Elec 475 Lab 1-MLP Autoencoder

Setareh Soltanieh

20370820

Dataset:

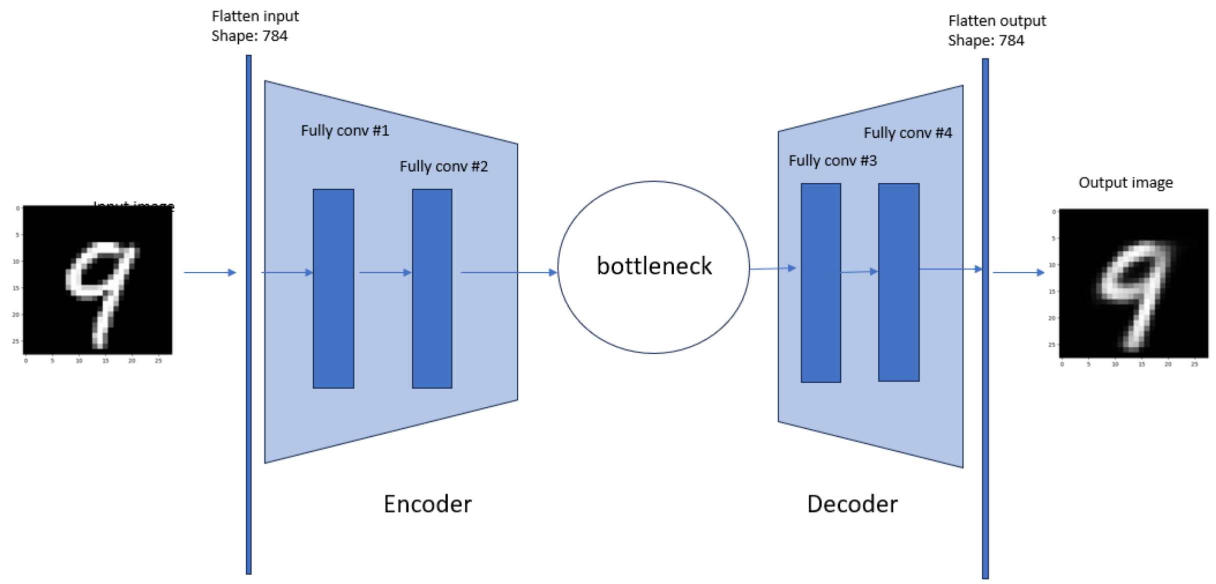
In this lab we are using MNIST dataset which is short for the 'Modified National Institute of Standards and Technology' dataset, is a well-known dataset in the field of machine learning and computer vision. It consists of a collection of grayscale images, each measuring 28x28 pixels. These images depict handwritten digits ranging from 0 to 9. Each image in the dataset is accompanied by a label that specifies the digit it represents. With a common split of 60,000 training images and 10,000 testing images.

Preprocessing:

We have employed a tensor transformation on our data to normalize our images, ensuring they are scaled within the range of 0 to 1.

Model:

The model that we have used for this dataset is an autoencoder. It will take a flatten input and it will apply a fully connected layer on the input and it will have the output of size 392 this fully connected is followed by a ReLU activation function, then we will apply another fully connected layer and we will have an output of 8 this is our bottleneck then we will apply a ReLU activation function. These two fully connected layers make our “Encoder” (They convert the input data to a smaller output which contains all the essential information about the input image.) Then, we will apply another fully connected layer that convert the 8 inputs to 392 followed by a ReLU activation function and then we will apply another fully connected layer with a sigmoid function this fully connected layer, converts the 392 input to 784 which is the same size as the input image. This part of our method is “Decoder”. Our model is shown in the following figure.



