Report 5: Recognition using Deep Networks

> Created by Hui Hu @hui hwoo

Description

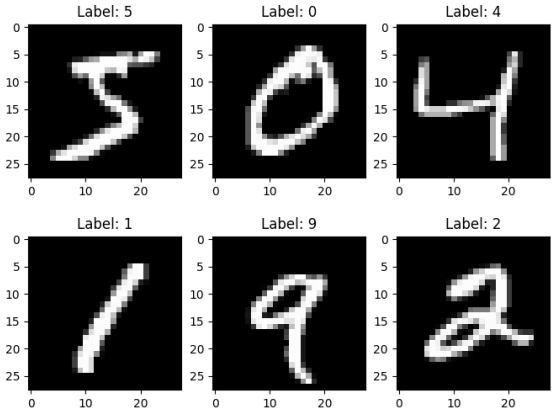
The objective of this project is to build, train, analyze, and modify a deep neural network for the recognition task using the MNIST digit dataset. The project involves tasks such as building and training the network, visualizing the dataset, making the network code repeatable, designing the network model, and training the model for five epochs. The final task of the project is to propose and design a final project.

Tasks

Task 1: Build and train a network to recognize digits

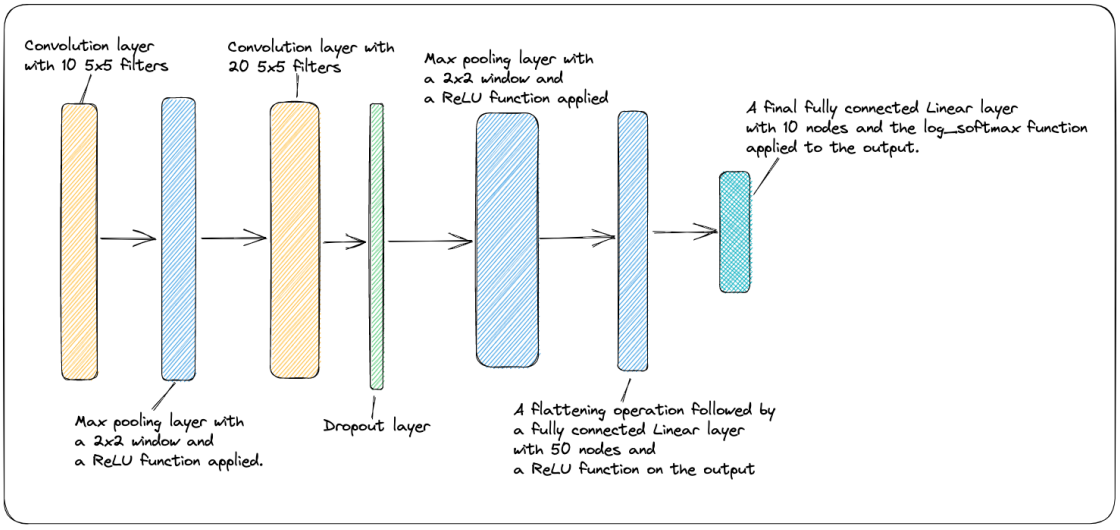
> Build and train a neural network to recognize digits using the MNIST dataset. The network should be built using pyTorch, and the code should be well-structured and repeatable. Train the model for five epochs and evaluate its performance on the test set. Finally, save the trained network to a file and test it on the first 10 examples in the test set.

- **Dataset:** the **MNIST** data base



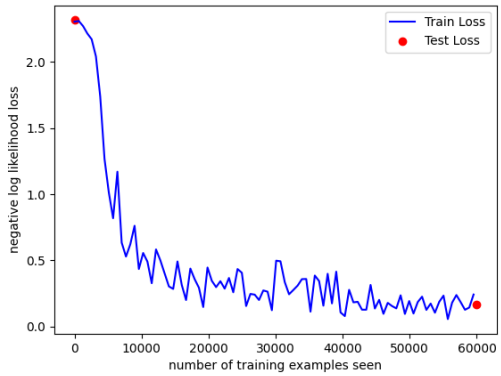
First six example digits

- **Network Model:**
  - A convolution layer with 10 5x5 filters
  - A max pooling layer with a 2x2 window and a ReLU function applied.
  - A convolution layer with 20 5x5 filters
  - A dropout layer with a 0.5 dropout rate (50%)
  - A max pooling layer with a 2x2 window and a ReLU function applied
  - A flattening operation followed by a fully connected Linear layer with 50 nodes and a ReLU function on the output
  - A final fully connected Linear layer with 10 nodes and the log_softmax function applied to the output.

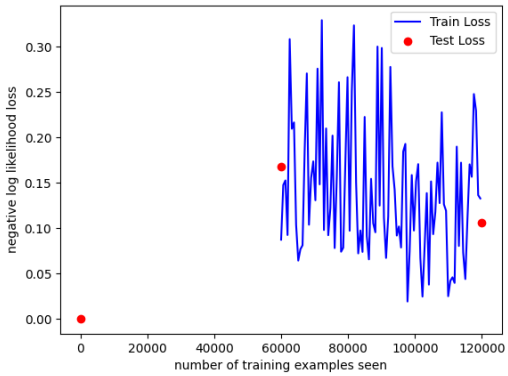


- **Train the model**

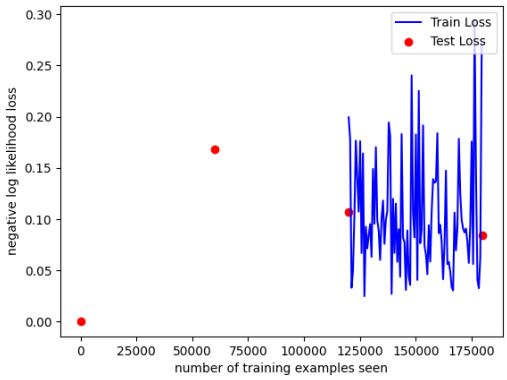
> Since the first value of test loss is meaningless, therefore, I set it 0 after epoch 1 when plotting the images.



