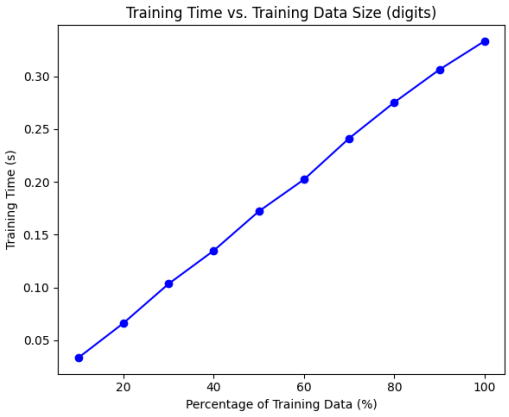
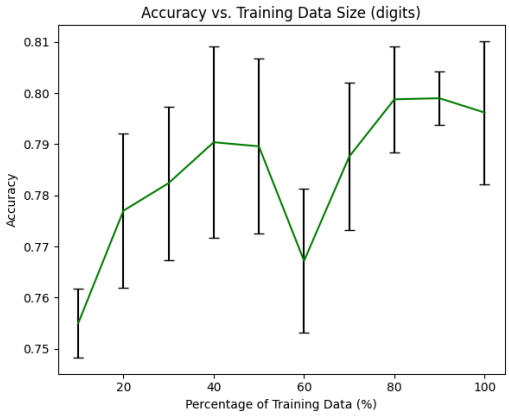
Face and Digit Classification Using Neural Networks

Varun Doreswamy (208008226), Nicholas Kushnir (207004422), Seth Yeh (207006882)

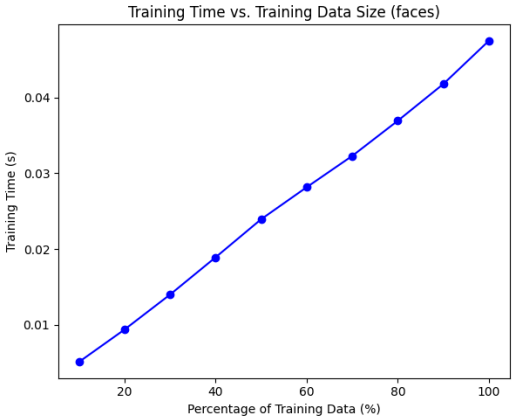
In this project, we used three classifiers to evaluate performance on digit and face recognition. The first classifier was a Perceptron classifier, the second was a 3-layer neural network, and third, a 3-layer PyTorch based neural network. Each of these classifiers were trained and analyzed on two sets of data, digitdata and facedata. Digitdata contains grayscale 28x28 images of digits, labeled as 0-9, while facedata contains data of 70x60 faces labeled 1 or 0, face or not face.

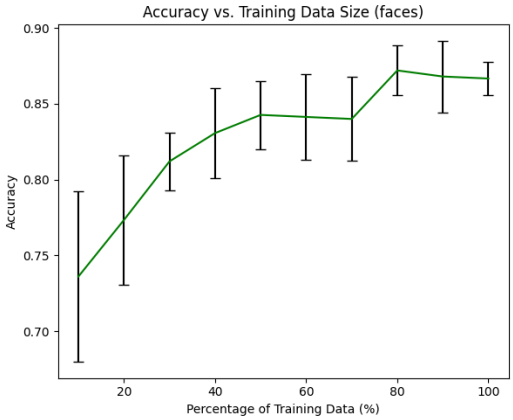
- **Perceptron Performance**
  - The perceptron classifier works by updating weights when a misclassification occurs, and then continues to classify based on the improvement.
  - **Digits Performance**





