Project 4: Introduction to Deep Learning

Submitted by : Gmon Kuzhiyanikkal
NU ID : 002724506

INTRODUCTION:

The purpose of this project is to introduce the tools of deep learning for visual and non-visual tasks. Initially, I was able to build a convolutional neural network for the MNIST digit recognition task. The report for this task should include a depiction of the network, a plot showing at least nine example outputs and their labels, and a plot of the training and testing error during training.

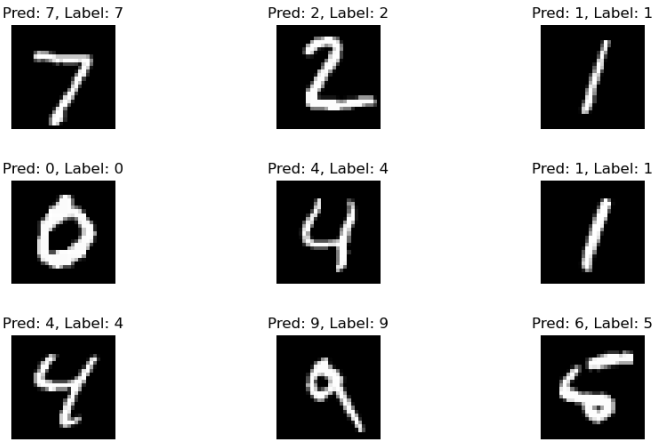
Secondly, I have to undertake some experimentation with the deep network for the MNIST task. The goal is to evaluate the effect of changing different aspects of the network, such as the number of convolution layers, the size of the convolution filters, and the activation function for each layer. I have tried to improve the accuracy of the network by varying different parameter.In addition to that, I have reused the MNIST digit recognition network built in step one to recognize three different Greek letters: alpha, beta, and gamma.

I have built two different networks to predict heart disease using the same data set from project 3. I have tried to compare the performance of those two models and a discussion of the network architectures, their relative computational requirements.As a part of first extensions, I have tried to build 4 more models with the same heart dates and tried to compare the accuracy and performance of the model.

As part of the second extension, I have tried to create and explore networks with the wine dataset. I have tried to compare the accuracy of different models with different numbers of interactions or epochs and come up with a better model.

1 MNIST Tutorial:

I have implemented the MNIST tutorial and after running the model, I was able to achieve a 99% accuracy from the model. A plot showing at least nine example outputs and their labels as generated by your network is given below:



From the figure, expect the last image, all the images are predicted correctly. I have trained the model with 10 epochs. The loss in the each epochs are given below:

```
Train Epoch: 1 [0/60000 (0%)]    Loss: 2.296739
Train Epoch: 1 [6400/60000 (11%)]        Loss: 0.241506
Train Epoch: 1 [12800/60000 (21%)]       Loss: 0.188616
Train Epoch: 1 [19200/60000 (32%)]       Loss: 0.235406
Train Epoch: 1 [25600/60000 (43%)]       Loss: 0.156336
Train Epoch: 1 [32000/60000 (53%)]       Loss: 0.040955
Train Epoch: 1 [38400/60000 (64%)]       Loss: 0.014988
Train Epoch: 1 [44800/60000 (75%)]       Loss: 0.105042
Train Epoch: 1 [51200/60000 (85%)]       Loss: 0.026415
Train Epoch: 1 [57600/60000 (96%)]       Loss: 0.115929
Test set: Average loss: 0.0008, Accuracy: 9833/10000 (98%)

Train Epoch: 2 [0/60000 (0%)]    Loss: 0.053834
Train Epoch: 2 [6400/60000 (11%)]        Loss: 0.042155
Train Epoch: 2 [12800/60000 (21%)]       Loss: 0.052861
Train Epoch: 2 [19200/60000 (32%)]       Loss: 0.046437
Train Epoch: 2 [25600/60000 (43%)]       Loss: 0.154088
Train Epoch: 2 [32000/60000 (53%)]       Loss: 0.141049
Train Epoch: 2 [38400/60000 (64%)]       Loss: 0.087820
Train Epoch: 2 [44800/60000 (75%)]       Loss: 0.038994
Train Epoch: 2 [51200/60000 (85%)]       Loss: 0.059565
Train Epoch: 2 [57600/60000 (96%)]       Loss: 0.183165
Test set: Average loss: 0.0005, Accuracy: 9894/10000 (99%)

Train Epoch: 3 [0/60000 (0%)]    Loss: 0.040525
Train Epoch: 3 [6400/60000 (11%)]        Loss: 0.217407
Train Epoch: 3 [12800/60000 (21%)]       Loss: 0.057605
Train Epoch: 3 [19200/60000 (32%)]       Loss: 0.015172
Train Epoch: 3 [25600/60000 (43%)]       Loss: 0.014660
Train Epoch: 3 [32000/60000 (53%)]       Loss: 0.151753
Train Epoch: 3 [38400/60000 (64%)]       Loss: 0.121777
Train Epoch: 3 [44800/60000 (75%)]       Loss: 0.221690
Train Epoch: 3 [51200/60000 (85%)]       Loss: 0.191037
Train Epoch: 3 [57600/60000 (96%)]       Loss: 0.005433
Test set: Average loss: 0.0004, Accuracy: 9913/10000 (99%)
```