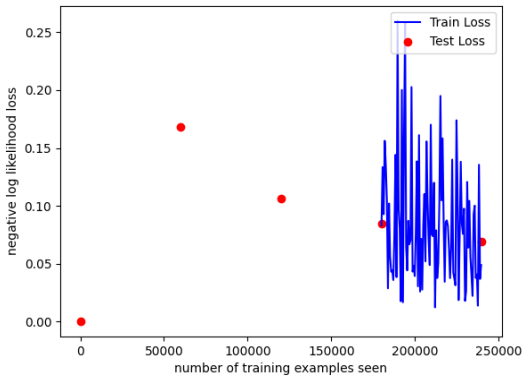
Epoch 1



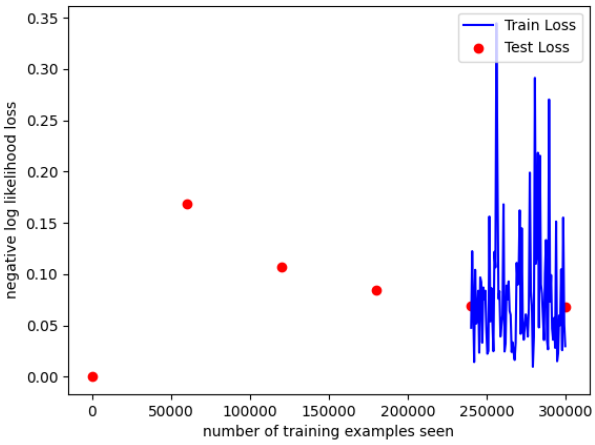
Epoch 2



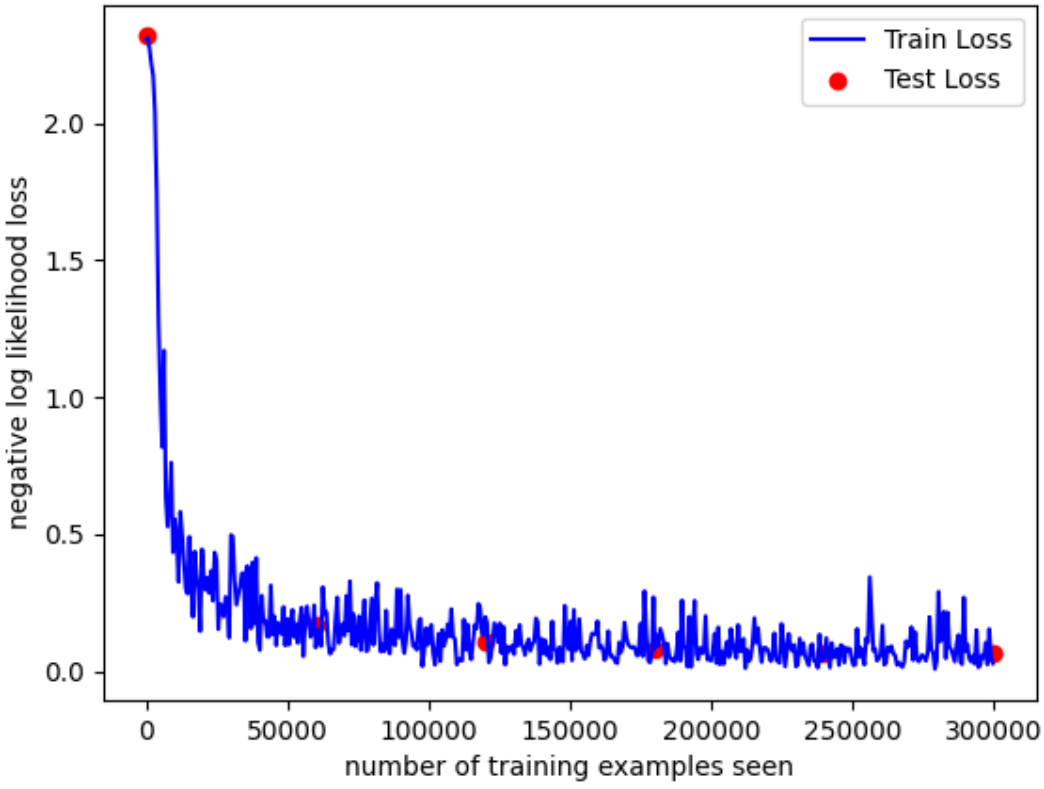
Epoch 3



Epoch 4



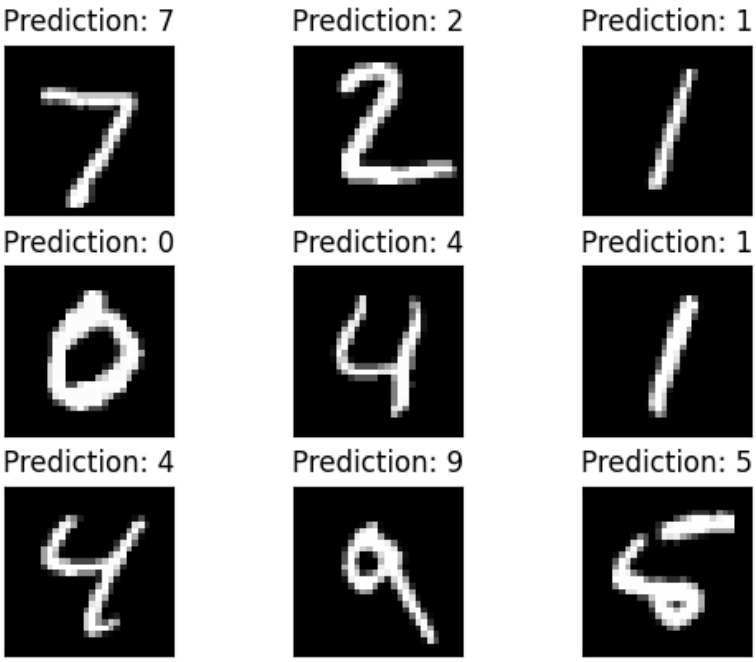
Epoch 5



Result after 5 epochs

As shown in above image, the negative log likelihood loss becomes smaller and smaller when training and modify the model weights, which produces a better recognition result.

- **Run model on the test set**

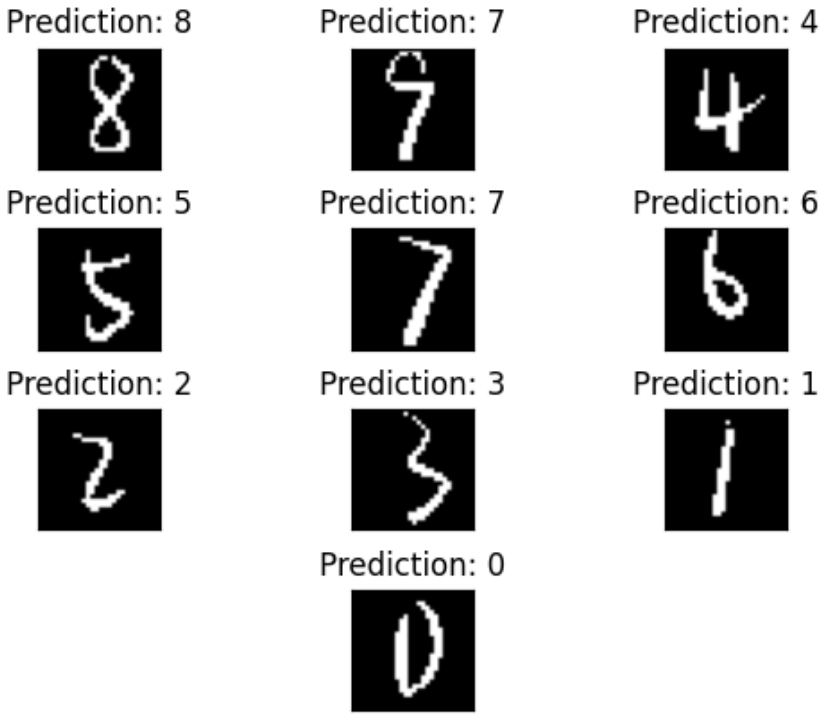


When running model on test dataset, model can always gives a good result.

Index	Output values	index of the max output value	correct label of the digit
1	[-10.56, -10.66, -6.41, -5.95, -14.19, -10.08, -20.64, **-0.01**, -10.34, -6.65]	7	7
2	[-5.39, -4.81, **-0.01**, -6.91, -13.72, -12.3, -8.5, -11.52, -7.75, -17.63]	2	2
3	[-7.72, **-0.01**, -7.53, -8.91, -5.48, -8.64, -6.83, -5.03, -7.13, -8.47]	1	1
4	[**-0.0**, -11.76, -7.37, -9.8, -11.35, -9.68, -7.53, -9.23, -9.23, -9.37]	0	0
5	[-10.65, -9.59, -11.11, -10.31, **-0.01**, -9.58, -9.73, -6.96, -10.67, -5.27]	4	4
6	[-10.04, **-0.0**, -10.56,	1	1

	-11.42, -6.8, -11.81, -9.97, -6.45, -9.09, -10.57]		
7	[-14.63, -7.31, -11.75, -13.27, **-0.01**, -9.57, -11.15, -7.31, -7.01, -5.69]	4	1
8	[-9.4, -7.73, -6.53, -2.78, -3.18, -4.67, -11.62, -5.43, -4.47, **-0.14**]	9	9
9	[-6.85, -12.15, -9.63, -10.07, -10.11, **-0.01**, -5.22, -10.07, -7.78, -7.07]	5	5

