```
### Importing the dataset - in the form of batches - using dataloader package - mini batch descent - size =64
import torch
from torchvision import datasets, transforms
from torch.utils.data import DataLoader

import matplotlib.pyplot as plt
import numpy as np
transform = transforms.Compose([transforms.ToTensor(),
transforms.Normalize((0.5,), (0.5,))])
train_dataset = datasets.MNIST(root='./data', train=True,
transform=transform, download=True)
test_dataset = datasets.MNIST(root='./data', train=False,
transform=transform, download=True)
train_loader = DataLoader(dataset=train_dataset, batch_size=64,
shuffle=True)
test_loader = DataLoader(dataset=test_dataset, batch_size=64,
shuffle=False)

import numpy as np

class MLP:
    def __init__(self, input_layer_size=784,
first_hidden_layer_size=500,
                 second_hidden_layer_size=250,
third_hidden_layer_size=100,
                 output_layer_size=10, learning_rate=0.01,
batch_size=64):

        self.input_layer_size = input_layer_size
        self.first_hidden_layer_size = first_hidden_layer_size
        self.second_hidden_layer_size = second_hidden_layer_size
        self.third_hidden_layer_size = third_hidden_layer_size
        self.output_layer_size = output_layer_size
        self.learning_rate = learning_rate
        self.batch_size = batch_size

        self.W1 = np.random.uniform(-
np.sqrt(6/(self.input_layer_size+self.first_hidden_layer_size)),
np.sqrt(6/(self.input_layer_size+self.first_hidden_layer_size)),
(self.input_layer_size, self.first_hidden_layer_size))
        self.b1 = np.zeros((1, self.first_hidden_layer_size))
        self.W2 = np.random.uniform(-
np.sqrt(6/(self.first_hidden_layer_size+self.second_hidden_layer_size)
),

np.sqrt(6/(self.first_hidden_layer_size+self.second_hidden_layer_size)
),
                                    (self.first_hidden_layer_size,
self.second_hidden_layer_size))
```

```
        self.b2 = np.zeros((1, self.second_hidden_layer_size))

        self.W3 = np.random.uniform(-
np.sqrt(6/(self.second_hidden_layer_size+self.third_hidden_layer_size)
),

np.sqrt(6/(self.second_hidden_layer_size+self.third_hidden_layer_size)
),
                                    (self.second_hidden_layer_size,
self.third_hidden_layer_size))
        self.b3 = np.zeros((1, self.third_hidden_layer_size))

        self.W4 = np.random.uniform(-
np.sqrt(6/(self.third_hidden_layer_size+self.output_layer_size)),

np.sqrt(6/(self.third_hidden_layer_size+self.output_layer_size)),
