CS440- Final Project

Project: Face and Digit Classification

By: Shubham Patel (sp2136); Deep Shah (djs504)

Introduction:

In this project, our team implemented two classification algorithms for detecting faces and classifying digits: the Perceptron and a Two-layer Neural Network. We trained and tested these algorithms on two datasets: handwritten numerical digits and edge-detected face images.

Implementation:

According to the project guidelines, we created the Perceptron and Two-layer Neural Network algorithms from beginning. We also created features for each problem, including features that were simply composed of raw pixels. To make sure the programs worked with the algorithms, we created programs to extract features from each image.

Data:

- Preprocessing:

To begin with, the dimensions of two datasets and every image are acquired. Subsequently, each file undergoes conversion into a matrix of size [i, j, k], where 'i' denotes the sample count, 'j' represents the row number, and 'k' indicates the column number. Utilizing this data structure, we can conveniently derive the feature map by employing the sum pooling technique for each image. Finally, within the images (as illustrated in Fig. 1), symbols "#" and "+" are substituted with "1", while vacant spaces are replaced with "0".

- Feature extraction:

Due to the small size of single-digit images, every pixel in such images is considered a feature. However, face images typically have larger dimensions, which can negatively impact training speed and model accuracy. Consequently, in the implementation of both algorithms, the sum pooling method is also applied to face images. This approach aims to mitigate the challenges posed by larger image sizes. A comparison is then made between the effectiveness of utilizing single pixel features versus employing sum pooling.

Methods:

Face image detection involves binary classification, while detecting digits involves multi-class classification.

Algorithms:

1. Perceptron:

The fundamental concept of the perceptron revolves around identifying a hyperplane (as denoted by Equation (1)) that segregates input samples into two categories: positive $(where\ f > 0\ and\ \hat{y} = 1)$ and negative $(where\ f < 0\ and\ \hat{y} = -1)$.

$$f = w * \varphi + b \ (1)$$

During training, the parameters w and b are initially set to random values and 1, respectively. Subsequently, if the predicted output $\hat{y}$ differs from the true label y, w and b are updated (as per Equations (3) and (4)) using their gradients (as per Equations (5) and (6)) with respect to the loss function (Equation (2)).

$$loss = -\Sigma iyi(w\varphi i + b)\ (2)$$

$$w = w - \Delta w\ (3)$$

$$b = b - \Delta b\ (4)$$

$$\Delta w = -yi\varphi i\ (5)$$

$$\Delta b = -yi\ (6)$$

The updating process continues until every y_i matches its corresponding $\hat{y}_i$. The resulting w and b are then used to predict the labels for test samples and determine the model's accuracy (prediction error). It's important to note that the perceptron's linear hyperplane is limited to binary classification. To address this, the perceptron method is used by incorporating a max function, introducing non-linearity, for label determination. This overcomes the perceptron's inherent binary classification limitation. Additionally, pixel values (features) are used as φ.

2. Neural Network:

The neural network consists of an input layer, a hidden layer, and an output layer.

From input layer to hidden layer:

$$f = w * x + b\ (11)$$

Here, w = weights, b = bias and x = the input features.

at hidden layer (production of activation values):

$$activation = sigmoid(f)\ (12)$$

from hidden layer to output:

$$\hat{y} = argmax(activation)\ (13)$$

The optimization of w and b is achieved through differential with respect to the loss function (back propagation), where the loss function is defined as cross-entropy.

$$loss = \frac{-1}{m}\Sigma_i y_i log\hat{y}_i + (1 - y_i)\log(1 - \hat{y}_i)\ \ (14)$$

$$w = w - \Delta w\ (15)$$

$$b = b - \Delta b\ (16)$$

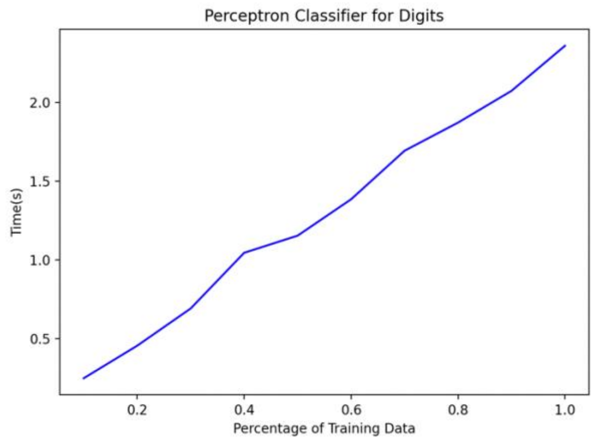
$$\Delta = \frac{1}{m} \sum_i \ (\hat{y}_i - y_i) x_i \ (17)$$