- **Run model on the new inputs**



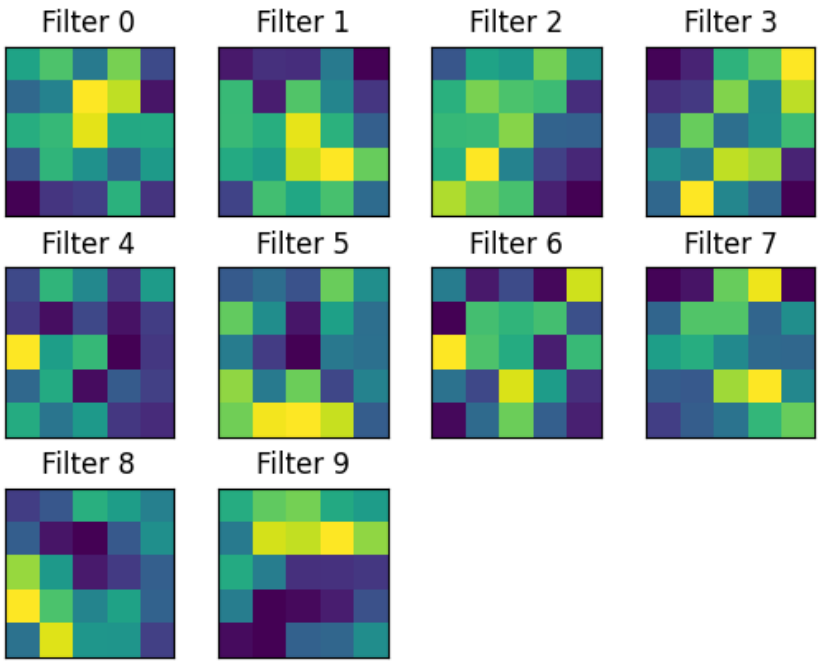
When running model on my handwritten digits, it will always confuses 7 and 9, but for any other digits, model can give correct answers.

At first the accuracy rate was only 10%, I found it's problem of normalization, after modification of the value of mean and std, I increased the rate to 90%.

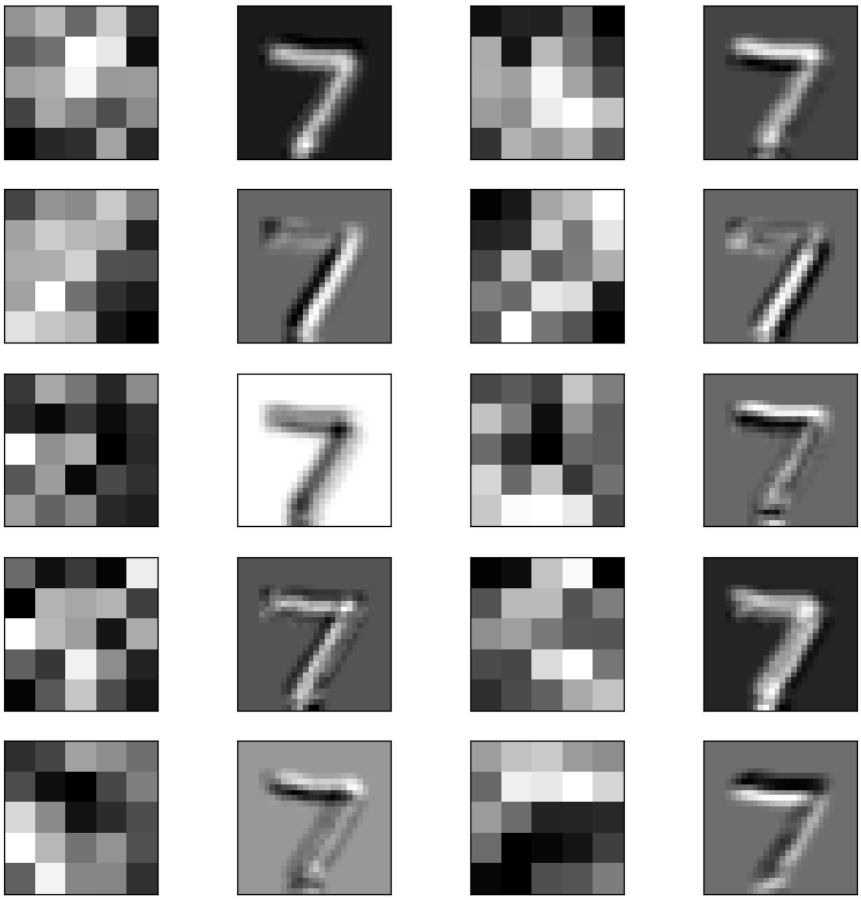
Task 2: Examine network

> The second task requires analyzing the neural network's data processing methods. First, read in the trained network and print its structure and layer names. Analyze the first layer by accessing its weights, visualizing the ten filters using pyplot, and applying them to the first training example image using OpenCV's filter2D function. The resulting plot should be included in the report, along with observations on whether the results are logical given the filters.

