Final Project - Face and Digit Classification

Arya Desai (ahd61) & Aryaman Patel (arp236)

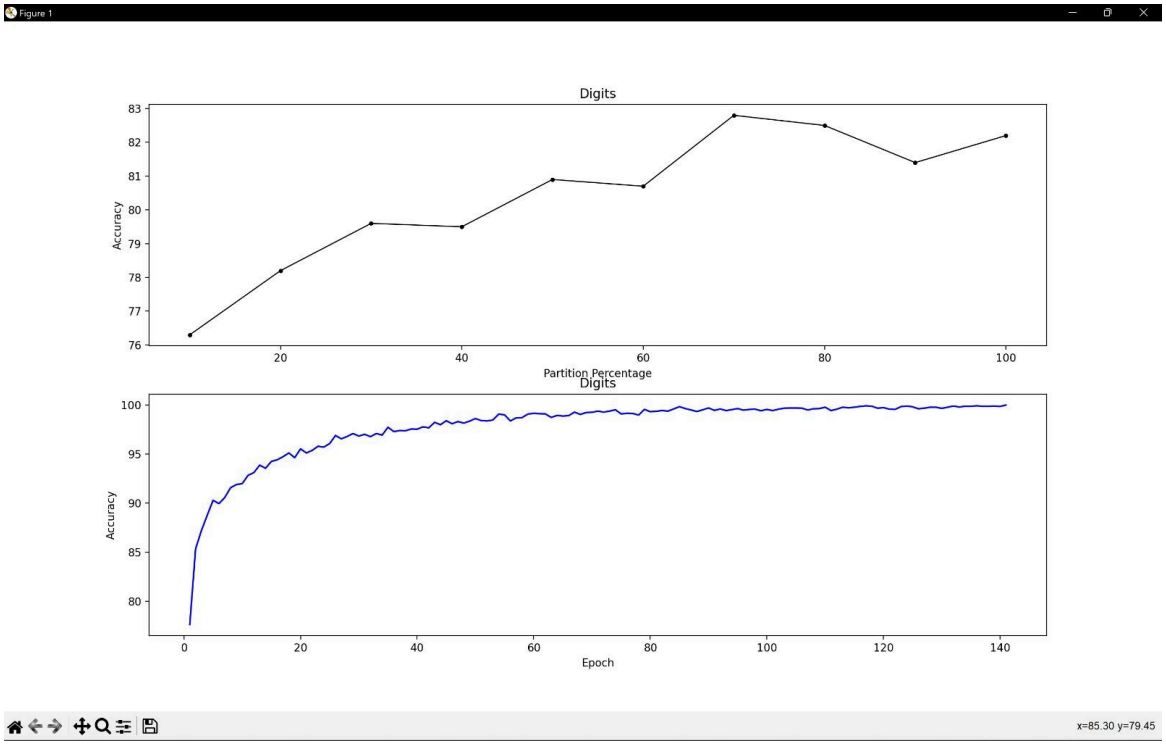
In this project, we will design and implement three classifiers: a two-layer neural network, a perceptron classifier, and a classifier of our choice. These classifiers will be tested on two distinct image datasets: one containing scanned handwritten digit images and the other comprising face images where edges have already been detected. The goal is to demonstrate the classifiers' performance in optical character recognition (OCR) and simplified face detection tasks. OCR involves extracting text from image sources, akin to the technology used by postal services for routing mail by zip codes. Notably, some systems achieve over 99% classification accuracy in OCR tasks. Face detection, on the other hand, involves localizing faces within images or video frames and finds applications in various fields like human-computer interaction and surveillance. For this project, we will simplify the face detection task by providing pre-processed edge images, and the classifiers will determine whether the image contains a face or not. We will conduct the assignment using two fundamental algorithms: the perceptron and the two-layer neural network. These algorithms will be evaluated for their effectiveness in both OCR and face detection tasks.

