Computer Vision – Exercise 3

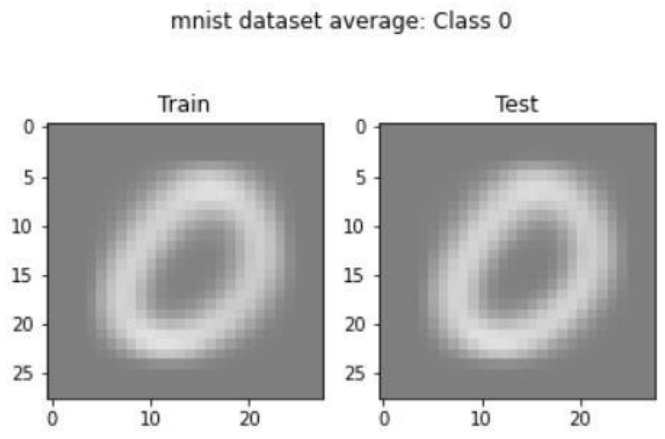
Kareem Jabareen - 211406343

Malik Egbaria - 318585627

Part 1:

- **Q1**:

  For this question, we calculated the average image for each class in MNIST both for the training dataset and the test one. After that we normalized the values in the averaged images by finding the minimum and the maximum values of each image.
  Then, using Matplotlib, we plotted the results as follows:



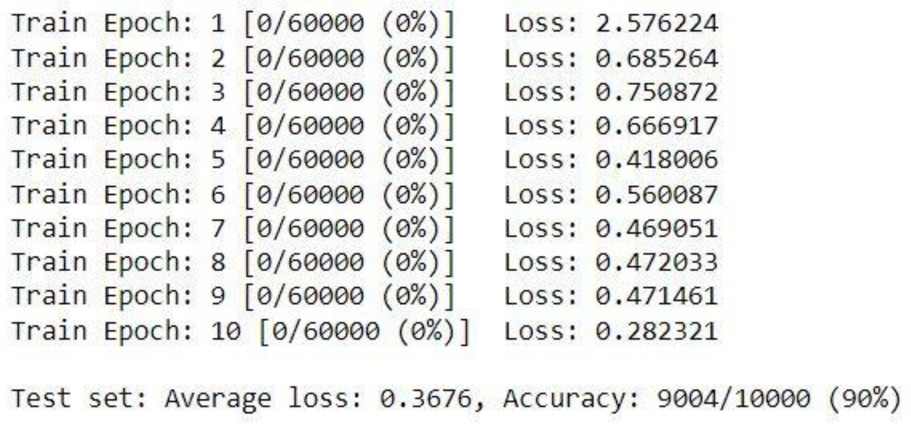
- **Q2**:

  A) Using the utilities of **Pytorch**, we created the **DataLoader** objects to load the data and process it, then we trained the CNN and plotted the results including the loss value. Eventually, we called the function called **test** to test our model and plotted its accuracy on the test dataset.

-

  After that we retrieved the weights from the network parameters and plotted them as normalized images.

B) We created a new method and called it **limited_train**, which reduced the number of training examples to just 50, then we used it for training our new model and got the following results:



```
Train Epoch: 1 [0/60000 (0%)]   Loss: 2.576224
Train Epoch: 2 [0/60000 (0%)]   Loss: 0.685264
Train Epoch: 3 [0/60000 (0%)]   Loss: 0.750872
Train Epoch: 4 [0/60000 (0%)]   Loss: 0.666917
Train Epoch: 5 [0/60000 (0%)]   Loss: 0.418006
Train Epoch: 6 [0/60000 (0%)]   Loss: 0.560087
Train Epoch: 7 [0/60000 (0%)]   Loss: 0.469051
Train Epoch: 8 [0/60000 (0%)]   Loss: 0.472033
Train Epoch: 9 [0/60000 (0%)]   Loss: 0.471461
Train Epoch: 10 [0/60000 (0%)]  Loss: 0.282321

Test set: Average loss: 0.3676, Accuracy: 9004/10000 (90%)
```