Train Epoch: 4 [0/60000 (0%)] Loss: 0.002924
Train Epoch: 4 [6400/60000 (11%)] Loss: 0.094755
Train Epoch: 4 [12800/60000 (21%)] Loss: 0.008768
Train Epoch: 4 [19200/60000 (32%)] Loss: 0.047505
Train Epoch: 4 [25600/60000 (43%)] Loss: 0.232998
Train Epoch: 4 [32000/60000 (53%)] Loss: 0.012868
Train Epoch: 4 [38400/60000 (64%)] Loss: 0.020838
Train Epoch: 4 [44800/60000 (75%)] Loss: 0.052229
Train Epoch: 4 [51200/60000 (85%)] Loss: 0.021965
Train Epoch: 4 [57600/60000 (96%)] Loss: 0.011334
Test set: Average loss: 0.0006, Accuracy: 9889/10000 (99%)

Train Epoch: 5 [0/60000 (0%)] Loss: 0.078027
Train Epoch: 5 [6400/60000 (11%)] Loss: 0.057481
Train Epoch: 5 [12800/60000 (21%)] Loss: 0.003735
Train Epoch: 5 [19200/60000 (32%)] Loss: 0.018742
Train Epoch: 5 [25600/60000 (43%)] Loss: 0.030598
Train Epoch: 5 [32000/60000 (53%)] Loss: 0.024907
Train Epoch: 5 [38400/60000 (64%)] Loss: 0.028470
Train Epoch: 5 [44800/60000 (75%)] Loss: 0.005778
Train Epoch: 5 [51200/60000 (85%)] Loss: 0.029744
Train Epoch: 5 [57600/60000 (96%)] Loss: 0.058500
Test set: Average loss: 0.0004, Accuracy: 9923/10000 (99%)

Train Epoch: 6 [0/60000 (0%)] Loss: 0.005848
Train Epoch: 6 [6400/60000 (11%)] Loss: 0.003285
Train Epoch: 6 [12800/60000 (21%)] Loss: 0.011501
Train Epoch: 6 [19200/60000 (32%)] Loss: 0.003402
Train Epoch: 6 [25600/60000 (43%)] Loss: 0.015848
Train Epoch: 6 [32000/60000 (53%)] Loss: 0.017931
Train Epoch: 6 [38400/60000 (64%)] Loss: 0.024691
Train Epoch: 6 [44800/60000 (75%)] Loss: 0.001660
Train Epoch: 6 [51200/60000 (85%)] Loss: 0.002579
Train Epoch: 6 [57600/60000 (96%)] Loss: 0.052931
Test set: Average loss: 0.0004, Accuracy: 9921/10000 (99%)

Train Epoch: 7 [0/60000 (0%)] Loss: 0.017654
Train Epoch: 7 [6400/60000 (11%)] Loss: 0.022920
Train Epoch: 7 [12800/60000 (21%)] Loss: 0.001074
Train Epoch: 7 [19200/60000 (32%)] Loss: 0.004729
Train Epoch: 7 [25600/60000 (43%)] Loss: 0.087334
Train Epoch: 7 [32000/60000 (53%)] Loss: 0.065466
Train Epoch: 7 [38400/60000 (64%)] Loss: 0.045613
Train Epoch: 7 [44800/60000 (75%)] Loss: 0.046889
Train Epoch: 7 [51200/60000 (85%)] Loss: 0.012502
Train Epoch: 7 [57600/60000 (96%)] Loss: 0.012244
Test set: Average loss: 0.0004, Accuracy: 9931/10000 (99%)

Train Epoch: 8 [0/60000 (0%)] Loss: 0.001164
Train Epoch: 8 [6400/60000 (11%)] Loss: 0.041305
Train Epoch: 8 [12800/60000 (21%)] Loss: 0.024487
Train Epoch: 8 [19200/60000 (32%)] Loss: 0.066766
Train Epoch: 8 [25600/60000 (43%)] Loss: 0.012525
Train Epoch: 8 [32000/60000 (53%)] Loss: 0.047772
Train Epoch: 8 [38400/60000 (64%)] Loss: 0.007660
Train Epoch: 8 [44800/60000 (75%)] Loss: 0.003227
Train Epoch: 8 [51200/60000 (85%)] Loss: 0.000807
Train Epoch: 8 [57600/60000 (96%)] Loss: 0.008013
Test set: Average loss: 0.0004, Accuracy: 9915/10000 (99%)
Train Epoch: 9 [0/60000 (0%)] Loss: 0.007664
Train Epoch: 9 [6400/60000 (11%)] Loss: 0.010291
Train Epoch: 9 [12800/60000 (21%)] Loss: 0.019112
Train Epoch: 9 [19200/60000 (32%)] Loss: 0.079929
Train Epoch: 9 [25600/60000 (43%)] Loss: 0.081415
Train Epoch: 9 [32000/60000 (53%)] Loss: 0.000420
Train Epoch: 9 [38400/60000 (64%)] Loss: 0.025371
Train Epoch: 9 [44800/60000 (75%)] Loss: 0.002754
Train Epoch: 9 [51200/60000 (85%)] Loss: 0.133592
Train Epoch: 9 [57600/60000 (96%)] Loss: 0.001290