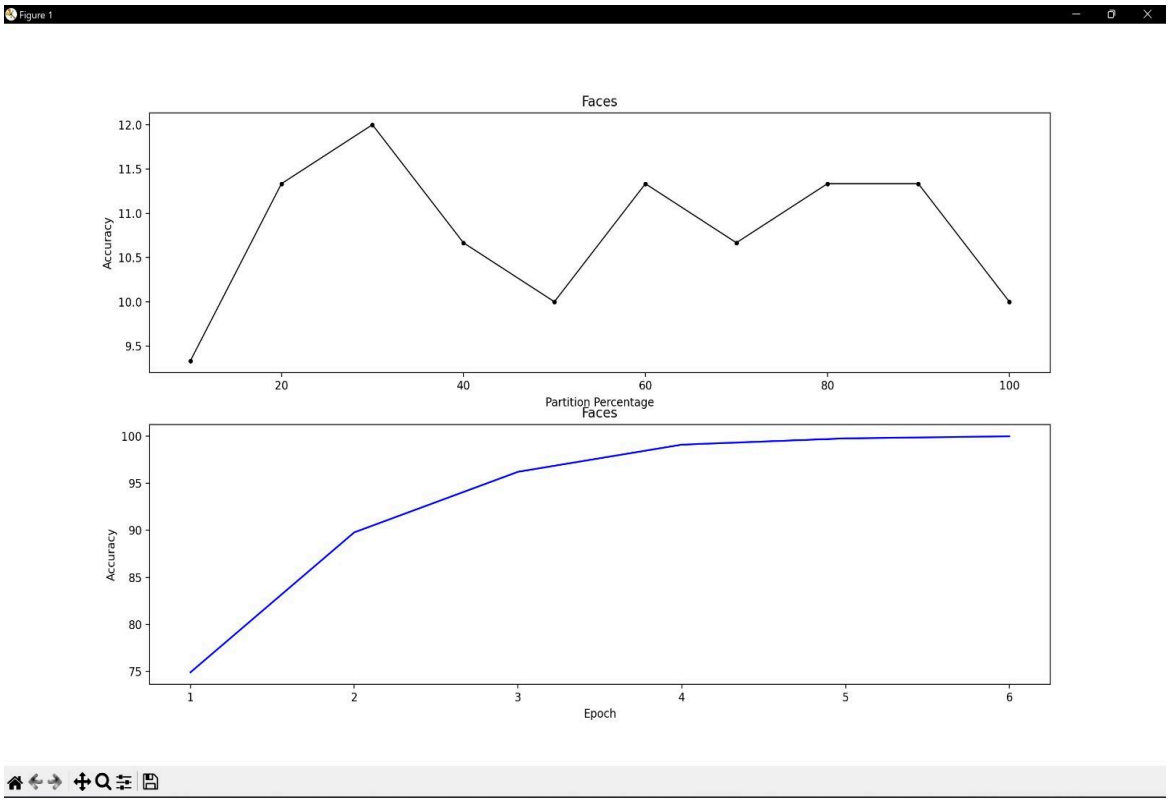
Perceptron Algorithm

A perceptron is one of the simplest types of artificial neural networks. It's a binary classifier used for supervised learning of binary classifiers, meaning it determines whether an input belongs to one class or another based on its features.

This Python script implements a perceptron classifier for two distinct datasets: handwritten digit images and face images. The code begins by loading and preprocessing the image and label data using a custom module called `readdata.py`, which transforms the images into feature matrices and prepares the corresponding label data. The perceptron model is then trained on the training data using a specified number of epochs, with weights updated based on misclassifications during each epoch. The training process is monitored for accuracy, and the accuracy values for each epoch are recorded for analysis. Subsequently, the trained perceptron model is tested on the separate test dataset to evaluate its classification performance. Multiple experiments are conducted, gradually increasing the percentage of training data used, and the accuracy of the perceptron classifier is plotted against the percentage of training data employed, offering insights into its performance sensitivity to training dataset size. Additionally, the script visualizes the accuracy of the perceptron over each epoch during training, illustrating its convergence behavior. Finally, the script reports the testing accuracies achieved by the perceptron classifier for each experiment, along with the total training time and overall execution time for running the

experiments. Overall, this script provides a comprehensive analysis of the perceptron classifier's performance on both handwritten digit and face image datasets, offering insights into its efficacy and behavior under varying training conditions.





