Intro to AI - Final Project Report

Authors (RUID & NetID): Janine Yanes - 209008173, jqy2; Aadi Gopi - 207003115, ag1903; Jason Cai - 208004639, jc2829

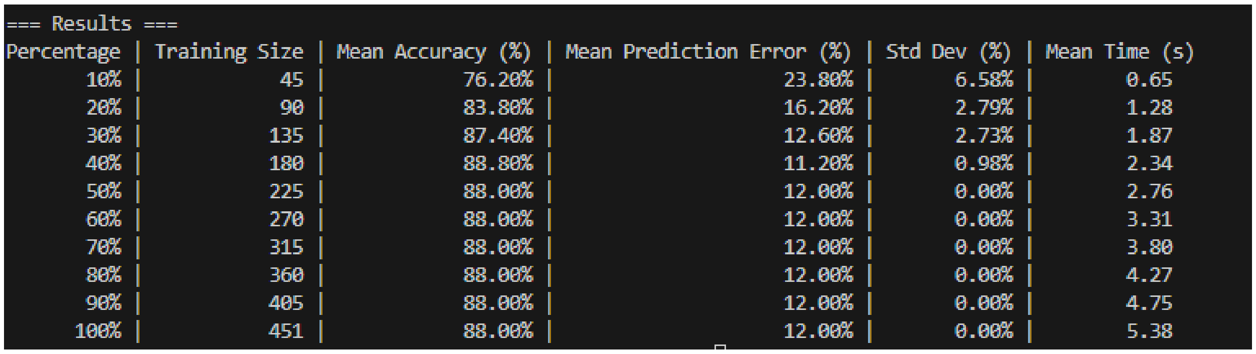
Perceptron

In our implementation of Perceptron, our algorithm maintains separate weight vectors for each class label, with each feature (pixel) in the given data corresponding to a specific weight within the vectors. Starting with all weights at zero, the algorithm computes a score for each class (by calculating the dot product between the input feature vector and each class's weight vector), then predicts the class with the highest score. During the training process, for each example in the training data, the algorithm predicts a class using the current weights and updates the weights according to the prediction: if the prediction is incorrect, the algorithm decreases the weights for the incorrectly predicted class and increases weights for the correct one. This gradually adjusts the predictions to better separate the classes over multiple iterations.

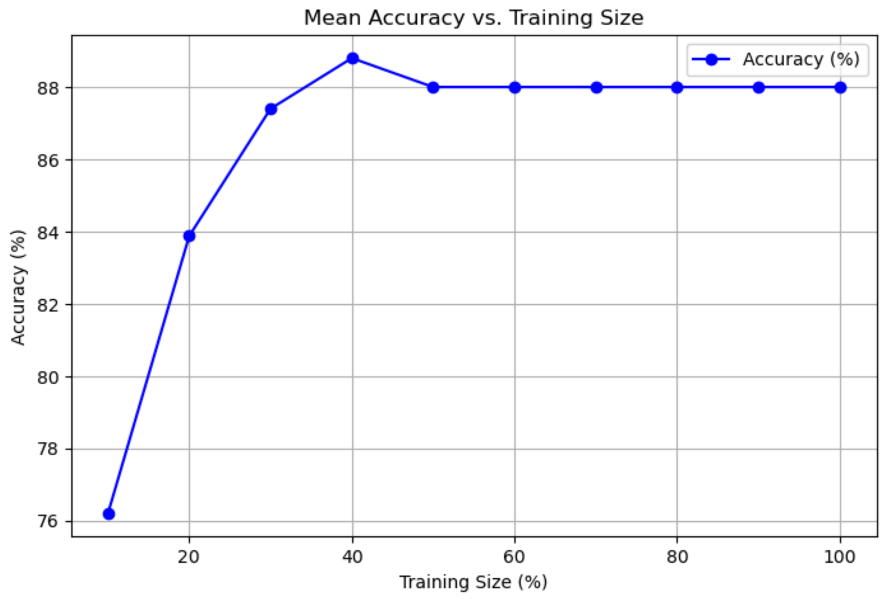
Perceptron Observations

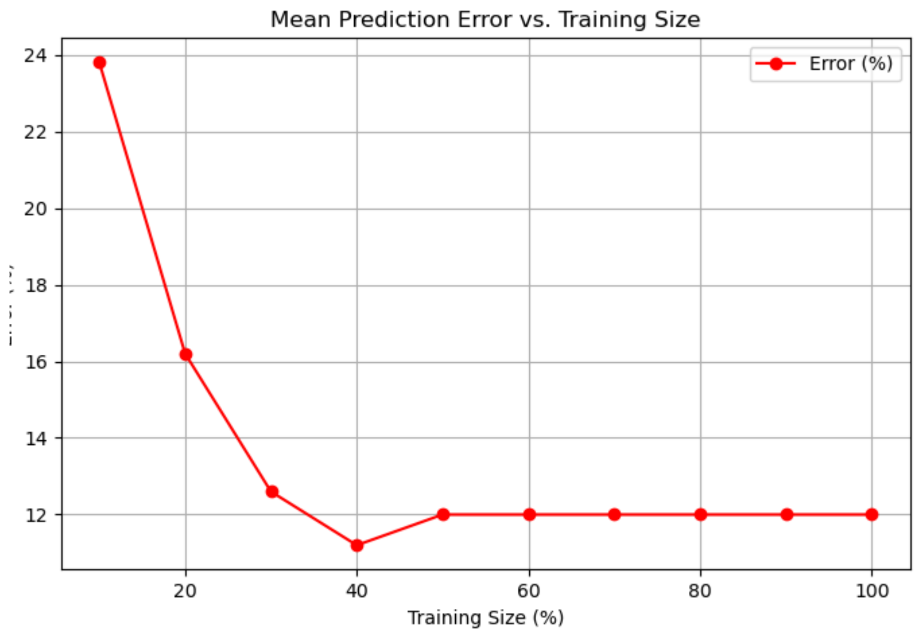
From our results, we observed that training time increased as the percentage of the training data used increased, while the standard deviation of the accuracy for each iteration went down. Surprisingly, we also saw that while the mean accuracy of the runs generally increased as the percentage of the training data used increased, it was highest when we used about half of the training size rather than when we used all the training data. Lastly, we also observed that that perceptron seemed to have converged when we tested it on the faces data set: as we used more of the training data, the standard deviation of the accuracy became 0%.