Test set: Average loss: 0.0005, Accuracy: 9918/10000 (99%)
Train Epoch: 10 [0/60000 (0%)] Loss: 0.070100
Train Epoch: 10 [6400/60000 (11%)] Loss: 0.086580
Train Epoch: 10 [12800/60000 (21%)] Loss: 0.038342
Train Epoch: 10 [19200/60000 (32%)] Loss: 0.005987
Train Epoch: 10 [25600/60000 (43%)] Loss: 0.022172
Train Epoch: 10 [32000/60000 (53%)] Loss: 0.065338
Train Epoch: 10 [38400/60000 (64%)] Loss: 0.002396
Train Epoch: 10 [44800/60000 (75%)] Loss: 0.001664

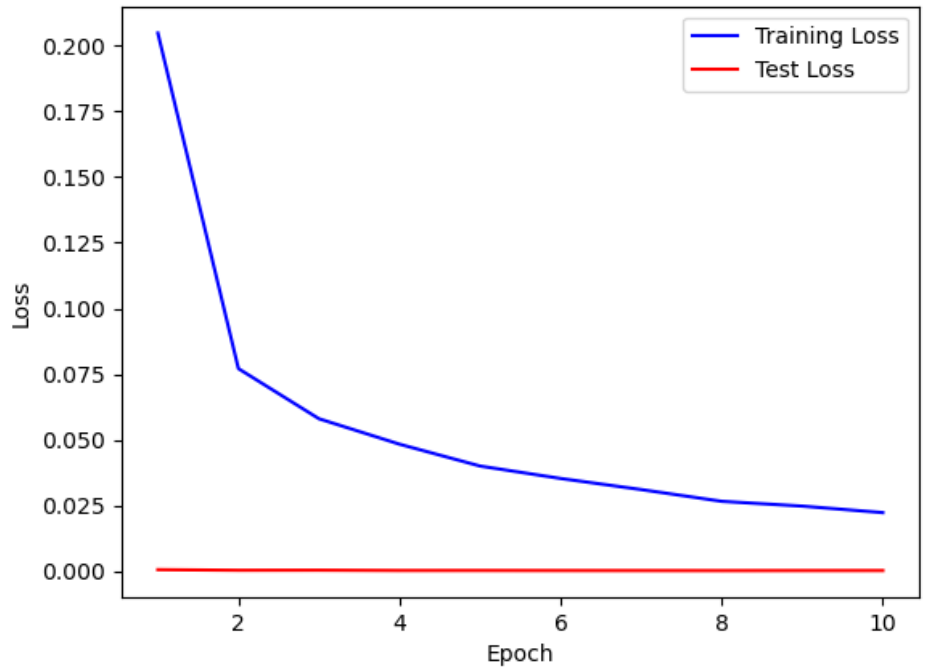
Train Epoch: 10 [51200/60000 (85%)] Loss: 0.001912
Train Epoch: 10 [57600/60000 (96%)] Loss: 0.008794

The depiction of the network is printed out as below:

```
Train Epoch: 10 [44800/60000 (75%)]     Loss: 0.001664
Train Epoch: 10 [51200/60000 (85%)]     Loss: 0.001912
Train Epoch: 10 [57600/60000 (96%)]     Loss: 0.008794
Test set: Average loss: 0.0004, Accuracy: 9924/10000 (99%)

Net(
  (conv1): Conv2d(1, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv2): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (fc1): Linear(in_features=3136, out_features=128, bias=True)
  (fc2): Linear(in_features=128, out_features=10, bias=True)
  (dropout): Dropout(p=0.5, inplace=False)
)
(base) gmonk@Gmons-MacBook-Air PM4 %
```

I have tried to plot the training and testing loss. I could find out that the training loss has reduced as the number of the epochs has increased. The plot is given below:



PS: python program to generate this is output is ‘1.py’

2.Experiment with Network Variations:

I have taken MNIST Fashion data set instead of the digits data set and the Three dimensions that i have chosen for experimenting with the networks are:

1. Varying the number of epochs of training
2. The number of convolution layers
3. The activation function for each layer

A.Develop a plan:

I have used the linear search strategy for the evaluation and I have kept one of the dimensions to change, while keeping all the other dimensions as constant. I have tried to print out the accuracy of the network in each of the cases. Their prediction and the final result of the implementation are given below.

B.Predict the results:

I could predict the following for each of the dimensions and it given below:

1.Varying the number of epochs of training:

Hypothesis: Increasing the number of epochs will result in higher accuracy, but may also increase the risk of overfitting and increase training time.I have tried for different epoch ranges that are from 5,10,25,50 and 100. I have kept the other dimension as constant.

2.The number of convolution layers:

Hypothesis: As we increase the number of convolution layers, the model should be able to learn more complex features in the images, resulting in improved accuracy. I have created additional convolution layers and I will keep the number of epochs constant at 10 and use the ReLU activation function in all layers.