Two Layer Neural Network

A two-layer neural network, also known as a shallow neural network or a multi-layer perceptron (MLP), is a fundamental architecture in deep learning. It comprises an input layer, a hidden layer, and an output layer. The input layer represents the features of the data, with each neuron corresponding to a feature. In the hidden layer, neurons receive inputs from the input layer, compute weighted sums of these inputs, and apply an activation function to introduce non-linearity. This layer learns to extract relevant features from the input data. The output layer generates predictions based on the learned features, with the number of neurons determined by the nature of the task (e.g., binary or multi-class classification, regression). During training, the network adjusts its weights and biases using optimization algorithms like gradient descent and backpropagation, which iteratively minimize a loss function measuring the disparity between predicted and true outputs. Activation functions like sigmoid, hyperbolic tangent, or ReLU introduce non-linearities crucial for learning complex relationships in data. While capable of approximating diverse functions and patterns, two-layer neural networks may struggle with highly complex tasks, often requiring deeper architectures for improved performance and representation learning.

Digits Neural Network

This Python script orchestrates a two-layer neural network to classify digits from the MNIST dataset. It initiates by preprocessing the training and testing data, encoding labels into one-hot vectors, and flattening the images. Optimization functions are then employed to iteratively update the weights and biases of the neural network using gradient descent and backpropagation. The sigmoid activation function introduces non-linearity into the network, essential for capturing complex patterns in the data. The model training process initializes the network parameters and refines them through multiple iterations, returning optimized weights and biases. The script evaluates the trained model's performance on the test data, computing accuracy metrics to gauge its effectiveness. Finally, matplotlib is utilized to generate visualizations depicting the relationship between training time and test accuracy as a function of the percentage of training data used. While proficient in illustrating the capabilities of a two-layer neural network for digit classification, further refinements such as hyperparameter tuning and exploration of alternative activation functions could enhance model performance and robustness. Additionally, consideration of more advanced architectures like convolutional neural networks (CNNs) may be warranted for tasks requiring deeper feature extraction and hierarchical learning. Here are our results :

