Handwritten Digit Recognition Using Neural Networks

TEAM MEMBERS: Anjali Pathak: MSCS
Mohitha Gowda: MSCS

CS6140 - Machine Learning, Project Presentation

What is Neural Network

What is Neural Network?

Neural Networks in simple terms:

- **A Supervised Machine Learning Model.**
- **Inspired by the way our brain works.**
- **Solves problems and makes decisions.**
- **Learns from examples to recognize patterns**

Problem Statement

Problem Statement

- **A fundamental problem in the field of machine learning and computer vision.**
- **Involves developing a system that can accurately recognize and classify handwritten digits from 0 to 9.**
- **Aim : to build a neural network from scratch that can accurately recognize a given image as one of the possible 10 digits .**

Significance of the Problem

Significance of the Problem

- Has numerous real-world applications:
  1. Digitizing historical documents
  2. Automating data entry
  3. Enabling signature verification
- Serves as a foundational concept in the development of more complex image recognition systems.

Background

Previous Solutions and Approaches

- Advancements in handwritten digit recognition using deep neural networks.
- Common models: Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs).
- Frameworks: TensorFlow, PyTorch.

Drawbacks of Existing Solutions

- Complexity and computational expenses.
- Dependency on pre-trained models.

Proposed Solution

Model/Framework/Approach Description

- Develop a neural network using Python and NumPy.
- One or two hidden layers, adjustable hyperparameters.
- Standard activation functions (ReLU), softmax output layer.
- Training with gradient descent-based optimization algorithms.

Data

Dataset Description and Types

- Use the MNIST dataset.
- 60,000 training images, 10,000 testing images.
- Grayscale 28x28 pixel handwritten digits (0-9).

Current Progress

Building the Neural Network

```
# Function to randomly initialize parameters (weights and biases)
# for the input layer and the first hidden layer
def init_params():
  W1 = np.random.rand(10, 784) - 0.5
  b1 = np.random.rand(10, 1) - 0.5
  W2 = np.random.rand(10, 10) - 0.5
  b2 = np.random.rand(10, 1) - 0.5
  return W1, b1, W2, b2

# Defining Activation Function (ReLU / Rectified Linear Unit)
# to activate the nodes of the first hidden layer
def ReLU(Z):
  return np.maximum(0,Z)

# Defining Softmax Function to calculate the probability of each node
# being the recognized digit in the output layer (0 - 9)
def softmax(Z):
  return np.exp(Z) / sum(np.exp(Z))

# Defining Forward Propogation Function, i.e.,
# one iteration of digit recognition
# starting from input layer with 784 nodes through output layer with 10 nodes
def forward_prop(W1, b1, W2, b2, X):
  Z1 = W1.dot(X) + b1
  A1 = ReLU(Z1)
  Z2 = W2.dot(A1) + b2
  A2 = softmax(Z2)
  return Z1, A1, Z2, A2

# Defining One Hot Encoding Function to one-hot encode the target label Y
# as a combination of 0s and 1s
def one_hot(Y):
  one_hot_Y = np.zeros((Y.size, Y.max() + 1))
  one_hot_Y[np.arange(Y.size), Y] = 1
  one_hot_Y = one_hot_Y.T
  return one_hot_Y
```

[7]

```
# Function to calculate the derivative of activation function
def deriv_ReLU(Z):
  return Z > 0

# Defining Back Propogation function, which basically calculates the derivatives
# in order to update the parameters later using learning rate alpha,
# so that our model can better learn and improve its predictions
def back_prop(Z1, A1, Z2, A2, W1, W2, X, Y):
  m = Y.size
  one_hot_Y = one_hot(Y)
  dZ2 = A2 - one_hot_Y
  dW2 = 1 / m * dZ2.dot(A1.T)
  db2 = 1 / m * np.sum(dZ2)
  dZ1 = W2.T.dot(dZ2) * deriv_ReLU(Z1)
  dW1 = 1 / m * dZ1.dot(X.T)
  db1 = 1 / m * np.sum(dZ1)
  return dW1, db1, dW2, db2

# Defing Update Parameters function to update the weights and biases for
# the input layer and the first hidden layer as learned with the help of
# back propogation function and the learning rate, alpha, after each iteration
# of forward propogation function
def update_params(W1, b1, W2, b2, dW1, db1, dW2, db2, alpha):
  W1 = W1 - alpha * dW1
  b1 = b1 = alpha * db1
  W2 = W2 - alpha * dW2
  b2 = b2 = alpha * db2
  return W1, b1, W2, b2
```