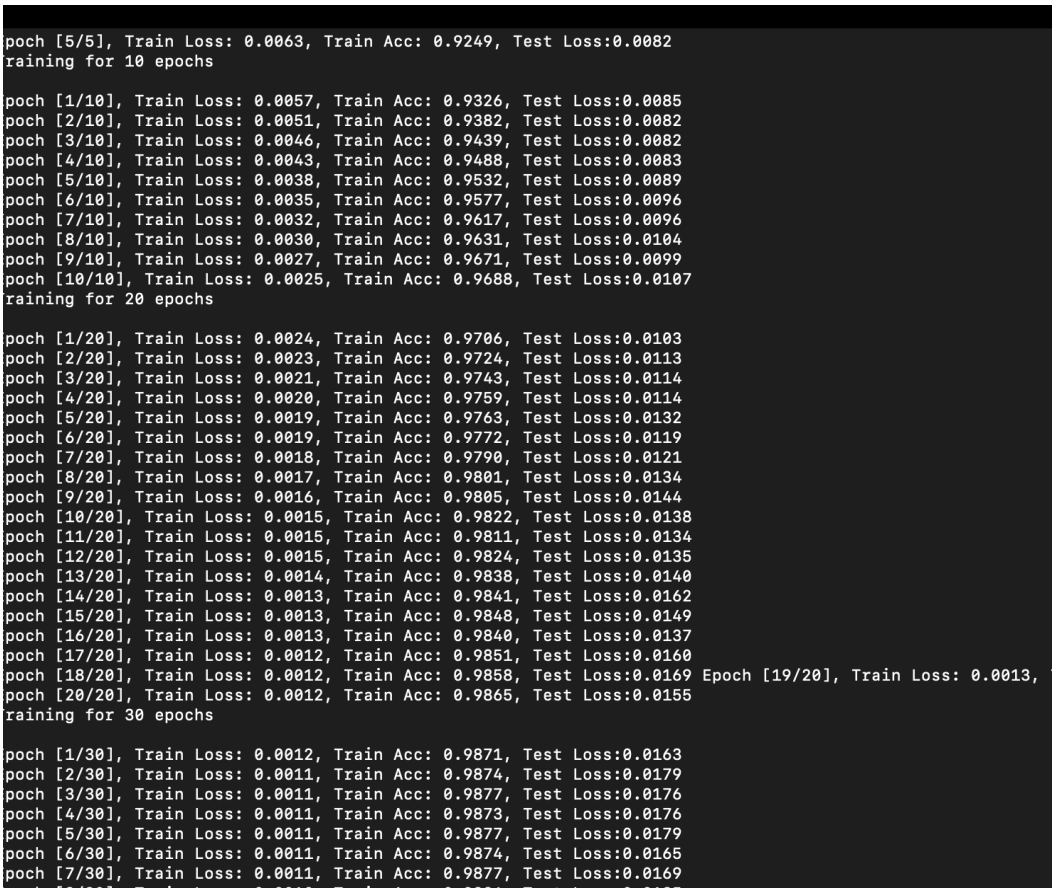
3.The activation function for each layer:

Hypothesis: Using a nonlinear activation function like ReLU should result in better accuracy compared to a linear activation function like linear. I have created 3 variations of the model with different activation functions for all layers. I have kept the number of epochs constant at 10 and use 3 convolution layers.

C.Execute your plan:

1.Varying the number of epochs of training:

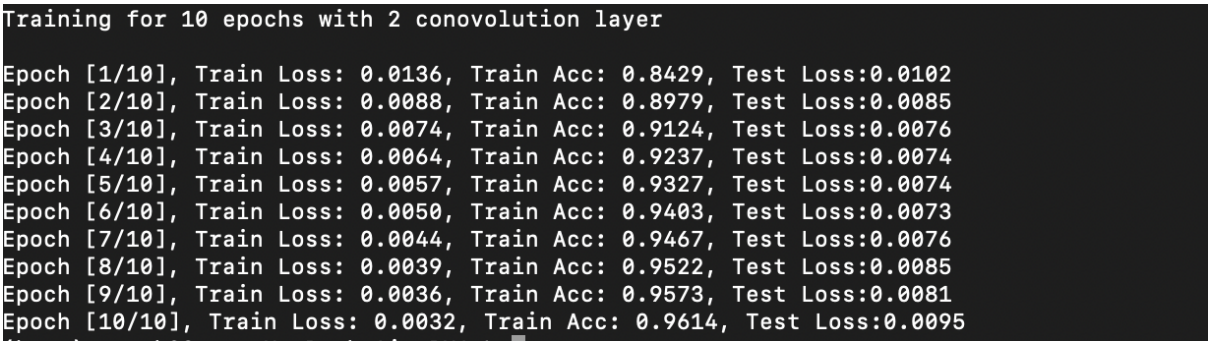
As predicted the accuracy of the model has increased from 84% to 98%, even when increasing the epoch to 30, I have coded the program till 100 epochs, i.1(5,10,25,30,50,100), but the time taken to train this model has increased significantly for higher epoch value. I have kept the use of the ReLU activation function in all layers and also I have used 1 convolution layer. The output of the program along with the accuracy is given below:



PS: python program to run this is python3 2a.py

2.The number of convolution layers:

In this method, I have tried to add 2 convolution layers, keeping the epoch to 10 the ReLU activation function in all layers. I tried to print the accuracy of the model in each of the convolution layers. The output of the program is given as below:



From the hypothesis, we can say that the accuracy has increased significantly. Now the model is able to learn more complex features in the images.