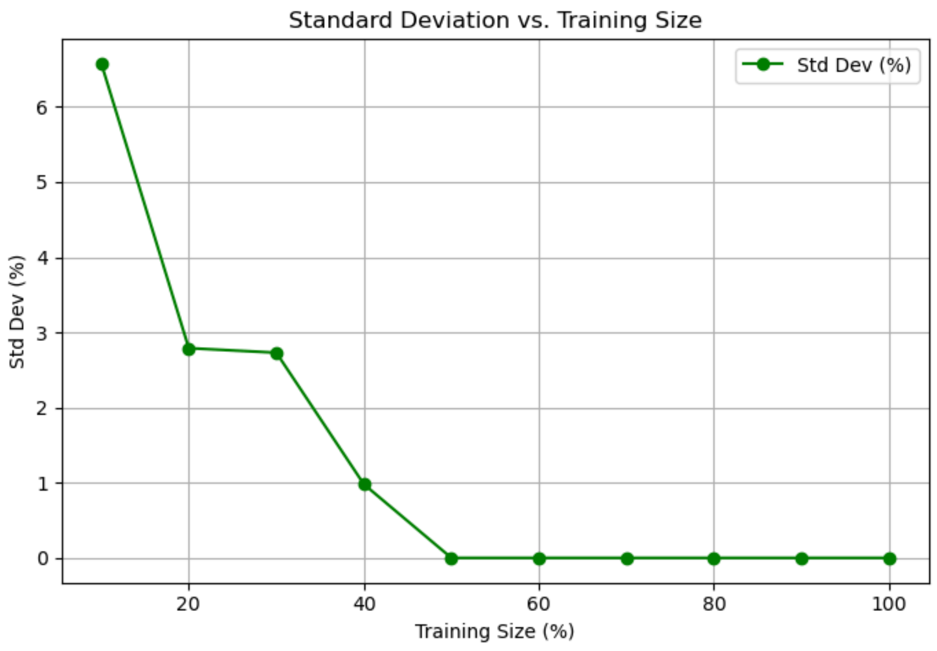
Faces Perceptron Results



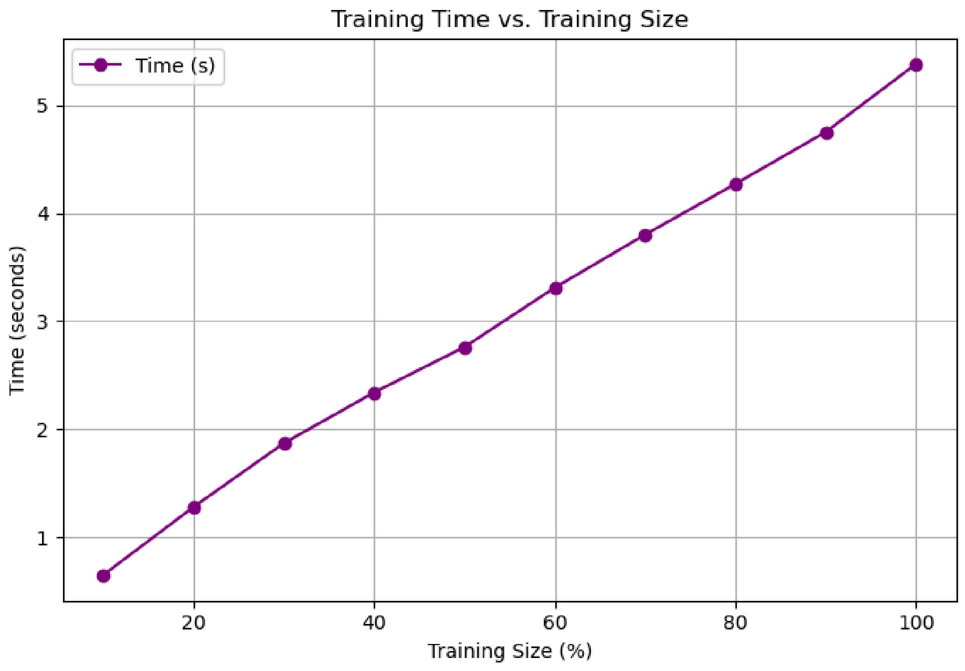
```
=== Results ===
Percentage | Training Size | Mean Accuracy (%) | Mean Prediction Error (%) | Std Dev (%) | Mean Time (s)
       10% |            45 |            76.20% |                    23.80% |       6.58% |          0.65
       20% |            90 |            83.80% |                    16.20% |       2.79% |          1.28
       30% |           135 |            87.40% |                    12.60% |       2.73% |          1.87
       40% |           180 |            88.80% |                    11.20% |       0.98% |          2.34
       50% |           225 |            88.00% |                    12.00% |       0.00% |          2.76
       60% |           270 |            88.00% |                    12.00% |       0.00% |          3.31
       70% |           315 |            88.00% |                    12.00% |       0.00% |          3.80
       80% |           360 |            88.00% |                    12.00% |       0.00% |          4.27
       90% |           405 |            88.00% |                    12.00% |       0.00% |          4.75
      100% |           451 |            88.00% |                    12.00% |       0.00% |          5.38
```



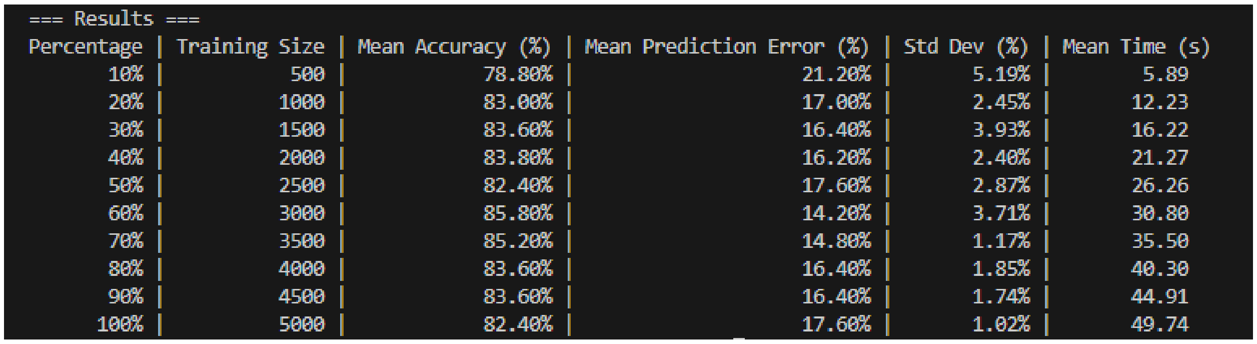




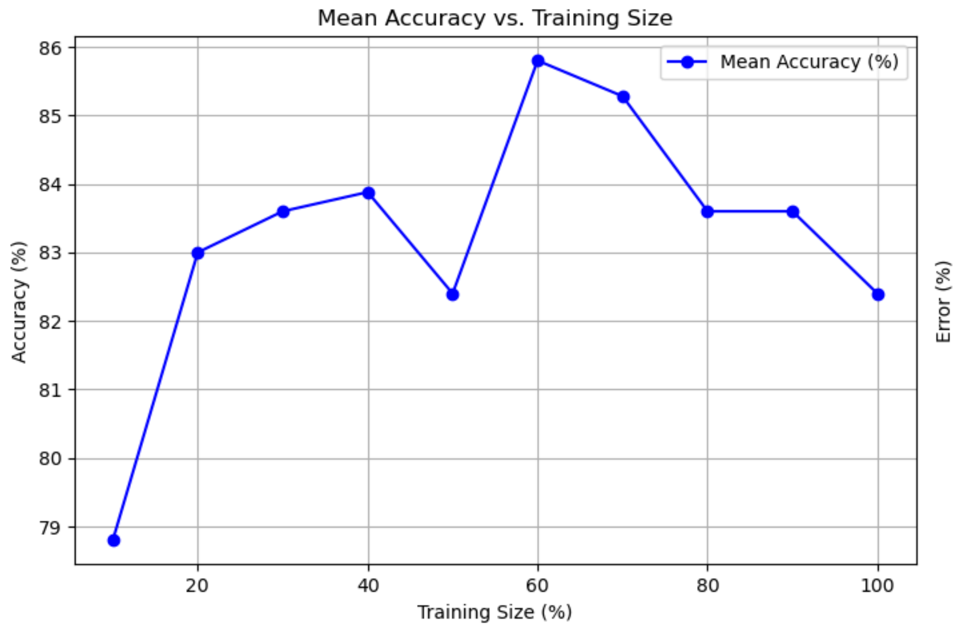
Training Size (%)