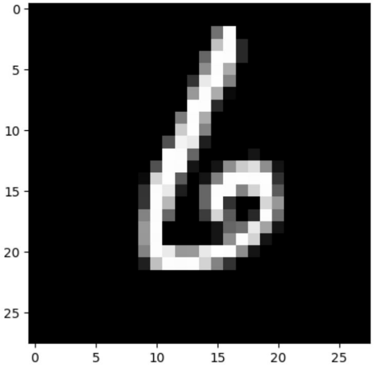
```python
# Function to return the predicted labels
def get_predictions(A2):
  return np.argmax(A2, 0)

# Function to calculate the accuracy of the model
def get_accuracy(predictions, Y):
  print(predictions, Y,"\nAccuracy: ")
  return np.sum(predictions == Y) / Y.size

# Defining the gradient descent function that takes the input training set,
# training set labels, number of iterations to be performed and learning rate
# as arguments.
def gradient_descent(X, Y, iterations, alpha):
  W1, b1, W2, b2 = init_params()
  for i in range(iterations):
    Z1, A1, Z2, A2 = forward_prop(W1, b1, W2, b2, X)
    dW1, db1, dW2, db2 = back_prop(Z1, A1, Z2, A2, W1, W2, X, Y)
    W1, b1, W2, b2 = update_params(W1, b1, W2, b2, dW1, db1, dW2, db2, alpha)
    if (i % 100 == 0):
      print("Iteration: " , i)
      print("Accuracy: ", get_accuracy(get_predictions(A2), Y))
  return W1, b1, W2, b2
```

```
W1, b1, W2, b2 = gradient_descent(X_train, Y_train, 500, 0.1)
```

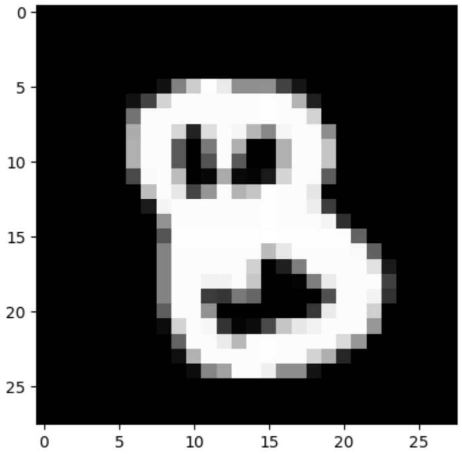
```
Iteration:  0
[2 7 2 ... 2 2 2] [5 6 6 ... 7 6 7]
Accuracy:
Accuracy:  0.08773170731707317
Iteration:  100
[2 6 6 ... 7 6 7] [5 6 6 ... 7 6 7]
Accuracy:
Accuracy:  0.5009756097560976
Iteration:  200
[2 6 6 ... 7 6 7] [5 6 6 ... 7 6 7]
Accuracy:
Accuracy:  0.7002926829268292
Iteration:  300
[5 6 6 ... 7 6 7] [5 6 6 ... 7 6 7]
Accuracy:
Accuracy:  0.7744634146341464
Iteration:  400
[5 6 6 ... 7 6 7] [5 6 6 ... 7 6 7]
Accuracy:
Accuracy:  0.8093658536585366
```

```
#verify the prediction
test_prediction(2, W1, b1, W2, b2)
test_prediction(8, W1, b1, W2, b2)
test_prediction(5, W1, b1, W2, b2)
test_prediction(4, W1, b1, W2, b2)
```