                                    (self.third_hidden_layer_size,
self.output_layer_size))
        self.b4 = np.zeros((1, self.output_layer_size))

    def sigmoid(self, z):
        return 1 / (1 + np.exp(-z))

    def softmax(self, z):
        exp_z = np.exp(z - np.max(z, axis=1, keepdims=True))
        return exp_z / np.sum(exp_z, axis=1, keepdims=True)

    def forward(self, X):
        self.z1 = np.dot(X, self.W1) + self.b1
        self.Activation_1 = self.sigmoid(self.z1)
        self.z2 = np.dot(self.Activation_1, self.W2) + self.b2
        self.Activation_2 = self.sigmoid(self.z2)
        self.z3 = np.dot(self.Activation_2, self.W3) + self.b3
        self.Activation_3 = self.sigmoid(self.z3)
        self.z4 = np.dot(self.Activation_3, self.W4) + self.b4
        self.Activation_4 = self.softmax(self.z4)
        return self.Activation_4



    def backward(self, X, Y):
        errorz4 = self.Activation_4 - Y
        partial_derivative_w_4 = np.dot(self.Activation_3.T, errorz4)
/ self.batch_size
        partial_derivative_b_4 = np.sum(errorz4, axis=0,
keepdims=True) / self.batch_size

        errorz3 = np.dot(errorz4, self.W4.T) * self.Activation_3 * (1
- self.Activation_3)
        partial_derivative_w_3 = np.dot(self.Activation_2.T, errorz3)
```

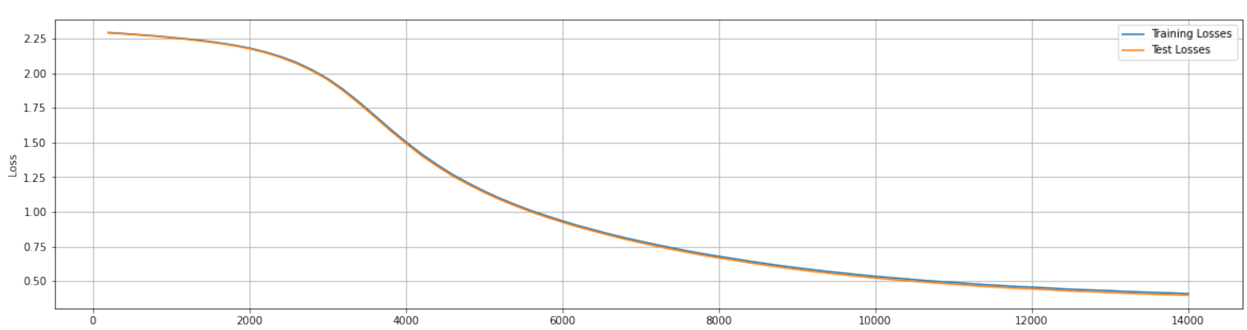
```
/ self.batch_size
        partial_derivative_b_3 = np.sum(errorz3, axis=0,
keepdims=True) / self.batch_size

        errorz2 = np.dot(errorz3, self.W3.T) * self.Activation_2 * (1
- self.Activation_2)
        partial_derivative_w_2 = np.dot(self.Activation_1.T, errorz2)
/ self.batch_size
        partial_derivative_b_2 = np.sum(errorz2, axis=0,
keepdims=True) / self.batch_size

        errorz1 = np.dot(errorz2, self.W2.T) * self.Activation_1 * (1
- self.Activation_1)
        partial_derivative_w_1 = np.dot(X.T, errorz1) /
self.batch_size
        partial_derivative_b_1 = np.sum(errorz1, axis=0,
keepdims=True) / self.batch_size


        self.W4 -= self.learning_rate * partial_derivative_w_4
        self.b4 -= self.learning_rate * partial_derivative_b_4
        self.W3 -= self.learning_rate * partial_derivative_w_3
        self.b3 -= self.learning_rate * partial_derivative_b_3
        self.W2 -= self.learning_rate * partial_derivative_w_2
        self.b2 -= self.learning_rate * partial_derivative_b_2
        self.W1 -= self.learning_rate * partial_derivative_w_1
        self.b1 -= self.learning_rate * partial_derivative_b_1


    def entropy_class_loss(self, Y, Y_hat):

