Jenna Leali Assignment 4

https://colab.research.google.com/drive/1HaqxHO5yMnMjQAAq3v2srTA9lBtq2oL7?usp=sharing

The MNIST dataset is divided into training and test sets. Each set comprises a series of images (28 x 28-pixel images of handwritten digits) and their respective labels (0 - 9 values representing which digit the image corresponds to).

```
from tensorflow.keras.datasets import mnist
import matplotlib.pyplot as plt
import numpy as np
```

a) Use the mnist function in keras.datasets to load and split the MNIST dataset into the training and testing sets. Name the sets as x_train, y_train, and x_test, y_test. Print the following: The number of images in each training and testing set and the image width and height.

```
(x_train, y_train), (x_test, y_test) = mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datas
11490434/11490434 [==============================] - 0s 0us/step
```

```
print(x_train.shape, y_train.shape) # shape and size of training images
print(x_test.shape, y_test.shape) # shape and size of testing images
print(x_train.shape[0], x_train.shape[1], x_train.shape[2]) # get individual valu
```

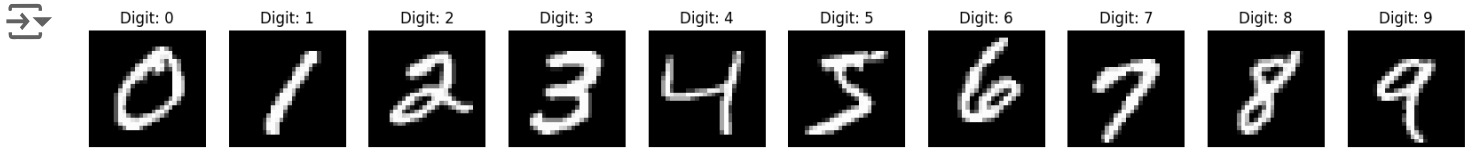
```
(60000, 28, 28) (60000,)
(10000, 28, 28) (10000,)
60000 28 28
```

b) Write a function that takes two inputs: 1) images of ten digits and 2) their corresponding labels and plots a figure with 10 subplots for each 0-9 digits. Each subplot has the number of the handwritten digit in its title.

```
def img_plt(images, labels):
    fig, axes = plt.subplots(1, 10, figsize=(20, 4))
    for i in range(10):
        index = np.where(labels == i)[0][0]
        axes[i].imshow(images[index], cmap='gray')
        axes[i].set_title('Digit: ' + str(i))
        axes[i].axis('off')
    plt.show()
```

c) Create a loop to call the function in (b) and plot images from the training set to create a figure that includes all the 10 digits (0-9).

```
img_plt(x_train, y_train)
```



d) Select the 0 and 8 digits from the training and testing sets and name them:

x_train_01, y_train_01 and x_test_01, y_test_01.

```
train_indices_01 = np.where((y_train == 0) | (y_train == 8))[0]
x_train_01 = x_train[train_indices_01]
y_train_01 = y_train[train_indices_01]

test_indices_01 = np.where((y_test == 0) | (y_test == 8))[0]
x_test_01 = x_test[test_indices_01]
y_test_01 = y_test[test_indices_01]

print("x_train_01:", x_train_01.shape)
print("y_train_01:", y_train_01.shape)
print("x_test_01:", x_test_01.shape)
print("y_test_01:", y_test_01.shape)
```

```
x_train_01: (11774, 28, 28)
y_train_01: (11774,)
x_test_01: (1954, 28, 28)
y_test_01: (1954,)
```

e) In machine learning, we typically divide the training set into two training and validation sets to adjust the machine learning model parameters. In your code, randomly select 500 training images and their corresponding labels (from x_train_01 and y_train_01) as the validation set and name them x_valid_01 and y_valid_01, respectively. Name the remaining training images and their labels as x_train_01 and y_train_01, respectively. Note: there are no overlaps between the two sets.

```
from sklearn.model_selection import train_test_split


x_train_01, x_valid_01, y_train_01, y_valid_01 = train_test_split(x_train_01, y_t

print("New training set size:", x_train_01.shape[0])
print("Validation set size:", x_valid_01.shape[0])
```

```
New training set size: 11274
Validation set size: 500
```

f) Print the number of images in each training, validation, and testing set.