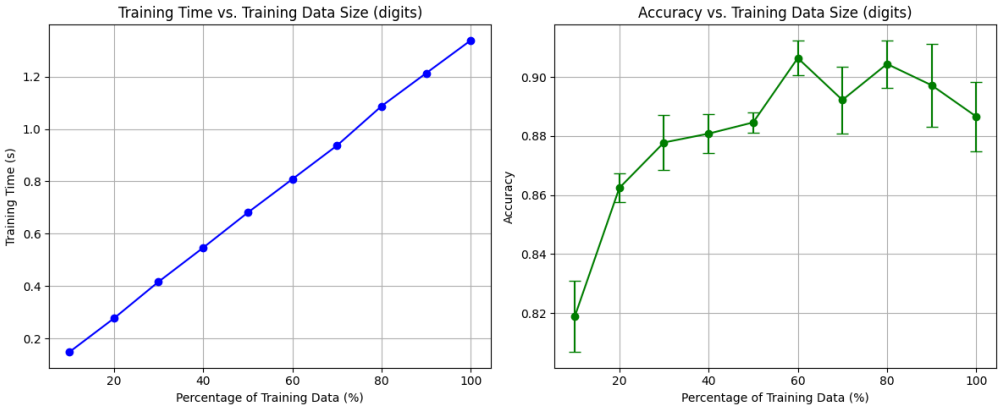
    - Accuracy grew from ~75% to ~80% with more data, but was somewhat inconsistent.
    - Training time is linearly consistent with training data %.
  - **Faces Performance**





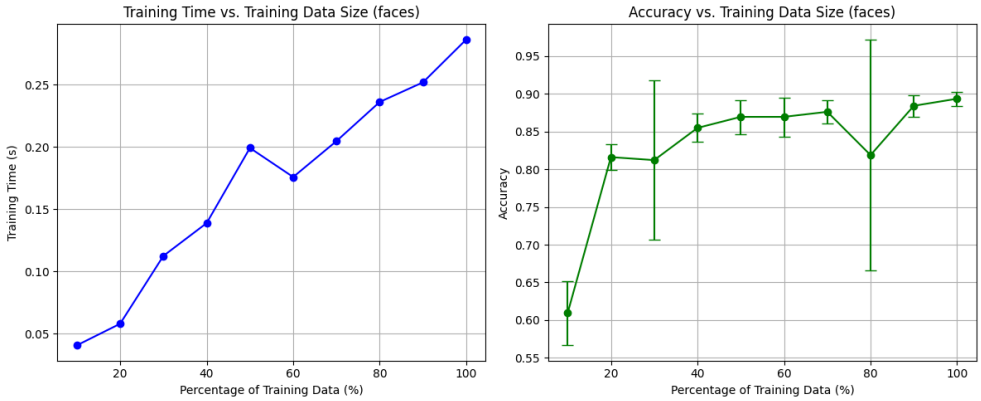
    - Accuracy grew from ~73% to ~87% as more data was used

        - Training time remained nearly negligible across all training sizes (<0.05s)
    - **Overall Conclusions**
        - For complex patterns like the various digits, the Perceptron fails to classify them quickly.
        - On the other hand, in binary tasks like face or not face, the Perceptron algorithm works well with fast convergence.
- **<u>Three-Layer Neural Network</u>**
    - When implementing our own three layer neural network we decided to use two different activation functions, relu for the hidden layers and softmax for the output layer. When training the network we noticed much better performance when using relu as our activation function so we decided to use that for the hidden layers. Furthermore, softmax was used as our activation function for the output layer because it is known to be used for multiclass probability distributions which we had to deal with due to half of the data being digits. We also decided to use a momentum gradient descent as it helps accelerate the training process by smoothing out noisy gradients and allows the model to traverse through local minima more efficiently. The only difference from a standard gradient descent is that it gives the model a “velocity” that helps it continue moving in the direction of the most recent strong gradients, even if the current gradient is small or noisy. The backward pass manually calculates gradients via chain rule, applying derivatives through each layer to propagate error and adjust weights. The training process iteratively feeds data through the network, computes loss using cross-entropy, and updates weights over multiple epochs. After training, the model’s accuracy is evaluated on a test set. To analyze the learning process, the algorithm trains the network on varying portions of the dataset (10% to 100%) and records performance metrics such as accuracy and training time. Finally, it visualizes the relationship between data usage and model performance using plots.
    - **Digits Performance**