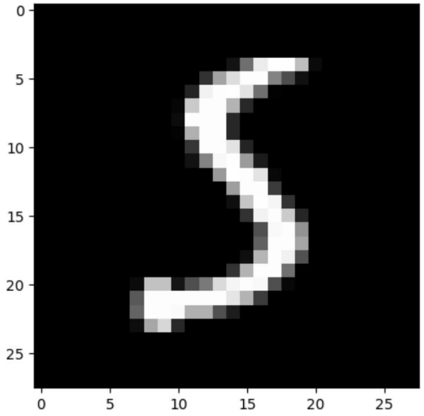
```
Prediction:  [6]
Label:  6
```



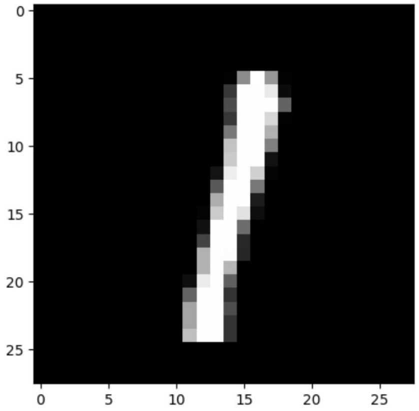

```
Prediction:  [8]
Label:  8
```




```
Prediction:  [3]
Label:  5
```




```
Prediction:  [1]
Label:  1
```

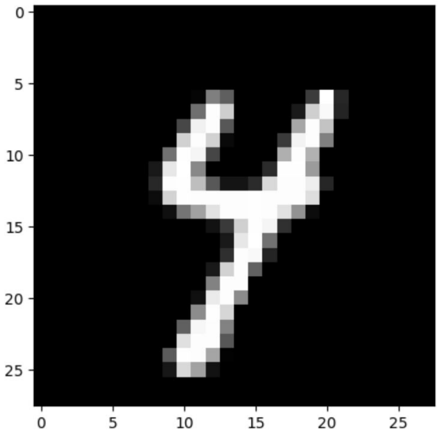


```python
#accuracy on test dataset

predict_test_data = predict(X_dev, W1, b1, W2, b2)
get_accuracy(predict_test_data, Y_dev)
```

