Final Project

Tri-An Nguyen
RUID: 221008675
netid: tdn39

Sebastian Massella
RUID: 207000623
netid: sxm4

Arjan Abazovic
RUID: 209009268
netid: aa2386

Introduction

In this project, we implement the perception algorithm, a three-layer neural network, and a three-layer neural network using PyTorch to detect faces and classify digits. We train these algorithms first on 10% of the training data, then 20%, up to 100%, to investigate the results as a function of the percentage of training data used. We compare the performance of the algorithms on the test data, and report the average training time and the average test accuracy (and standard deviation) as a function of the percentage of training data used.

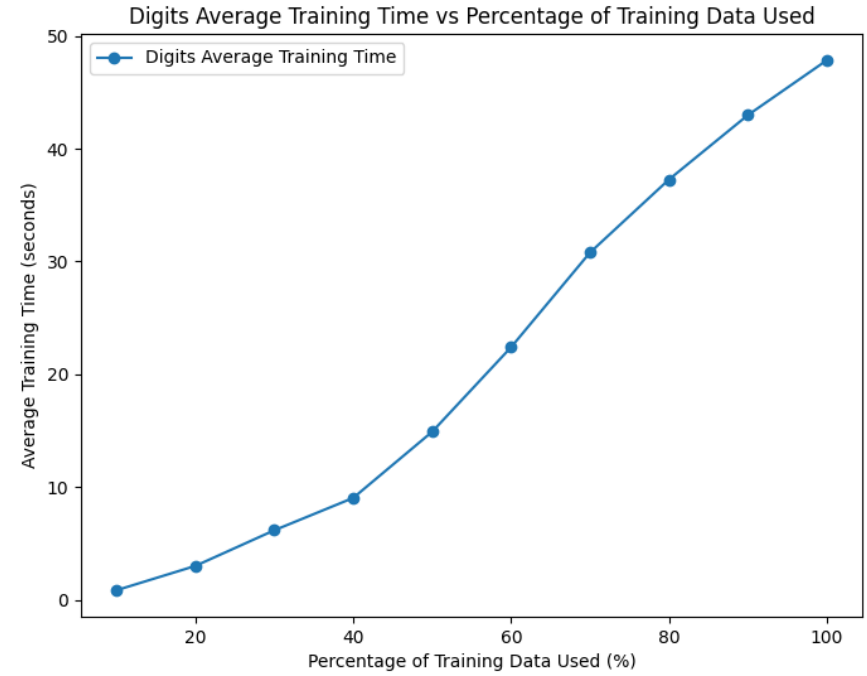
Data Preparation

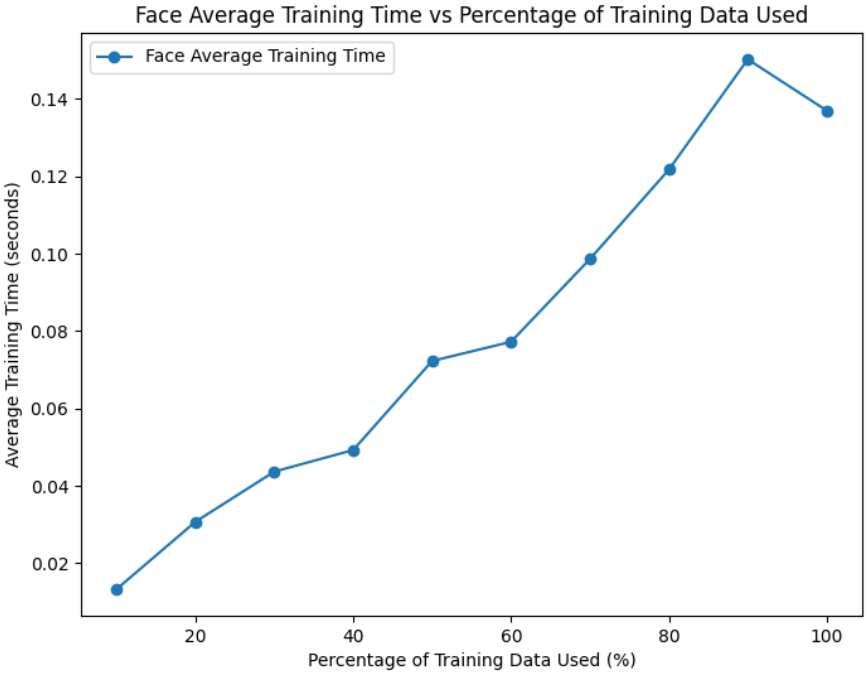
To extract the digit and face data from the given text images, we used the data processing code provided by UC Berkeley `https://ai.berkeley.edu/classification.html`. After loading the data from their corresponding files, we created digit and face data features. We chose straightforward features for both datasets by making each pixel a feature. For the digit train and test sets, we set the value of a pixel feature to 0 if the pixel is blank, 1 if the pixel is gray, and 2 if the pixel is black. Similarly, for the face train and test sets, we set the value of a pixel feature to 0 if the pixel is blank, and 1 if the pixel is an edge.

