```
### 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)

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

class MLP_Tanh:
    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)
),
```

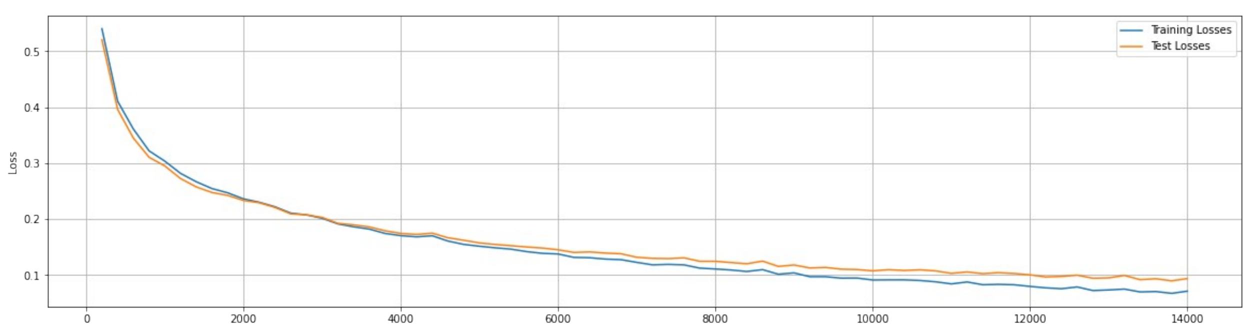
```
np.sqrt(6/(self.first_hidden_layer_size+self.second_hidden_layer_size)
),
                                    (self.first_hidden_layer_size,
self.second_hidden_layer_size))
        self.b2 = np.zeros((1, self.second_hidden_layer_size))

        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 tanh(self, z):
        return np.tanh(z)

    def tanh_differentiated(self, z):
        return 1 - np.tanh(z) ** 2

    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.tanh(self.z1)
        self.z2 = np.dot(self.Activation_1, self.W2) + self.b2
        self.Activation_2 = self.tanh(self.z2)
        self.z3 = np.dot(self.Activation_2, self.W3) + self.b3
        self.Activation_3 = self.tanh(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.tanh_differentiated(self.z3)
        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.tanh_differentiated(self.z2)
        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.tanh_differentiated(self.z1)
        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 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()


model = MLP_Tanh()
train(model, train_loader, test_loader, epochs=15)
```

```
Current Iteration 200, Training Loss: 0.5401, Test Loss: 0.5204
Current Iteration 400, Training Loss: 0.4105, Test Loss: 0.3960
Current Iteration 600, Training Loss: 0.3610, Test Loss: 0.3448
Current Iteration 800, Training Loss: 0.3220, Test Loss: 0.3106
Epoch 1/15
Current Iteration 1000, Training Loss: 0.3036, Test Loss: 0.2951
Current Iteration 1200, Training Loss: 0.2818, Test Loss: 0.2726
Current Iteration 1400, Training Loss: 0.2668, Test Loss: 0.2576
Current Iteration 1600, Training Loss: 0.2546, Test Loss: 0.2477
Current Iteration 1800, Training Loss: 0.2471, Test Loss: 0.2425