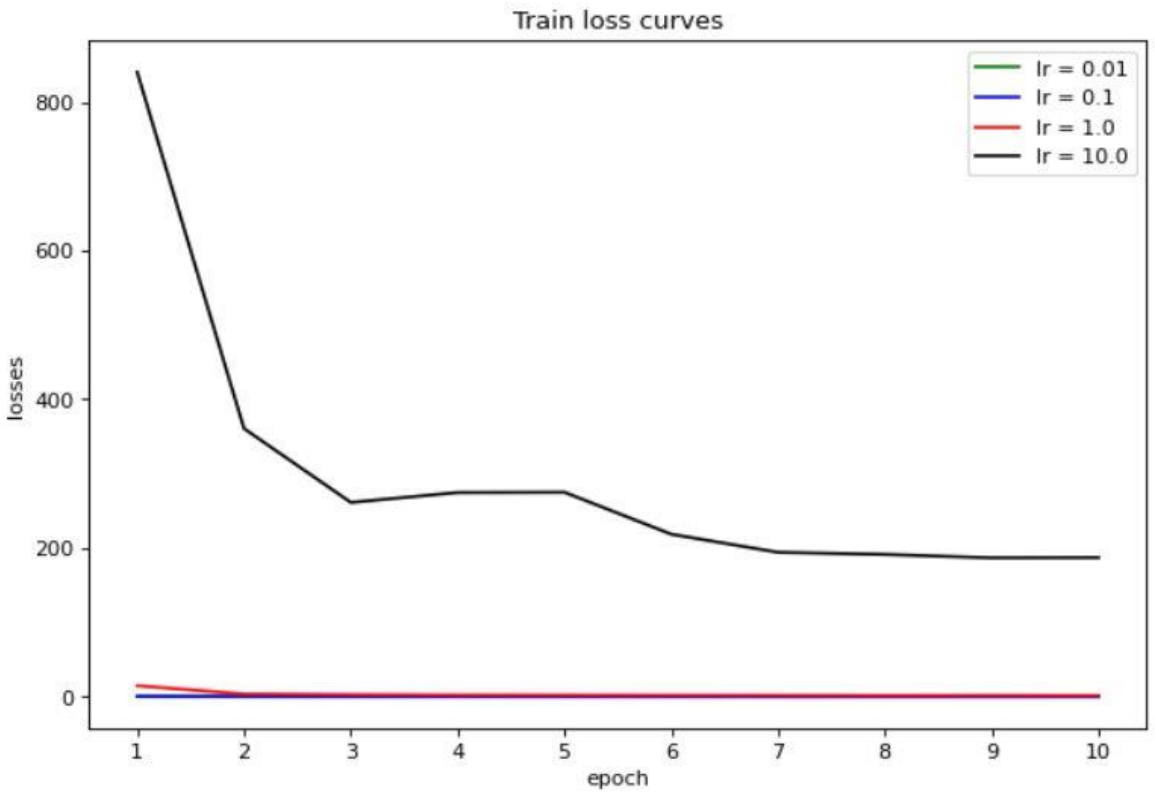
<u>Results demonstration</u>: Despite training our model on just 50 examples in each epoch, our model still managed to get similar accuracy to the one it got first when we trained it on full epochs. This shows that 60000 examples per epoch is more than needed for the model we are building.

- **Q3**:

  A) We defined a New Class called **MultiLayerNet** in which we added a new hidden linear layer with 1000 Units. In the Forward Function, we called the mentioned layer and applied the tanh non-linearity.

  B) We created a new Model using the mentioned Class above and trained it on MNIST four times with different learning rate each time [0.01, 0.1, 1.0, 10.0] and tested each one of them.

  Eventually, we Plotted the training loss curve for each learning rate and got the following results:
-

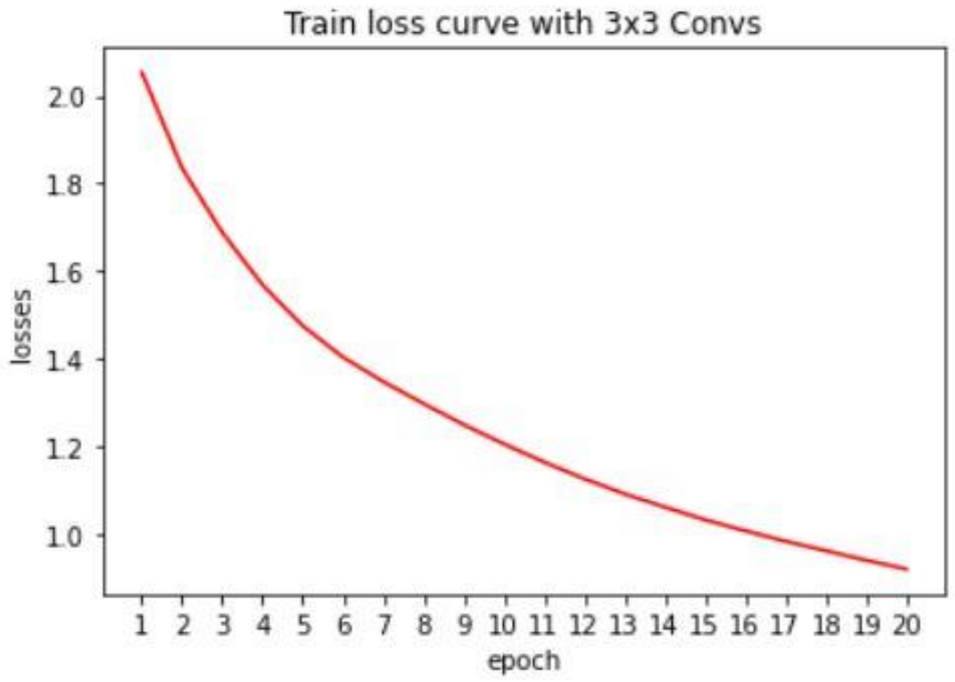


For the test accuracy, the learning rates 1.0, 0.1 and the 0.01 we got the best results and they were so close, that's because **outliers** have less impact in these models than in the other layers with great learning rate.