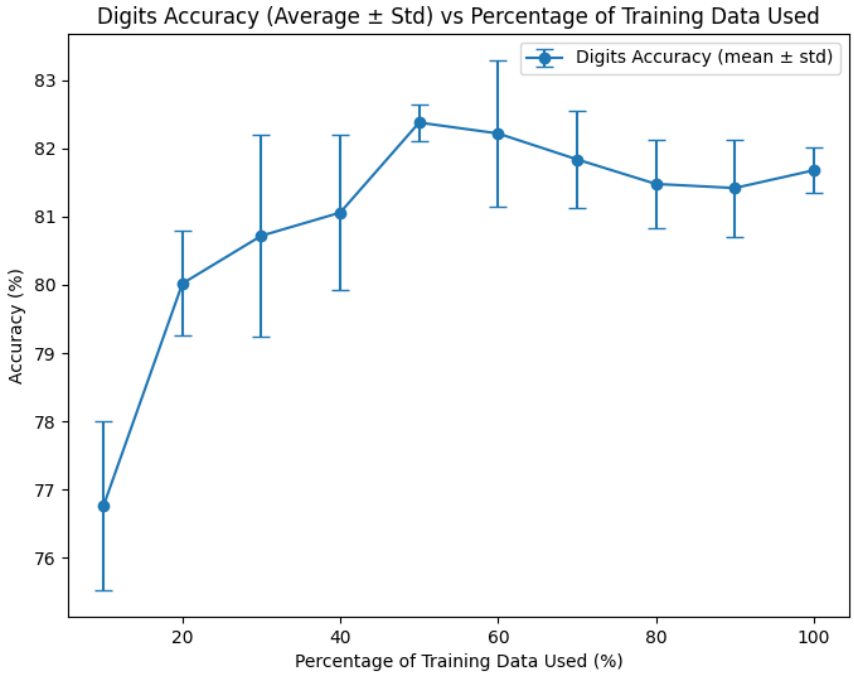
Perceptron Algorithm

We trained the perceptron algorithm using a percentage of the training data (10%, 20%, ..., 100%), 5 times each, to gather training time and test accuracy averages. Before inputting the training data into the algorithm to train, we randomly selected data samples. We implemented the perceptron algorithm by following the steps shown in the lecture. First, we initialized the weights to zero and the maximum epochs to 100, to give our simple algorithm enough time to train on relatively complex datasets. Then we went through every data point in the training data, computing the dot product between the weights and the data sample. If the product matches the label, we continue; if not, we update the weights accordingly. This process continues until we make a pass on the training data without making any updates, or the maximum number of epochs is reached. We evaluated the trained model on the test set.

Below are plots showing average training time and test accuracy per percentage of training data for each dataset, followed by the output log.


Digits Average Training Time vs Percentage of Training Data Used
Digits Average Training Time
Average Training Time (seconds)
Percentage of Training Data Used (%)
0
10
20
30
40
50
20
40
60
80
100


Face Average Training Time vs Percentage of Training Data Used
Face Average Training Time
Average Training Time (seconds)
Percentage of Training Data Used (%)
0.02
0.04
0.06
0.08
0.10
0.12
0.14
20
40
60
80
100


Digits Accuracy (Average ± Std) vs Percentage of Training Data Used
Digits Accuracy (mean ± std)
Accuracy (%)
Percentage of Training Data Used (%)
83
82
81
80
79
78
77
76
20
40
60
80
100


Face Accuracy (Average ± Std) vs Percentage of Training Data Used
Face Accuracy (mean ± std)
Accuracy (%)
Percentage of Training Data Used (%)
90.0
87.5
85.0
82.5
80.0
77.5
75.0
72.5
70.0
20
40
60
80
100

```
Percentage: 10%, Digits Average Training Time: 0.86 seconds, Face Average Training Time: 0.01 seconds
Percentage: 20%, Digits Average Training Time: 3.01 seconds, Face Average Training Time: 0.03 seconds
Percentage: 30%, Digits Average Training Time: 6.17 seconds, Face Average Training Time: 0.04 seconds
Percentage: 40%, Digits Average Training Time: 9.04 seconds, Face Average Training Time: 0.05 seconds
Percentage: 50%, Digits Average Training Time: 14.90 seconds, Face Average Training Time: 0.07 seconds
Percentage: 60%, Digits Average Training Time: 22.40 seconds, Face Average Training Time: 0.08 seconds
Percentage: 70%, Digits Average Training Time: 30.79 seconds, Face Average Training Time: 0.10 seconds
Percentage: 80%, Digits Average Training Time: 37.26 seconds, Face Average Training Time: 0.12 seconds
Percentage: 90%, Digits Average Training Time: 42.97 seconds, Face Average Training Time: 0.15 seconds
Percentage: 100%, Digits Average Training Time: 47.84 seconds, Face Average Training Time: 0.14 seconds
Percentage: 10%, Digits Average Accuracy: 76.76 ± 1.24, Face Average Accuracy: 73.20 ± 2.93
Percentage: 20%, Digits Average Accuracy: 80.02 ± 0.77, Face Average Accuracy: 77.20 ± 3.69
Percentage: 30%, Digits Average Accuracy: 80.72 ± 1.48, Face Average Accuracy: 83.07 ± 2.22
Percentage: 40%, Digits Average Accuracy: 81.06 ± 1.14, Face Average Accuracy: 82.67 ± 1.84
Percentage: 50%, Digits Average Accuracy: 82.38 ± 0.27, Face Average Accuracy: 84.93 ± 2.22
Percentage: 60%, Digits Average Accuracy: 82.22 ± 1.07, Face Average Accuracy: 86.00 ± 1.98
Percentage: 70%, Digits Average Accuracy: 81.84 ± 0.70, Face Average Accuracy: 85.33 ± 1.40
Percentage: 80%, Digits Average Accuracy: 81.48 ± 0.66, Face Average Accuracy: 87.60 ± 1.61
Percentage: 90%, Digits Average Accuracy: 81.42 ± 0.71, Face Average Accuracy: 86.80 ± 1.54
Percentage: 100%, Digits Average Accuracy: 81.68 ± 0.34, Face Average Accuracy: 86.80 ± 1.95
```

Based on the average training time plots and output logs, the average training time for both datasets linearly increases with the percentage of data used. This is expected as the more training data, the more computational effort is required. The training time for the digits dataset is also much higher than the face dataset for the same percentage, which is also expected, as there is more training data for the digits dataset.