Epoch 2/15
Current Iteration 2000, Training Loss: 0.2363, Test Loss: 0.2334
Current Iteration 2200, Training Loss: 0.2303, Test Loss: 0.2294
Current Iteration 2400, Training Loss: 0.2222, Test Loss: 0.2209
Current Iteration 2600, Training Loss: 0.2109, Test Loss: 0.2095
Current Iteration 2800, Training Loss: 0.2078, Test Loss: 0.2079
Epoch 3/15
Current Iteration 3000, Training Loss: 0.2016, Test Loss: 0.2032
Current Iteration 3200, Training Loss: 0.1916, Test Loss: 0.1926
Current Iteration 3400, Training Loss: 0.1864, Test Loss: 0.1899
Current Iteration 3600, Training Loss: 0.1823, Test Loss: 0.1862
Epoch 4/15
Current Iteration 3800, Training Loss: 0.1746, Test Loss: 0.1793
Current Iteration 4000, Training Loss: 0.1707, Test Loss: 0.1747
Current Iteration 4200, Training Loss: 0.1685, Test Loss: 0.1728
Current Iteration 4400, Training Loss: 0.1705, Test Loss: 0.1751
Current Iteration 4600, Training Loss: 0.1610, Test Loss: 0.1668
Epoch 5/15
Current Iteration 4800, Training Loss: 0.1550, Test Loss: 0.1624
Current Iteration 5000, Training Loss: 0.1517, Test Loss: 0.1576
Current Iteration 5200, Training Loss: 0.1489, Test Loss: 0.1548
Current Iteration 5400, Training Loss: 0.1465, Test Loss: 0.1526
Current Iteration 5600, Training Loss: 0.1421, Test Loss: 0.1504
Epoch 6/15
Current Iteration 5800, Training Loss: 0.1391, Test Loss: 0.1485
Current Iteration 6000, Training Loss: 0.1379, Test Loss: 0.1453
Current Iteration 6200, Training Loss: 0.1318, Test Loss: 0.1407
Current Iteration 6400, Training Loss: 0.1312, Test Loss: 0.1416
Epoch 7/15
Current Iteration 6600, Training Loss: 0.1288, Test Loss: 0.1395
Current Iteration 6800, Training Loss: 0.1277, Test Loss: 0.1383
Current Iteration 7000, Training Loss: 0.1229, Test Loss: 0.1320
Current Iteration 7200, Training Loss: 0.1185, Test Loss: 0.1301
Current Iteration 7400, Training Loss: 0.1193, Test Loss: 0.1296
Epoch 8/15
Current Iteration 7600, Training Loss: 0.1185, Test Loss: 0.1310
Current Iteration 7800, Training Loss: 0.1127, Test Loss: 0.1249