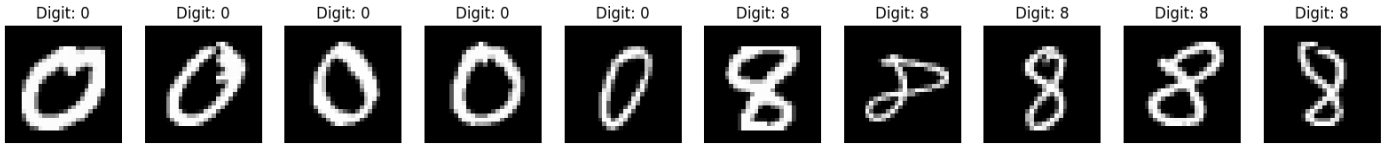
```
print("Number of images in the training set:", x_train_01.shape[0])
print("Number of images in the validation set:", x_valid_01.shape[0])
print("Number of images in the testing set:", x_test_01.shape[0])
```

```
Number of images in the training set: 11274
Number of images in the validation set: 500
Number of images in the testing set: 1954
```

g) Use the function created in part (b) to plot 10 images from the validation set.

```
def img_plt_01(images, labels):
    fig, axes = plt.subplots(1, 10, figsize=(20, 4))
    for i, digit in enumerate([0, 8]):
        digit_indices = np.where(labels == digit)[0][:5]
        for j, index in enumerate(digit_indices):
            axes[i * 5 + j].imshow(images[index], cmap='gray')
            axes[i * 5 + j].set_title('Digit: ' + str(digit))
            axes[i * 5 + j].axis('off')
    plt.show()

img_plt_01(x_valid_01, y_valid_01)
```



h) Convert each image in the three training, validation, and testing sets to one attribute by calculating the average of all the pixel values in the center 4x4 grid of the image.

```
def calculate_center_4x4_average(images):
    center_images = images[:, 12:16, 12:16]
    averages = np.mean(center_images, axis=(1, 2))
    return averages

x_train_01_avg = calculate_center_4x4_average(x_train_01)
x_valid_01_avg = calculate_center_4x4_average(x_valid_01)
x_test_01_avg = calculate_center_4x4_average(x_test_01)
```