1. **Analyze the first layer**



2. **Show the effect of the filters**



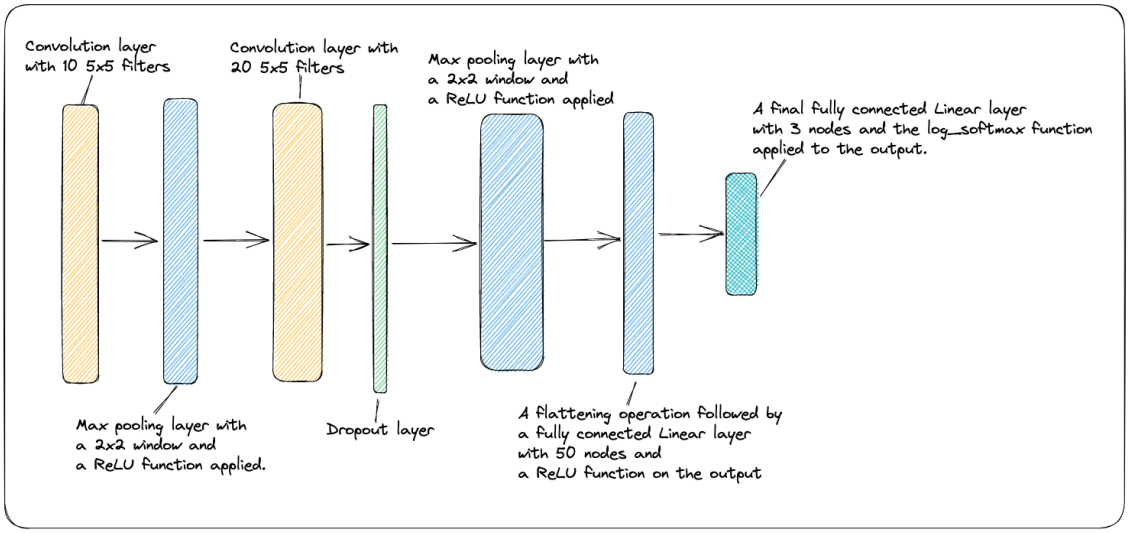
In a convolutional neural network, each filter is used to detect a specific feature or pattern in the input data. The filters in the first layer of a CNN are designed to detect low-level features such as edges, corners, and blobs in the input image. These low-level features are then combined and processed by deeper layers to detect more complex features and eventually classify the input. In the case of the given ML model, the 10 5x5 filters in the first layer are used to detect different low-level features in the input image.

Task 3: Transfer Learning on Greek Letters

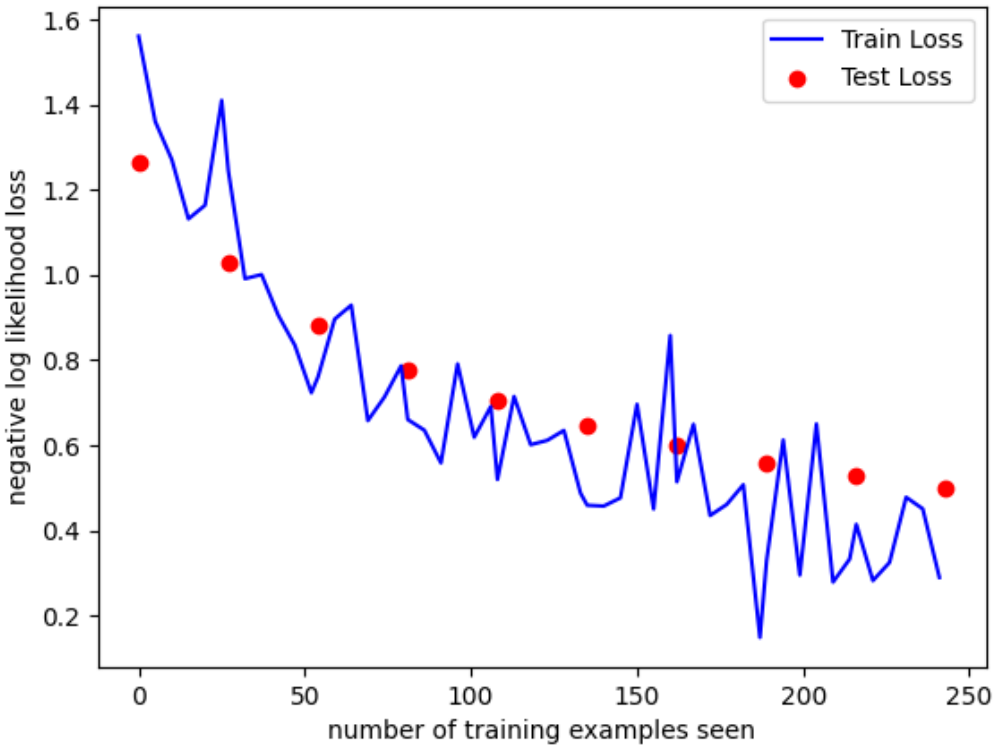
> Task 3 involves re-using the MNIST digit recognition network built in Task 1 to recognize three different Greek letters: alpha, beta, and gamma. The last layer of the pre-trained network

> should be replaced with a new Linear layer with three nodes. The pre-trained weights should be loaded and the network weights frozen. The task is to determine the number of epochs required to perfectly identify the 27 examples of the Greek letters provided.