Current Iteration 8000, Training Loss: 0.1111, Test Loss: 0.1248
Current Iteration 8200, Training Loss: 0.1093, Test Loss: 0.1225
Current Iteration 8400, Training Loss: 0.1066, Test Loss: 0.1204
```

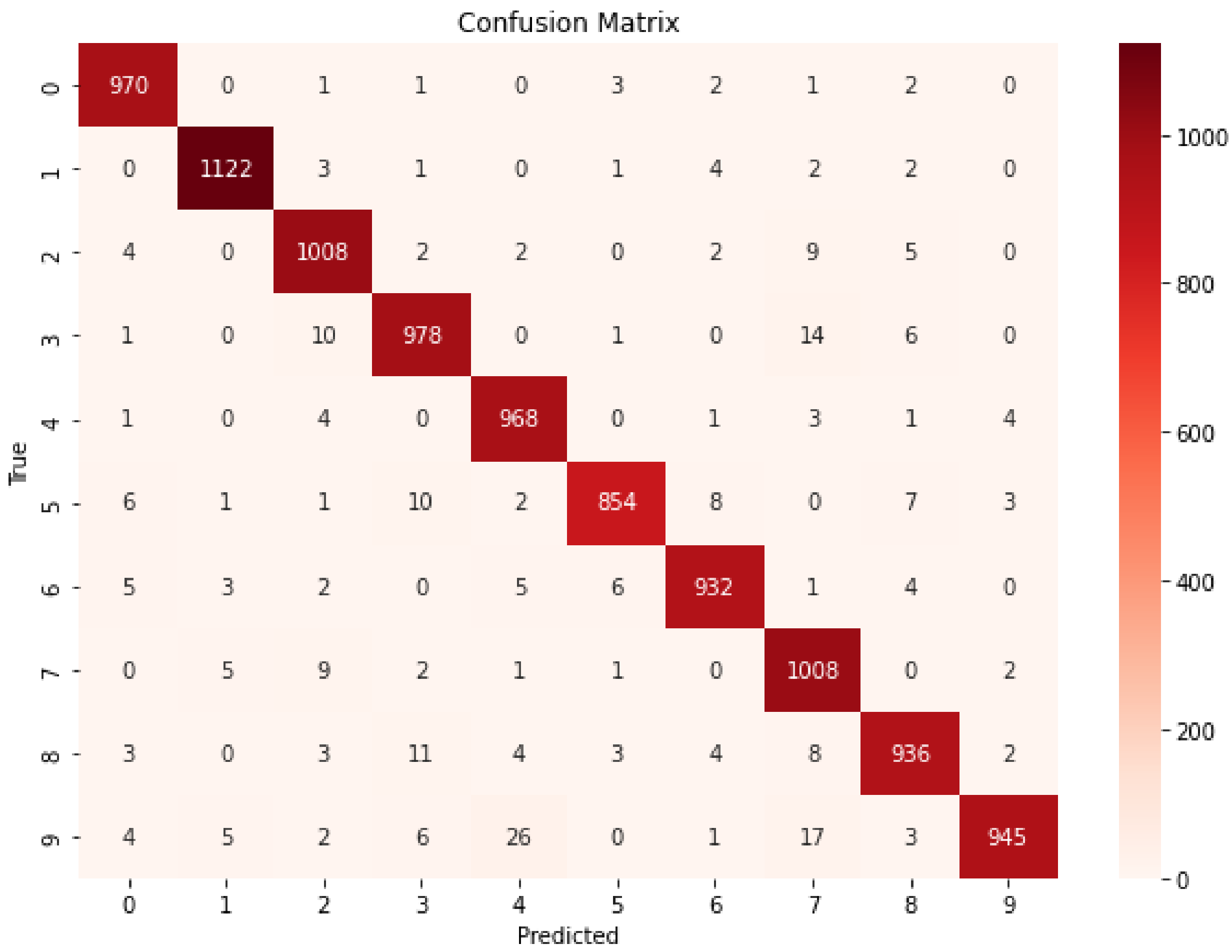
```
Epoch 9/15
Current Iteration 8600, Training Loss: 0.1100, Test Loss: 0.1251
Current Iteration 8800, Training Loss: 0.1018, Test Loss: 0.1157
Current Iteration 9000, Training Loss: 0.1042, Test Loss: 0.1182
Current Iteration 9200, Training Loss: 0.0973, Test Loss: 0.1129
Epoch 10/15
Current Iteration 9400, Training Loss: 0.0973, Test Loss: 0.1139
Current Iteration 9600, Training Loss: 0.0947, Test Loss: 0.1108
Current Iteration 9800, Training Loss: 0.0949, Test Loss: 0.1101
Current Iteration 10000, Training Loss: 0.0914, Test Loss: 0.1078
Current Iteration 10200, Training Loss: 0.0917, Test Loss: 0.1099
Epoch 11/15
Current Iteration 10400, Training Loss: 0.0917, Test Loss: 0.1085
Current Iteration 10600, Training Loss: 0.0906, Test Loss: 0.1097
Current Iteration 10800, Training Loss: 0.0882, Test Loss: 0.1077
Current Iteration 11000, Training Loss: 0.0845, Test Loss: 0.1033
Current Iteration 11200, Training Loss: 0.0879, Test Loss: 0.1058
Epoch 12/15
Current Iteration 11400, Training Loss: 0.0830, Test Loss: 0.1028
Current Iteration 11600, Training Loss: 0.0837, Test Loss: 0.1046
Current Iteration 11800, Training Loss: 0.0829, Test Loss: 0.1030
Current Iteration 12000, Training Loss: 0.0800, Test Loss: 0.1006
Epoch 13/15
Current Iteration 12200, Training Loss: 0.0774, Test Loss: 0.0968
Current Iteration 12400, Training Loss: 0.0759, Test Loss: 0.0978
Current Iteration 12600, Training Loss: 0.0790, Test Loss: 0.0999
Current Iteration 12800, Training Loss: 0.0726, Test Loss: 0.0945
Current Iteration 13000, Training Loss: 0.0737, Test Loss: 0.0952
Epoch 14/15
Current Iteration 13200, Training Loss: 0.0752, Test Loss: 0.0995
Current Iteration 13400, Training Loss: 0.0701, Test Loss: 0.0921
