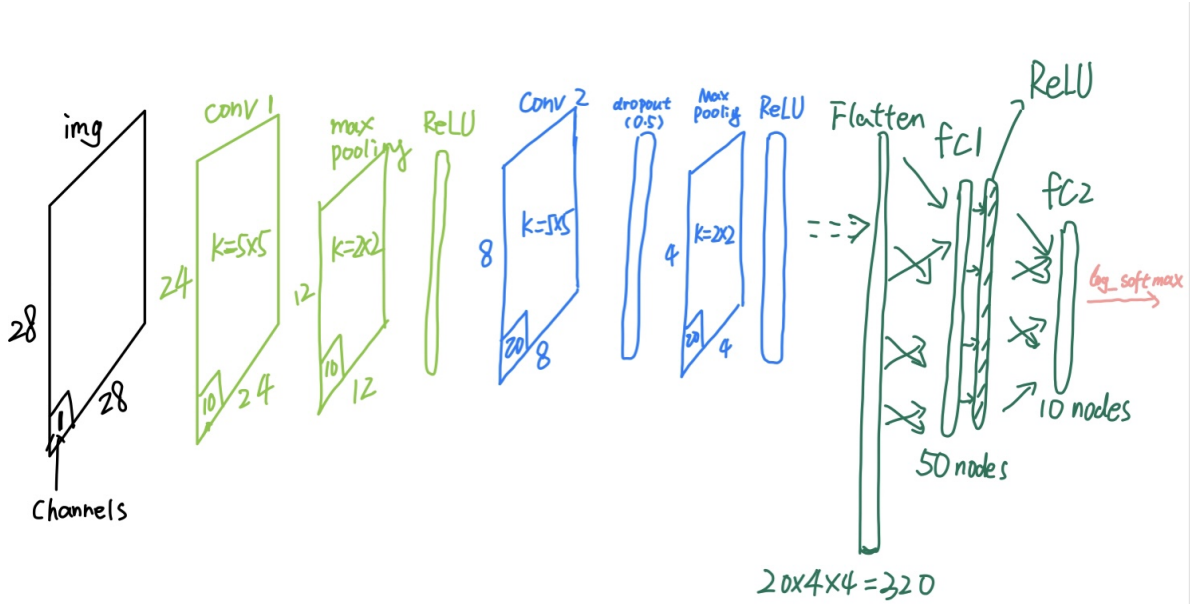
Here are the first six examples of the dataset that I loaded.



C.Build a network model

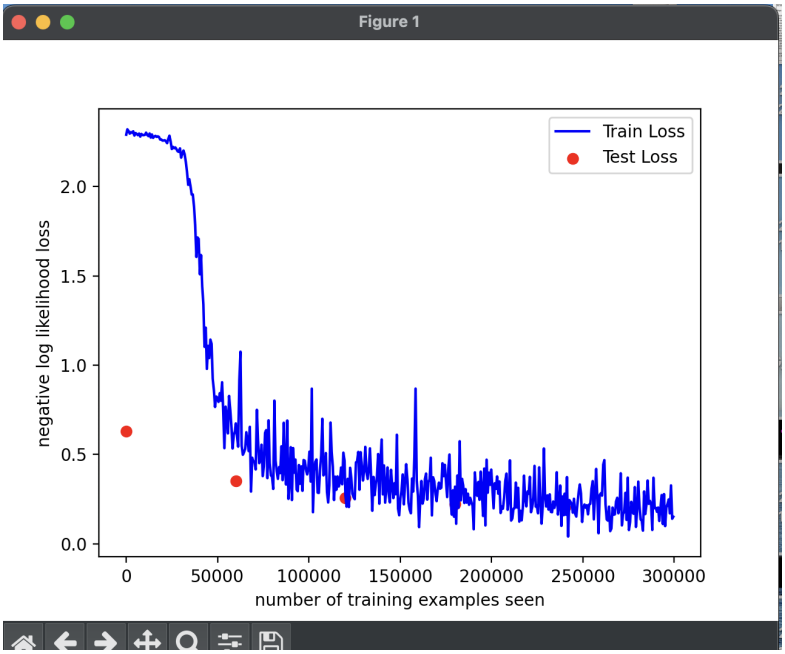
For the model, I implemented it as per the instruction required. There are 3 main parts: myStack is a nn.Sequential that holds all the convolution layers and pooling layers for the feature computing, fc1 and fc2 are two fully connected layers and the final output is with 10 nodes corresponding to 10 digits.