- **Model**

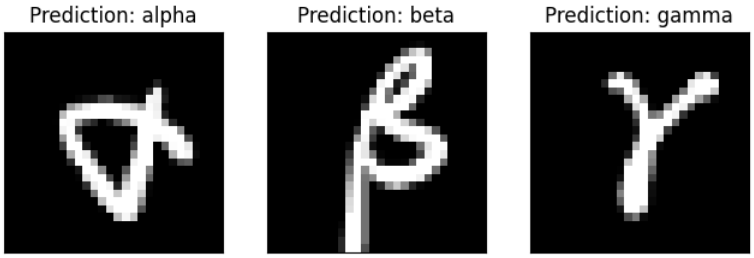


- **The training error**



Plot of the training error

- **The result of handwritten Greek letters**



Although the accuracy is enough high after 4 epochs, I still need at least 9 epochs to guarantee all the three handwritten Greek letters get correct answers.

```
Test set: Avg. loss: 1.1506, Accuracy: 4/27 (15%)
Test set: Avg. loss: 0.9896, Accuracy: 14/27 (52%)
Test set: Avg. loss: 0.8653, Accuracy: 21/27 (78%)
Test set: Avg. loss: 0.7776, Accuracy: 23/27 (85%)
Test set: Avg. loss: 0.7090, Accuracy: 25/27 (93%)
Test set: Avg. loss: 0.6543, Accuracy: 25/27 (93%)
Test set: Avg. loss: 0.6081, Accuracy: 25/27 (93%)
```

```
Test set: Avg. loss: 0.5728, Accuracy: 25/27 (93%)
Test set: Avg. loss: 0.5415, Accuracy: 26/27 (96%)
Test set: Avg. loss: 0.5106, Accuracy: 26/27 (96%)
```

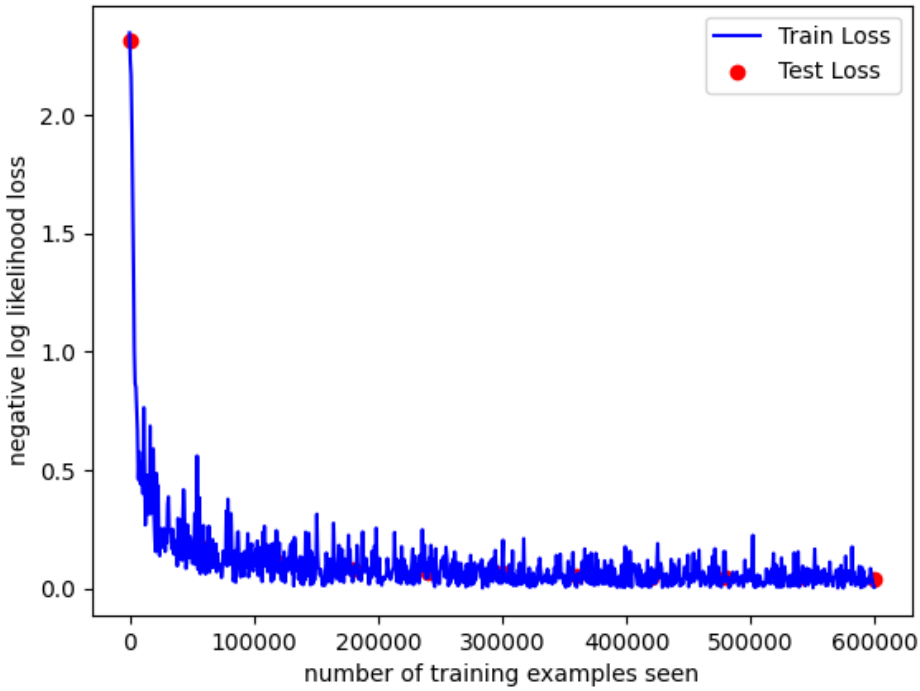
Task 4: Design customized experiment

1. ### The number of epochs of training

   The number of epochs is a hyperparameter that controls the number of times the entire dataset is passed through the model during training. Each epoch consists of one forward pass and one backward pass (i.e., one training step) for each batch of data.

   Increasing the number of epochs allows the model to see the data more times, which can improve the model's performance. However, increasing the number of epochs can also lead to overfitting, where the model becomes too specialized to the training data and performs poorly on new data. Therefore, it is important to find the right balance of epochs to achieve the best performance without overfitting. This balance can be found through experimentation and validation on a held-out set of data.

   After several experiments, I found the optima vlaue of epoch is 6, after 6 epochs, the accuracy will not significantly increase, so we could balance the time and accuracy, and make epochs be 7 to get more effectiveness and efficiency.




```
Test set: Avg. loss: 2.3125, Accuracy: 964/10000 (10%)
Test set: Avg. loss: 0.1726, Accuracy: 9618/10000 (96%)
Test set: Avg. loss: 0.1076, Accuracy: 9778/10000 (98%)
Test set: Avg. loss: 0.0839, Accuracy: 9827/10000 (98%)
Test set: Avg. loss: 0.0717, Accuracy: 9838/10000 (98%)
Test set: Avg. loss: 0.0674, Accuracy: 9842/10000 (98%)
Test set: Avg. loss: 0.0546, Accuracy: 9873/10000 (99%)
Test set: Avg. loss: 0.0498, Accuracy: 9872/10000 (99%)
Test set: Avg. loss: 0.0488, Accuracy: 9892/10000 (99%)
Test set: Avg. loss: 0.0459, Accuracy: 9859/10000 (99%)
Test set: Avg. loss: 0.0435, Accuracy: 9884/10000 (99%)
```

- **The dropout rates of the Dropout layer**

  The Dropout layer randomly drops (sets to zero) a fraction of the neurons in the previous layer during training. The effect of using Dropout is that it prevents overfitting by reducing the complex co-adaptations of neurons, forcing the remaining neurons to learn more robust features. The dropout rate determines the probability that a neuron will be dropped, with typical values ranging from 0.1 to 0.5. A higher dropout rate can increase regularization and reduce overfitting, but it can also result in underfitting if too many neurons are dropped, while a

lower dropout rate may not provide enough regularization to prevent overfitting. In practice, the dropout rate is often chosen through hyperparameter tuning on a validation set.