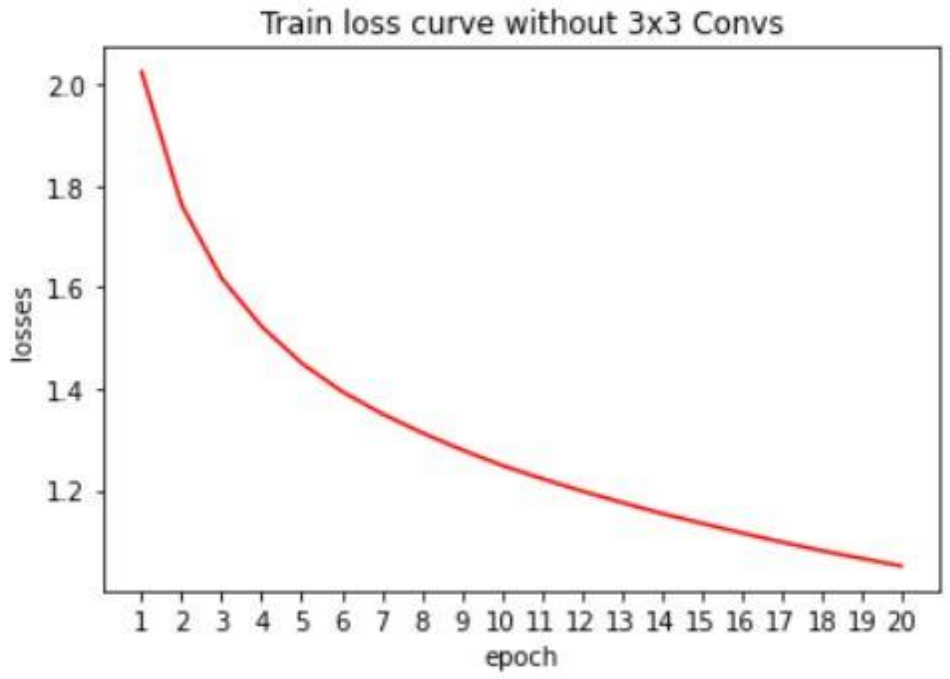
The same reason explains why we got terrible accuracy and high loss when we used learning rate of 10.0 (accuracy and loss values appear for each epoch in the notebook).

- **Q4**:

A) When we used the 3x3 convolution Layers, we got the following output:



However, when we dropped the 3x3 convolutional layers, we got the following output:



Explanation: We got similar loss curves and close test accuracy. When we used the convolutional layers, we got an accuracy of 62%, and we got an accuracy of 61% for the other model. However, we still can see that using the two convolutional layers improves the accuracy of the model, that is because these layers reshape the input so we can process data in larger dimensions.

B) For the model described in this question, we have the following parameters:

For the convolutional layers, the number of parameters depends on the kernel size, the input size, the output size, and the bias.

1. First convolutional layer:

   Parameters = (5 * 5 * 3 + 1) * 16 = 76*16 = 1216

2. Second convolutional layer:

   Parameters = (3 * 3 * 16 + 1) * 64 = 145 * 64 = 9280

3. Last convolutional layer:

   Parameters = (3 * 3 * 64 + 1) * 64 = 577 * 64 = 36928

For pooling layers and flatten layers there is no parameter you could learn, so we get:

Pooling layer parameters = Flatten layer parameters = 0

For the linear layers:

1. First linear layer:

   Parameters = (#inputs + 1) * #outputs (we add one for the bias).

   ⇨ Parameters = (1600 + 1) * 64 = 1601 * 64 = 102464

2. Last linear layer:

   In a similar way to the first linear layer parameters, we get:

   Parameters = (64 + 1) * 10 = 65 * 10 = 650

#Parameters in the whole model = 1216 + 9280 + 36928 + 102464 + 650 = 150538.

Part 2:

For this part, we added the followings to the base code:

1. Adam optimizer for getting better accuracy.
2. Three more convolutional layers in the Net class for performance improvement.

In addition, we modified some of the **hyper-parameters**:

1. Increased the number of epochs for getting better accuracy.
2. Reduced the learning rate to avoid overfitting.
3. Changed the crop size.
4. Increased the dim feature.
5. Changed the interpolation mode to **bicubic**.

More explanation on our decisions exists in the **notebook for the second part**.