Based on the average test accuracy plots and output logs, the test accuracy generally increases for both datasets; however, the digits dataset plateaus much quicker. The digits data accuracy increases to about 81% using 40% of the training data, and stays around 81%-82% accuracy for the rest of the training. The face data accuracy, on the other hand, increases throughout most of the training, only plateauing around 90%-100%. The output log shows that the standard deviation for both datasets generally decreases with the percentage of training data, which shows the accuracy is becoming more consistent. The face dataset's test accuracy is higher than the digit dataset after 30% of the data is used.

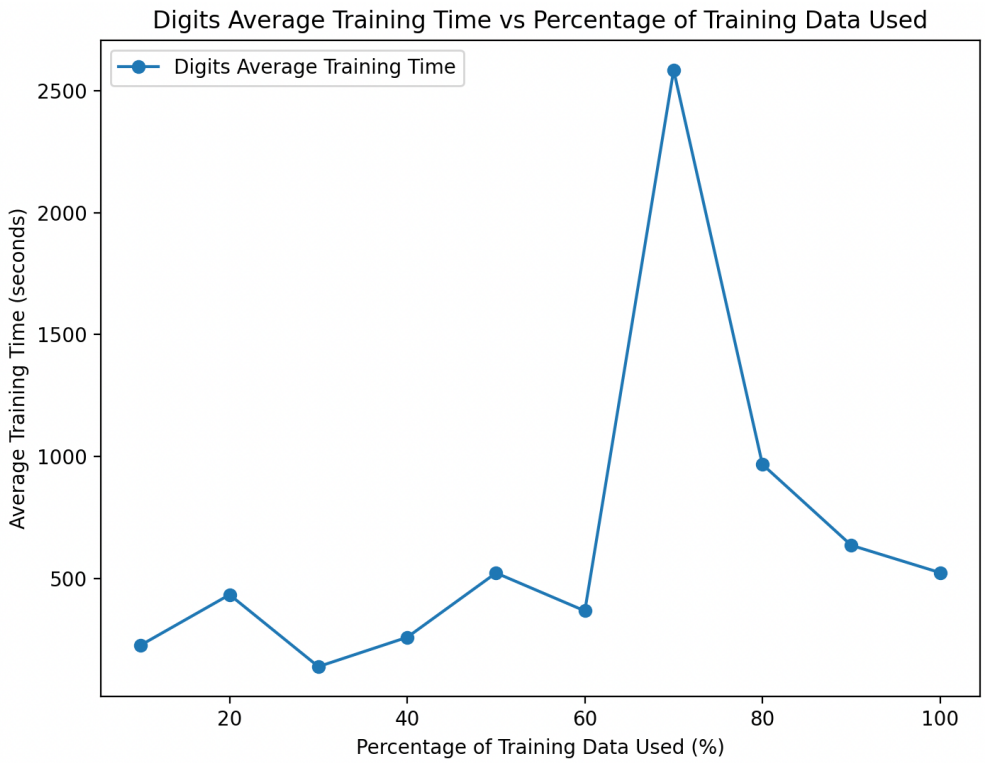
Three-layer Neural Network

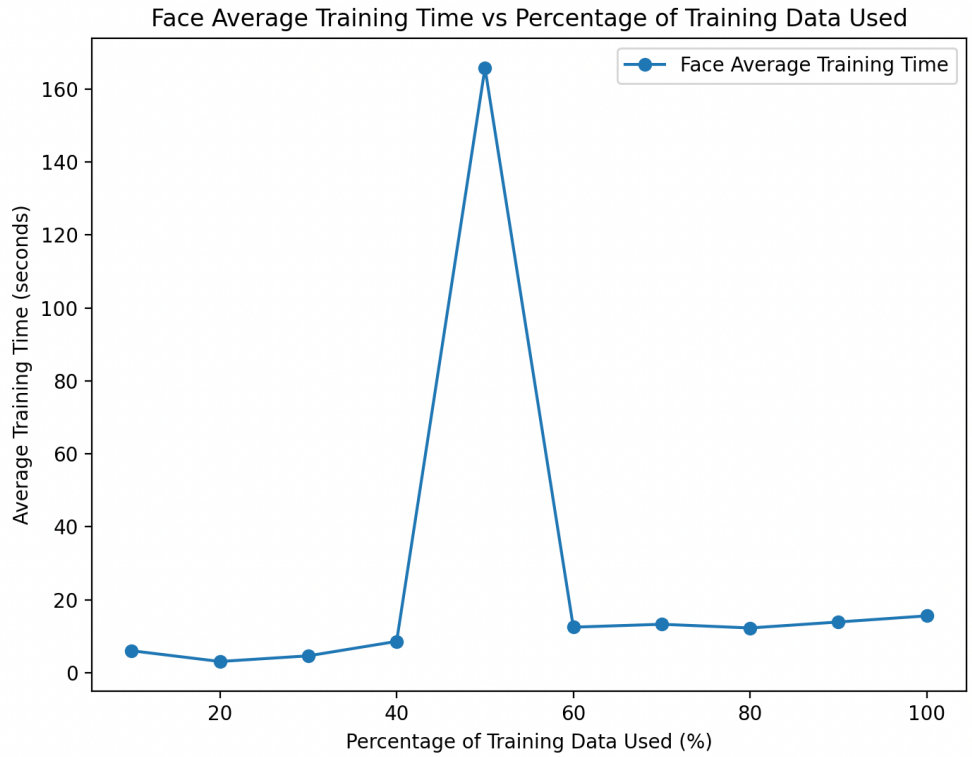
First, we defined our neural network class containing an input layer, two hidden layers, and an output layer, all as type `np.longdouble` to mitigate underflow. We used Sigmoid as the activation function and initialized the weights in the network using Xavier Initialization to help prevent vanishing gradients during training. We used 20 neurons in each hidden layer because it produced good results while keeping model complexity low. We also experimented with different values of L2 regularization and found 0.0 regularization to achieve the best results, as our model did not have an overfitting problem. We adjusted the learning rates to find the learning rate that balanced convergence speed and accuracy. We found the digits dataset worked well with a 0.5 learning rate, while the face dataset worked well with a 0.7 learning rate.