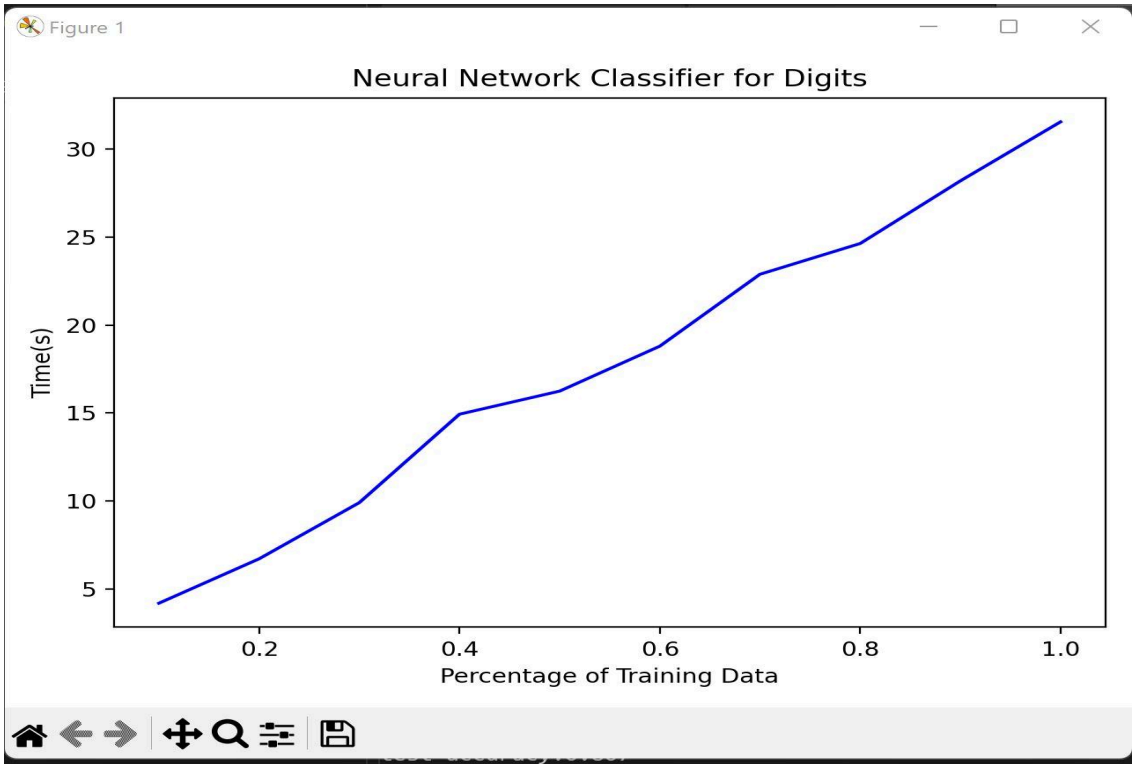


Figure 1
Neural Network Classifier for Digits
Time(s)
30
25
20
15
10
5
0.2
0.4
0.6
0.8
1.0
Percentage of Training Data