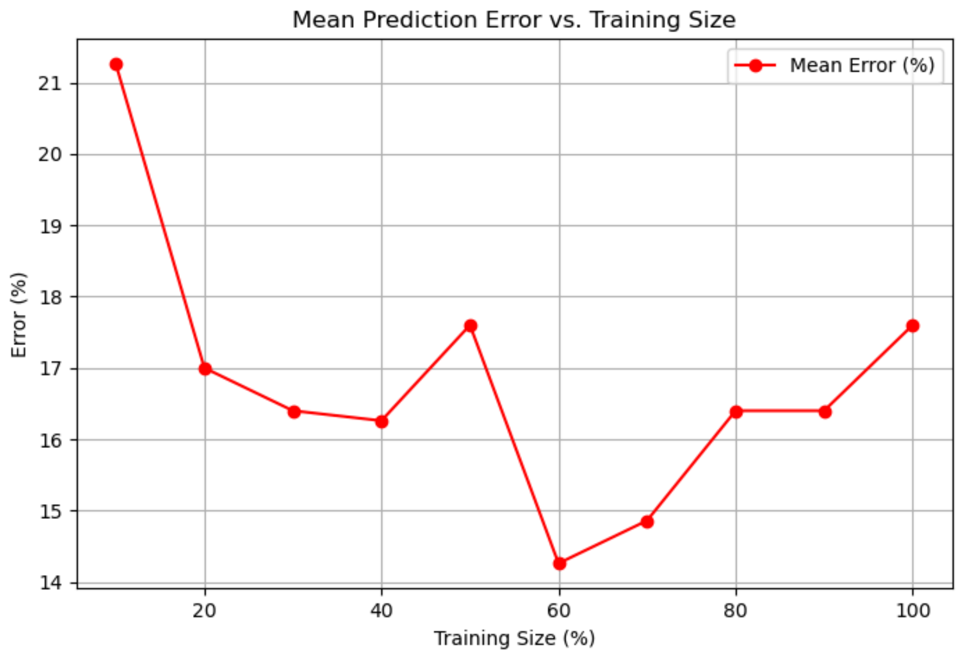


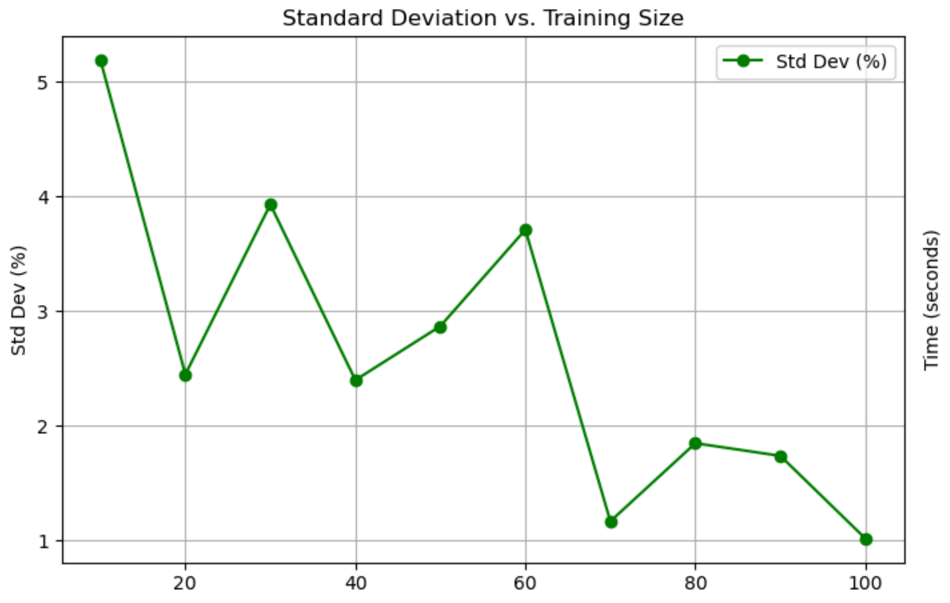
Digits Perceptron Results

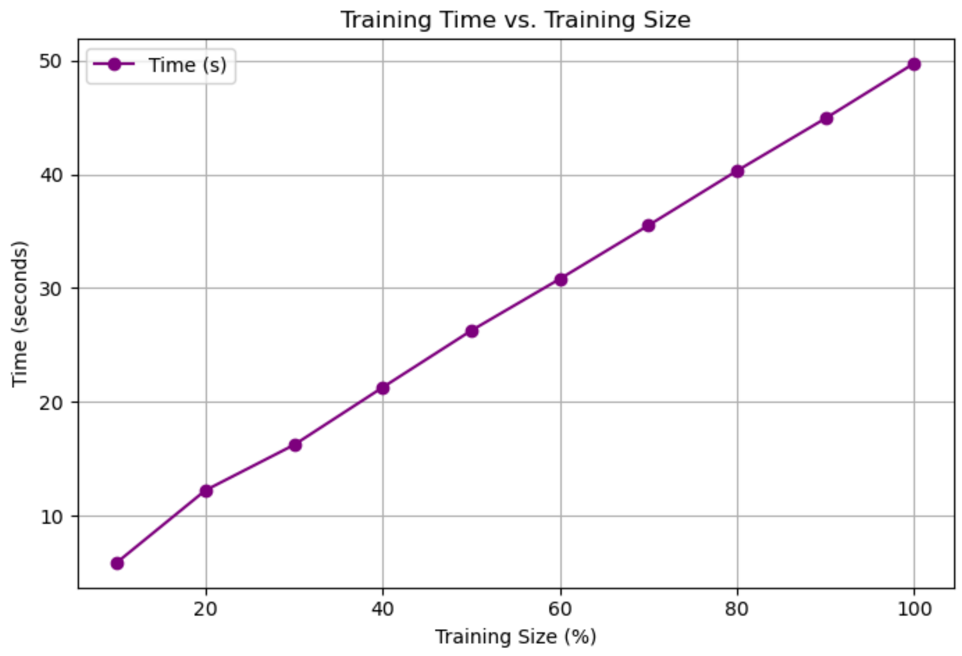


```
=== Results ===
Percentage | Training Size | Mean Accuracy (%) | Mean Prediction Error (%) | Std Dev (%) | Mean Time (s)
       10% |           500 |            78.80% |                    21.20% |       5.19% |          5.89
       20% |          1000 |            83.00% |                    17.00% |       2.45% |         12.23
       30% |          1500 |            83.60% |                    16.40% |       3.93% |         16.22
       40% |          2000 |            83.80% |                    16.20% |       2.40% |         21.27
       50% |          2500 |            82.40% |                    17.60% |       2.87% |         26.26
       60% |          3000 |            85.80% |                    14.20% |       3.71% |         30.80
       70% |          3500 |            85.20% |                    14.80% |       1.17% |         35.50
       80% |          4000 |            83.60% |                    16.40% |       1.85% |         40.30
       90% |          4500 |            83.60% |                    16.40% |       1.74% |         44.91
      100% |          5000 |            82.40% |                    17.60% |       1.02% |         49.74
```









Neural Network

The network processes input data through multiple layers of weighted connections while incorporating two hidden layers, each containing 100 neurons. During initialization, weights are set using He initialization: each weight is randomly chosen from the standard normal distribution and then scaled by $\sqrt{2/n}$ (with n being the size of the previous layer), such that the set of initial weights $W \sim N(0, \sigma^2 = 2/n)$. The forward pass computes activations through the network, where the inputs are multiplied by weights, added to biases, then passed through ReLU (such that ReLU(x)=max(0,x)), which in turn introduces non-linearity within the hidden layers, and then goes into a softmax function in the output layer to produce probability distributions across classes.

The training occurs through a mini batch gradient descent over multiple epochs. For each batch, the algorithm performs a forward pass to compute predictions, then a backward pass to calculate gradients of the loss function with respect to each weight and bias. We use these gradients to update the parameters. Furthermore, we also incorporate a lambda to help prevent overfitting by punishing/penalizing large weight values. During validation and classification, the trained network processes inputs through the forward pass only, and predictions are made by selecting the class with the highest output probability.

Pytorch Neural Network

The PyTorch version of the neural network stacks layers in sequence: a linear layer mapping inputs to the first hidden layer, followed by ReLU activation, and then another linear layer to the second hidden layer with ReLU activation, and a final linear layer producing the output logits. Unlike the previous implementation that required manual forward and backward propagation, this version leverages PyTorch's automatic differentiation system.

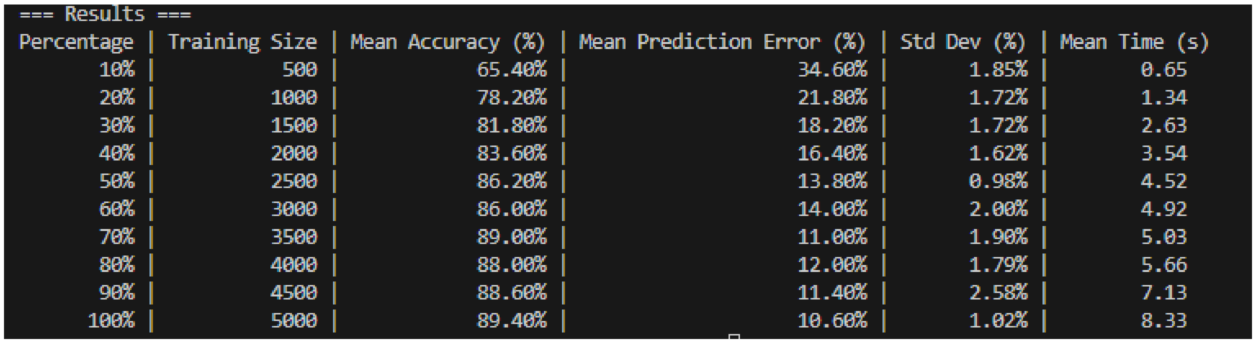
Training using PyTorch makes things much more streamline because of its built in methods. The CrossEntropyLoss function makes calculating losses much simpler, the Adam optimizer allows for more efficient weight updating, and the DataLoader allows for an efficient batch processing. During each training epoch, the model computes predictions, losses, backpropagation gradients, and updates parameters automatically through the optimizer. We also implemented validation at regular intervals to monitor the training progress.