PS; python program to run this is python3 2b.py

3.The activation function for each layer:

I have kept the epoch to 10 and the number of convolution layers to 1. In order to vary the activation function, i have used the ReLU, sigmoid, tanh activation function for the experiment and the output of the accuracy for each of these activation function are given below:

```
Activation function: relu

 Epoch: 1, Train loss: 1.6803631630025184, Train accuracy: 0.7932, Test loss: 1.6331657563583761, Test accuracy: 0.8329
 Epoch: 2, Train loss: 1.6080850886383544, Train accuracy: 0.85735, Test loss: 1.602664460109759, Test accuracy: 0.8629
 Epoch: 3, Train loss: 1.59050918286289, Train accuracy: 0.8748333333333334, Test loss: 1.5924628218518029, Test accuracy: 0.8706
 Epoch: 4, Train loss: 1.5762396893267439, Train accuracy: 0.88805, Test loss: 1.5816354283803626, Test accuracy: 0.8807
 Epoch: 5, Train loss: 1.5690268280663724, Train accuracy: 0.89465, Test loss: 1.575139015535765, Test accuracy: 0.8872
 Epoch: 6, Train loss: 1.5623151928122871, Train accuracy: 0.9013166666666667, Test loss: 1.5783764516250998, Test accuracy: 0.8831
 Epoch: 7, Train loss: 1.5559906740940965, Train accuracy: 0.9079333333333334, Test loss: 1.5679937797256662, Test accuracy: 0.8938
 Epoch: 8, Train loss: 1.5500060319900513, Train accuracy: 0.9136333333333333, Test loss: 1.5711599799651135, Test accuracy: 0.8903
 Epoch: 9, Train loss: 1.547442600162807, Train accuracy: 0.9159333333333334, Test loss: 1.5665109233011174, Test accuracy: 0.8965
 Epoch: 10, Train loss: 1.542420938833436, Train accuracy: 0.9213333333333333, Test loss: 1.559439238113693, Test accuracy: 0.9021

Activation function: sigmoid

 Epoch: 1, Train loss: 1.6916519741513836, Train accuracy: 0.7806166666666666, Test loss: 1.628048919424226, Test accuracy: 0.8366
 Epoch: 2, Train loss: 1.6013981030185593, Train accuracy: 0.8638833333333333, Test loss: 1.5985888152182857, Test accuracy: 0.8658
 Epoch: 3, Train loss: 1.5846179322139033, Train accuracy: 0.8798166666666667, Test loss: 1.5860071634944481, Test accuracy: 0.8783
 Epoch: 4, Train loss: 1.5728129898307166, Train accuracy: 0.8912333333333333, Test loss: 1.57909994487521, Test accuracy: 0.885
 Epoch: 5, Train loss: 1.563305750330374, Train accuracy: 0.9006, Test loss: 1.582621144343026, Test accuracy: 0.8824
 Epoch: 6, Train loss: 1.5569867975930416, Train accuracy: 0.9069333333333334, Test loss: 1.5712147860587398, Test accuracy: 0.892
 Epoch: 7, Train loss: 1.5506565626750368, Train accuracy: 0.9129333333333334, Test loss: 1.5636078677599943, Test accuracy: 0.8984
 Epoch: 8, Train loss: 1.545641140642959, Train accuracy: 0.9176666666666666, Test loss: 1.5648826722857319, Test accuracy: 0.8978
 Epoch: 9, Train loss: 1.5412988464461206, Train accuracy: 0.9223166666666667, Test loss: 1.5665721983849248, Test accuracy: 0.8964
 Epoch: 10, Train loss: 1.537095388115596 2, Train accuracy: 0.9266, Test loss: 1.560930208314823, Test accuracy: 0.9004

Activation function: tanh

 Epoch: 1, Train loss: 1.7503397815517272, Train accuracy: 0.7191666666666666, Test loss: 1.709948423542554, Test accuracy: 0.7525
 Epoch: 2, Train loss: 1.6790050085165353, Train accuracy: 0.7847833333333334, Test loss: 1.6721242304089703, Test accuracy: 0.7911
 Epoch: 3, Train loss: 1.6357976076191167, Train accuracy: 0.8288166666666666, Test loss: 1.588866937009594, Test accuracy: 0.8761
 Epoch: 4, Train loss: 1.5744057225265 99, Train accuracy: 0.89095, Test loss: 1.5808914100067526, Test accuracy: 0.8823
 Epoch: 5, Train loss: 1.5655718475008316, Train accuracy: 0.8985666666666666, Test loss: 1.571438197848163, Test accuracy: 0.8924
 Epoch: 6, Train loss: 1.5590630391005005, Train accuracy: 0.9050666666666667, Test loss: 1.5684021364284466, Test accuracy: 0.894
 Epoch: 7, Train loss: 1.5543422861648266, Train accuracy: 0.9092833333333333, Test loss: 1.569276853452755, Test accuracy: 0.8942
 Epoch: 8, Train loss: 1.5498278690045322, Train accuracy: 0.91405, Test loss: 1.5625142523005038, Test accuracy: 0.9001
 Epoch: 9, Train loss: 1.5447887530459015, Train accuracy: 0.9183833333333333, Test loss: 1.5650143668621401, Test accuracy: 0.8973
 Epoch: 10, Train loss: 1.54013520085227, Train accuracy: 0.9229166666666667, Test loss: 1.558812574495243, Test accuracy: 0.9039
(base) gmonk@Gmons-MacBook-Air PM4 %
```