        loss = -np.sum(Y * np.log(Y_hat)) / self.batch_size
        return loss

def entropy_class_loss(self, Y, Y_hat):
        loss = -np.sum(Y * np.log(Y_hat)) / self.batch_size
        return loss



def train(model, train_loader, test_loader, epochs,
plot_interval=200):
    training_loss_list = []
    test_loss_list = []
    iteration_counts = []

    current_iteration = 0

    for epoch in range(epochs):
        for batch_idx, (images, labels) in enumerate(train_loader):
```

```
            X = images.view(-1, 28*28).numpy()
            Y = np.eye(10)[labels]
            Y_hat = model.forward(X)
            loss = model.entropy_class_loss(Y, Y_hat)
            model.backward(X, Y)
            current_iteration += 1
            if current_iteration % plot_interval == 0:
                iteration_counts.append(current_iteration)
                total_train_loss = 0
                total_train_samples = 0
                for train_images, train_labels in train_loader:
                    X_train = train_images.view(-1, 28*28).numpy()
                    Y_train = np.eye(10)[train_labels]
                    Y_train_hat = model.forward(X_train)
                    total_train_loss +=
model.entropy_class_loss(Y_train, Y_train_hat) * X_train.shape[0]
                    total_train_samples += X_train.shape[0]
                avg_train_loss = total_train_loss /
total_train_samples
                training_loss_list.append(avg_train_loss)

                total_test_loss = 0
                total_test_samples = 0
                for test_images, test_labels in test_loader:
                    X_test = test_images.view(-1, 28*28).numpy()
                    Y_test = np.eye(10)[test_labels]
                    Y_test_hat = model.forward(X_test)
                    total_test_loss +=
model.entropy_class_loss(Y_test, Y_test_hat) * X_test.shape[0]
                    total_test_samples += X_test.shape[0]
                avg_test_loss = total_test_loss / total_test_samples
                test_loss_list.append(avg_test_loss)

                print(f' Current Iteration {current_iteration},
Training Loss: {avg_train_loss:.4f}, Test Loss: {avg_test_loss:.4f}')

        print(f'Epoch {epoch+1}/{epochs} ')

    plt.figure(figsize=(20, 5))
    plt.plot(iteration_counts, training_loss_list, label='Training
Losses')
    plt.plot(iteration_counts, test_loss_list, label='Test Losses')
    plt.xlabel('Iterations')
    plt.ylabel('Loss')

    plt.legend()
    plt.grid(True)