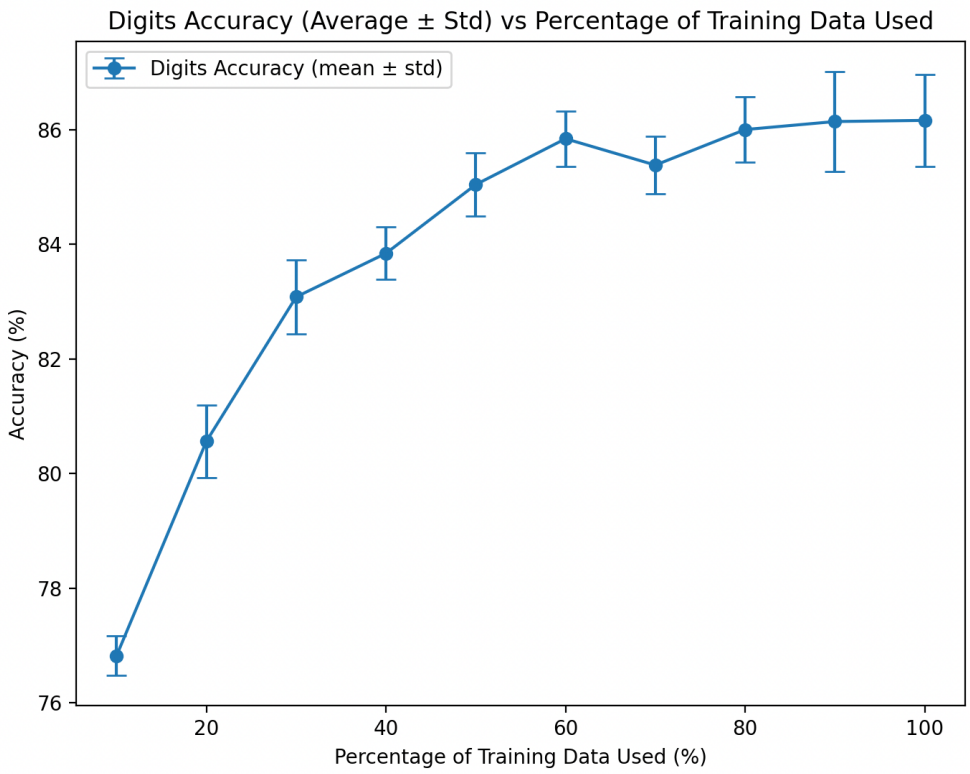
We suspect the following reason for such high learning rates: due to the use of the sigmoid function, we encountered vanishing gradients as described above. So having a high learning rate enabled us to still learn and change the weights, even when the gradient was super small. Additionally, we set the learning rate for the digits dataset lower because there is more nuance as there are 10 output neurons.