The following is a diagram of the network:



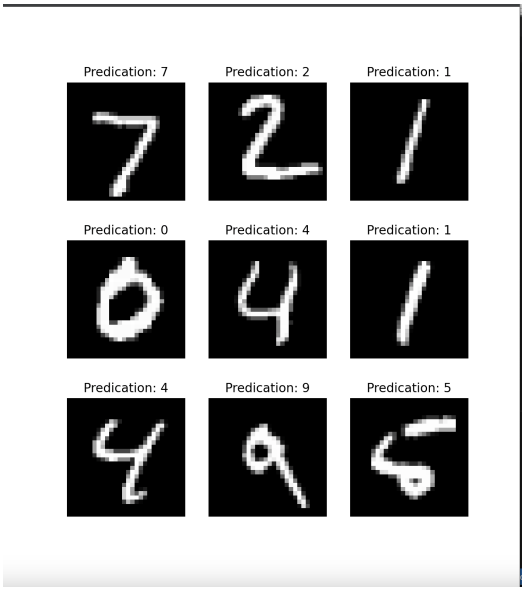
D. Train the model

After loading the training and test data and building the model, we can train the model. After setting the epochs=5, batch_size=64, and learning_rate=0.01. After that, we saved the model for later use. Here is the result of running the training and testing.



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

In a separate file, we loaded the network and tested it with 10 examples from the test set. Before we run the test, the model has been set to evaluation mode. The following are the results.

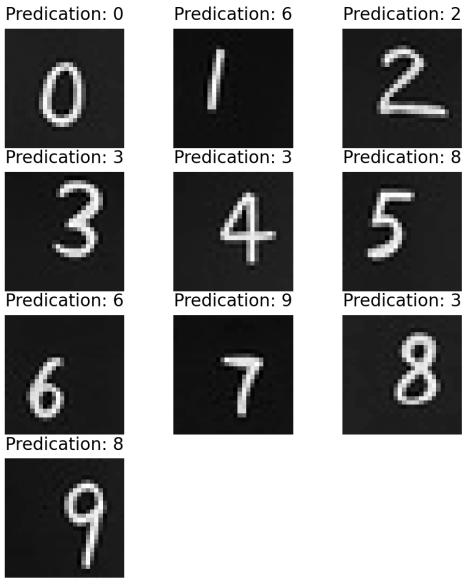


As we can see from the graph, the model can perfectly recognize the digits from the test set, and the accuracy is 100%.

G. Test the network on new inputs

In this task, we handwrote 10 digits [0~9] of our own and used the ImageMagick package to crop and resize each digit image into 28x28 size.

In the program, all the digit images are first pre-processed by OpenCV functions to grayscale and then inversed and normalized to match the intensities of the dataset. After these processes, the digits were still clearly visible. So we conducted the test with them, and the following are the results.

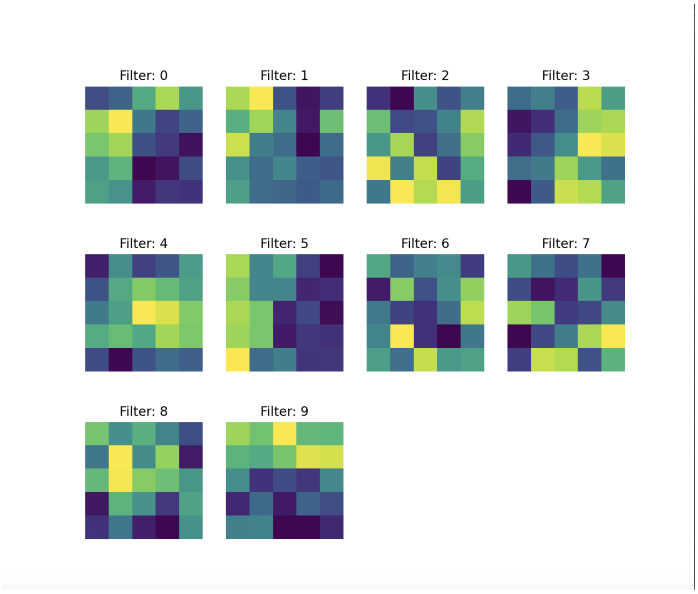


As we can tell from the result above. Although the new input digits have the same properties(intensity) as the test set and the digits can be clearly seen. But the test accuracy is not good, there were only 4 of them being correctly recognized, which makes the accuracy only 40%.