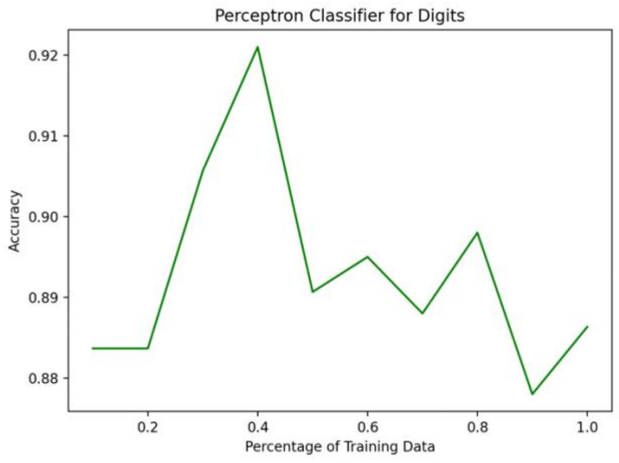
$$\Delta b = \frac{1}{m} \sum_i \ (\hat{y}_i - y_i) \ (18)$$

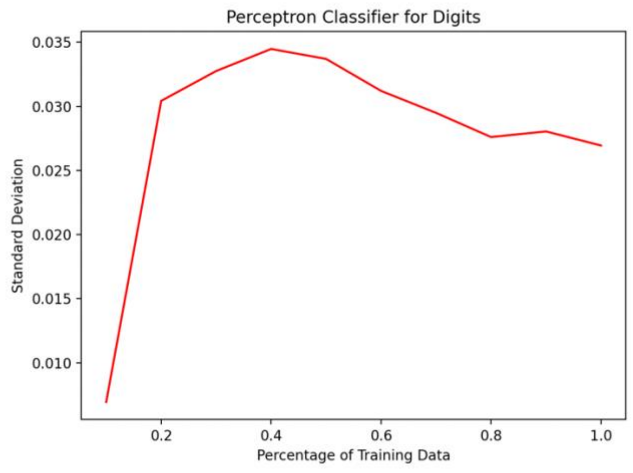
The optimization function undergoes 2000 iterations before producing the final w and b values. Predictions for test data labels are made using Equation (13). In multi-class classification for digits, labels are converted to one-hot format to suit the loss function (Equation 14). When comparing perceptron to a neural network, perceptron behaves like a basic single layer network, yet constrained to binary classification.

Result:

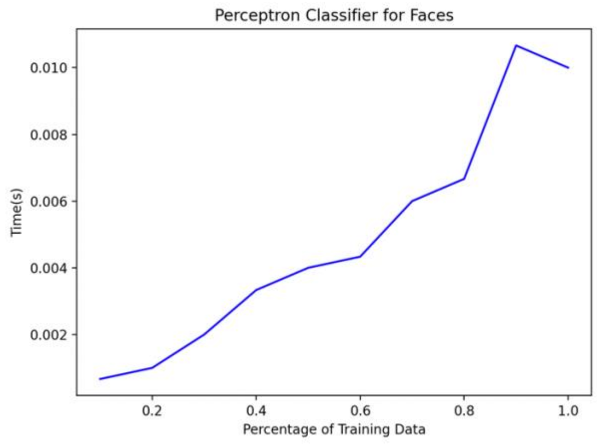
- **Perceptron**
  - **Digits**

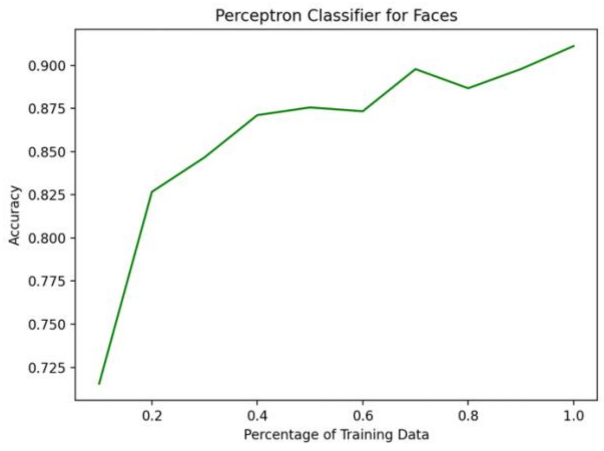


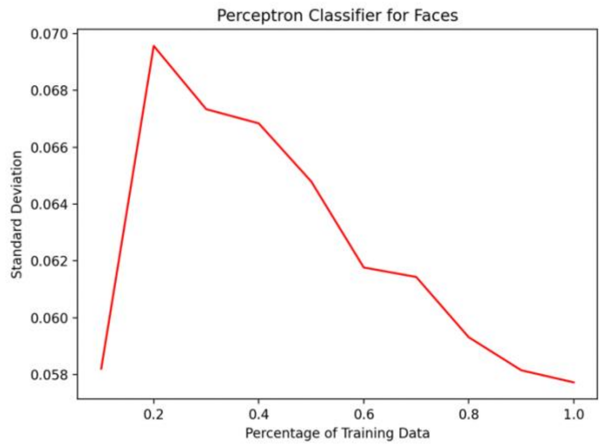




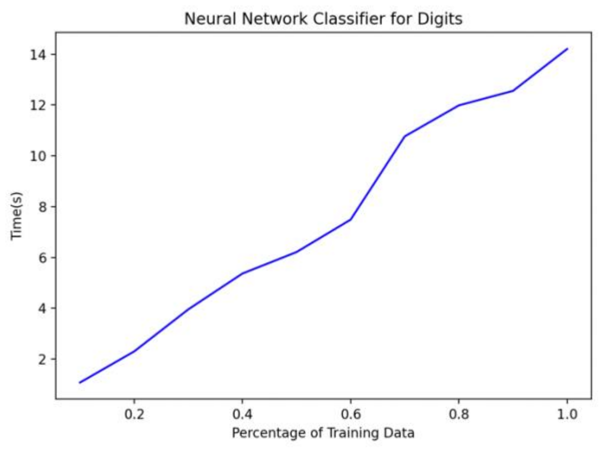
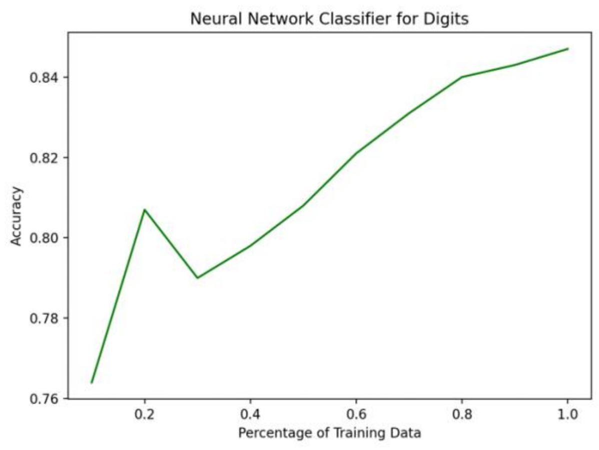
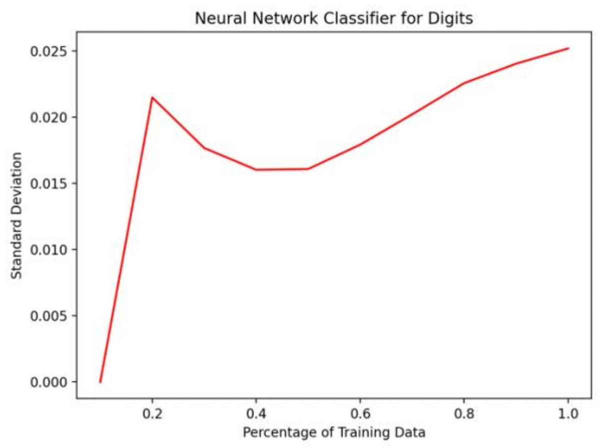
- **Faces**