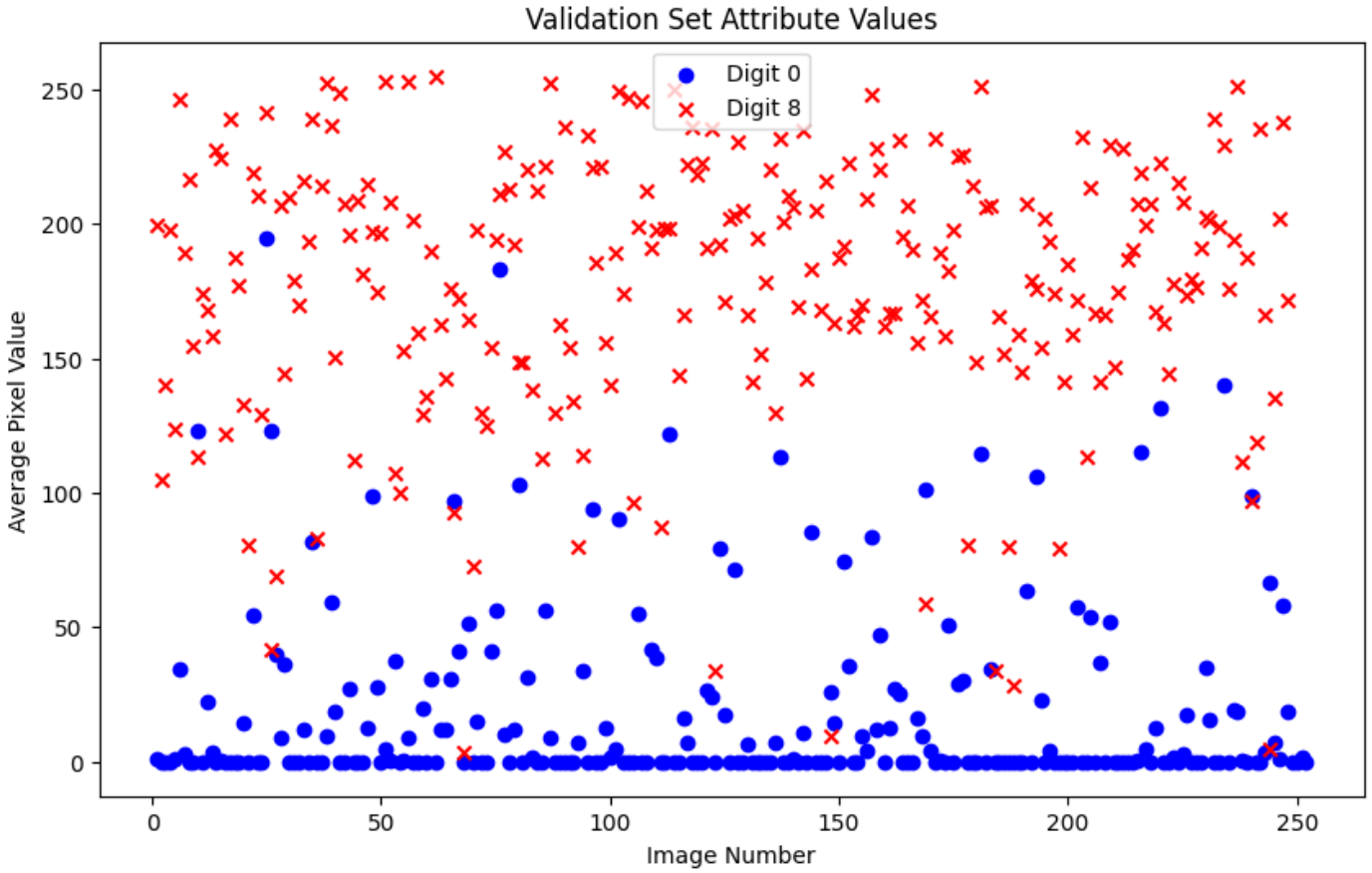
i) Plot the attribute values of the 500 images in the validation set that you calculated in part (h). Use different colors and shapes for 0's and 8's. The x-axis in your figure is the image number (1 to 500), and the y-axis is the calculated attribute. Label the axes and add legends appropriately.

```
import matplotlib.pyplot as plt
def plt_val_set(x_valid_avg, y_valid, digit_0, digit_1):
    avg_digit_0 = x_valid_avg[y_valid == digit_0]
    avg_digit_1 = x_valid_avg[y_valid == digit_1]

    x_axis_0 = range(1, len(avg_digit_0) + 1)
    x_axis_1 = range(1, len(avg_digit_1) + 1)

    plt.figure(figsize=(10, 6))
    plt.scatter(x_axis_0, avg_digit_0, color='blue', marker='o', label=f'Digit {d
    plt.scatter(x_axis_1, avg_digit_1, color='red', marker='x', label=f'Digit {di
    plt.title('Validation Set Attribute Values')
    plt.xlabel('Image Number')
    plt.ylabel('Average Pixel Value')
    plt.legend()
    plt.show()

plt_val_set(x_valid_01_avg, y_valid_01, 0, 8)
```



j) Based on your observation from the plot (i.e., validation set), guess a threshold on the attribute you think would differentiate the two classes with the highest accuracy.