Task 2: Examine your network

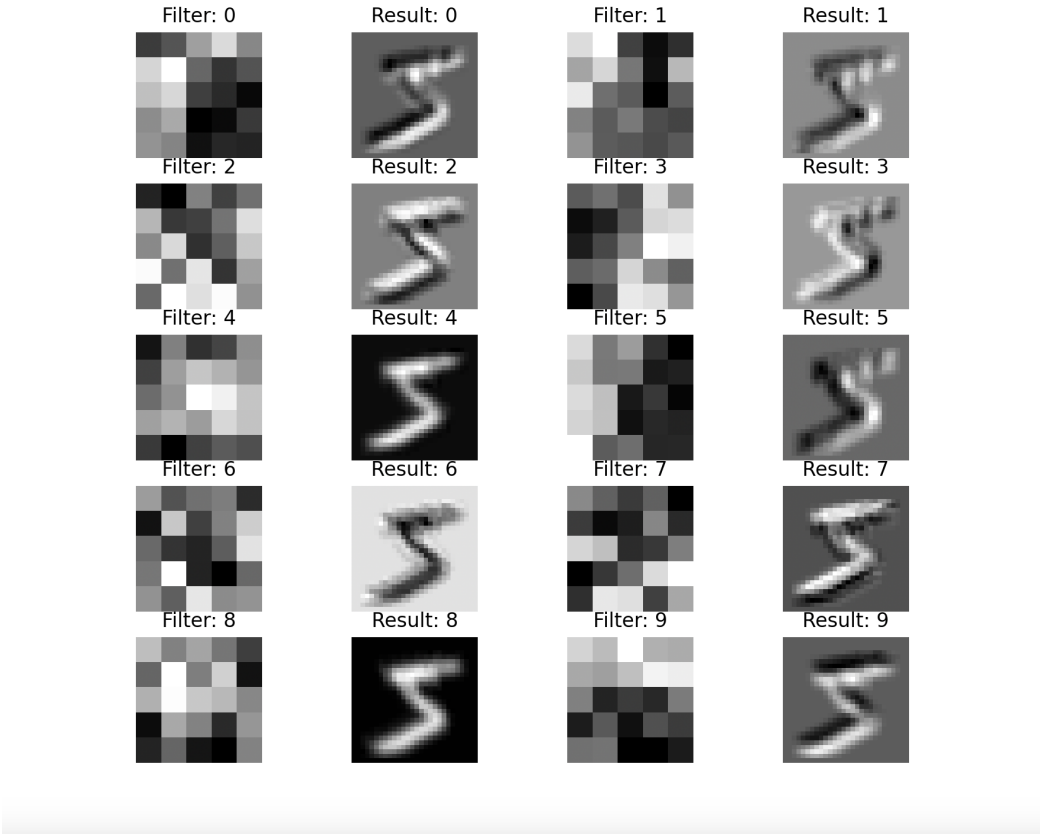
A. Analyze the first layer

In this task, we got the weights and visualized the weights of the first convolution layer, and the following are the results.



B. Show the effect of the filters

After acquiring the filters of the convolution layer, we explicitly applied them to the first training example image to see their effects on the input. Here is the result.



From the result above, we can see that results 0~3, result-5, result-7, and result-9, are emphasizing the edges of the digits from different orientations, just like the Gabor filters. And

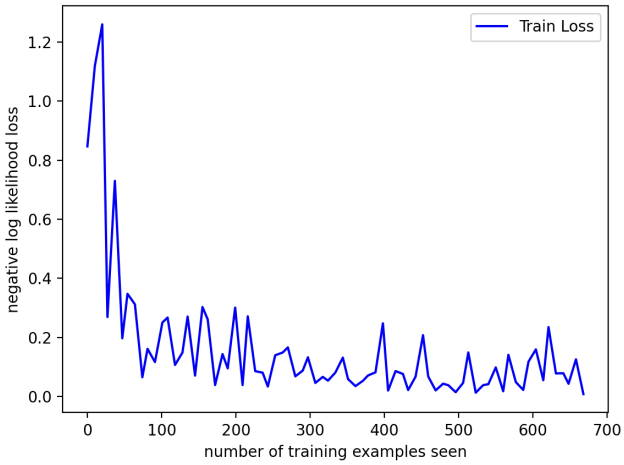
for result-4 and result-8, it segmented the digit and background into black and white, which is close to a “thresholding” on the image. And for result-6, it is a reverse of the intensity. Therefore, all the filters are close to the filters that we learned from classical computer vision, which makes the results reasonable.