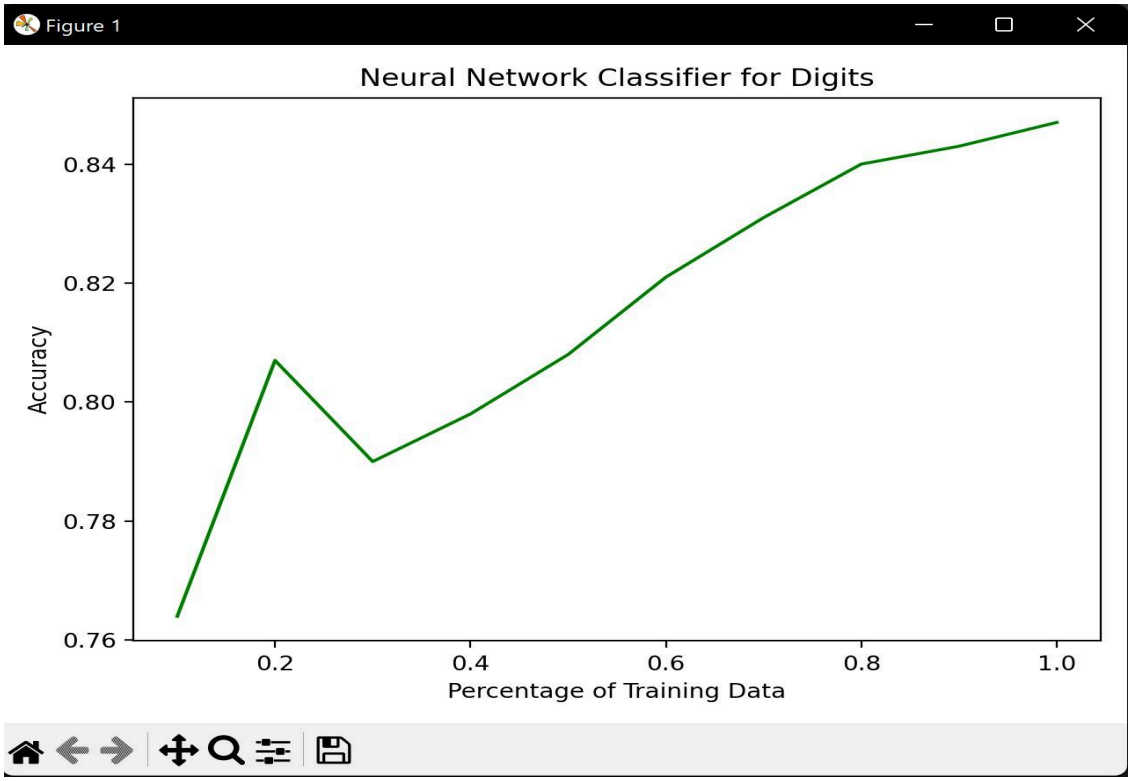
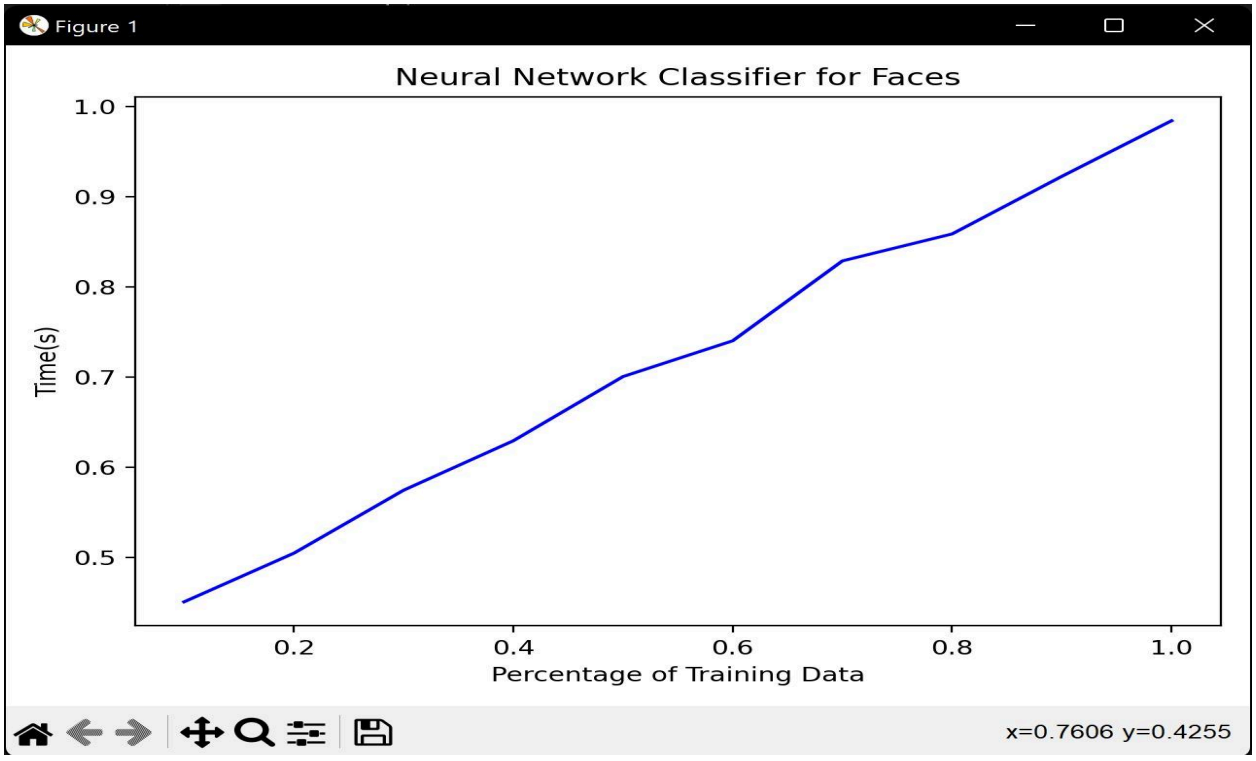
