```
### 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

import numpy as np

class MLP_ReLU:
    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 relu(self, z):
        return np.maximum(0, z)

    def relu_differentiated(self, z):
        return np.where(z > 0, 1, 0)

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

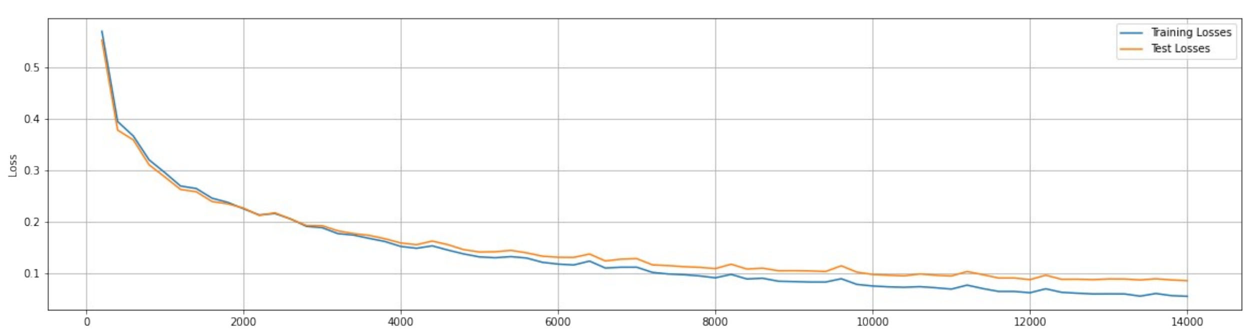
```
        partial_derivative_w_4 = np.dot(self.Activation_3.T, errorz4)
/ self.batch_size
        partial_derivative_b_4 = np.sum(errorz4, axis=0,
keepdims=True) / self.batch_size

        errorz3 = np.dot(errorz4, self.W4.T) *
self.relu_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.relu_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.relu_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()
```

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

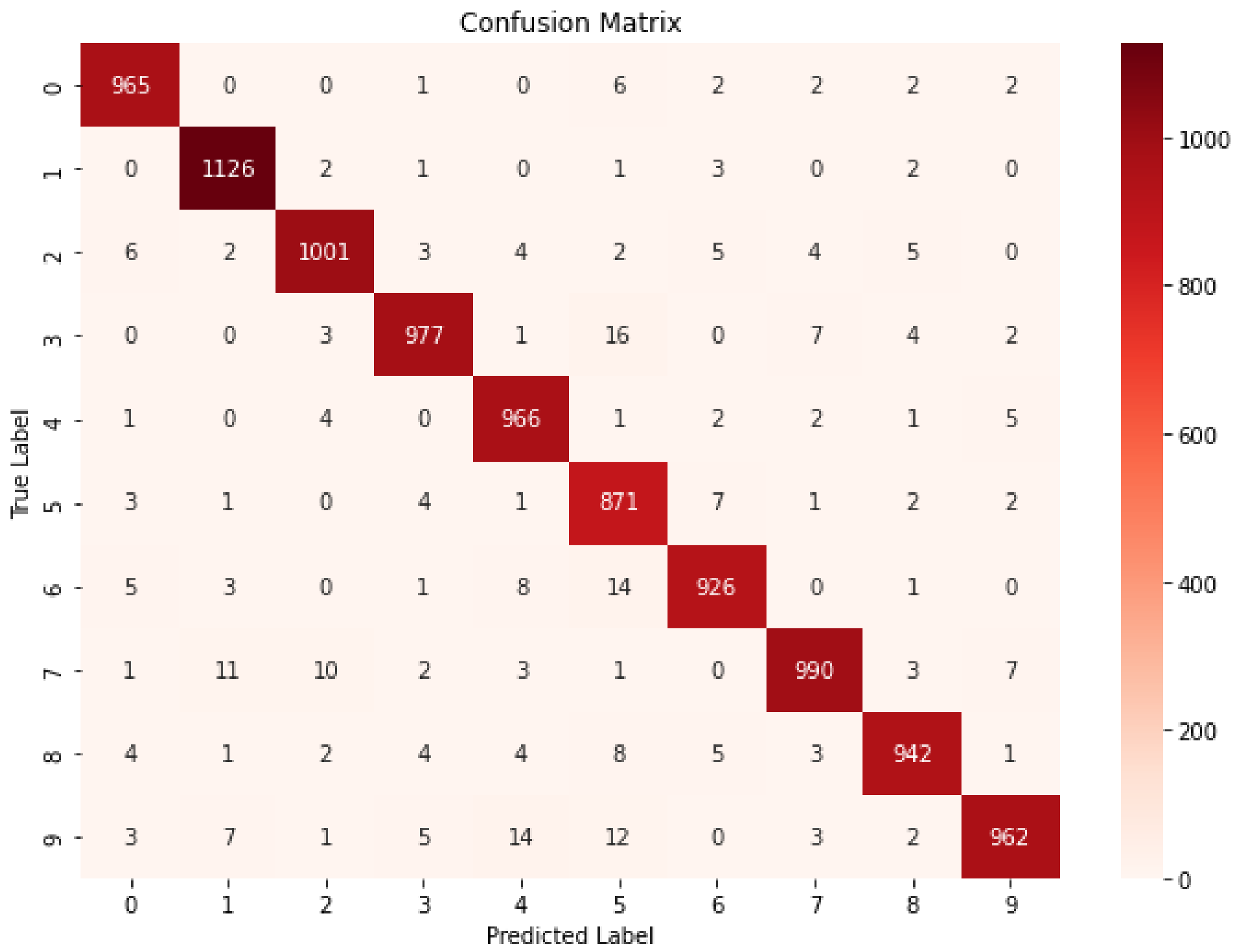
```
Current Iteration 200, Training Loss: 0.5703, Test Loss: 0.5531
Current Iteration 400, Training Loss: 0.3947, Test Loss: 0.3780
Current Iteration 600, Training Loss: 0.3663, Test Loss: 0.3588
Current Iteration 800, Training Loss: 0.3200, Test Loss: 0.3104
Epoch 1/15
Current Iteration 1000, Training Loss: 0.2952, Test Loss: 0.2868
Current Iteration 1200, Training Loss: 0.2691, Test Loss: 0.2624
Current Iteration 1400, Training Loss: 0.2643, Test Loss: 0.2580
Current Iteration 1600, Training Loss: 0.2454, Test Loss: 0.2389
Current Iteration 1800, Training Loss: 0.2373, Test Loss: 0.2345
Epoch 2/15
Current Iteration 2000, Training Loss: 0.2248, Test Loss: 0.2262
Current Iteration 2200, Training Loss: 0.2128, Test Loss: 0.2119
Current Iteration 2400, Training Loss: 0.2156, Test Loss: 0.2170
Current Iteration 2600, Training Loss: 0.2049, Test Loss: 0.2048
Current Iteration 2800, Training Loss: 0.1906, Test Loss: 0.1923
Epoch 3/15
Current Iteration 3000, Training Loss: 0.1881, Test Loss: 0.1920