Task 3: Transfer Learning on Greek Letters

In this task, the network is modified and re-trained to recognize the greek letters: alpha, beta, and gamma.The following is the modified network:

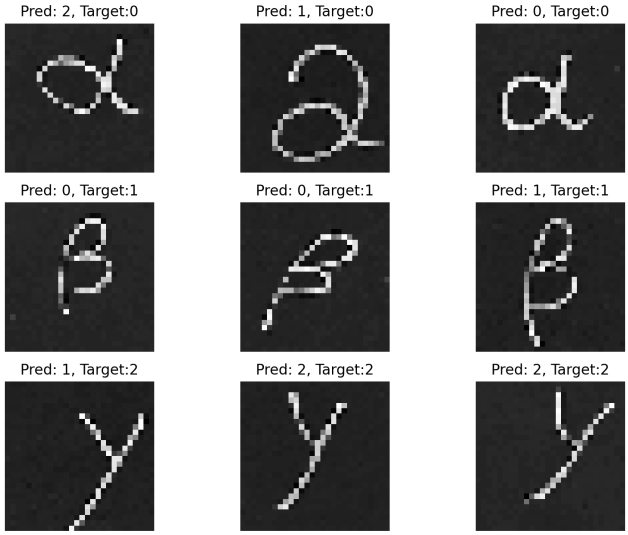
```
MyNetwork(
  (my_stack): Sequential(
    (0): Conv2d(1, 10, kernel_size=(5, 5), stride=(1, 1))
    (1): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (2): ReLU()
    (3): Conv2d(10, 20, kernel_size=(5, 5), stride=(1, 1))
    (4): Dropout(p=0.5, inplace=False)
    (5): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (6): ReLU()
  )
  (fc1): Linear(in_features=320, out_features=50, bias=True)
  (fc2): Linear(in_features=50, out_features=3, bias=True)
)
```

The first result is the training error.



Here is the graph of training loss when the epochs are set to 25. However, actually, according to the accuracy logged in the terminal, the accuracy of recognition reaches 100% as early as epoch-9, however, the relatively stable 100% accuracy is after 12 epochs of training. Therefore, considering not to over-learning, we decided to use 16 epochs to train the model and proceed with the test on new greek letters.

Here is the result of recognizing new greek letters that I wrote.



However, the result is not satisfying, the model can only correctly recognize 1/3 of the alpha, 1/3 of the beta, and 2/3 of the gamma, which makes the overall accuracy on new input only 44.4%. This result is not satisfying, therefore, we explored this problem more deeply in the extensions.

Task 4: Design your own experiment

Experiment 1: The effect of filter number and filter size

Experiment dimensions:

The size of the convolution filters

The number of convolution filters in a layer

Materials:

MNIST fashion data set

Network developed from previous tasks

Metrics:

The average loss of the test set, the accuracy of the test result

Design of experiment:

The size of filters of the first layer will have 3 options: 3, 5, and 7.

The size of filters of the second layer will have 3 options: 3, 5, and 7.

The number of filters in the first layer has 4 options, 5, 10, 15, and 20, and the number of filters in the second layer is double that of the first layer, 10, 20, 30, and 40.

Therefore, for this experiment, there will be 3*3*4 = 36 variations

Hypothesis:

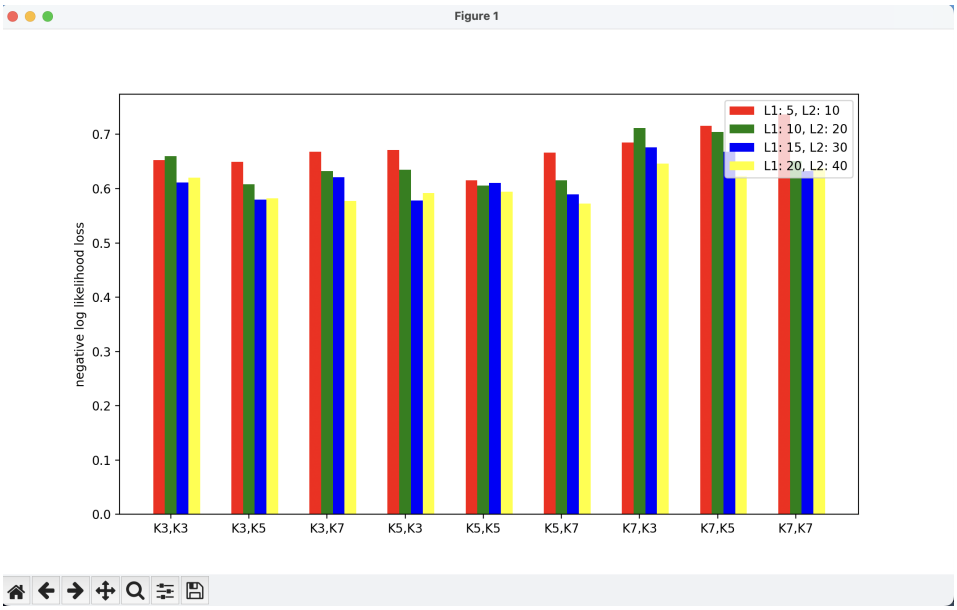
The size of the filter is smaller the better. Because when the size of the filter is smaller, the information is less blurred with the information around, and it can be more accurate.