Current Iteration 13600, Training Loss: 0.0708, Test Loss: 0.0937
Current Iteration 13800, Training Loss: 0.0676, Test Loss: 0.0900
Current Iteration 14000, Training Loss: 0.0715, Test Loss: 0.0940
Epoch 15/15
```



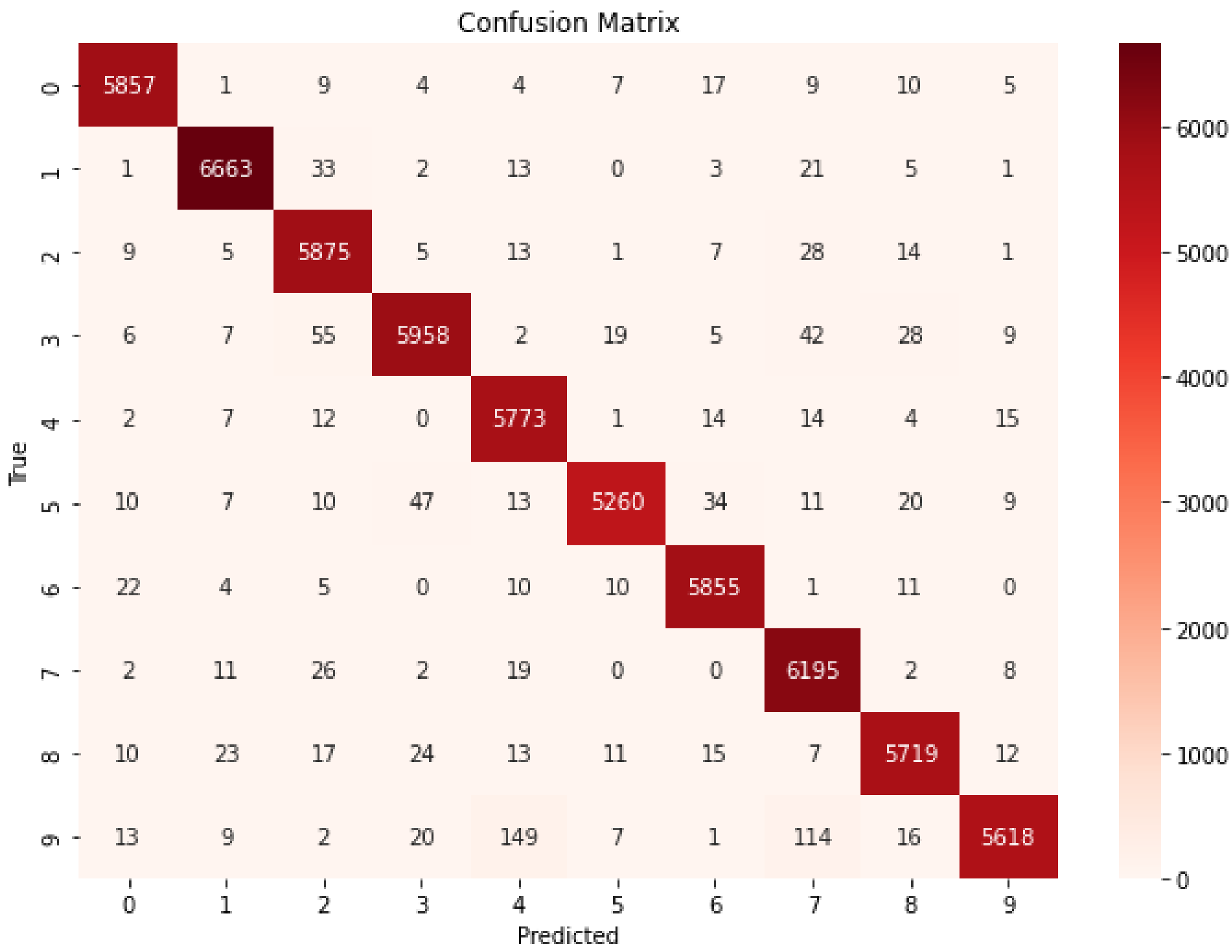

```
from sklearn.metrics import confusion_matrix, accuracy_score
import seaborn as sns
```

```python
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,test_loader)
```

```
Final Test Accuracy: 97.21%
Confusion Matrix:
 [[ 970    0    1    1    0    3    2    1    2    0]
 [    0 1122    3    1    0    1    4    2    2    0]
 [    4    0 1008    2    2    0    2    9    5    0]
 [    1    0   10  978    0    1    0   14    6    0]
 [    1    0    4    0  968    0    1    3    1    4]
 [    6    1    1   10    2  854    8    0    7    3]
 [    5    3    2    0    5    6  932    1    4    0]
 [    0    5    9    2    1    1    0 1008    0    2]
 [    3    0    3   11    4    3    4    8  936    2]
 [    4    5    2    6   26    0    1   17    3  945]]
```




```
final_test(model,train_loader)

Final Test Accuracy: 97.95%
Confusion Matrix:
 [[5857    1    9    4    4    7   17    9   10    5]
 [   1 6663   33    2   13    0    3   21    5    1]
 [   9    5 5875    5   13    1    7   28   14    1]
 [   6    7   55 5958    2   19    5   42   28    9]
 [   2    7   12    0 5773    1   14   14    4   15]
 [  10    7   10   47   13 5260   34   11   20    9]
 [  22    4    5    0   10   10 5855    1   11    0]
 [   2   11   26    2   19    0    0 6195    2    8]
 [  10   23   17   24   13   11   15    7 5719   12]
 [  13    9    2   20  149    7    1  114   16 5618]]
```



Confusion Matrix
5857
1
9
4
4
7
17
9
10
5
1
6663
33
2
13
0
3
21
5
1
9
5
5875
5
13
1
7
28
14
1
6
7
55
5958
2
19
5
42
28
9
2
7
12
0
5773
1
14
14
4
15
10
7
10
47
13
5260
34
11
20
9
22
4
5
0
10
10
5855
1
11
0
2
11
26
2
19
0
0
6195
2
8
10
23
17
24
13
11
15
7
5719
12
13
9
2
20
149
7
1
114
16
5618
True
Predicted
0
1
2
3
4
5
6
7
8
9
0
1000
2000
3000
4000
5000
6000