Current Iteration 3200, Training Loss: 0.1765, Test Loss: 0.1819
Current Iteration 3400, Training Loss: 0.1736, Test Loss: 0.1765
Current Iteration 3600, Training Loss: 0.1672, Test Loss: 0.1730
Epoch 4/15
Current Iteration 3800, Training Loss: 0.1611, Test Loss: 0.1664
Current Iteration 4000, Training Loss: 0.1515, Test Loss: 0.1583
Current Iteration 4200, Training Loss: 0.1479, Test Loss: 0.1549
Current Iteration 4400, Training Loss: 0.1527, Test Loss: 0.1620
Current Iteration 4600, Training Loss: 0.1444, Test Loss: 0.1549
Epoch 5/15
Current Iteration 4800, Training Loss: 0.1369, Test Loss: 0.1453
Current Iteration 5000, Training Loss: 0.1312, Test Loss: 0.1407
Current Iteration 5200, Training Loss: 0.1295, Test Loss: 0.1412
Current Iteration 5400, Training Loss: 0.1317, Test Loss: 0.1438
Current Iteration 5600, Training Loss: 0.1289, Test Loss: 0.1390
Epoch 6/15
Current Iteration 5800, Training Loss: 0.1207, Test Loss: 0.1325
Current Iteration 6000, Training Loss: 0.1171, Test Loss: 0.1304
Current Iteration 6200, Training Loss: 0.1155, Test Loss: 0.1302
Current Iteration 6400, Training Loss: 0.1230, Test Loss: 0.1368
Epoch 7/15
Current Iteration 6600, Training Loss: 0.1095, Test Loss: 0.1232
Current Iteration 6800, Training Loss: 0.1110, Test Loss: 0.1268
Current Iteration 7000, Training Loss: 0.1111, Test Loss: 0.1278
Current Iteration 7200, Training Loss: 0.1010, Test Loss: 0.1156
Current Iteration 7400, Training Loss: 0.0979, Test Loss: 0.1142
Epoch 8/15
Current Iteration 7600, Training Loss: 0.0965, Test Loss: 0.1121
```

```
Current Iteration 7800, Training Loss: 0.0941, Test Loss: 0.1108
Current Iteration 8000, Training Loss: 0.0905, Test Loss: 0.1084
Current Iteration 8200, Training Loss: 0.0971, Test Loss: 0.1169
Current Iteration 8400, Training Loss: 0.0882, Test Loss: 0.1075
Epoch 9/15
Current Iteration 8600, Training Loss: 0.0896, Test Loss: 0.1092