The more filters the better, more filters will allow more channels of features to be captured, and therefore more aspects of the images will be considered.

Therefore, the best combination might be layer 1: 20 filters with kernel size 3x3 and layer 2: 40 filters with kernel size 3x3.

Result:

The following is the diagram of average loss on the test set under different combinations.



The average loss of each combination

From the result, we can find out that, as for the size of filters, a smaller filter size is better than the large one. For example, the combinations with a first layer that have a kernel size K7 (K7+K3 K7+K5, and K7+ K7) have a higher loss than other combinations.

As for the number of filters, we can see the difference inside each group. The common phenomenon is that when the number of filters increases, the avg loss decreases. Therefore, increasing the number of filters inside each convolution layer is good for the model.

From the result, the original combination: (L1:10, K5, L2:20, K5) has an average loss of 0.6057 and an accuracy of 0.8036. And the optimal combination: (L1:20, K5, L2:40, K7) has an average loss of 0.5724 and an accuracy of 0.8083.

The average loss is improved by 5.5%, and the accuracy is almost the same with only a 0.5% improvement.

B. The effect of dropout rate and dropout layer number

Experiment dimensions:

The dropout rates of the Dropout layer

Whether to add another dropout layer after the fully connected layer

Materials:

MNIST fashion data set

Network developed from previous tasks

The best combination from the previous experiment

Metrics:

The average loss of the test set, the accuracy of the test result

Design of experiment:

The dropout rate has 10 options 0, 0.1, 0.2, 0.3, 0.4,0.5, 0.6, 0.7, 0.8, 0.9

The second dimension for this experiment is whether to add another dropout layer after the fully connected layer, there will be 2 options: add(0.5 dropout rate) or not add(0 dropout rate). And the dropout layer will be added after the first fully connected layer

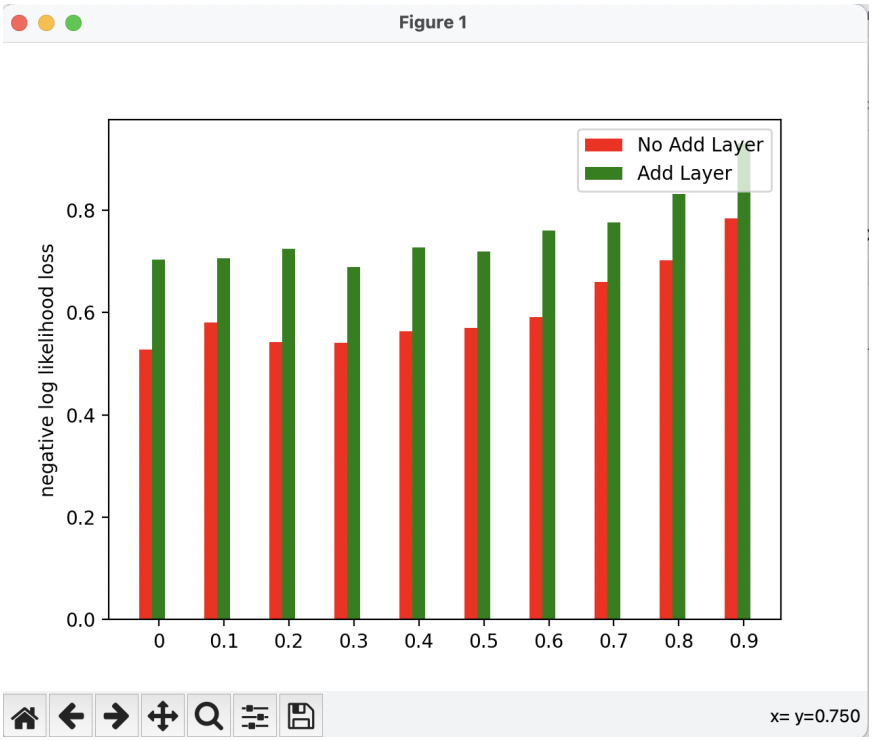
Therefore, there are 10*2 = 20 variations