The summary of our model is as follows:

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Linear-1               [-1, 1, 392]         307,720
            Linear-2                 [-1, 1, 8]           3,144
            Linear-3               [-1, 1, 392]           3,528
            Linear-4               [-1, 1, 784]         308,112
================================================================
Total params: 622,504
Trainable params: 622,504
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 0.01
Params size (MB): 2.37
Estimated Total Size (MB): 2.39
----------------------------------------------------------------
```

This model has 622504 parameters that we need to train them.

Training Details

We have declared a train function that takes the following inputs:

n_epochs: The total number of epochs that we want to train the model. In this case it is 50.

Optimizer: We have an optimizer that will minimize the loss function by adjusting the model’s parameter. In this case our optimizer is Adam. Our learning rate is also defined here that is 1e-3.

Model: Our desired model. In this case it is autoencoderMLP4Layer()

loss_fn: For calculating the difference between the model’s output and its ground truth we need to have a loss function. In this function our loss function is Mean Square Error.

train_loader: We have loaded our training dataset which contains 60000 images into DataLoader function that will create batchs of data and it will also shuffle it.

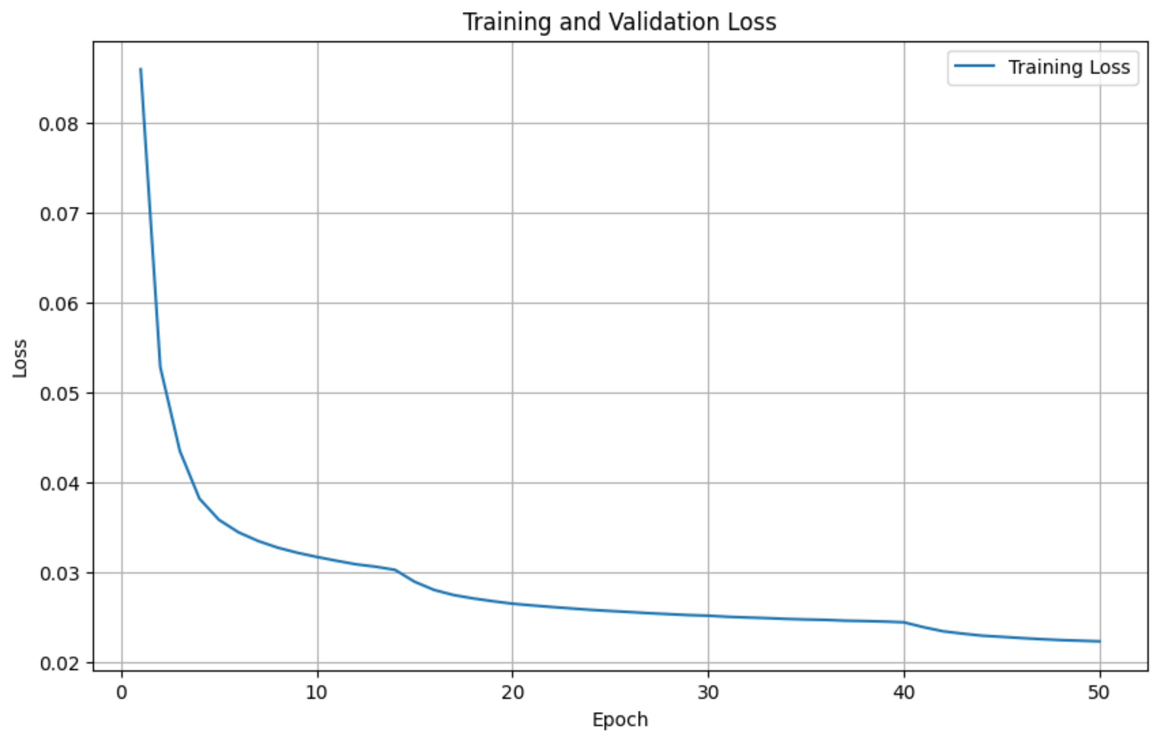
Scheduler: Our scheduler here is reduced on Plateau that will reduce the learning rate whenever the loss function haven’t changed for 5 epochs. It will reduce the learning rate by a weight decay of 1e-5.

Device: This component will demonstrate whether or not we have a GPU available or not.

In the training function we have a for loop that will iterate through each epochs, we have applied our model on each batch of data and calculate its loss and have updated the model’s weights. We have printed our loss for each epoch.

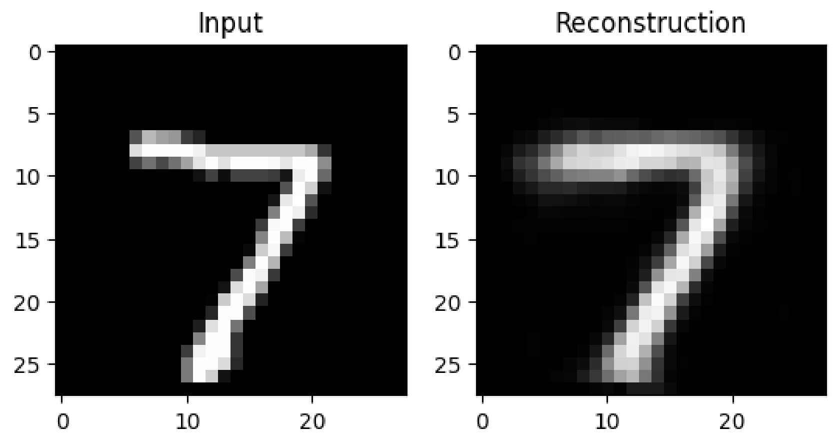
Results