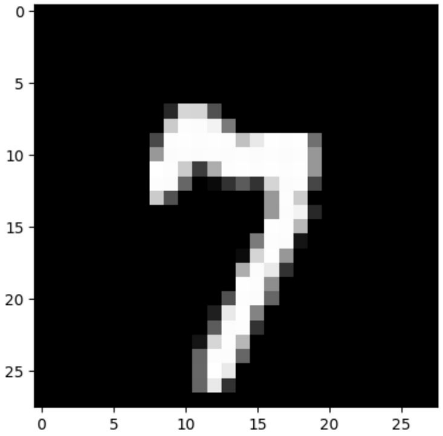
```
[7 8 2 5 0 1 2 8 2 2 5 5 9 5 3 1 3 6 7 4 2 7 6 5 7 3 7 0 1 8 5 1 5 2 8 1 2
 8 8 4 6 7 2 5 1 0 2 6 7 7 0 8 4 6 2 3 4 6 1 9 2 8 3 8 1 8 5 0 0 6 5 5 7 0
 9 3 3 2 1 7 8 7 4 3 6 4 1 5 6 8 1 4 3 2 9 3 4 4 6 1 5 7 1 1 4 7 3 0 8 4 9
 6 6 5 3 0 9 8 5 1 9 4 2 0 4 0 7 0 9 8 7 2 6 3 3 4 7 5 1 5 1 6 1 4 7 2 7 7
 8 0 1 4 3 4 2 6 7 7 6 6 1 4 3 0 9 8 4 4 7 3 7 7 7 1 8 2 2 2 1 0 8 8 6 9 6
 5 1 3 7 8 0 0 5 9 0 9 3 1 7 9 6 5 6 4 2 1 5 4 0 8 2 4 8 1 9 0 7 7 1 3 0 4
 7 8 4 6 7 6 7 7 4 7 8 6 1 8 0 7 0 2 7 3 6 5 1 0 1 9 6 9 9 0 1 1 5 9 1 9 7
 7 5 4 5 7 0 3 7 1 1 7 1 9 1 8 7 9 1 4 7 8 3 2 8 4 1 1 1 1 1 2 2 1 3 7 8 8
 6 1 0 4 8 9 8 4 6 8 0 1 4 3 2 2 5 9 0 9 0 0 5 3 8 0 6 0 2 5 1 7 8 2 0 3 3
 8 7 1 8 8 9 6 9 3 0 8 1 1 8 6 5 5 3 5 2 0 3 5 3 6 6 4 4 1 8 8 1 7 3 9 8 1
 9 1 0 7 7 6 2 9 9 8 5 4 4 4 5 1 9 8 7 9 6 8 7 5 7 4 7 3 2 6 6 3 7 8 4 8 9
 9 1 0 9 2 5 0 0 0 4 5 8 1 2 2 2 8 5 7 0 1 2 4 3 9 7 2 8 7 6 7 3 3 3 1 9 1
 4 6 9 7 5 1 5 2 0 5 9 6 5 6 1 2 3 7 1 1 2 4 2 4 0 2 9 3 1 4 4 3 9 2 9 4 9
 8 9 4 7 4 2 7 1 8 6 6 5 7 3 3 6 2 8 1 2 5 3 1 0 4 6 3 3 2 2 6 3 1 2 0 1 5
 3 6 9 5 4 5 0 8 9 4 7 5 0 3 7 3 9 0 7 2 4 6 2 4 5 0 7 4 7 3 5 8 0 7 0 1 5
 7 7 9 5 9 4 9 4 1 0 3 6 9 9 6 0 1 0 8 1 8 3 1 1 2 8 3 7 1 3 7 4 2 0 4 7 2
 5 3 3 3 5 3 6 9 6 1 8 2 3 9 6 8 2 7 6 8 2 0 9 9 9 7 8 3 9 2 9 5 6 6 1 8 3
 7 3 4 9 5 6 6 2 5 2 2 6 0 4 5 7 4 0 6 1 3 7 0 5 4 1 6 6 9 0 0 8 5 8 9 1 4
 7 1 5 1 3 0 5 9 3 0 6 7 3 7 2 2 0 7 6 5 5 5 3 0 0 3 1 2 1 4 3 7 2 2 8 7 7
 0 9 1 7 9 5 3 9 4 1 6 6 5 8 1 5 4 8 8 6 9 1 1 9 1 4 0 9 9 6 4 6 9 3 4 1 0
 9 4 3 8 9 2 3 0 0 3 3 1 2 1 7 0 7 4 7 4 2 5 1 8 3 6 5 7 2 5 5 6 9 4 5 6 7
 6 8 2 2 8 5 9 0 9 4 1 8 8 3 2 1 5 1 3 8 6 2 0 7 3 0 7 3 2 6 6 5 6 7 9 8 9
 0 7 6 4 3 3 4 0 7 9 4 0 4 2 1 5 4 1 9 1 3 5 0 5 5 1 8 4 3 1 4 4 8 5 7 3 5
 9 2 5 5 2 1 2 1 8 8 5 4 3 3 4 3 9 9 6 4 7 5 7 4 4 7 4 3 0 9 3 8 0 9 9 8 5
 3 5 8 2 9 9 5 6 7 1 5 2 1 1 0 5 6 7 4 1 5 9 5 5 7 1 9 4 9 0 1 6 5 5 7 0 6
 8 9 7 1 0 6 6 7 7 2 3 3 8 6 1 8 7 6 2 6 7 7 8 3 4 4 8 2 0 2 6 9 0 9 3 9 2
 1 4 2 4 2 7 4 5 9 6 5 2 3 1 0 3 6 5 1 5 3 7 1 6 3 3 6 6 1 8 2 3 4 8 6 5 9