Drop rate	Avg. loss	Accuracy
0.1	0.0424	9872/10000 (99%)
0.2	0.0475	9847/10000 (98%)
0.3	0.0403	9883/10000 (99%)
0.4	0.0426	9891/10000 (99%)
0.5	0.0478	9898/10000 (99%)
0.6	0.0590	9874/10000 (99%)
0.7	0.0619	9884/10000 (99%)
0.8	0.0811	9862/10000 (99%)
0.9	0.1241	9849/10000 (98%)
0.999999	2.3053	995/10000 (10%)

We could conclude that, when drop rate is larger than 0.5, the loss becomes worse and worse.

- **The batch size while training**

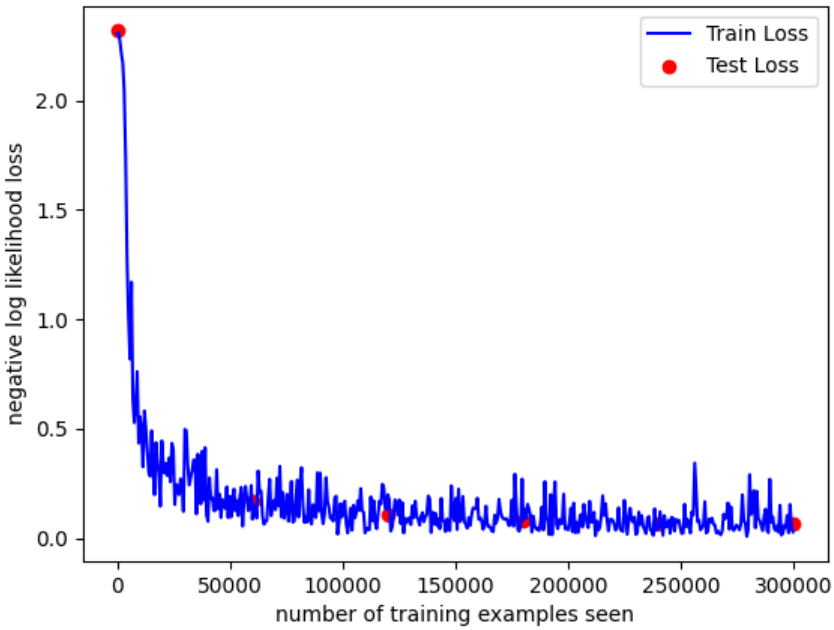
  The batch size refers to the number of training examples used in one forward/backward pass of the neural network. The effect of batch size on training is that:

  - A larger batch size can lead to faster training, as more examples can be processed in parallel, and the updates to the model's parameters are more stable as they are averaged over more examples.
  - A smaller batch size can lead to better generalization, as it introduces more randomness in the training process and prevents the model from overfitting to the training set.

  However, there are trade-offs with choosing a batch size. A larger batch size can require more memory and computational resources, which can be a limiting factor for training on certain hardware. A smaller batch size can lead to slower training, as fewer examples are processed in parallel, and can require more training epochs to reach the same level of performance as a larger batch size. Ultimately, the best batch size for a given problem depends on factors such as

the size and complexity of the dataset, the available hardware resources, and the desired level of performance and generalization.

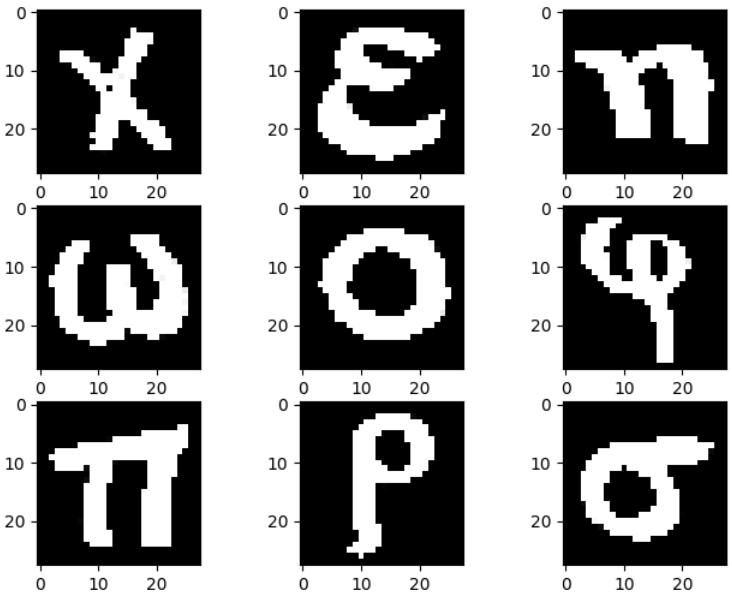
When I first try batch size = 1, it takes really long time to get the final result. Then I tried the batch size = 64, and batch size = 128, the only difference is the speed, the result are almost the same.