    plt.show()
```

```python
model = MLP()
train(model, train_loader, test_loader, epochs=15)
```

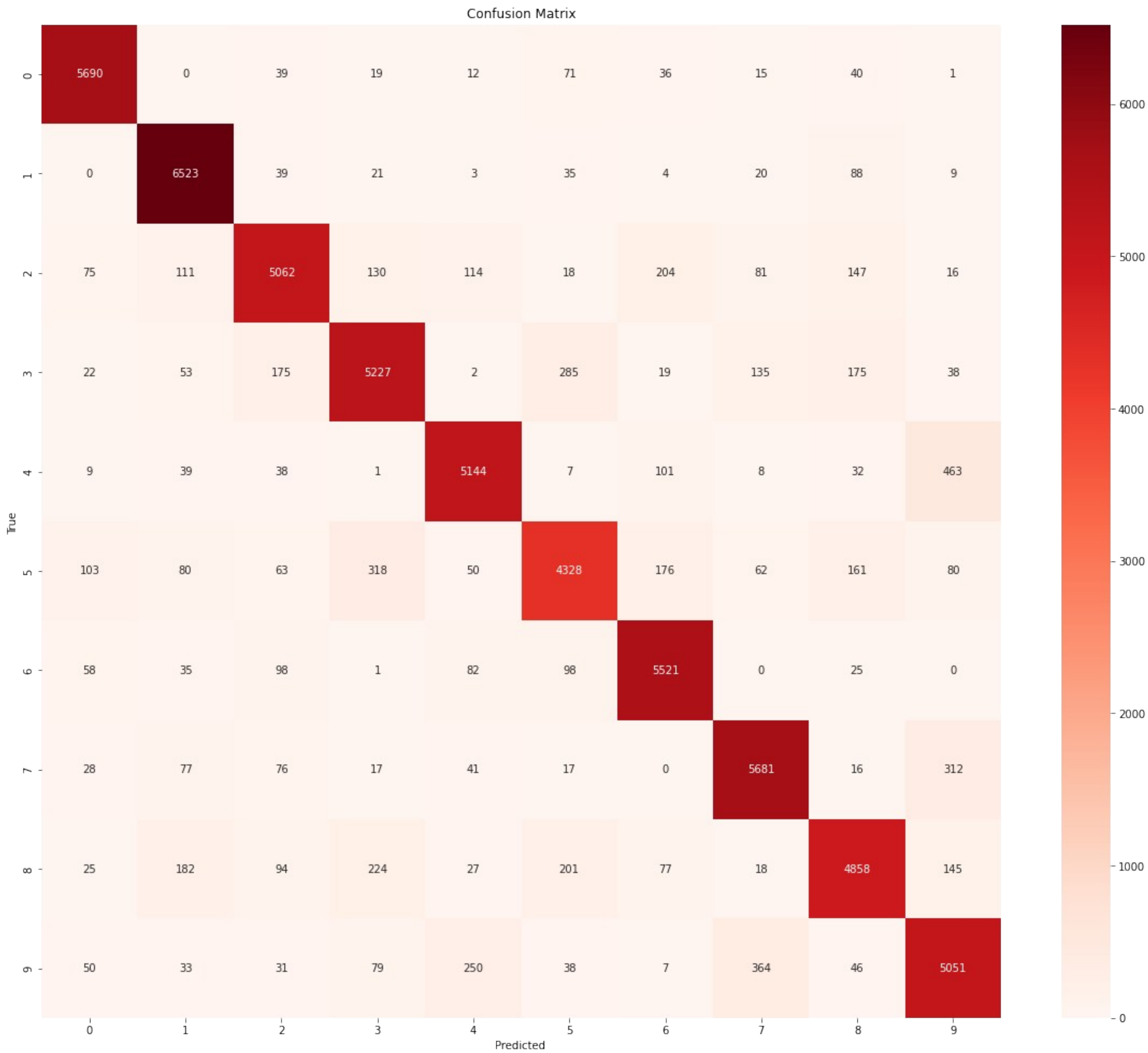
```
 Current Iteration 200, Training Loss: 2.2930, Test Loss: 2.2907
 Current Iteration 400, Training Loss: 2.2854, Test Loss: 2.2829
 Current Iteration 600, Training Loss: 2.2774, Test Loss: 2.2749
 Current Iteration 800, Training Loss: 2.2687, Test Loss: 2.2662
Epoch 1/15
 Current Iteration 1000, Training Loss: 2.2585, Test Loss: 2.2553
 Current Iteration 1200, Training Loss: 2.2476, Test Loss: 2.2440
 Current Iteration 1400, Training Loss: 2.2349, Test Loss: 2.2313
 Current Iteration 1600, Training Loss: 2.2204, Test Loss: 2.2164
 Current Iteration 1800, Training Loss: 2.2029, Test Loss: 2.1987
Epoch 2/15
 Current Iteration 2000, Training Loss: 2.1814, Test Loss: 2.1768
 Current Iteration 2200, Training Loss: 2.1540, Test Loss: 2.1490
 Current Iteration 2400, Training Loss: 2.1200, Test Loss: 2.1140
 Current Iteration 2600, Training Loss: 2.0781, Test Loss: 2.0710
 Current Iteration 2800, Training Loss: 2.0251, Test Loss: 2.0179
Epoch 3/15
 Current Iteration 3000, Training Loss: 1.9617, Test Loss: 1.9541
 Current Iteration 3200, Training Loss: 1.8840, Test Loss: 1.8742
 Current Iteration 3400, Training Loss: 1.7938, Test Loss: 1.7832
 Current Iteration 3600, Training Loss: 1.6974, Test Loss: 1.6864
Epoch 4/15
 Current Iteration 3800, Training Loss: 1.5990, Test Loss: 1.5874
 Current Iteration 4000, Training Loss: 1.5058, Test Loss: 1.4948
 Current Iteration 4200, Training Loss: 1.4181, Test Loss: 1.4065
 Current Iteration 4400, Training Loss: 1.3410, Test Loss: 1.3297
 Current Iteration 4600, Training Loss: 1.2690, Test Loss: 1.2584
Epoch 5/15
 Current Iteration 4800, Training Loss: 1.2075, Test Loss: 1.1976
 Current Iteration 5000, Training Loss: 1.1504, Test Loss: 1.1410
 Current Iteration 5200, Training Loss: 1.0982, Test Loss: 1.0892
 Current Iteration 5400, Training Loss: 1.0522, Test Loss: 1.0436
 Current Iteration 5600, Training Loss: 1.0092, Test Loss: 1.0001
Epoch 6/15
 Current Iteration 5800, Training Loss: 0.9711, Test Loss: 0.9616
 Current Iteration 6000, Training Loss: 0.9349, Test Loss: 0.9257
 Current Iteration 6200, Training Loss: 0.9011, Test Loss: 0.8911
 Current Iteration 6400, Training Loss: 0.8707, Test Loss: 0.8612
Epoch 7/15
 Current Iteration 6600, Training Loss: 0.8406, Test Loss: 0.8312
 Current Iteration 6800, Training Loss: 0.8132, Test Loss: 0.8028
 Current Iteration 7000, Training Loss: 0.7880, Test Loss: 0.7772