We could see that at epoch 10, sigmoid function has higher training accuracy than Relu activation function, but the testing accuracy is better for the Relu activation function at epoch 10, thus we can say that hypothesis that i have predicted is true,Using a nonlinear activation function like ReLU should result in better accuracy compared to a linear activation function like linear.

PS: python program to run this is python3 2c.py

3)Transfer Learning on Greek Letters:

I have Load the pre-trained weights into the network and Freeze the network weights. In addition to that, I have replaced the last layer with a new Linear layer with three nodes. The modified network is printed below:

```
Net(
  (conv1): Conv2d(1, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv2): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (fc1): Linear(in_features=3136, out_features=128, bias=True)
  (fc2): Linear(in_features=128, out_features=3, bias=True)
  (dropout): Dropout(p=0.5, inplace=False)
)
```

I have used 5 epochs to get better accuracy. If I am increasing the epoch the prediction rate is getting worse. The output of the loss in epochs are given below:

Train Epoch: 1 [0/60000 (0%)] Loss: 2.310515
Train Epoch: 1 [6400/60000 (11%)] Loss: 0.245814
Train Epoch: 1 [12800/60000 (21%)] Loss: 0.311361
Train Epoch: 1 [19200/60000 (32%)] Loss: 0.228982
Train Epoch: 1 [25600/60000 (43%)] Loss: 0.260466
Train Epoch: 1 [32000/60000 (53%)] Loss: 0.060529
Train Epoch: 1 [38400/60000 (64%)] Loss: 0.159612
Train Epoch: 1 [44800/60000 (75%)] Loss: 0.075302
Train Epoch: 1 [51200/60000 (85%)] Loss: 0.061980
Train Epoch: 1 [57600/60000 (96%)] Loss: 0.246970

Train Epoch: 2 [0/60000 (0%)] Loss: 0.141240
Train Epoch: 2 [6400/60000 (11%)] Loss: 0.140599
Train Epoch: 2 [12800/60000 (21%)] Loss: 0.029053
Train Epoch: 2 [19200/60000 (32%)] Loss: 0.129817
Train Epoch: 2 [25600/60000 (43%)] Loss: 0.039444
Train Epoch: 2 [32000/60000 (53%)] Loss: 0.214712
Train Epoch: 2 [38400/60000 (64%)] Loss: 0.068680
Train Epoch: 2 [44800/60000 (75%)] Loss: 0.066766
Train Epoch: 2 [51200/60000 (85%)] Loss: 0.131892
Train Epoch: 2 [57600/60000 (96%)] Loss: 0.120739

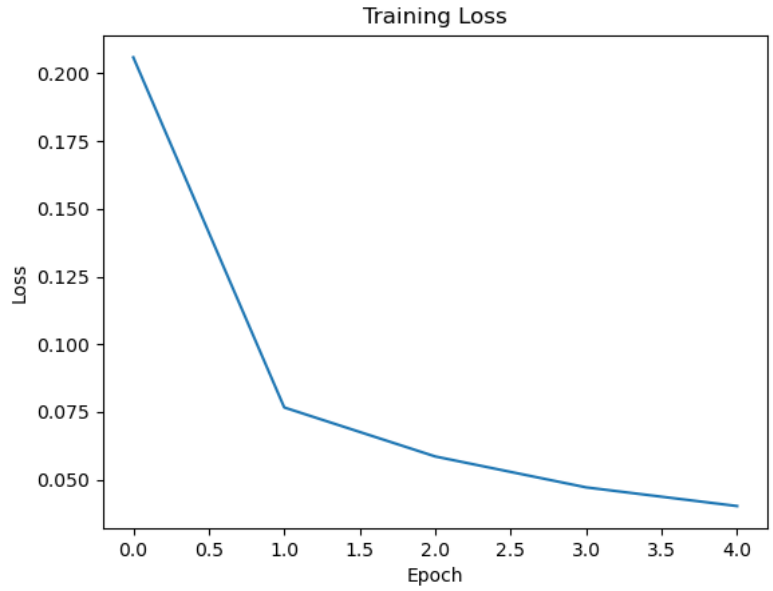
Train Epoch: 3 [0/60000 (0%)] Loss: 0.177079
Train Epoch: 3 [6400/60000 (11%)] Loss: 0.015323
Train Epoch: 3 [12800/60000 (21%)] Loss: 0.085086
Train Epoch: 3 [19200/60000 (32%)] Loss: 0.071666
Train Epoch: 3 [25600/60000 (43%)] Loss: 0.026826
Train Epoch: 3 [32000/60000 (53%)] Loss: 0.107801
Train Epoch: 3 [38400/60000 (64%)] Loss: 0.084415
Train Epoch: 3 [44800/60000 (75%)] Loss: 0.035397
Train Epoch: 3 [51200/60000 (85%)] Loss: 0.068085
Train Epoch: 3 [57600/60000 (96%)] Loss: 0.045404