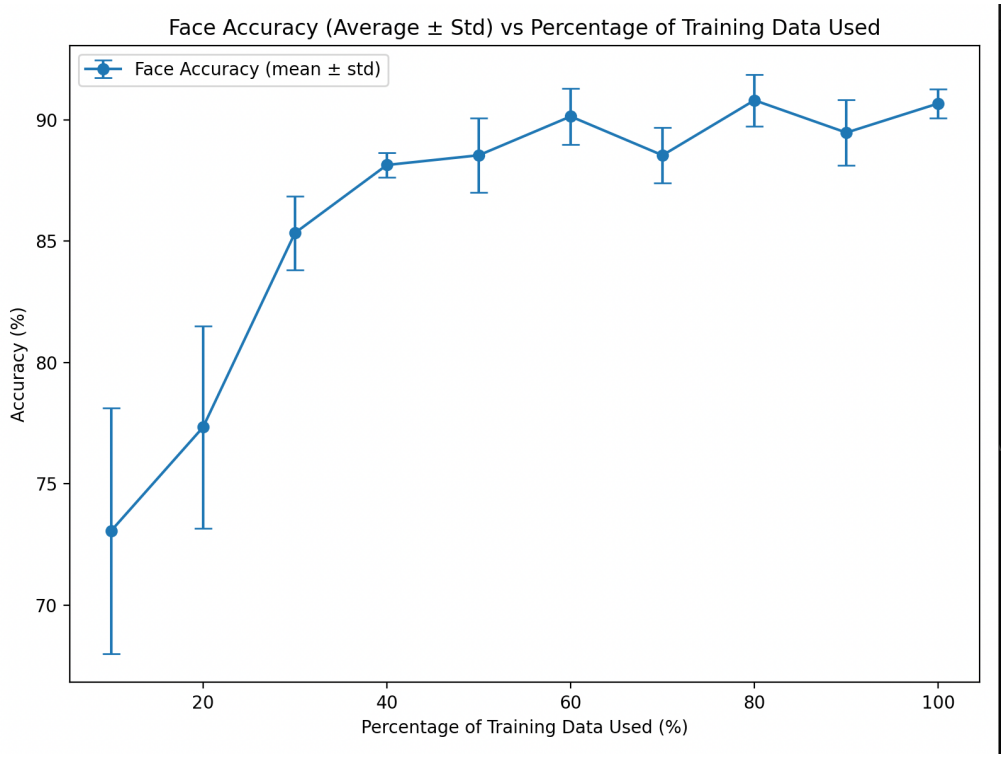
We trained the neural network using a percentage of the training data (10%, 20%, ..., 100%), 5 times each, to gather training time and test accuracy averages. Before inputting the training data into the model to train, we randomly selected data samples. We trained the digits dataset for 1000 epochs and the face dataset for 100 epochs. In each epoch, we manually computed the forward pass, the loss/cost, the backward pass, and updated the weights. Once training was completed, we evaluated the model on the test set.

Below are plots showing average training time and test accuracy per percentage of training data for each dataset, followed by the output log.


Digits Average Training Time vs Percentage of Training Data Used
Digits Average Training Time
Average Training Time (seconds)
2500
2000
1500
1000
500
20
40
60
80
100
Percentage of Training Data Used (%)


Face Average Training Time vs Percentage of Training Data Used
Face Average Training Time
Average Training Time (seconds)
160
140
120
100
80
60
40
20
0
20
40
60
80
100
Percentage of Training Data Used (%)


Digits Accuracy (Average ± Std) vs Percentage of Training Data Used
Digits Accuracy (mean ± std)
Accuracy (%)
Percentage of Training Data Used (%)
76
78
80
82
84
86
20
40
60
80
100


Face Accuracy (Average ± Std) vs Percentage of Training Data Used
Face Accuracy (mean ± std)
Accuracy (%)
Percentage of Training Data Used (%)
70
75
80
85
90
20
40
60
80
100

```
Percentage: 10%, Digits Average Training Time: 225.81 seconds, Face Average Training Time: 6.05 seconds
Percentage: 20%, Digits Average Training Time: 432.76 seconds, Face Average Training Time: 3.11 seconds
Percentage: 30%, Digits Average Training Time: 137.54 seconds, Face Average Training Time: 4.65 seconds
Percentage: 40%, Digits Average Training Time: 257.94 seconds, Face Average Training Time: 8.61 seconds
Percentage: 50%, Digits Average Training Time: 521.74 seconds, Face Average Training Time: 165.76 seconds
Percentage: 60%, Digits Average Training Time: 366.26 seconds, Face Average Training Time: 12.51 seconds
Percentage: 70%, Digits Average Training Time: 2584.96 seconds, Face Average Training Time: 13.30 seconds
Percentage: 80%, Digits Average Training Time: 967.17 seconds, Face Average Training Time: 12.30 seconds
Percentage: 90%, Digits Average Training Time: 635.36 seconds, Face Average Training Time: 13.93 seconds
Percentage: 100%, Digits Average Training Time: 523.13 seconds, Face Average Training Time: 15.61 seconds
Percentage: 10%, Digits Average Accuracy: 76.82 ± 0.34, Face Average Accuracy: 73.07 ± 5.05
Percentage: 20%, Digits Average Accuracy: 80.56 ± 0.63, Face Average Accuracy: 77.33 ± 4.17
Percentage: 30%, Digits Average Accuracy: 83.08 ± 0.65, Face Average Accuracy: 85.33 ± 1.52
Percentage: 40%, Digits Average Accuracy: 83.84 ± 0.46, Face Average Accuracy: 88.13 ± 0.50
Percentage: 50%, Digits Average Accuracy: 87.04 ± 4.02, Face Average Accuracy: 88.53 ± 1.54
Percentage: 60%, Digits Average Accuracy: 85.84 ± 0.48, Face Average Accuracy: 90.13 ± 1.15
Percentage: 70%, Digits Average Accuracy: 85.38 ± 0.50, Face Average Accuracy: 88.53 ± 1.15
Percentage: 80%, Digits Average Accuracy: 86.00 ± 0.57, Face Average Accuracy: 90.80 ± 1.07
Percentage: 90%, Digits Average Accuracy: 86.14 ± 0.87, Face Average Accuracy: 89.47 ± 1.36
Percentage: 100%, Digits Average Accuracy: 86.16 ± 0.80, Face Average Accuracy: 90.67 ± 0.60
```

Based on the average training time plots and output logs, the training time generally increases with the percentage of training data used, as expected. The training time for the digits dataset doesn't linearly increase, which could be for reasons such as vanishing gradients or the computer that was used for training. The face dataset does have that linear relationship between the training time and portion of training data used, besides the one spike. Also, the training for the digits dataset took significantly longer than the face dataset. To speed up training, the ReLU activation function could be used, which mitigates vanishing gradients.