 6] [7 5 2 5 9 1 2 3 2 2 5 5 9 5 3 1 3 6 7 9 4 5 6 5 7 5 7 0 1 3 5 5 5 2 9 1 2
```

```
 8 8 4 6 7 2 3 1 0 2 6 7 7 0 8 6 6 3 3 9 2 1 9 2 5 1 5 1 8 5 0 0 6 5 5 7 0
 9 3 3 2 1 7 8 7 4 3 6 4 1 5 6 8 1 4 3 2 7 3 4 7 6 1 5 7 1 1 4 7 3 0 8 4 9
 6 2 6 3 0 9 8 5 1 7 4 2 0 4 5 7 0 9 8 7 2 6 5 8 4 7 5 1 5 1 6 2 4 7 2 7 7
 8 0 1 4 3 4 8 4 7 9 6 6 1 4 3 0 4 8 4 4 7 3 7 7 7 1 5 5 2 2 1 0 8 0 6 9 6
 5 1 3 7 5 0 0 3 4 0 9 3 1 7 9 6 5 6 4 2 1 5 4 0 5 2 7 3 1 9 0 3 7 1 3 3 4
 7 8 4 6 7 6 7 7 6 7 8 6 1 8 0 7 0 2 7 5 6 5 1 0 1 9 6 9 9 0 1 1 5 9 1 9 9
 7 5 4 5 7 0 8 7 1 1 7 1 9 1 8 7 9 1 8 7 1 3 2 9 2 1 1 1 2 1 2 5 1 3 7 8 0
 6 1 0 4 8 7 8 4 6 8 0 1 4 3 2 2 5 9 0 9 0 0 5 3 8 0 6 0 2 5 1 7 8 2 5 3 3
 8 7 1 8 8 2 6 9 3 0 8 1 1 2 6 8 8 3 5 2 0 3 5 3 6 5 4 4 1 8 8 1 7 3 5 8 1
 9 1 0 7 7 2 2 9 9 8 5 6 4 4 5 1 9 8 7 8 6 8 7 5 3 4 7 3 2 6 6 3 7 8 2 4 9
 9 1 0 9 2 8 0 0 0 4 5 5 1 2 3 2 8 5 7 0 1 2 4 3 9 7 2 8 7 6 7 3 3 8 1 9 1
 4 6 7 7 5 1 5 2 0 5 9 6 5 6 1 2 3 7 1 1 2 4 2 4 0 3 9 3 1 4 4 3 7 2 9 4 9
 0 9 7 7 4 2 7 1 8 6 6 5 7 3 3 6 2 8 1 8 7 3 1 0 4 5 3 3 2 2 6 3 1 2 5 1 5
 3 6 9 5 4 8 0 8 9 4 7 5 0 3 7 3 9 0 7 6 4 6 2 4 5 0 7 4 7 3 5 8 0 7 0 1 5
 7 7 0 5 9 4 9 9 1 0 3 2 9 9 2 0 1 0 8 1 8 3 1 1 2 8 3 1 1 3 7 4 3 0 4 7 2
 5 3 3 3 5 3 6 9 6 1 8 2 3 9 6 8 3 7 6 8 2 0 9 7 9 7 8 0 9 2 9 3 6 6 1 8 3
 7 2 4 9 5 2 6 2 5 2 2 6 0 4 5 7 4 0 6 1 1 7 8 5 9 1 6 6 9 0 0 8 5 2 7 1 9
 5 1 5 1 3 6 5 7 3 0 6 7 5 2 2 2 0 7 6 5 5 3 3 0 0 3 1 2 1 4 3 7 2 2 8 7 7
 0 9 1 7 9 8 3 9 4 1 6 6 5 1 1 5 4 8 8 8 9 1 1 4 1 4 0 9 9 8 4 6 9 3 9 1 0
 9 4 3 8 9 2 3 0 0 3 3 1 2 1 7 4 7 4 2 4 2 5 1 8 3 6 8 7 2 5 5 6 9 4 3 6 7
 6 8 2 2 2 5 9 0 3 4 1 8 8 3 2 9 3 1 3 8 6 2 0 7 5 0 7 3 2 6 6 5 6 7 9 3 9
 0 7 6 4 3 3 4 0 8 9 4 0 4 3 7 3 4 1 9 1 3 5 0 5 5 1 8 4 3 1 4 4 8 5 7 3 5
 9 2 8 5 2 1 2 1 8 8 5 4 3 3 4 3 9 4 6 4 7 5 7 9 4 7 4 8 0 9 3 8 0 9 9 8 8
 3 5 8 2 8 8 5 6 7 6 5 2 1 1 0 5 5 7 4 1 5 9 8 5 7 3 9 8 9 0 1 6 8 5 7 0 5
 8 9 7 1 0 6 6 7 7 2 3 3 8 6 3 8 7 6 2 6 7 7 8 3 4 9 8 2 0 7 6 9 0 9 3 9 2
 1 4 3 9 2 7 2 5 9 6 5 3 3 1 0 3 6 5 1 5 3 7 1 6 3 3 6 6 1 0 2 3 9 8 6 8 4
 2]
Accuracy:
0.848
```