Hypothesis:

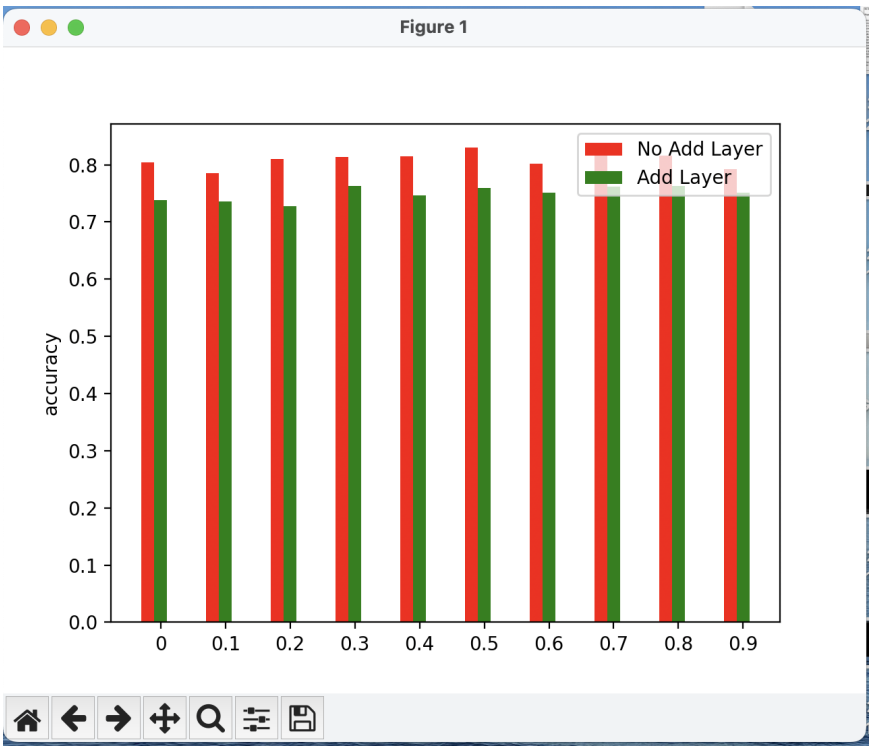
We expect to see that a medium dropout rate will be an optimal one. for example 0.4~0.6.

And we also expect that adding a dropout layer after the fully connected layer will help with decrease the average loss and increase accuracy.

Result:



The average loss of each variation



Accuracy of each combination

In the dropout dimension, from the result of average loss, we can tell that, the average loss has a “Nike” shape, which first decreases when the dropout rate increases, and then increases when the dropout rate increases. This result indicates that a medium dropout rate(0.3~0.5) is good for the average loss. From the result of the accuracy, we can also

find out a similar issue, the medium dropout rate has relatively better accuracy, and the accuracy peaks when the dropout rate is 0.5.

For the dimension of whether to add a dropout layer after the fully connected layer. We can find out that, in both average loss and accuracy, when adding the new dropout layer(green bars), is always worse than not adding the new layer. It leads to a higher loss and lower accuracy.

According to the result, the best combination will be a medium dropout rate of 0.5 and not add a new dropout layer after the fully connected layer, which is the current design.

Extensions

Extension 1:

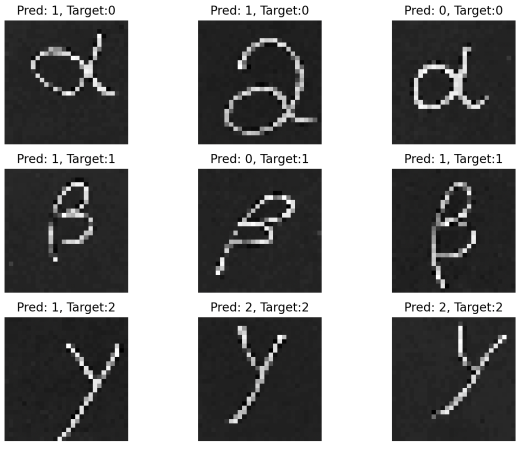
In task 4, we experimented with 4 dimensions rather than the required 3 dimensions, which could be considered as an extension.