 Current Iteration 7200, Training Loss: 0.7633, Test Loss: 0.7527
 Current Iteration 7400, Training Loss: 0.7406, Test Loss: 0.7297
Epoch 8/15
 Current Iteration 7600, Training Loss: 0.7181, Test Loss: 0.7080
 Current Iteration 7800, Training Loss: 0.6978, Test Loss: 0.6865
```

```
 Current Iteration 8000, Training Loss: 0.6790, Test Loss: 0.6681
 Current Iteration 8200, Training Loss: 0.6615, Test Loss: 0.6502
 Current Iteration 8400, Training Loss: 0.6430, Test Loss: 0.6321
Epoch 9/15
 Current Iteration 8600, Training Loss: 0.6269, Test Loss: 0.6155
 Current Iteration 8800, Training Loss: 0.6108, Test Loss: 0.5993
 Current Iteration 9000, Training Loss: 0.5965, Test Loss: 0.5846
 Current Iteration 9200, Training Loss: 0.5827, Test Loss: 0.5705
Epoch 10/15
 Current Iteration 9400, Training Loss: 0.5695, Test Loss: 0.5574
 Current Iteration 9600, Training Loss: 0.5580, Test Loss: 0.5460
 Current Iteration 9800, Training Loss: 0.5458, Test Loss: 0.5336
 Current Iteration 10000, Training Loss: 0.5347, Test Loss: 0.5228
 Current Iteration 10200, Training Loss: 0.5252, Test Loss: 0.5125
Epoch 11/15
 Current Iteration 10400, Training Loss: 0.5158, Test Loss: 0.5033
 Current Iteration 10600, Training Loss: 0.5064, Test Loss: 0.4946
 Current Iteration 10800, Training Loss: 0.4979, Test Loss: 0.4857
 Current Iteration 11000, Training Loss: 0.4912, Test Loss: 0.4783
 Current Iteration 11200, Training Loss: 0.4823, Test Loss: 0.4694
Epoch 12/15
 Current Iteration 11400, Training Loss: 0.4745, Test Loss: 0.4619
 Current Iteration 11600, Training Loss: 0.4685, Test Loss: 0.4559
 Current Iteration 11800, Training Loss: 0.4614, Test Loss: 0.4486
 Current Iteration 12000, Training Loss: 0.4570, Test Loss: 0.4441
Epoch 13/15
 Current Iteration 12200, Training Loss: 0.4512, Test Loss: 0.4388
 Current Iteration 12400, Training Loss: 0.4444, Test Loss: 0.4318
 Current Iteration 12600, Training Loss: 0.4396, Test Loss: 0.4266
 Current Iteration 12800, Training Loss: 0.4354, Test Loss: 0.4234
 Current Iteration 13000, Training Loss: 0.4315, Test Loss: 0.4179
Epoch 14/15
 Current Iteration 13200, Training Loss: 0.4256, Test Loss: 0.4132
 Current Iteration 13400, Training Loss: 0.4216, Test Loss: 0.4088
 Current Iteration 13600, Training Loss: 0.4176, Test Loss: 0.4051
 Current Iteration 13800, Training Loss: 0.4142, Test Loss: 0.4020
 Current Iteration 14000, Training Loss: 0.4102, Test Loss: 0.3987
Epoch 15/15
```

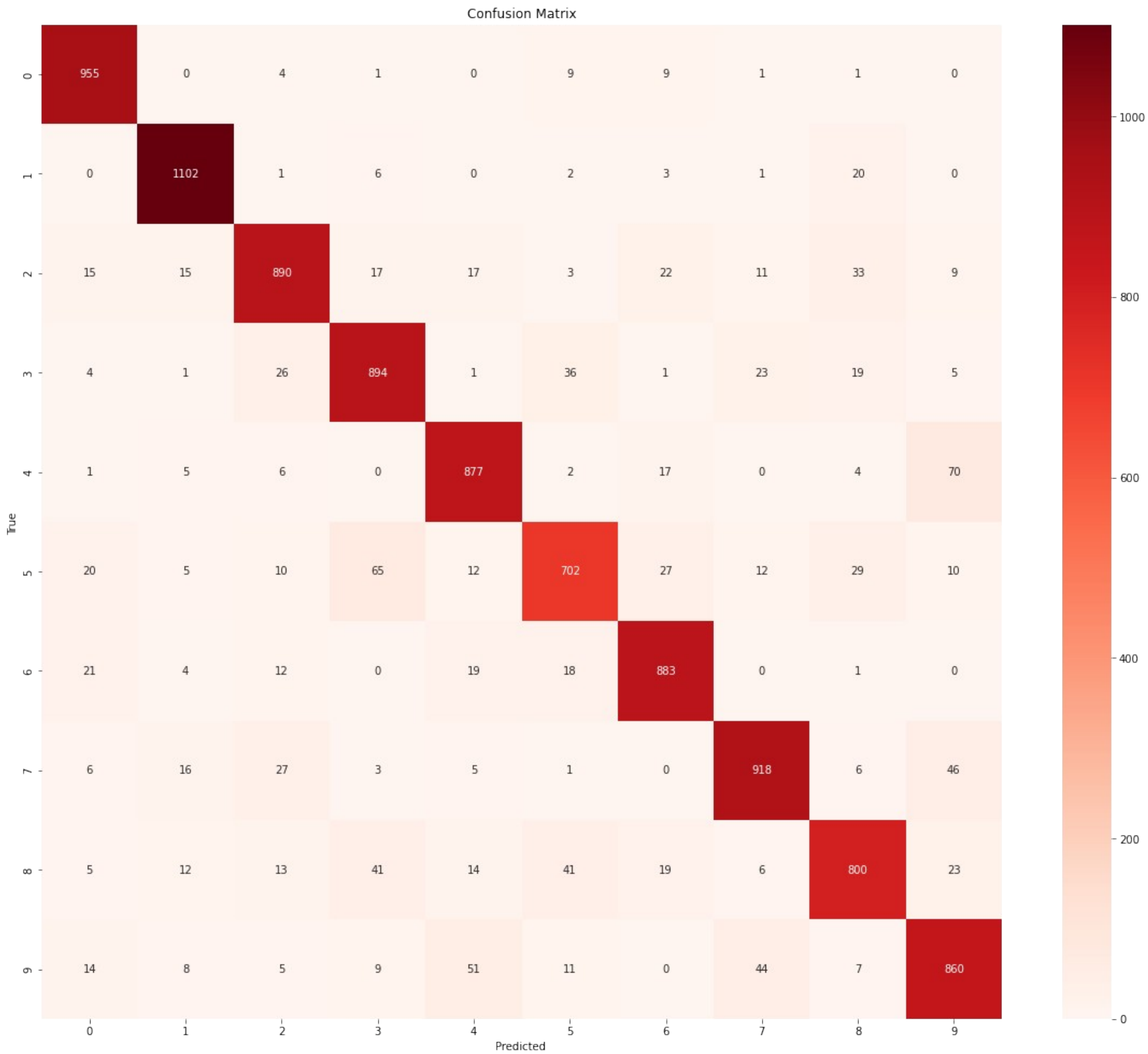