For classification, the model processes the test data through the forward pass and selects the class with the highest output score. The PyTorch implementation achieves the same functionality as the base/non-pytorch neural network but with cleaner code, better optimization and access to the extensive features and utilities of the PyTorch framework, making it more maintainable.

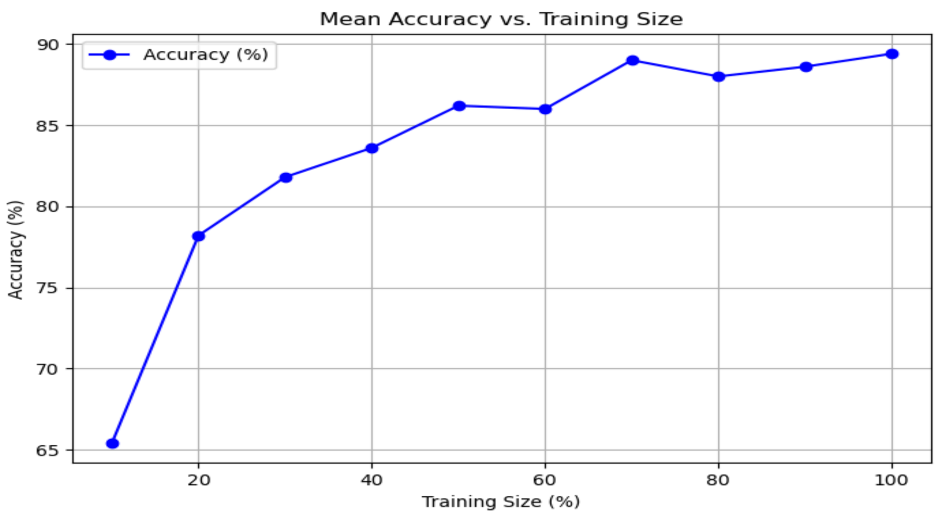
Observations of Neural Network and Pytorch Neural Network

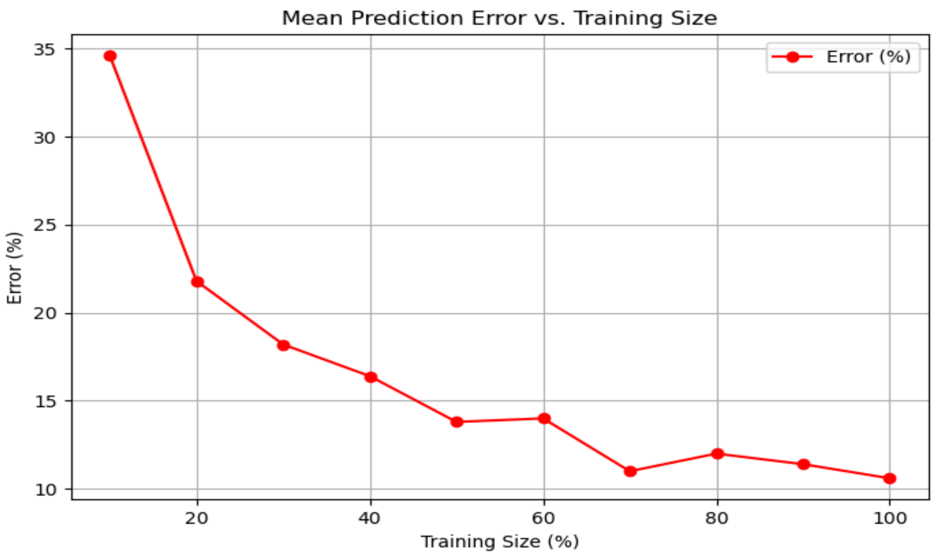
Some observations we made for both were that as the training size increases generally the prediction error decreases, while the standard deviation for each set also generally decreases. We also found that the neural network was not as good as predicting faces compared to digits: when identifying faces, it took more of the training data to reach the same mean accuracy compared to digits and also had a higher standard deviation.This could be due to the larger training data set for digits.

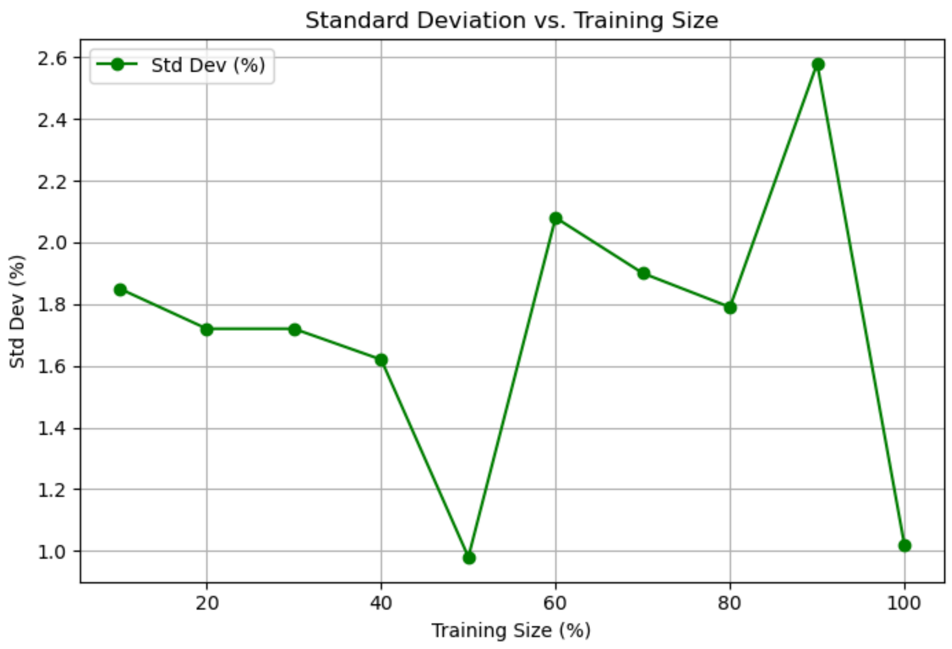
Digits Neural Network Results

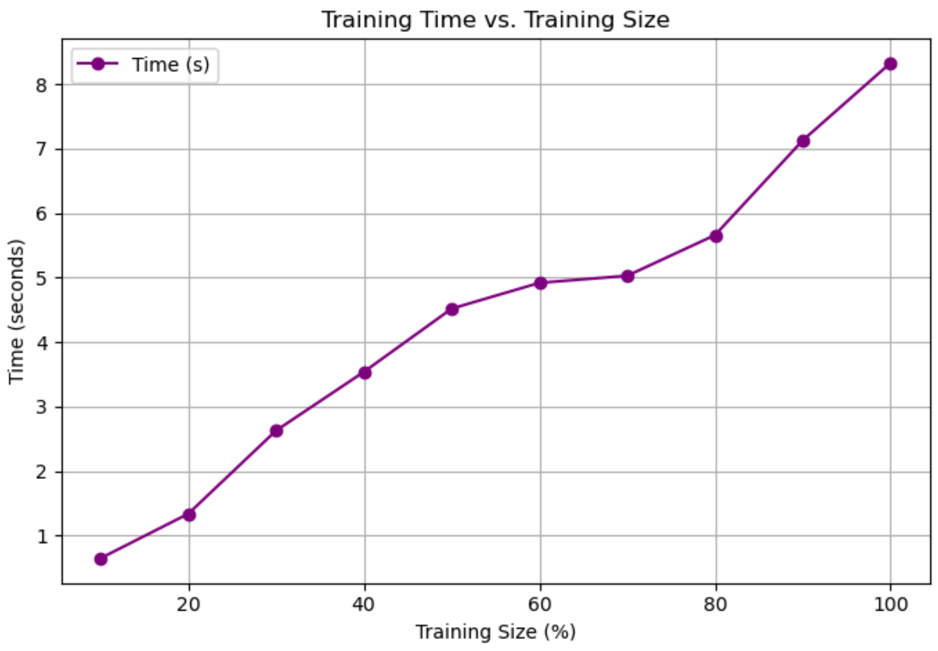


```
=== Results ===
Percentage | Training Size | Mean Accuracy (%) | Mean Prediction Error (%) | Std Dev (%) | Mean Time (s)
       10% |           500 |            65.40% |                    34.60% |       1.85% |          0.65
       20% |          1000 |            78.20% |                    21.80% |       1.72% |          1.34
       30% |          1500 |            81.80% |                    18.20% |       1.72% |          2.63
       40% |          2000 |            83.60% |                    16.40% |       1.62% |          3.54
       50% |          2500 |            86.20% |                    13.80% |       0.98% |          4.52
       60% |          3000 |            86.00% |                    14.00% |       2.00% |          4.92
       70% |          3500 |            89.00% |                    11.00% |       1.90% |          5.03
       80% |          4000 |            88.00% |                    12.00% |       1.79% |          5.66
       90% |          4500 |            88.60% |                    11.40% |       2.58% |          7.13
      100% |          5000 |            89.40% |                    10.60% |       1.02% |          8.33
```