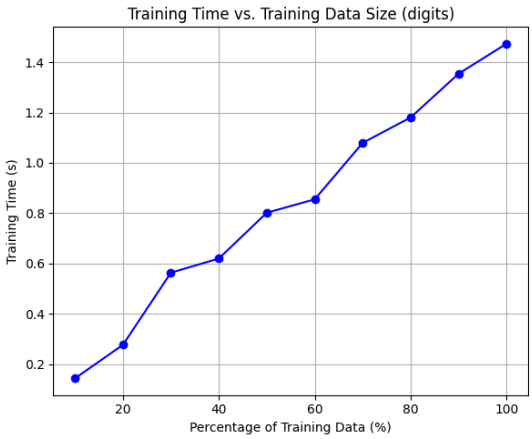
        - Accuracy reached ~91%, with more variance early on in the training, but stabilizing after 50% of the training data.
        - Not completely stable with accuracy after 60%.
        - Training time was also linear, peaking at 1.4 seconds at 100%.

- **Faces Performance**

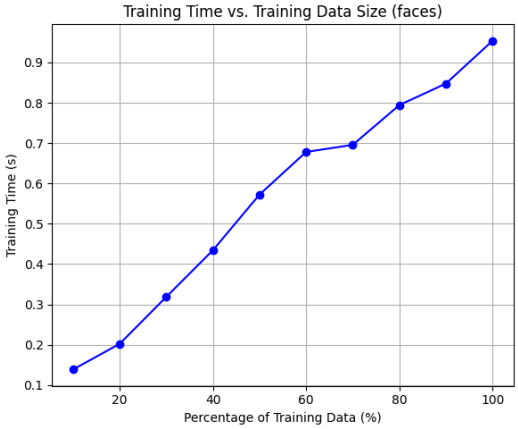


  -
  - Accuracy reached ~89% with higher variation at smaller data sizes.
  - Faces were quicker, likely because of the smaller dataset, and Overfitting could have happened at around 80% training data, because of smaller model size.
- **Conclusions**
  - This three-layer neural network model can more accurately classify digits and faces at a reasonable pace (slower than a perceptron) but still somewhat fast.
  - Better at classifying digits than faces, perhaps suggesting the algorithm is better at multi-level or complex classifications.

- **<u>Three-Layer Neural Network using PyTorch</u>**
  - The images are first converted into numerical tensors and organized using a custom Dataset, which makes it easy to batch and randomly sample the data with DataLoaders. The neural network has two hidden layers with ReLU activations and an output layer designed for classification, using CrossEntropyLoss to handle softmax and calculate errors. Training uses the Adam (adaptive moment estimation) optimizer along with PyTorch's automatic differentiation to compute gradients and update weights. The adam optimizer keeps track of the average of past gradients which helps smooth updates. This works similarly to the momentum gradient we used in our own neural network before. It also keeps track of the average of the squared gradients which helps slow down updates for parameters that receive large gradients, resulting in stable learning. The model's performance is tested by measuring accuracy on a separate test set. To better understand how training size affects results, experiments are run using different amounts of training data, from 10% up to 100%.
  - **Digits Performance**



-
    - Accuracy improved smoothly with data, reaching ~91% at 100%, also had the most stable accuracy curve
    - Training time was longer than Perceptron and the previous model, but still efficient.
- **Faces Performance**



-
    - Accuracy exceeded 90% with only 60% data, and remained relatively stable.
    - Outperformed previous models on consistency and convergence.
- **Conclusion**
    - Performance was very accurate and stable, even at lower data sizes, and converged relatively quickly, reaching 90% in just nearly 50% of the data size.
    - It was also similarly accurate between faces and digits, meaning the algorithm is flexible
    - Between the three, the PyTorch implementation had the best overall speed, accuracy, consistency, and performance.

Final Conclusions

- While all models increased with training model size, most models had diminishing returns after ~80%

- Perceptron is fast and decent for binary tasks, but slower and limited to multi-class classifications.
- The three-layer neural network is flexible, generally consistent, and efficient, but is sensitive to sample size and learning rate
- The PyTorch neural network had a fast convergence, low variance, and high accuracy, making it the overall best pick.