Extension 2: the effect of training material on recognizing greek letters

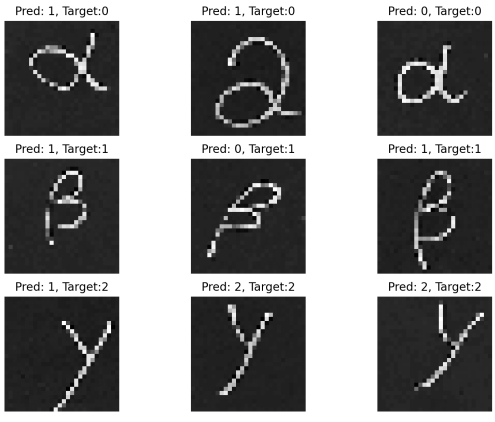
In task 3, we transformed our model so that we can run it to train and test with greek letters. However, the test result was not ideal, the accuracy was only 44.4%, slightly higher than the random guess. However, after the training, the accuracy of the training material itself was 100% accurate.

Therefore, we have a hypothesis of why this happens.

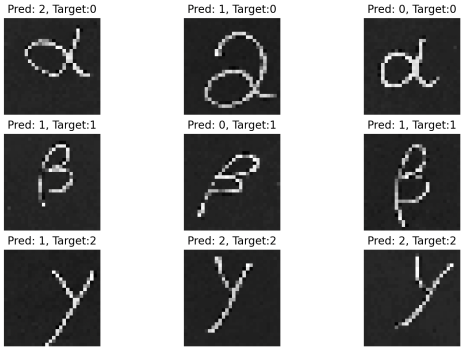
Firstly, we thought might be our 16 epochs is too high, although it makes the training set accurate, but actually overlearning the material. Therefore, we conducted an experiment on this dimension and trained the model for 3, 8, 12, and 16 epochs(16 was the setting of task 3)



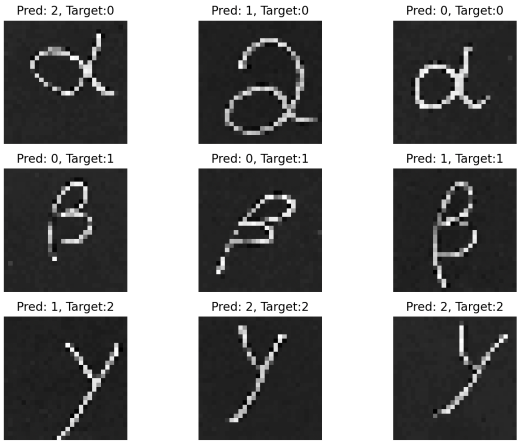
Epochs=3, Accuracy 5/9 = 55.6%



Epochs=8, Accuracy: 5/9 = 55.6%



Epoxhes=12, Accuracy 5/9 = 55.6%

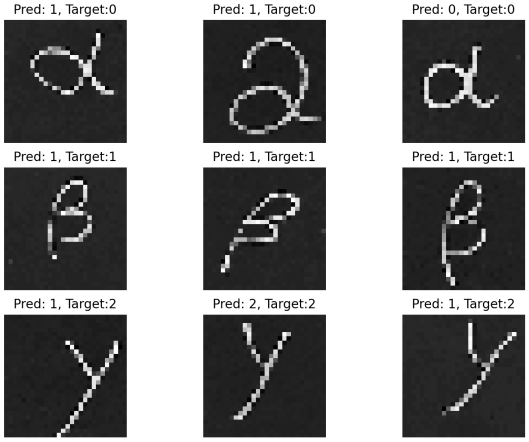


Epochs=16, Accuracy 4/9 = 44.4%

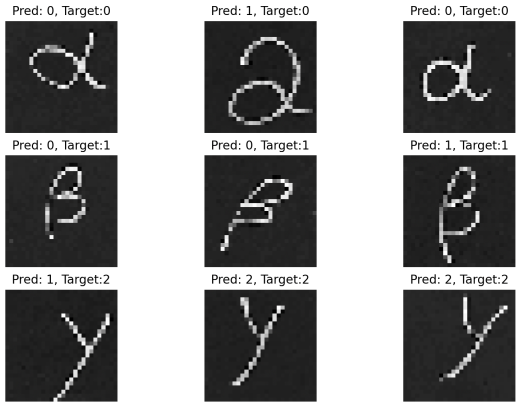
From the result above, we found out that all the epochs that are lower than 16 have better accuracy when trying to recognize the new greek letters. Therefore, when the accuracy of the training set itself reaches 100% accuracy, overlearning happens.