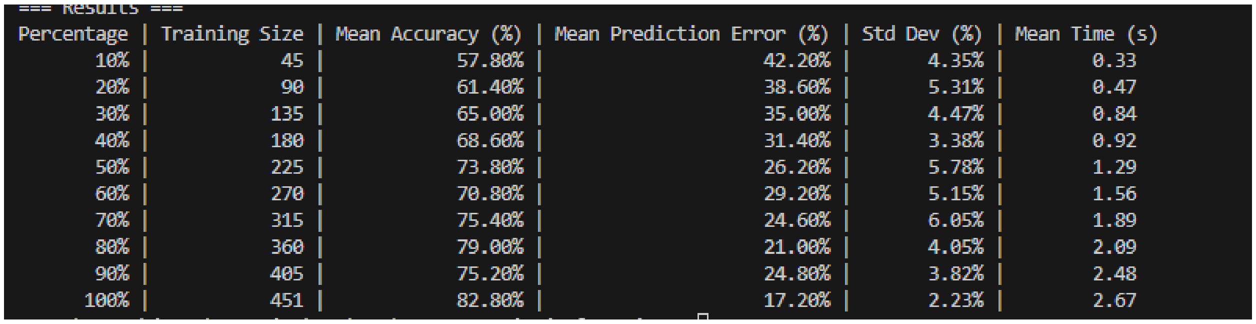




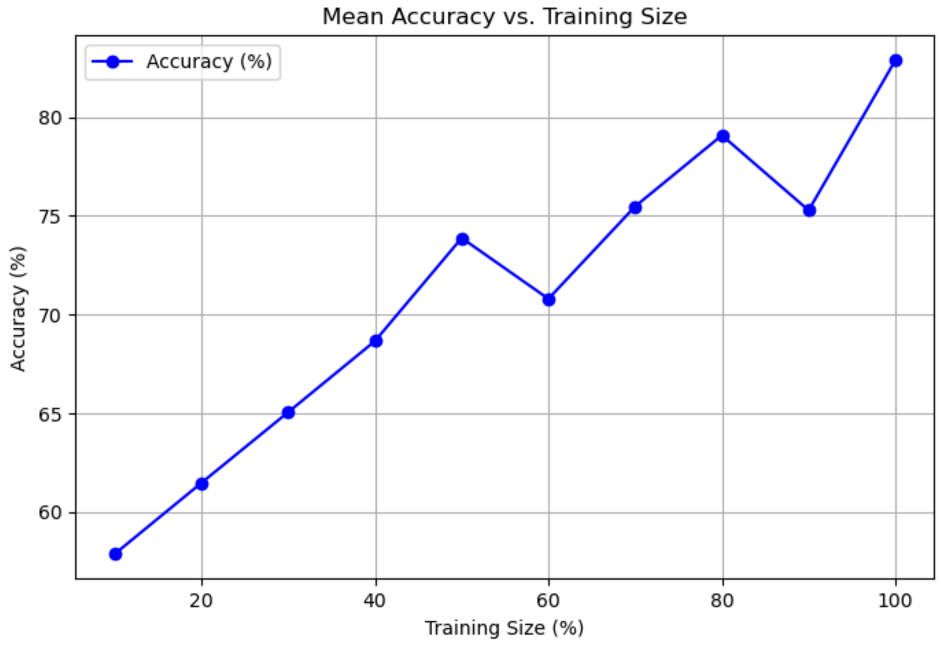

Standard Deviation vs. Training Size
Std Dev (%)
Std Dev (%)
Training Size (%)
1.0
1.2
1.4
1.6
1.8
2.0
2.2
2.4
2.6
20
40
60
80
100

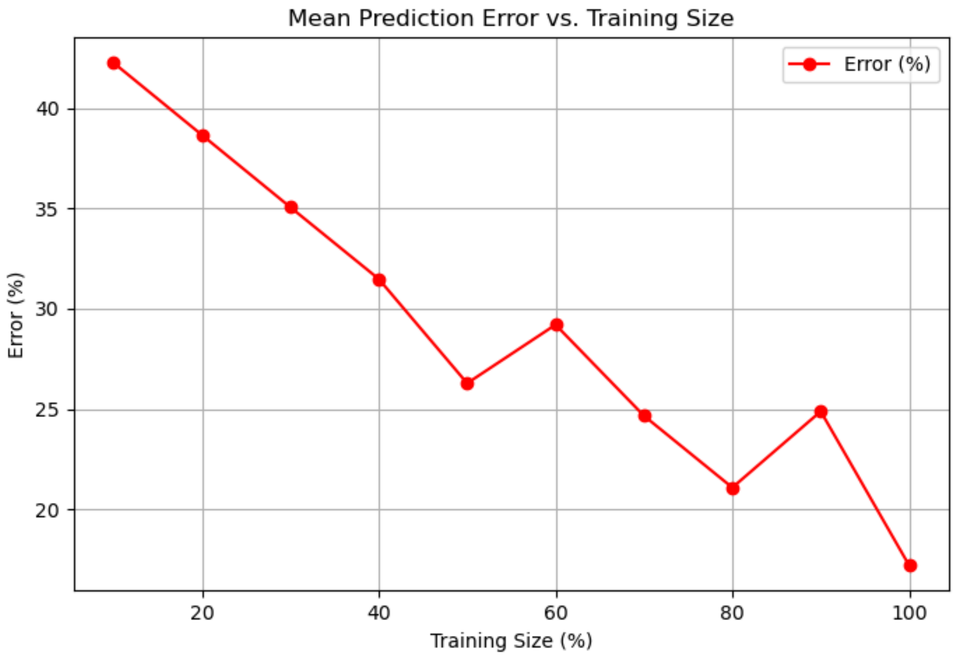

Training Time vs. Training Size
Time (s)
Time (seconds)
Training Size (%)
1
2
3
4
5
6
7
8
20
40
60
80
100