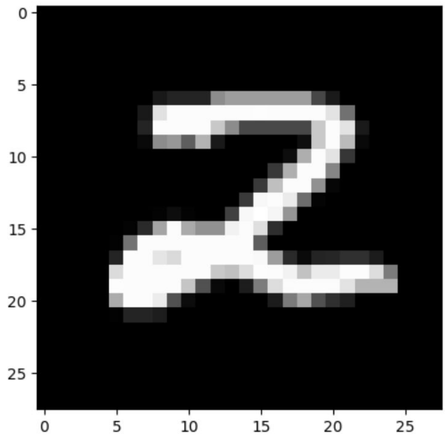
Prediction: [9]
Label: 4



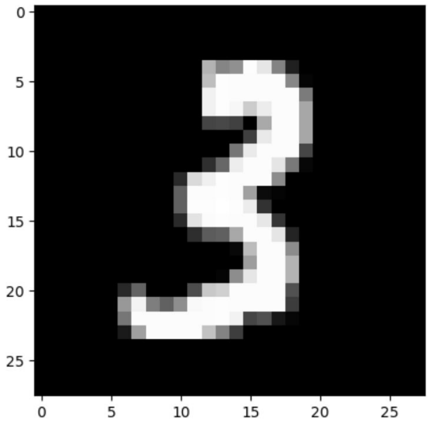
Prediction: [7]
Label: 7



Prediction: [2]
Label: 2



Prediction: [3]
Label: 3



Modifications

Enhancing Model Complexity

Additional Hidden Layers:

- **Consider the impact of introducing more hidden layers. -** The network architecture was iteratively modified to include two hidden layers with 128 and 64 neurons, respectively.

- **Evaluate the network's ability to capture more intricate features in handwritten digits.**
  - **- Noise Robustness**
  - **- Intra-digit variability**

Optimization and Fine-Tuning

Hyperparameter Tuning:

- **Conduct a thorough exploration of hyperparameters to optimize the model's performance.**

- **Fine-tune learning rates, activation functions, and layer sizes for improved accuracy.**