However, what is obvious is that even if we lowered the epochs, the accuracy is still low, and only 5/9 letters will be recognized correctly. Therefore, we have our second hypothesis: the learning material itself. We are training the model with the letters given by the assignment, however, the test cases are all newly written. My writing is different from that of the provider. Therefore, we decided to extend the training material and try to see if it can improve the recognition result.

We first added 3 of each type of handwritten greek letter (different from test examples but written under similar conditions) to the training set, we call it additional_v1. And then we added 3 more to make the additional_v2, which has 15 examples for each letter. Here is the result.



Epochs= 16, 3 additional examples each
Accuracy 5/9 = 55.6%



Epochs= 16, 6 additional examples each
Accuracy 5/9 = 55.6%

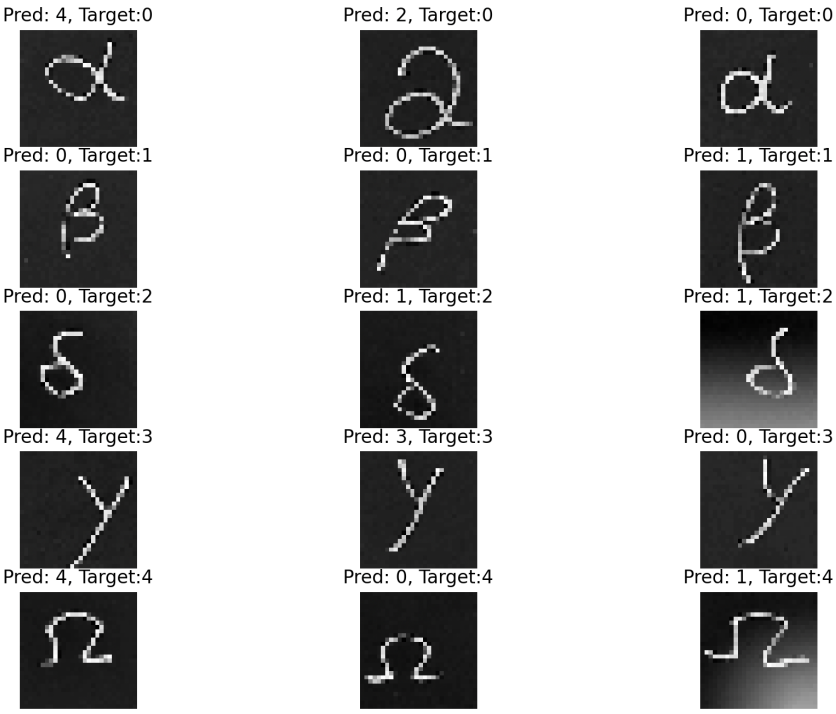
From the result, we can find out that no matter whether we add 3 examples for each letter or we add 6 examples for each letter, the accuracy still didn't improve much, only one more letter can be recognized compared with the original dataset.

However, we still find something interesting that in the addtional_v1, all the betas are correctly recognized, however, when not adding the new examples, only 1/3 of the beta can be recognized. Therefore, the new examples did affect the recognition, but due to the small size of the training dataset, the effect is not stable.

Extension 3: add new letters to the set and recognize them

We also created two more greek letters "delta" and "omega", and added them to the training set(10 examples each) and test set(3 examples each).

Here's the result:



Similar to the result of the other greek letters, the recognition accuracy is pretty low. The possible reason can be the limited amount of the examples and even the model itself should be modified to adapt the greek letters.

Reflections

From this assignment, I learned about how to build, train, and test a deep network. After this assignment, I am familiar with the workflow of using the deep network to do recognition. More

importantly, I am more familiar with the concepts of deep networks, I now know how the convolution layers, dropout layers, max-pooling layers, and fully connected layers work. Furthermore, the experimenting part helps me to have a taste of doing research on improving the performance of a model. By trying different variations and seeing the result, I had a lot of fun and also learned more about each component in a deep network.

Acknowledgments

The PyTorch tutorial: https://pytorch.org/tutorials/beginner/basics/intro.html

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