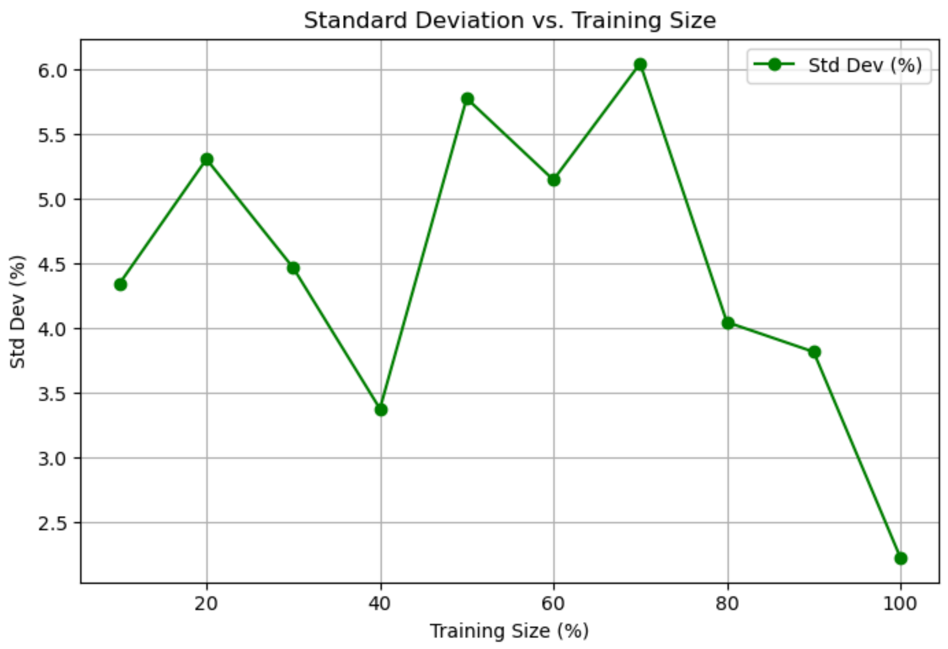
Faces Neural Network Results

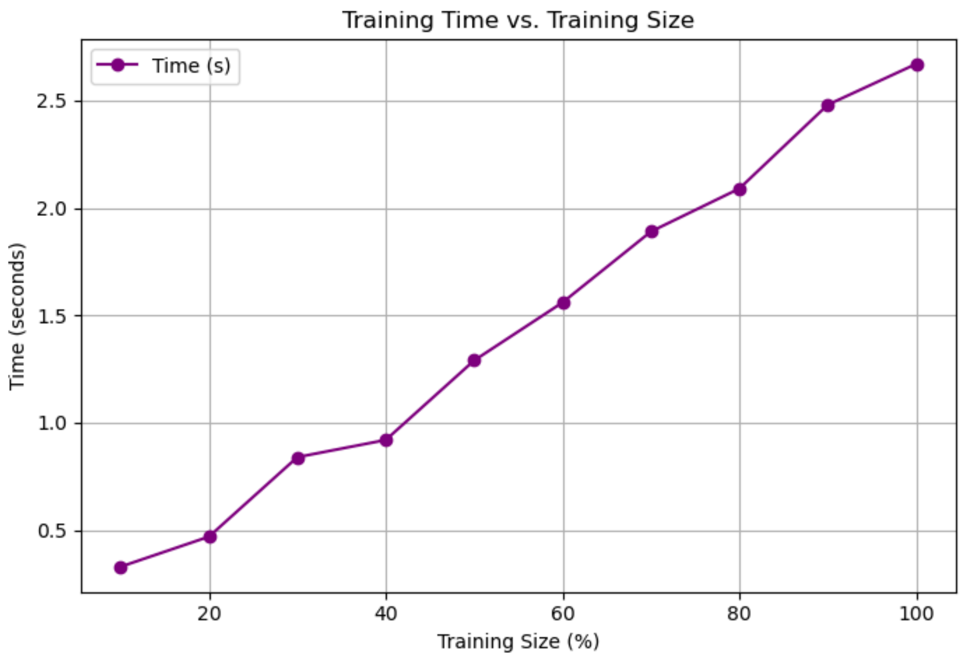


```
=== Results ===
Percentage | Training Size | Mean Accuracy (%) | Mean Prediction Error (%) | Std Dev (%) | Mean Time (s)
       10% |            45 |            57.80% |                    42.20% |       4.35% |          0.33
       20% |            90 |            61.40% |                    38.60% |       5.31% |          0.47
       30% |           135 |            65.00% |                    35.00% |       4.47% |          0.84
       40% |           180 |            68.60% |                    31.40% |       3.38% |          0.92
       50% |           225 |            73.80% |                    26.20% |       5.78% |          1.29
       60% |           270 |            70.80% |                    29.20% |       5.15% |          1.56
       70% |           315 |            75.40% |                    24.60% |       6.05% |          1.89
       80% |           360 |            79.00% |                    21.00% |       4.05% |          2.09
       90% |           405 |            75.20% |                    24.80% |       3.82% |          2.48
      100% |           451 |            82.80% |                    17.20% |       2.23% |          2.67
```

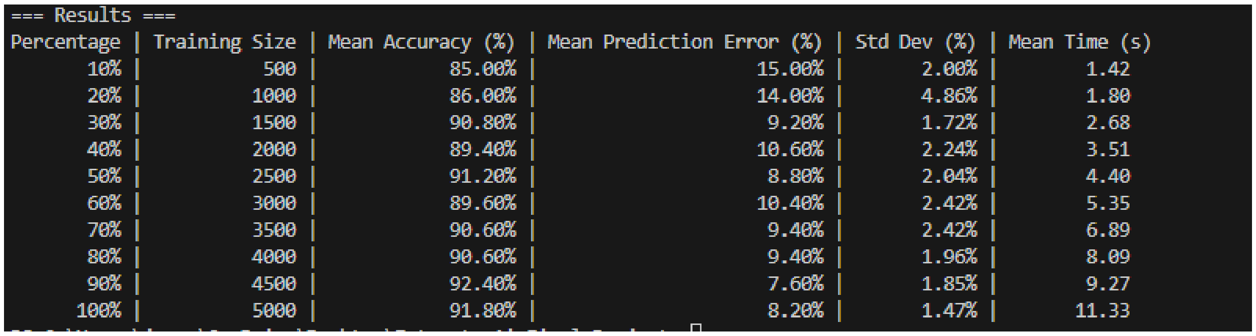




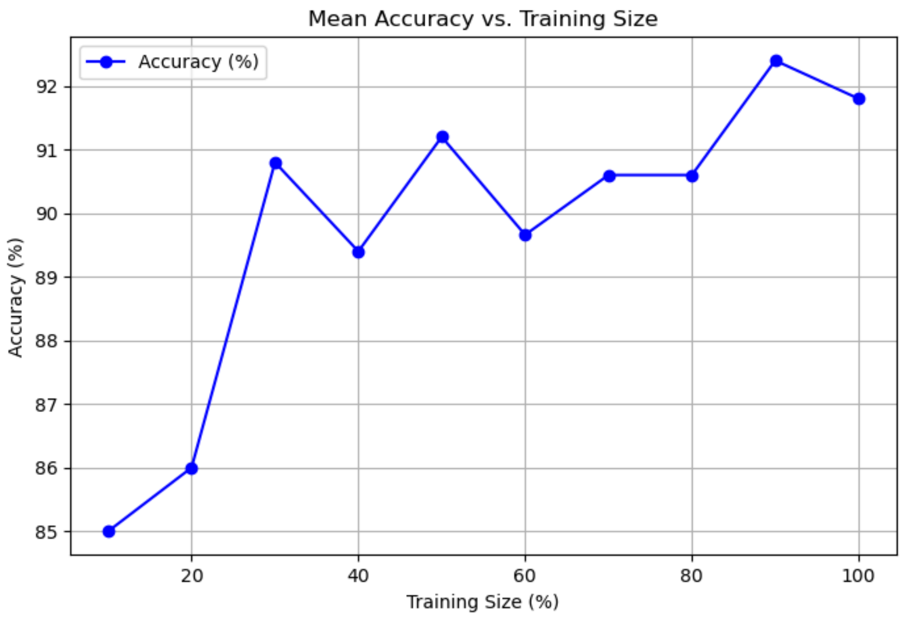

Standard Deviation vs. Training Size
Std Dev (%)
Std Dev (%)
Training Size (%)
6.0
5.5
5.0
4.5
4.0
3.5
3.0
2.5
20
40
60
80
100

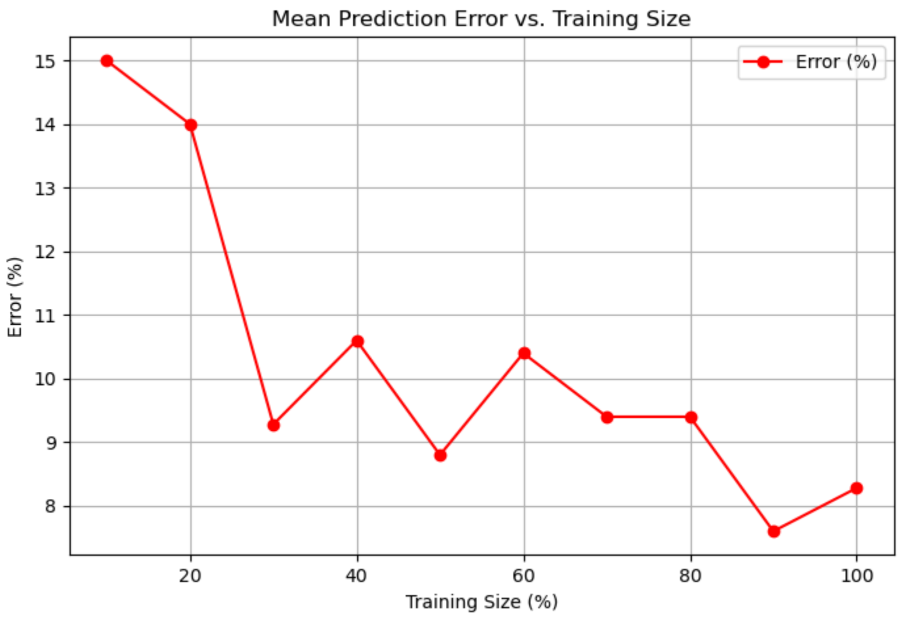

Training Time vs. Training Size
Time (s)
Time (seconds)
Training Size (%)
2.5
2.0
1.5
1.0
0.5
20
40
60
80
100