Train Epoch: 4 [0/60000 (0%)] Loss: 0.016249
Train Epoch: 4 [6400/60000 (11%)] Loss: 0.024045
Train Epoch: 4 [12800/60000 (21%)] Loss: 0.042912
Train Epoch: 4 [19200/60000 (32%)] Loss: 0.034178
Train Epoch: 4 [25600/60000 (43%)] Loss: 0.027417
Train Epoch: 4 [32000/60000 (53%)] Loss: 0.019944
Train Epoch: 4 [38400/60000 (64%)] Loss: 0.037909
Train Epoch: 4 [44800/60000 (75%)] Loss: 0.064628
Train Epoch: 4 [51200/60000 (85%)] Loss: 0.004067
Train Epoch: 4 [57600/60000 (96%)] Loss: 0.005707

Train Epoch: 5 [0/60000 (0%)] Loss: 0.023213

```
Train Epoch: 5 [6400/60000 (11%)]       Loss: 0.114275
Train Epoch: 5 [12800/60000 (21%)]      Loss: 0.006750
Train Epoch: 5 [19200/60000 (32%)]      Loss: 0.032771
Train Epoch: 5 [25600/60000 (43%)]      Loss: 0.017058
Train Epoch: 5 [32000/60000 (53%)]      Loss: 0.072814
Train Epoch: 5 [38400/60000 (64%)]      Loss: 0.049530
Train Epoch: 5 [44800/60000 (75%)]      Loss: 0.016427
Train Epoch: 5 [51200/60000 (85%)]      Loss: 0.028952
Train Epoch: 5 [57600/60000 (96%)]      Loss: 0.002727
```

In addition to that, I have tried to plot the training error with the epoch and i could see that the training loss has reduced significantly when the epoch is in the 5th iteration. the diagram is given below:



I have tried to predict the output with the greek letter and the output from the program is given below:

```
[W NNPACK.cpp:53] Could not initialize NNPACK! Reason: Unsupported hardwa
Predicted class: 0, Greek letter: alpha
Predicted class: 0, Greek letter: alpha
Predicted class: 0, Greek letter: alpha
Predicted class: 0, Greek letter: alpha
Predicted class: 0, Greek letter: alpha
Predicted class: 0, Greek letter: alpha
Predicted class: 0, Greek letter: alpha
Predicted class: 0, Greek letter: alpha
Predicted class: 0, Greek letter: alpha
Predicted class: 1, Greek letter: beta
Predicted class: 1, Greek letter: beta
Predicted class: 0, Greek letter: alpha
Predicted class: 0, Greek letter: alpha
```

PS: python program to run this is python3 3.py

4. Heart Disease Prediction Using an ANN:

Two models are created to predict the presence of heart disease in patients based on a set of features.the first model has two hidden layers with 32 and 16 neurons, respectively. The second model has four hidden layers with 64, 32, 16, and 8 neurons, respectively. Both models use the rectified linear unit (ReLU) activation function for their hidden layers and the sigmoid activation function for their output layer.The models are trained using the Adam optimizer and the binary cross-entropy loss function.

Model 2 has more layers and more neurons, which makes it more computationally expensive than Model 1. Therefore, in terms of computational requirements, Model 2 is likely to take more time and resources to train than Model 1. However, the additional layers and neurons in Model 2 may lead to better performance and higher accuracy on the test data.

In terms of the network architecture, the first model is relatively simple with only two hidden layers. The second model is more complex with four hidden layers, which may allow it to capture more complex relationships between the input features and the target variable.

I have tried to vary the number of the epochs for both the models, when epoch is 100, model 1 has shown a higher accuracy. I am still not able to achieve a better accuracy from project 3. If I increase the number of iterations and add more convolution networks, I will be able to achieve it. The output of this program is given below:

```
models with 100 iterations

Model 1 Test accuracy: 0.8369565217391305
Model 2 Test accuracy: 0.8260869565217391

models with 50 iterations

Model 1 Test accuracy: 0.7717391304347826
Model 2 Test accuracy: 0.7934782608695652

models with 10 iterations

Model 1 Test accuracy: 0.625
Model 2 Test accuracy: 0.5815217391304348
(base) gmonk@Gmons-MacBook-Air PM4 %
```