```python
from sklearn.metrics import confusion_matrix, accuracy_score
import seaborn as sns
def final_test(model, test_loader):
    all_preds = []
    all_labels = []
    for images, labels in test_loader:
        X = images.view(-1, 28*28).numpy()
        Y_hat = model.forward(X)
        preds = np.argmax(Y_hat, axis=1)
        all_preds.append(preds)
        all_labels.append(labels.numpy())
    all_preds = np.concatenate(all_preds)
    all_labels = np.concatenate(all_labels)
    accuracy = accuracy_score(all_labels, all_preds)
    print(f'Final Test Accuracy: {accuracy * 100:.2f}%')
    conf_matrix = confusion_matrix(all_labels, all_preds)
    print('Confusion Matrix:\n', conf_matrix)
    plt.figure(figsize=(20, 17))
    sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Reds')
    plt.xlabel('Predicted ')
    plt.ylabel('True ')
    plt.title('Confusion Matrix')
    plt.show()

final_test(model,train_loader)
```

```
Final Test Accuracy: 88.48%
Confusion Matrix:
 [[5690    0   39   19   12   71   36   15   40    1]
 [   0 6523   39   21    3   35    4   20   88    9]
 [  75  111 5062  130  114   18  204   81  147   16]
 [  22   53  175 5227    2  285   19  135  175   38]
 [   9   39   38    1 5144    7  101    8   32  463]
 [ 103   80   63  318   50 4328  176   62  161   80]
 [  58   35   98    1   82   98 5521    0   25    0]
 [  28   77   76   17   41   17    0 5681   16  312]
 [  25  182   94  224   27  201   77   18 4858  145]
 [  50   33   31   79  250   38    7  364   46 5051]]
```




```
final_test(model, test_loader)

Final Test Accuracy: 88.81%
Confusion Matrix:
 [[ 955    0    4    1    0    9    9    1    1    0]
 [    0 1102    1    6    0    2    3    1   20    0]
 [   15   15  890   17   17    3   22   11   33    9]
 [    4    1   26  894    1   36    1   23   19    5]
 [    1    5    6    0  877    2   17    0    4   70]
 [   20    5   10   65   12  702   27   12   29   10]
 [   21    4   12    0   19   18  883    0    1    0]
 [    6   16   27    3    5    1    0  918    6   46]
 [    5   12   13   41   14   41   19    6  800   23]
 [   14    8    5    9   51   11    0   44    7  860]]
```


Confusion Matrix
True
Predicted
0
1
2
3
4
5
6
7
8
9
955 0 4 1 0 9 9 1 1 0
0 1102 1 6 0 2 3 1 20 0
15 15 890 17 17 3 22 11 33 9
4 1 26 894 1 36 1 23 19 5
1 5 6 0 877 2 17 0 4 70
20 5 10 65 12 702 27 12 29 10
21 4 12 0 19 18 883 0 1 0
6 16 27 3 5 1 0 918 6 46
5 12 13 41 14 41 19 6 800 23
14 8 5 9 51 11 0 44 7 860
1000
800
600
400
200
0