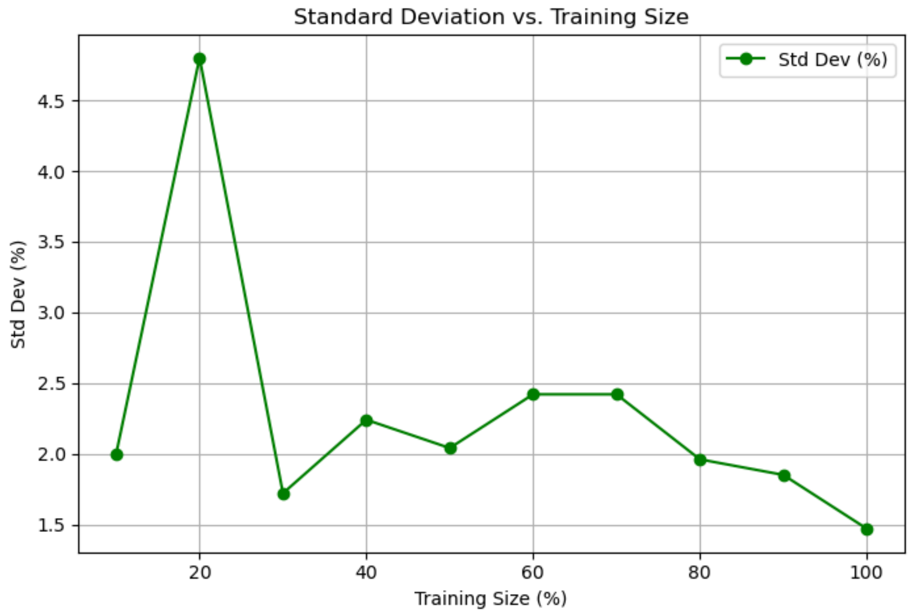
Digits Pytorch Neural Network Results

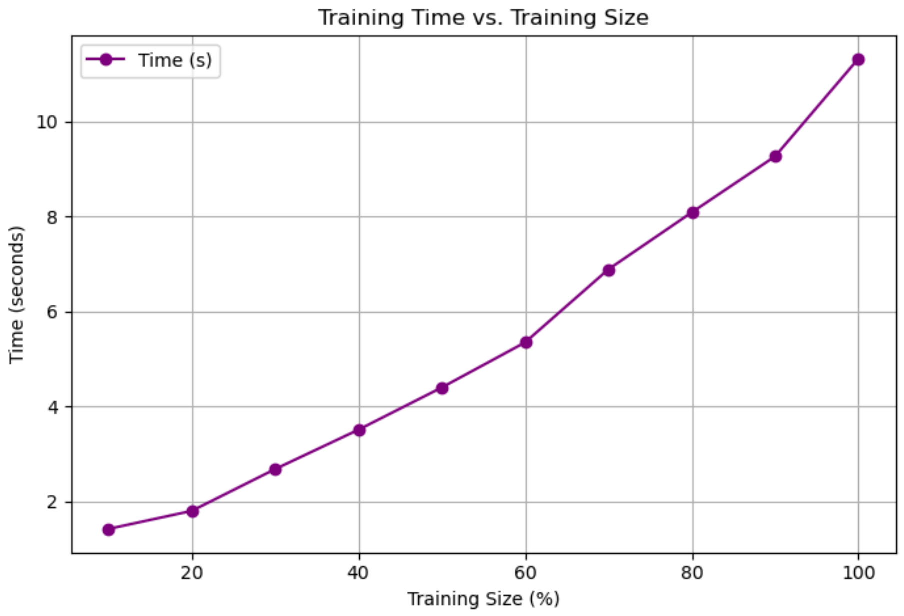


```
=== Results ===
Percentage | Training Size | Mean Accuracy (%) | Mean Prediction Error (%) | Std Dev (%) | Mean Time (s)
       10% |           500 |            85.00% |                    15.00% |       2.00% |          1.42
       20% |          1000 |            86.00% |                    14.00% |       4.86% |          1.80
       30% |          1500 |            90.80% |                     9.20% |       1.72% |          2.68
       40% |          2000 |            89.40% |                    10.60% |       2.24% |          3.51
       50% |          2500 |            91.20% |                     8.80% |       2.04% |          4.40
       60% |          3000 |            89.60% |                    10.40% |       2.42% |          5.35
       70% |          3500 |            90.60% |                     9.40% |       2.42% |          6.89
       80% |          4000 |            90.60% |                     9.40% |       1.96% |          8.09
       90% |          4500 |            92.40% |                     7.60% |       1.85% |          9.27
      100% |          5000 |            91.80% |                     8.20% |       1.47% |         11.33
```

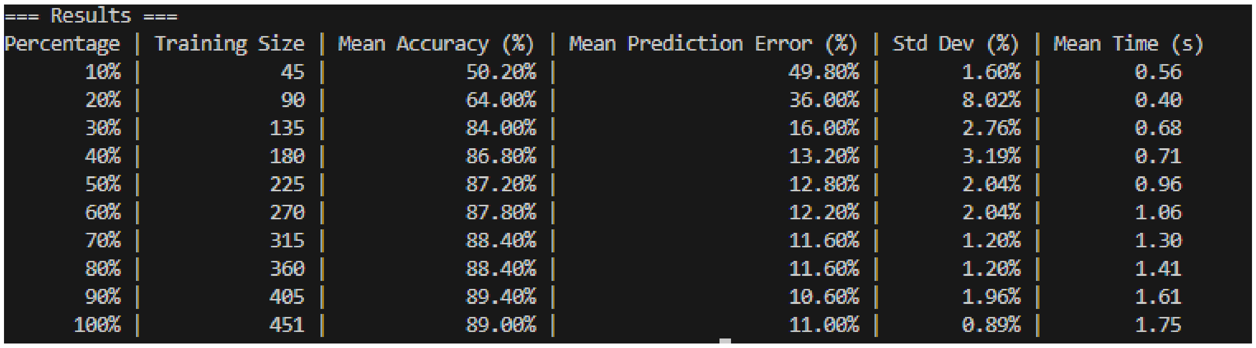




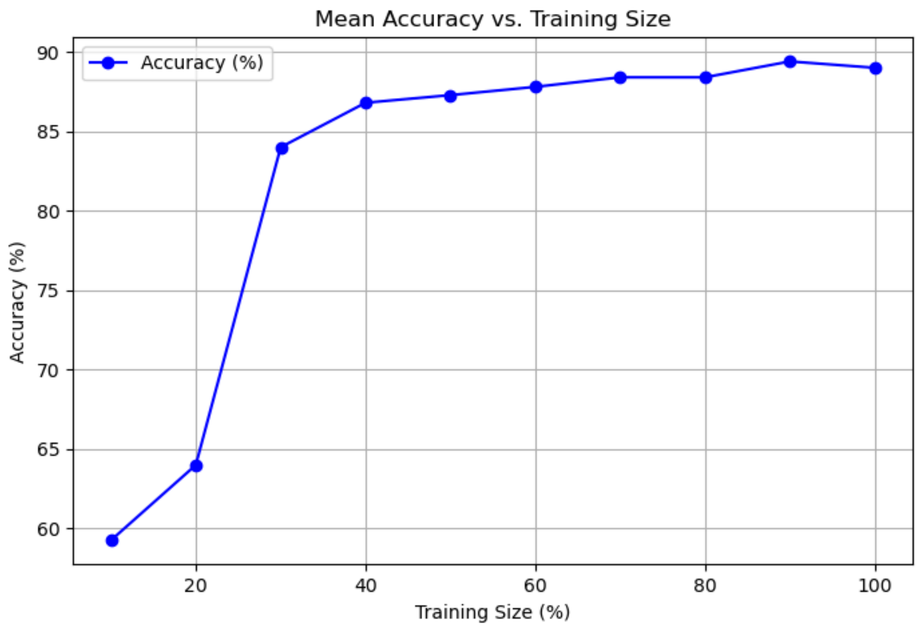

Standard Deviation vs. Training Size
Std Dev (%)
Std Dev (%)
Training Size (%)
1.5
2.0
2.5
3.0
3.5
4.0
4.5
20
40
60
80
100