CODE MODIFICATIONS

Project.ipynb

File Edit View Insert Runtime Tools Help All changes saved

Files

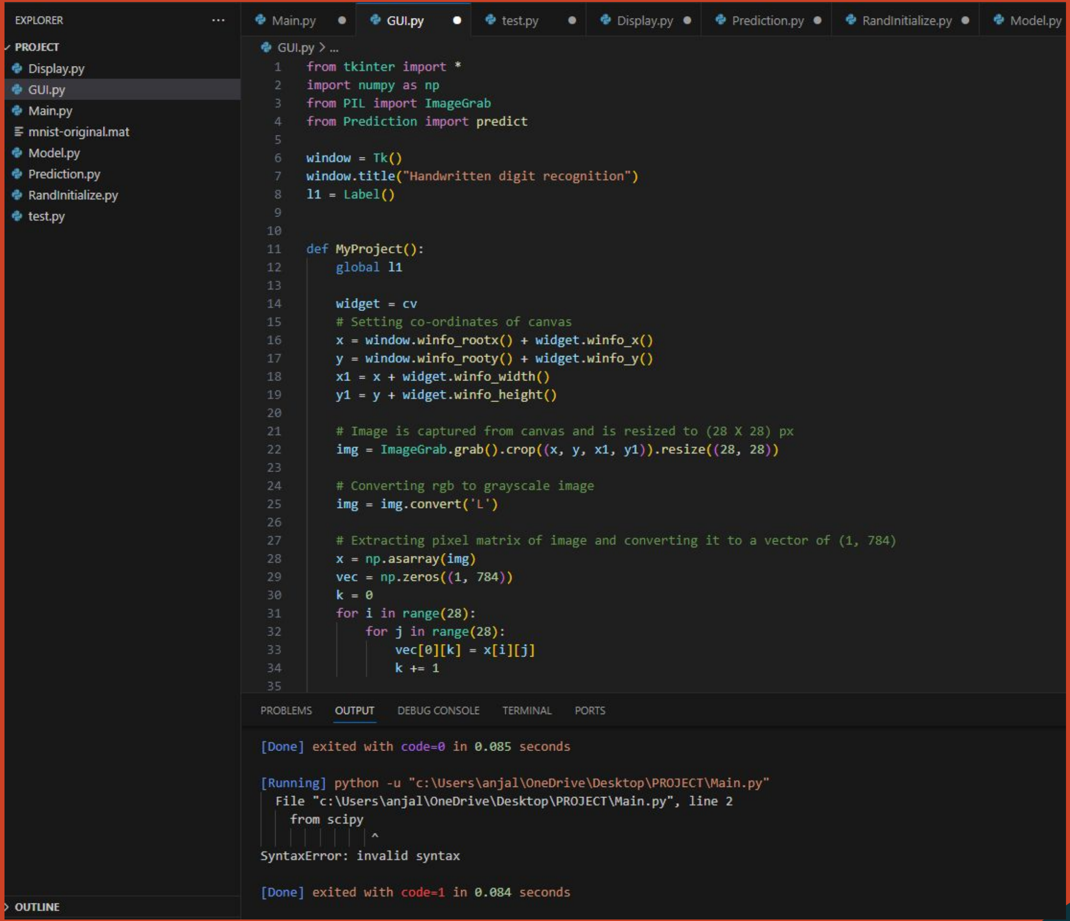
..
sample_data
Theta1.txt
Theta2.txt
Theta3.txt
mnist-original.mat