Based on the average test accuracy plots and output logs, the test accuracy generally increases with the percentage of training data used. For the digits dataset, the test accuracy increases but then begins to plateau around 80%-100%. The face dataset's test accuracy also increases, but then plateaus around 70%-100%. The standard deviation in accuracy is mostly below 1 for the digits dataset, showing consistency in accuracy. For the face dataset, the standard deviation starts large and then quickly decreases, indicating the accuracy is becoming more consistent with more data. The face detection task achieves higher accuracy than the digit recognition task past 20% training data.

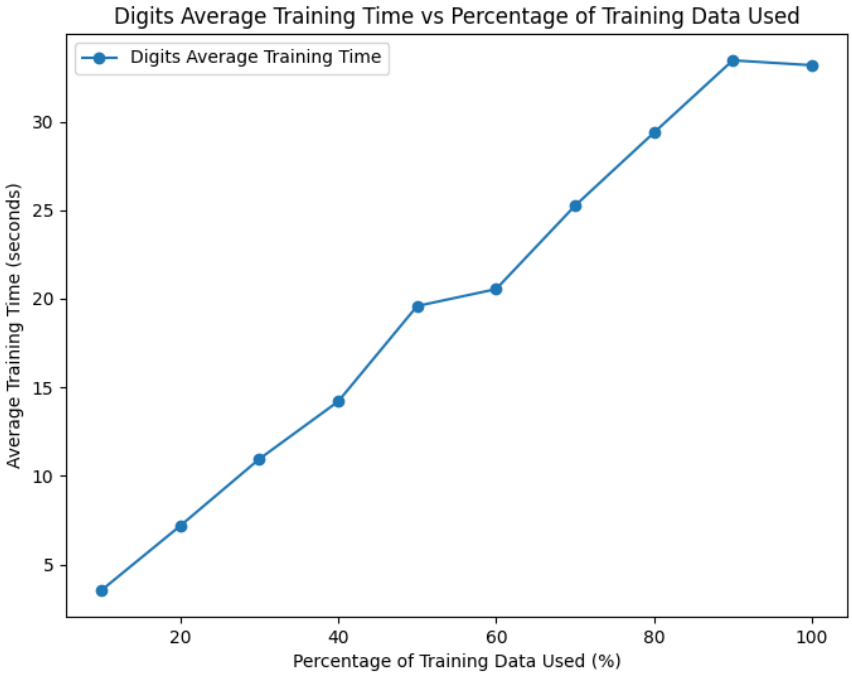
Three-layer Neural Network using PyTorch

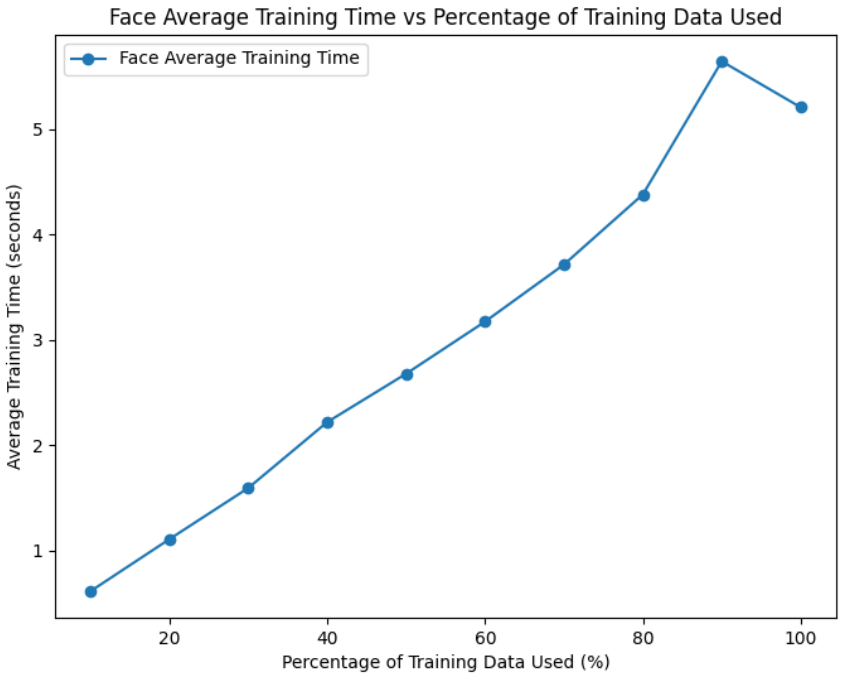
First, we defined our neural network class containing an input layer, two hidden layers, and an output layer. We experimented with different numbers of neurons in each hidden layer, such as {100, 100}, {100, 50}, {250, 250}, in the first and second layers, respectively. We found that the different numbers of neurons produced similar results; therefore, we went with {100, 100} for simplicity. The number of output neurons depends on the dataset. For the digits dataset, the output was of size 10, corresponding to the 10 digits, whereas, for the face dataset, the output size was 2, corresponding to detecting a face or not.