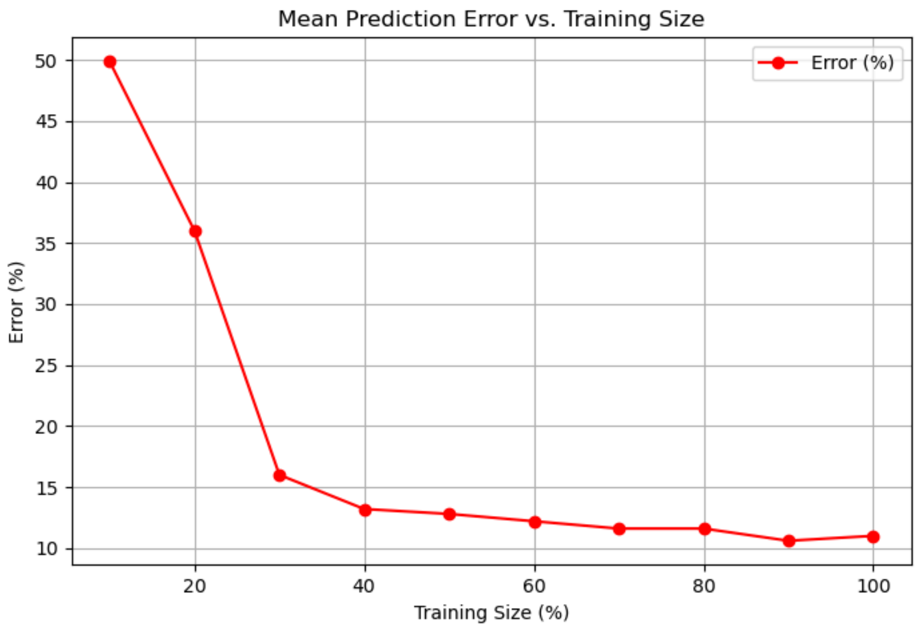

Training Time vs. Training Size
Time (s)
Time (seconds)
Training Size (%)
2
4
6
8
10
20
40
60
80
100

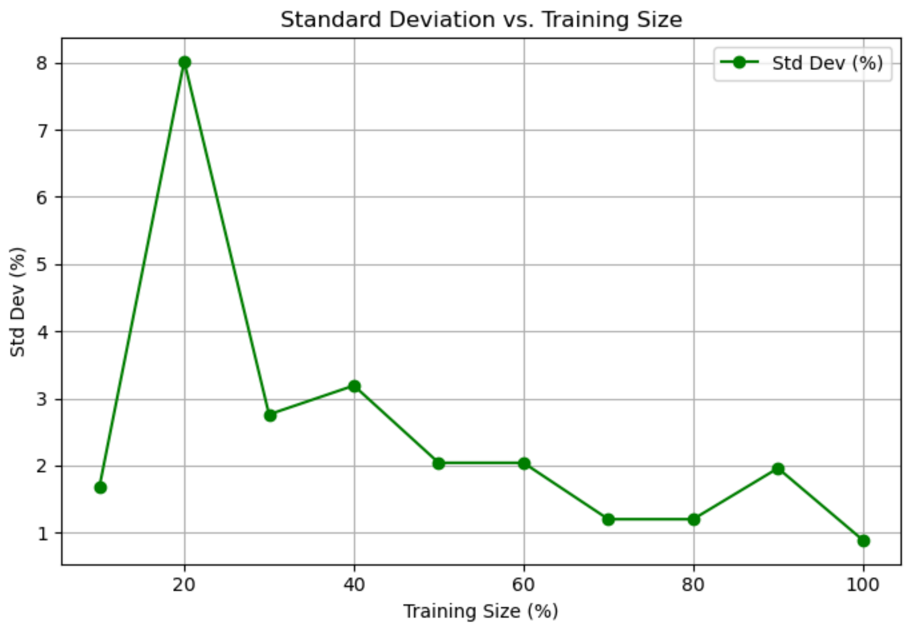
Faces Pytorch Neural Network Results

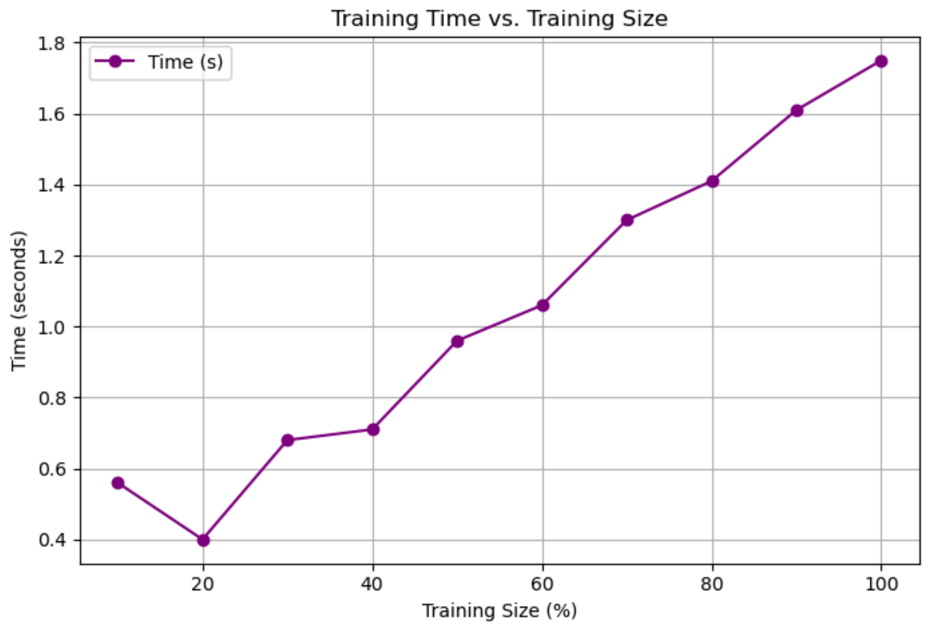


```
=== Results ===
Percentage | Training Size | Mean Accuracy (%) | Mean Prediction Error (%) | Std Dev (%) | Mean Time (s)
       10% |            45 |            50.20% |                    49.80% |       1.60% |          0.56
       20% |            90 |            64.00% |                    36.00% |       8.02% |          0.40
       30% |           135 |            84.00% |                    16.00% |       2.76% |          0.68
       40% |           180 |            86.80% |                    13.20% |       3.19% |          0.71
       50% |           225 |            87.20% |                    12.80% |       2.04% |          0.96
       60% |           270 |            87.80% |                    12.20% |       2.04% |          1.06
       70% |           315 |            88.40% |                    11.60% |       1.20% |          1.30
       80% |           360 |            88.40% |                    11.60% |       1.20% |          1.41
       90% |           405 |            89.40% |                    10.60% |       1.96% |          1.61
      100% |           451 |            89.00% |                    11.00% |       0.89% |          1.75
```









Perceptron vs Neural Networks

Due to the neural networks' learning capabilities, they were able to spend less time in comparison to perceptron as the training sizes increased. When we look at the graphs, an example we can see is for digits, the Perceptron took a maximum of 50 seconds, while the neural networks took a maximum of 11 seconds. The Neural networks were also slightly more accurate in comparison to Perceptron. We did have a case where the Perceptron algorithm had a convergence that led to a 0% standard deviation while no such incident happened with the neural networks, but otherwise there was no other meaningful difference between the standard deviations.