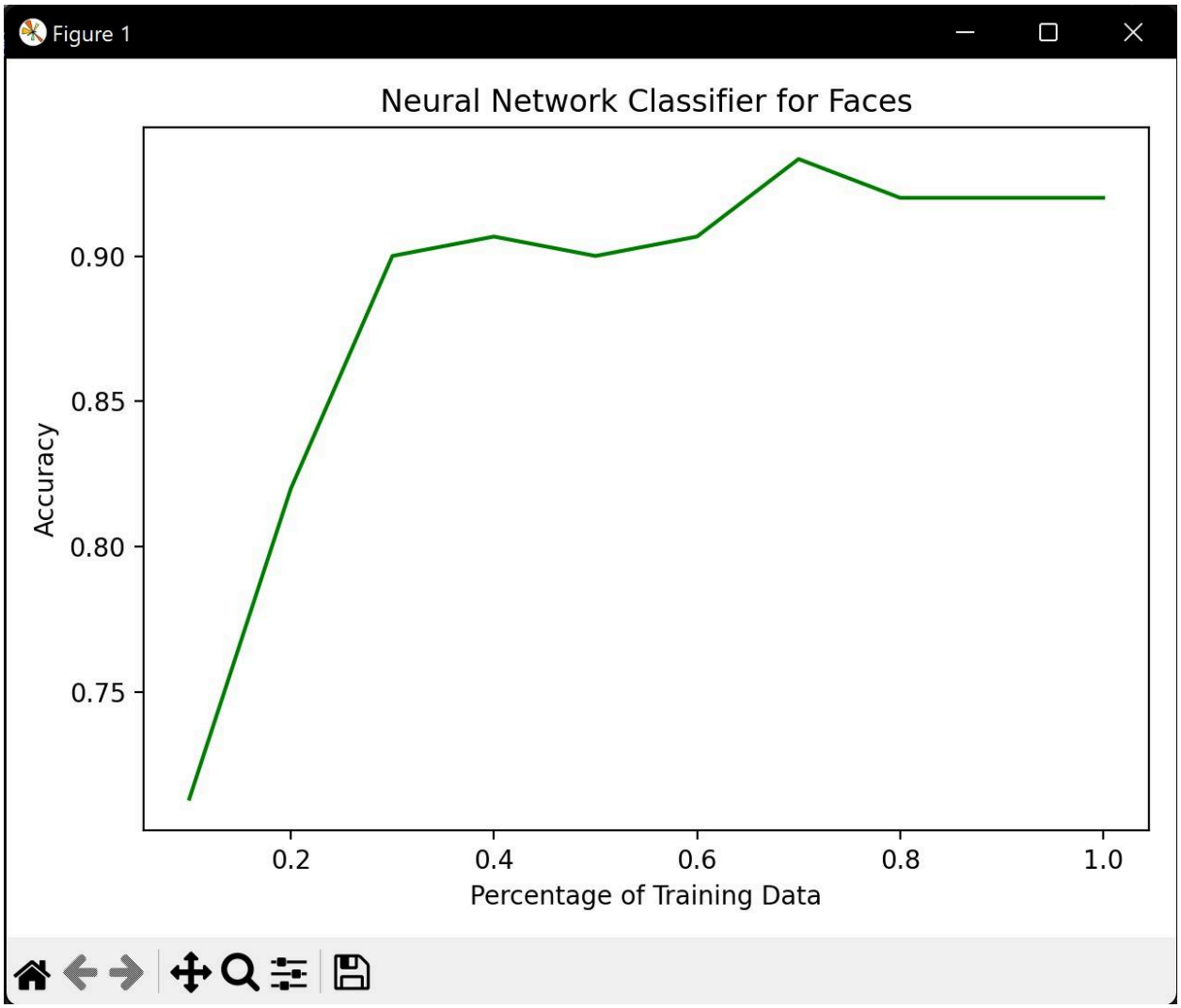