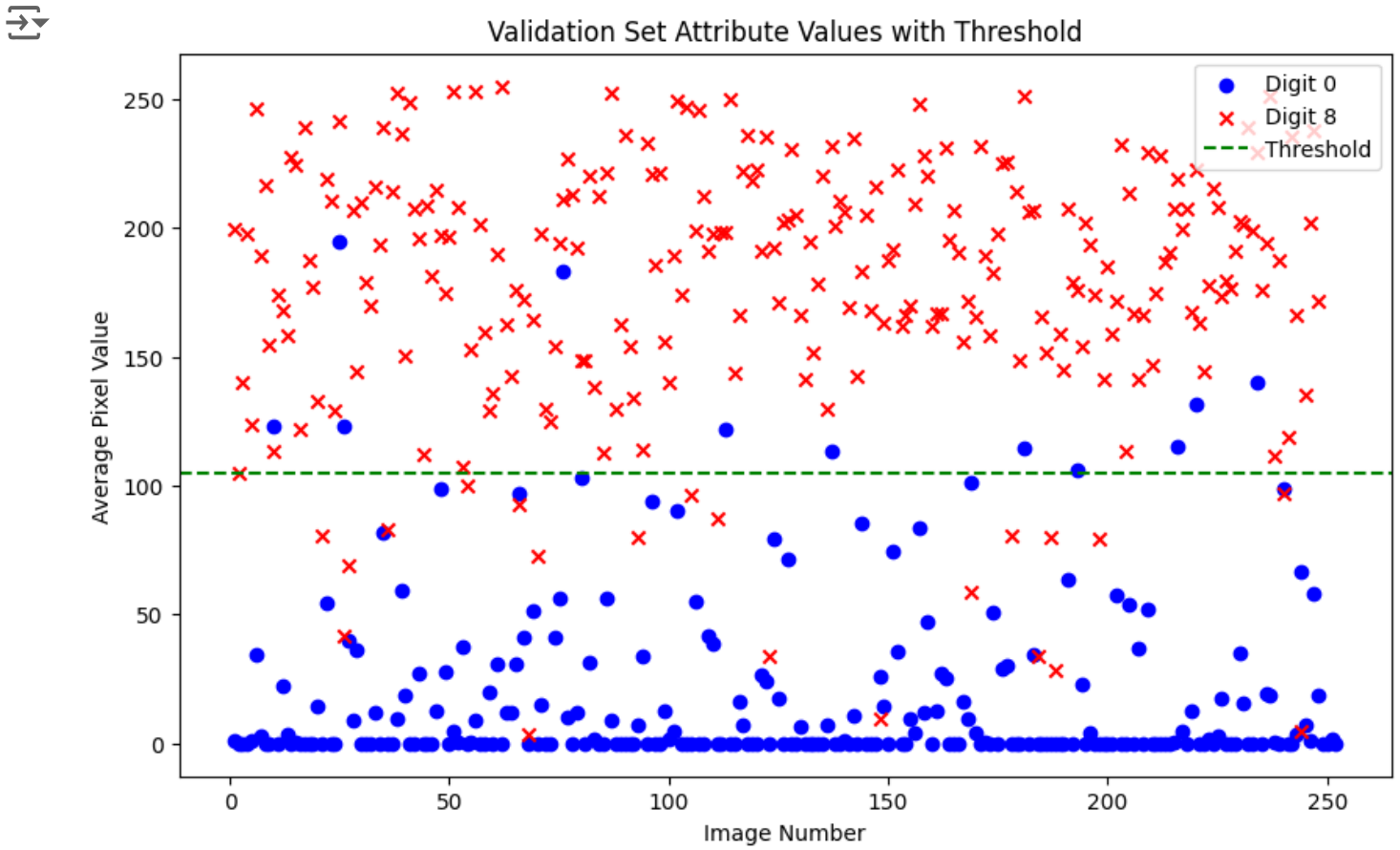
```
def plt_val_set_with_threshold(x_valid_avg, y_valid, digit_0, digit_1, threshold)

    avg_digit_0 = x_valid_avg[y_valid == digit_0]
    avg_digit_1 = x_valid_avg[y_valid == digit_1]
    x_axis_0 = range(1, len(avg_digit_0) + 1)
    x_axis_1 = range(1, len(avg_digit_1) + 1)
    plt.figure(figsize=(10, 6))
    plt.scatter(x_axis_0, avg_digit_0, color='blue', marker='o', label=f'Digit {d
    plt.scatter(x_axis_1, avg_digit_1, color='red', marker='x', label=f'Digit {di
    plt.axhline(y=threshold, color='green', linestyle='--', label='Threshold')
```

```
    plt.title('Validation Set Attribute Values with Threshold')
    plt.xlabel('Image Number')
    plt.ylabel('Average Pixel Value')
    plt.legend()

    plt.show()

plt_val_set_with_threshold(x_valid_01_avg, y_valid_01, 0, 8, 105)
```



k) Calculate the training, validation, and testing accuracies based on the selected threshold on the corresponding sets and print them.

```
def calculate_accuracy_with_threshold(x_values, y_values, threshold):
    predictions = np.where(x_values < threshold, 0, 8)
    correct_predictions = np.sum(predictions == y_values)
    accuracy = correct_predictions / len(y_values)

    return accuracy

threshold_value = 105
train_accuracy = calculate_accuracy_with_threshold(x_train_01_avg, y_train_01, thre
valid_accuracy = calculate_accuracy_with_threshold(x_valid_01_avg, y_valid_01, thre
test_accuracy = calculate_accuracy_with_threshold(x_test_01_avg, y_test_01, thresho

print(f'Training Accuracy: {train_accuracy:.2f}')
print(f'Validation Accuracy: {valid_accuracy:.2f}')
print(f'Testing Accuracy: {test_accuracy:.2f}')
```

```
Training Accuracy: 0.94
Validation Accuracy: 0.94
Testing Accuracy: 0.95
```

https://colab.research.google.com/drive/1HaqxHO5yMnMjQAAq3v2srTA9lBtq2oL7?usp=sharing