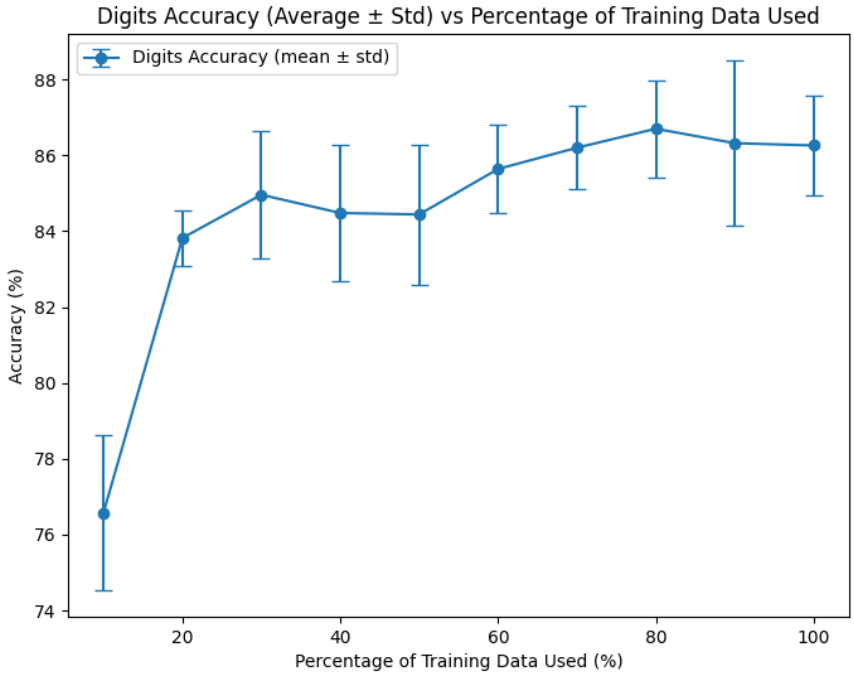
For the activation function, we chose Sigmoid, and for the loss function, we used cross-entropy loss. We used stochastic gradient descent as the optimizer with a 0.1 learning rate. We experimented with different learning rates, such as {0.1, 0.01, 0.001}, and found 0.1 significantly performs the best in speed and accuracy, which aligns with the fact that SGD works well with larger learning rates. We also experimented with different amounts of L2 regularization, such as {0.0, 0.01, 0.001}, and found that 0.0 L2 regularization performed the best in accuracy and significantly in training time.

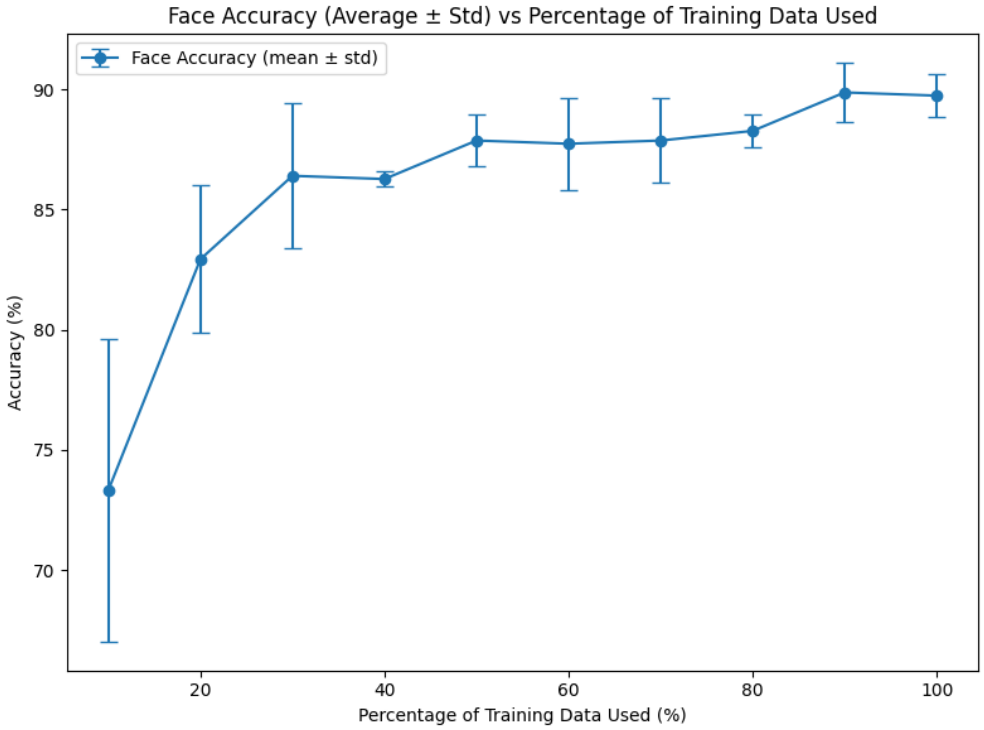
We trained the model using a percentage of the training data (10%, 20%, ..., 100%), 5 times each, to gather training time and test accuracy averages. Before inputting the training data into the model to train, we randomly select data samples. We transform the training data into tensors, then train the neural network for 10 epochs. The training process includes putting the data sample through the model and getting the output, calculating the loss, computing the gradients, and then updating the weights. Once the model finished training, we evaluated it on the test data.

Below are plots showing average training time and test accuracy per percentage of training data for each dataset, followed by the output log.