+ Code + Text

Model.py

```python
def neural_network(nn_params, input_layer_size, hidden_layer_one_size, hidden_layer_two_size, num_labels, X, y, lamb):
    # Weights are split back to Theta1, Theta2, Theta3
    Theta1 = np.reshape(nn_params[:hidden_layer_one_size * (input_layer_size + 1)],
                        (hidden_layer_one_size, input_layer_size + 1))
    Theta2 = np.reshape(nn_params[hidden_layer_one_size * (input_layer_size + 1):
                              hidden_layer_one_size * (input_layer_size + 1) + hidden_layer_two_size * (hidden_layer_one_size + 1)],
                        (hidden_layer_two_size, hidden_layer_one_size + 1))
    Theta3 = np.reshape(nn_params[hidden_layer_one_size * (input_layer_size + 1) + hidden_layer_two_size * (hidden_layer_one_size + 1):],
                        (num_labels, hidden_layer_two_size + 1))

    # Forward Propagation

    m = X.shape[0]
    one_matrix = np.ones((m,1))
    X = np.append(one_matrix, X, axis=1) # Adding bias unit to first layer
    a1 = X
    z2 = np.dot(X, Theta1.transpose())
    a2 = 1 / (1 + np.exp(-z2)) # Activation for second layer
    one_matrix = np.ones((m,1))
    a2 = np.append(one_matrix, a2, axis = 1) # Adding bias unit to first hidden layer
    z3 = np.dot(a2, Theta2.transpose())
    a3 = 1 / (1 + np.exp(-z3)) # Activation for third layer
    one_matrix = np.ones((m,1))
    a3 = np.append(one_matrix, a3, axis = 1) # Adding bias unit to second hidden layer
    z4 = np.dot(a3, Theta3.transpose())
    a4 = 1 / (1 + np.exp(-z4)) # Activation for fourth layer

    # Changing the y labels into vectors of boolean values.
    # For each label between 0 and 9, there will be a vector of length 10
    # where the ith element will be 1 if the label equals i
    y_vect = np.zeros((m,10))
    for i in range(m):
        y_vect[i, int(y[i])] = 1

    # Calculating cost function
    J = (1 / m) * (np.sum(np.sum(-y_vect * np.log(a4) - (1 - y_vect) * np.log(1 - a4)))) + \
        (lamb / (2 * m)) * (np.sum(np.sum(Theta1[:, 1:] ** 2)) + np.sum(np.sum(Theta2[:, 1:] ** 2)) + np.sum(np.sum(Theta3[:, 1:] ** 2)))
```

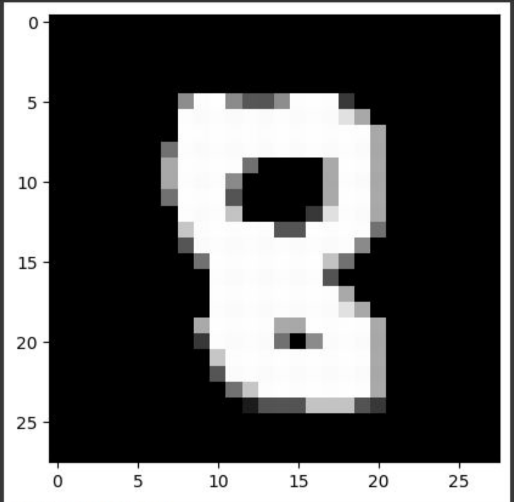
Disk 80.70 GB available



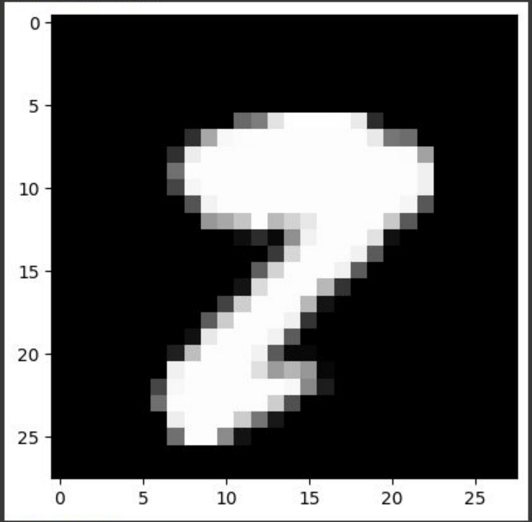
We tried to incorporate a GUI interface, which would enable the user to write handwritten digit on the screen in real time, which would then be feeded to the network for its prediction.

NEW RESULTS OBTAINED

NEW PREDICTIONS WITH MODIFIED ARCHITECTURE



Predicted Label: 8
Actual/True Label: 8.0
Iteration: 55000



Predicted Label: 8
Actual/True Label: 9.0
Iteration: 59000

Actual/True Label: 9.0
Test Set Accuracy: 97.190000
Iteration: 0

Training Set Accuracy: 98.908333
Precision = 0.9890833333333333

Thank You