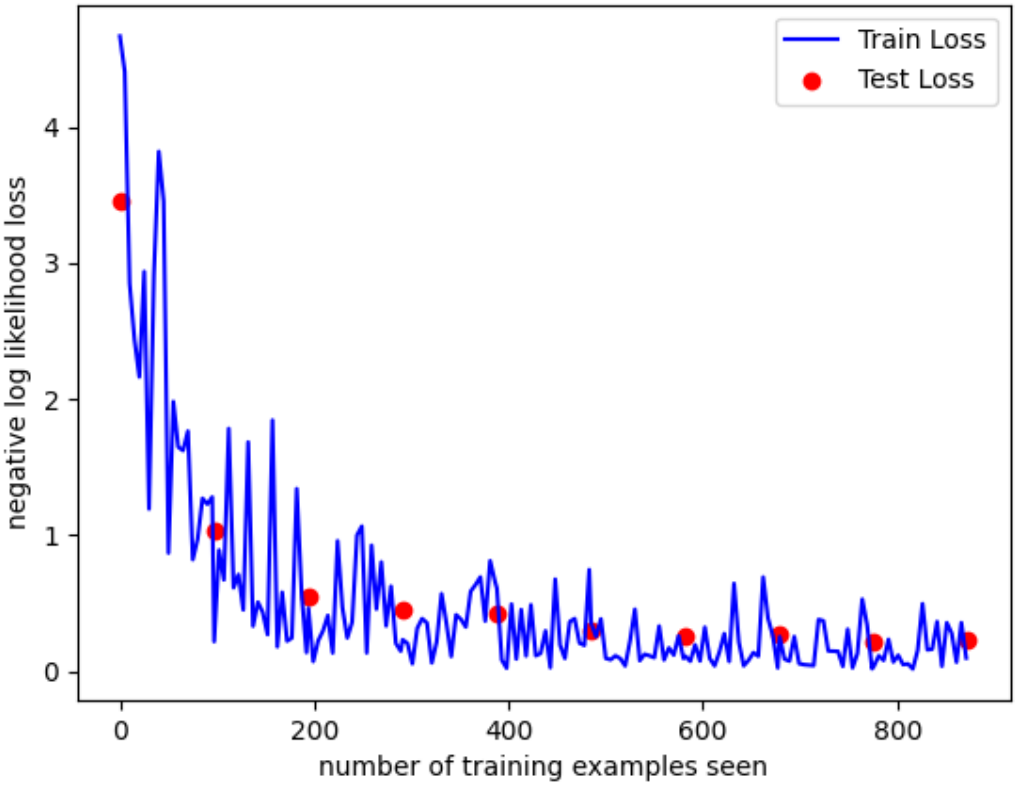
Extension

I add another 10 types of Greek letters, 97 in total, to the training dataset and 40 letters to test dataset. Below is the first 9 Greek letters. I got these image from Kaggle *Classification of Handwritten Greek Letters,* the link is https://www.kaggle.com/datasets/katianakontolati/classification-of-handwritten-greek-letters?resource=download.

Since the images from Kaggle are unlabeled, I have to recognize them one by one before training the model.



Following is the loss image, we could notice that we can get a pretty good result after only 3 epochs, after then, the accuracy almost keeps the same



Epoch	Loss	Accuracy
0	3.2359	4/40 (10%)
1	0.9438	24/40 (60%)
2	0.5023	35/40 (88%)
3	0.6416	31/40 (78%)
4	0.4149	35/40 (88%)
5	0.4850	35/40 (88%)
6	0.3593	33/40 (82%)
7	0.3423	36/40 (90%)
8	0.3498	35/40 (88%)
9	0.3351	35/40 (88%)
10	0.2649	39/40 (98%)

Transfer learning has several advantages:

1. Reduced Training Time: By reusing a pre-trained model, we can save a significant amount of time that would have been spent on training the model from scratch.
2. Improved Performance: A pre-trained model has already learned a lot of useful features from a large dataset. By using this knowledge, we can fine-tune the model on our specific task and achieve better performance than training a model from scratch on a smaller dataset.
3. Lower Data Requirements: Since a pre-trained model has already learned a lot of useful features, we can achieve good results even with a smaller amount of data.
4. Generalization: Pre-trained models have learned to generalize well to new data. By fine-tuning on our specific task, we can further improve the model's ability to generalize.
5. Access to State-of-the-Art Models: Transfer learning allows us to use state-of-the-art models that have been trained on large datasets, even if we do not have access to the same resources.

Reflection

Upon finishing the given tasks, I can confidently say that I have gained valuable insights and skills in building, training, analyzing, and modifying deep networks for a

recognition task. The project was designed in a way that helped me learn how to recognize digits using the MNIST dataset, which provided me with a good example of what deep networks can do.

This project was challenging, but it helped me gain the knowledge and skills I needed to build and train deep networks for a recognition task. This project has prepared me well for my final project, which I will be proposing and designing next.

Acknowledgement

[1] *OpenCV modules*. OpenCV. (n.d.). Retrieved February 10, 2023, from https://docs.opencv.org/4.x/

[2] https://stackoverflow.com/questions/43053923/replace-black-by-white-and-white-by-black-in-images

[3] https://pytorch.org/docs/stable/data.html#torch.utils.data.Dataset

[4] https://www.kaggle.com/datasets/katianakontolati/classification-of-handwritten-greek-letters?resource=download

[5] https://nextjournal.com/gkoehler/pytorch-mnist