The model was trained and worked well. The following is the training losses results.



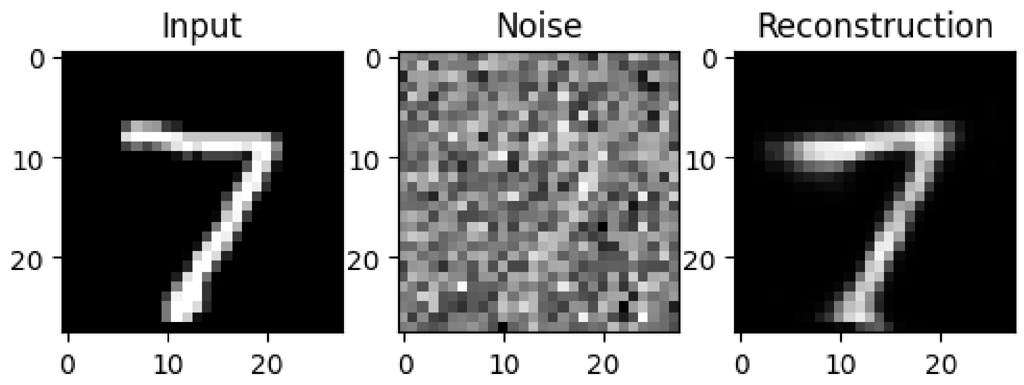
As it is demonstrated in the loss function's plot, our loss is decreasing and there are some bends in the plots one around the epoch 15 and one around the epoch 40. These bends can be demonstrated as the scheduler's functionality. On these epochs, scheduler has reduced the learning rate that will help the loss function to be opimized better.

Then, we have tested our model on our testing set. We have inputted a test image to our model and we have demonstrated the input and the resulted output in the following image.



As it is shown, this model has a very good performance, it has learned the number 7 and it can generate a new hand written digit.

Autoencoders can be used to remove image noise. We have applied a random noise to our images and then have applied the noisy images to the model and have its noise removed. In the following figure we have shown the input image and the noisy image and the output of the model.



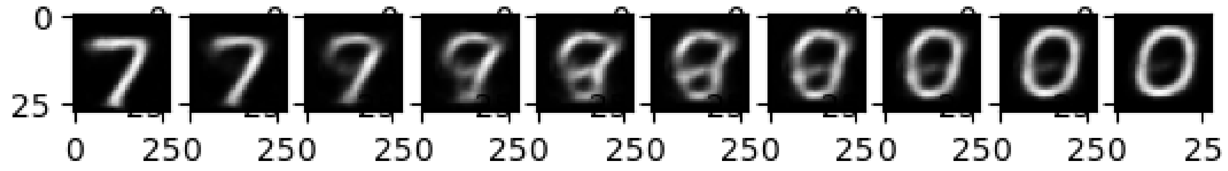
As it is shown, our model was successful in removing the noise from the noisy image and it can reconstruct the image.

Bottleneck Interpolation

In this section we have added two new methods to our model, encoder and decoder. We have talked about the details of encoder and decoder in the model section. In this section, we have defined a function that takes two images as input and calculate their bottlenech vector and have applied the following funciton on their bottleneck.

interpolated_bottleneck = bottleneck_1 + (i / n_interpolation) * (bottleneck_2 - bottleneck_1)

The previous function will take a weighted average from the input images and have new bottleneck. The number of calculated bottlenecks are as much as n_interpolations. When we have calculated the interpolated_bottleneck, we will pass it to our decoder, this will show our final image result. The following images are the results for 10 interpolations with input images of 7 and 0.



As it is shown, these images will look more a like their input image as it gets closser to them.