Digits Average Training Time vs Percentage of Training Data Used
Digits Average Training Time
Average Training Time (seconds)
5
10
15
20
25
30
20
40
60
80
100
Percentage of Training Data Used (%)


Face Average Training Time vs Percentage of Training Data Used
Face Average Training Time
Average Training Time (seconds)
1
2
3
4
5
20
40
60
80
100
Percentage of Training Data Used (%)


Digits Accuracy (Average ± Std) vs Percentage of Training Data Used
Digits Accuracy (mean ± std)
Accuracy (%)
Percentage of Training Data Used (%)
74
76
78
80
82
84
86
88
20
40
60
80
100


Face Accuracy (Average ± Std) vs Percentage of Training Data Used
Face Accuracy (mean ± std)
Accuracy (%)
Percentage of Training Data Used (%)
70
75
80
85
90
20
40
60
80
100

```
Percentage: 10%, Digits Average Training Time: 3.55 seconds, Face Average Training Time: 0.61 seconds
Percentage: 20%, Digits Average Training Time: 7.19 seconds, Face Average Training Time: 1.11 seconds
Percentage: 30%, Digits Average Training Time: 10.96 seconds, Face Average Training Time: 1.59 seconds
Percentage: 40%, Digits Average Training Time: 14.21 seconds, Face Average Training Time: 2.22 seconds
Percentage: 50%, Digits Average Training Time: 19.59 seconds, Face Average Training Time: 2.68 seconds
Percentage: 60%, Digits Average Training Time: 20.54 seconds, Face Average Training Time: 3.17 seconds
Percentage: 70%, Digits Average Training Time: 25.25 seconds, Face Average Training Time: 3.71 seconds
Percentage: 80%, Digits Average Training Time: 29.37 seconds, Face Average Training Time: 4.38 seconds
Percentage: 90%, Digits Average Training Time: 33.46 seconds, Face Average Training Time: 5.64 seconds
Percentage: 100%, Digits Average Training Time: 33.18 seconds, Face Average Training Time: 5.21 seconds
Percentage: 10%, Digits Average Accuracy: 76.58 ± 2.04, Face Average Accuracy: 73.33 ± 6.30
Percentage: 20%, Digits Average Accuracy: 83.82 ± 0.73, Face Average Accuracy: 82.93 ± 3.06
Percentage: 30%, Digits Average Accuracy: 84.96 ± 1.68, Face Average Accuracy: 86.40 ± 3.00
Percentage: 40%, Digits Average Accuracy: 84.48 ± 1.80, Face Average Accuracy: 86.27 ± 0.33
Percentage: 50%, Digits Average Accuracy: 84.44 ± 1.84, Face Average Accuracy: 87.87 ± 1.07
Percentage: 60%, Digits Average Accuracy: 85.64 ± 1.16, Face Average Accuracy: 87.73 ± 1.91
Percentage: 70%, Digits Average Accuracy: 86.20 ± 1.09, Face Average Accuracy: 87.87 ± 1.76
Percentage: 80%, Digits Average Accuracy: 86.70 ± 1.29, Face Average Accuracy: 88.27 ± 0.68
Percentage: 90%, Digits Average Accuracy: 86.32 ± 2.18, Face Average Accuracy: 89.87 ± 1.22
Percentage: 100%, Digits Average Accuracy: 86.26 ± 1.32, Face Average Accuracy: 89.73 ± 0.90
```

Based on the average training time plots and output logs, the training time linearly increases with the percentage of training data used. This is expected as training on more data requires more computations. Also, the training for the digits dataset takes longer than the face dataset for the same percentage, which makes sense since there is more data for the digits dataset.

Based on the average test accuracy plots and output logs, the test accuracy generally increases with the percentage of training data used. For the digits dataset, the test accuracy increases but then begins to plateau around 80%-100%. The face dataset's test accuracy also increases, but then plateaus around 90%-100%. The standard deviation in accuracy generally stays between 1 and 2 for the digits dataset. For the face dataset, the standard deviation starts large and then quickly decreases, indicating the accuracy is becoming more consistent with more data. The face detection task achieves slightly higher accuracy than the digit recognition task past 20%.

Results/Discussion

Overall, we achieved good test average accuracies on both datasets with all three methods. Using 100% of the training data, the lowest accuracy was 81.68% on the digits dataset using perception, and the highest accuracy was 90.67% on the face dataset using the manual three-layer neural network. We should note that the neural networks differed in the number of layers, number of epochs, and initialization of the weights as we sought to deal with the vanishing gradient issue. Thus all differences in results should be taken with this in mind. In general, we observed that the test accuracy does improve with more data, but up to a certain point. Often, once around 50%-70% of the data was used, the accuracy would start to plateau or decrease in one case. In general, the standard deviation would decrease as more training data was used, showing more consistent training. For the digits dataset, the standard deviation would decrease slightly and then stay within a certain range. The face dataset would start with a large standard deviation and then usually decrease as more data was used. In terms of training time, we observed that the training time was generally proportional to the percentage of training data used. This is expected as usually the more data used, the more computations are performed, requiring more time. In terms of the methods, we noticed the Perceptron was fastest, then the Pytorch neural network, and then the manual neural network. This was as expected since the Perceptron is very simple, Pytorch has a lot of built-in optimization, and our neural network used many more epochs and also did not have the same optimizations.

Conclusion/Lessons Learned

This project taught us a lot about machine learning/deep learning and how to conduct experiments. We learned about the tradeoff between the amount of training data used and the accuracy. More training data does not always result in better accuracy. We also learned the importance of picking the right hyperparameters like learning rate, activation function, weight initialization, and size of hidden layers. We also gained experience writing a neural

network from scratch and writing one using libraries, which was much easier to implement. We also learned the value of implementing things in parallel. When implementing our manual neural network, we implemented it by doing the forward pass of each training point in the batch individually, which was very slow. We could have instead done the whole batch at once with matrix operations. Finally, we learned about the complexity of even a small neural network.