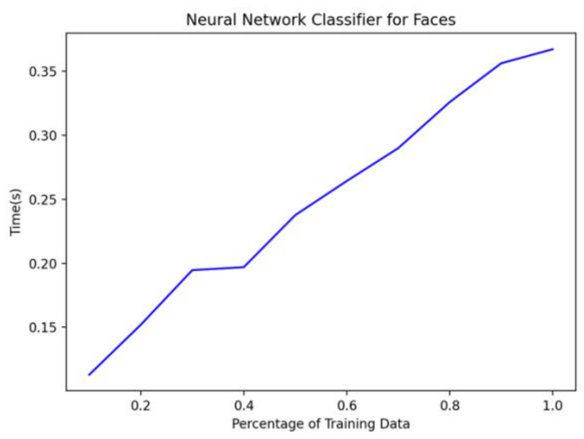


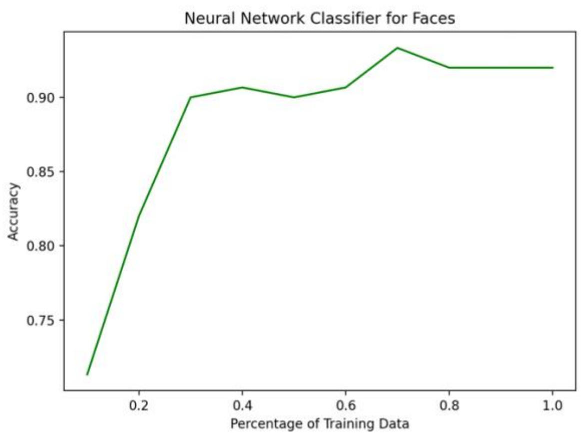


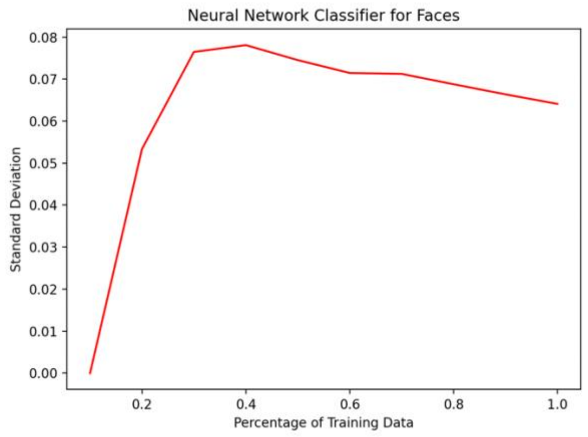
- **Neural Network**
  - **Digits**


Neural Network Classifier for Digits
Time(s)
Percentage of Training Data
14
12
10
8
6
4
2
0.2
0.4
0.6
0.8
1.0

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

Neural Network Classifier for Digits
Standard Deviation
Percentage of Training Data
0.025
0.020
0.015
0.010
0.005
0.000
0.2
0.4
0.6
0.8
1.0

- **Faces**







Accuracy:

	Face image	Digit image
Perceptron	0.872	0.895
Neural Network	0.894	0.820

Across all algorithms, we achieved accuracy levels passing 70% after sufficient training. The time required for training directly varied to the volume of training data; more data resulted in longer training durations, with the perceptron algorithm exhibiting the longest times. Accuracy didn't consistently align with the number of iterations. Typically, it began low, increased midway, converged, and experienced fluctuations. There were instances where accuracy dropped even after using the entire dataset. The standard deviation demonstrated an inverse relationship with the total amount of training data used. Sometimes, the standard deviation went up in case of overfitting.

Conclusion:

One of the key takeaways we’ve gathered is that simply adding more training data doesn’t automatically lead to higher accuracy. The quality of the training data is not certain, and its effectiveness remains unknown until tested. Moreover, dedicating more time to training a model doesn’t always result in better accuracy. Hence, there’s no need to extend the training process by increasing the number of iterations without any reason.