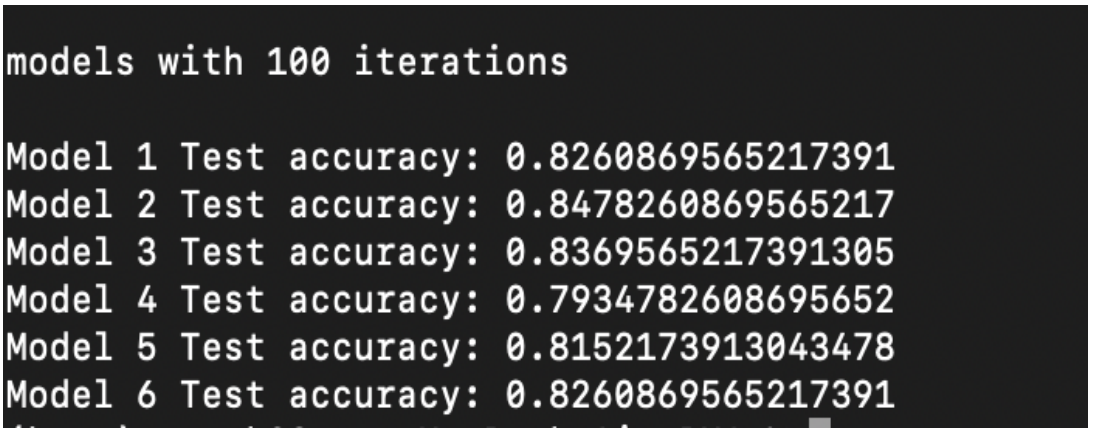
PS: program to run this python3 4.py

1)Extensions:

This is my first extension in this project. I have tried to explore more architectures for the Heart Disease data set. Some of the new model that i have created are given below:

1. Model with 3 hidden layers:
    - hidden_layer_sizes=(64, 32, 16)
    - activation='relu'
    - max_iter=100
2. Model with 4 hidden layers:
    - hidden_layer_sizes=(64, 32, 16, 8)
    - activation='tanh'
    - max_iter=100
3. Model with different solver:
    - hidden_layer_sizes=(32, 16)
    - activation='relu'
    - solver='lbfgs'
    - max_iter=100
4. Model with different alpha value:
    - hidden_layer_sizes=(32, 16)
    - activation='relu'
    - alpha=0.0001
    - max_iter=100

These are some of the architecture that i have explained and i tried to print the accuracy of the model. The output of the program is given below(model 1 and model 2 in the output are my previous architecture which i have explained above in the output):



Model 2 has higher accuracy with epoch=100.

PS: python program to run this is python3 extensions_1.py

2)Extensions:

The second extension I have used is that I have tried to use to explore creating networks for other data sets. This dataset contains measurements of various properties of wines and their corresponding quality ratings. It can be used for classification or regression tasks. Two models for the wine quality dataset and the architecture of the models are given below:

Model 1 has two hidden layers with 32 and 16 neurons respectively and an 'relu' activation function.

Model 2 has four hidden layers with 64, 32, 16, and 8 neurons respectively and an 'relu' activation function.

Both models are trained on the same training set and evaluated on the same test set using the accuracy score. The goal of the models is to predict the quality of wine, which is a numeric value between 0 and 10. The MLPClassifier is a type of neural network that can learn from the features of the dataset and make predictions based on them. I have tried both the models with 10, 50 and 100 epochs and tried to compare the accuracy. The output of the program is given below:

```
models with 100 iterations

Model 1 Test accuracy: 0.859375
Model 2 Test accuracy: 0.146875

models with 50 iterations

Model 1 Test accuracy: 0.853125
Model 2 Test accuracy: 0.853125

models with 10 iterations

Model 1 Test accuracy: 0.853125
Model 2 Test accuracy: 0.853125
(base) gmonk@Gmons-MacBook-Air PM4 %
```

Model 1 has performed better with 100 epochs and Model 2 accuracy is better in 50 epochs and it diminishes when the number of iteration has increased.

PS: python program to this code is python3 extensions_2.py

Summary(short reflection of what you learned):

- ☐ How to build a convolutional neural network for the MNIST digit recognition task and experiment with various network variations to optimise network performance and training time.
- ☐ How to use transfer learning to reuse a pre-trained MNIST network to recognize three different Greek letters: alpha, beta, and gamma.
- ☐ experimentation with the deep network for the MNIST task and how to increase the accuracy of the model
- ☐ How to come up with a plan for exploring different network architectures and metrics to use, and how to execute that plan to evaluate the network variations.
- ☐ How to develop a hypothesis for how the network will behave along each dimension, and how to test and evaluate that hypothesis
- ☐ How to automate the process of evaluating different network architectures and track the training and testing errors over time
- ☐ How to plot the training and testing errors and print out the modified network architecture.
- ☐ How to transform RGB images to grayscale, scale and crop them to the correct size, and invert the intensities to match the MNIST digits
- ☐ How to build different networks to predict heart disease using the same data set and compare their performance.
- ☐ As part of extensions, I was able to work with a wine dataset and try to create a network model. I was able to improve their accuracy.
- ☐ How to do experiments with the epochs of the model so that we can find a better accuracy.

Bibliography:

Some of the importance links and materials i have used for this projects are:

1. https://scikit-learn.org/stable/

2. Textbook by Muller and Guido

3. https://youtu.be/ukzFI9rgwfU

4.https://nextjournal.com/gkoehler/pytorch-mnist

5.https://pytorch.org/tutorials/beginner/transfer_learning_tutorial.html?highlight=transfer%20learning%20ant%20bees