Figure 1
Neural Network Classifier for Digits
Accuracy
0.84
0.82
0.80
0.78
0.76
0.2
0.4
0.6
0.8
1.0
Percentage of Training Data

Faces Neural Network

This Python script implements a two-layer neural network for facial recognition using the Yale Face Database. The code begins by loading facial images and corresponding labels from specified files and preprocesses the data by downsampling the images to reduce dimensionality. Gradient descent optimization is performed in the `optimization` function to update the weights and biases of the neural network based on backpropagation computed in the `propagation` function. Sigmoid activation is utilized to introduce non-linearity into the network, crucial for capturing complex patterns in the data. The `predict` function assigns binary class labels to the test data based on the learned network parameters. Model training is facilitated by the `model` function, initializing and training the network using the training data. Visualization of training time and test accuracy as functions of the percentage of training data used is achieved through the `plot` function. Finally, the script orchestrates the entire process in the main function, encompassing data loading, model training, evaluation, and result plotting. While demonstrating the basic implementation of a neural network for facial recognition, further enhancements such as hyperparameter tuning and exploring alternative architectures could bolster performance, particularly for more complex tasks. Additionally, comprehensive evaluation metrics beyond accuracy may offer deeper insights into the model's efficacy and generalization capabilities.





Difference of two Algorithms

The perceptron, inspired by early neural network concepts, consists of a single layer of neurons directly connected to an output node. It operates by computing a weighted sum of input features and applying a step function to determine the output, making it suitable for binary classification tasks. However, its simplicity limits its capability to learn only linear decision boundaries, which restricts its effectiveness on more complex datasets with non-linear patterns.

On the other hand, a two-layer neural network, or multi-layer perceptron (MLP), extends beyond the perceptron's architecture by incorporating an additional hidden layer between the input and output layers. This hidden layer introduces non-linear activation functions, such as sigmoid or ReLU, allowing the network to capture and represent complex, non-linear relationships in the data. As a result, the two-layer neural network can learn and adapt to more intricate patterns, making it suitable for a broader range of tasks, including multi-class classification and

regression. Its hierarchical architecture enables it to extract hierarchical features from the data, enhancing its flexibility and performance compared to the perceptron.

To conclude let’s discuss the difference of both algorithms when detecting digits. The graphs compare the performance of a neural network classifier and a perceptron on digit datasets. The first graph, which focuses on the neural network classifier, reveals a positive correlation between the amount of training data and accuracy. As the percentage of training data increases, the accuracy consistently improves, reaching about 84%. However, training time also rises steadily, which is expected given the complexity of the model. In the second graph, the perceptron classifier displays a similar relationship between the training data percentage and accuracy, also reaching about 84%. However, the perceptron exhibits more fluctuations, with a noticeable dip early on. This variation indicates that perceptrons are more sensitive to changes in training data partitioning.The final graph combines both models for a comprehensive comparison. The perceptron's top plot illustrates pronounced accuracy variations across different data partitions, while the neural network's bottom plot, mapping accuracy against training epochs, demonstrates a steady and consistent improvement. In summary, neural networks show more stability and robustness in accuracy trends, although they require more training time. Perceptrons, while achieving comparable peak accuracy, are more affected by data partitioning, leading to greater fluctuations. Ultimately, neural networks provide a more consistent performance but at the cost of increased training time.

To conclude let’s discuss the difference of both algorithms when detecting faces. The graphs provided offer insights into the performance differences between a neural network and a perceptron model when classifying faces. The neural network classifier achieves rapid improvements in accuracy, stabilizing around 90% accuracy after training with just 40% of the data. This accuracy remains consistent despite a slight dip when the training data reaches 80%, suggesting that the neural network can efficiently generalize. However, the training time increases linearly with the percentage of training data, starting at approximately 0.5 seconds and rising to 1 second. On the other hand, the perceptron classifier shows fluctuating accuracy across partition percentages, but when evaluated across training epochs, it gradually improves, reaching nearly 98% accuracy after six epochs. This indicates that while the perceptron requires more training iterations to achieve high accuracy, it ultimately outperforms the neural network in this regard. In summary, the neural network offers quicker stabilization in accuracy, reaching a consistent level more efficiently. However, the perceptron requires more epochs to train effectively, yet achieves a higher overall accuracy in the end. Thus, the neural network is faster in generalization, while the perceptron ultimately delivers higher accuracy with more training.