Current Iteration 8800, Training Loss: 0.0838, Test Loss: 0.1042
Current Iteration 9000, Training Loss: 0.0831, Test Loss: 0.1043
Current Iteration 9200, Training Loss: 0.0823, Test Loss: 0.1039
Epoch 10/15
Current Iteration 9400, Training Loss: 0.0822, Test Loss: 0.1027
Current Iteration 9600, Training Loss: 0.0886, Test Loss: 0.1138
Current Iteration 9800, Training Loss: 0.0776, Test Loss: 0.1015
Current Iteration 10000, Training Loss: 0.0745, Test Loss: 0.0970
Current Iteration 10200, Training Loss: 0.0729, Test Loss: 0.0955
Epoch 11/15
Current Iteration 10400, Training Loss: 0.0720, Test Loss: 0.0943
Current Iteration 10600, Training Loss: 0.0732, Test Loss: 0.0980
Current Iteration 10800, Training Loss: 0.0711, Test Loss: 0.0955
Current Iteration 11000, Training Loss: 0.0685, Test Loss: 0.0940
Current Iteration 11200, Training Loss: 0.0762, Test Loss: 0.1027
Epoch 12/15
Current Iteration 11400, Training Loss: 0.0695, Test Loss: 0.0966
Current Iteration 11600, Training Loss: 0.0640, Test Loss: 0.0901
Current Iteration 11800, Training Loss: 0.0639, Test Loss: 0.0901
Current Iteration 12000, Training Loss: 0.0614, Test Loss: 0.0870
Epoch 13/15
Current Iteration 12200, Training Loss: 0.0691, Test Loss: 0.0955
Current Iteration 12400, Training Loss: 0.0623, Test Loss: 0.0876
Current Iteration 12600, Training Loss: 0.0605, Test Loss: 0.0878
Current Iteration 12800, Training Loss: 0.0591, Test Loss: 0.0869
Current Iteration 13000, Training Loss: 0.0592, Test Loss: 0.0882
Epoch 14/15
Current Iteration 13200, Training Loss: 0.0591, Test Loss: 0.0881
Current Iteration 13400, Training Loss: 0.0548, Test Loss: 0.0863
Current Iteration 13600, Training Loss: 0.0599, Test Loss: 0.0886
Current Iteration 13800, Training Loss: 0.0557, Test Loss: 0.0863
Current Iteration 14000, Training Loss: 0.0545, Test Loss: 0.0848
Epoch 15/15
```

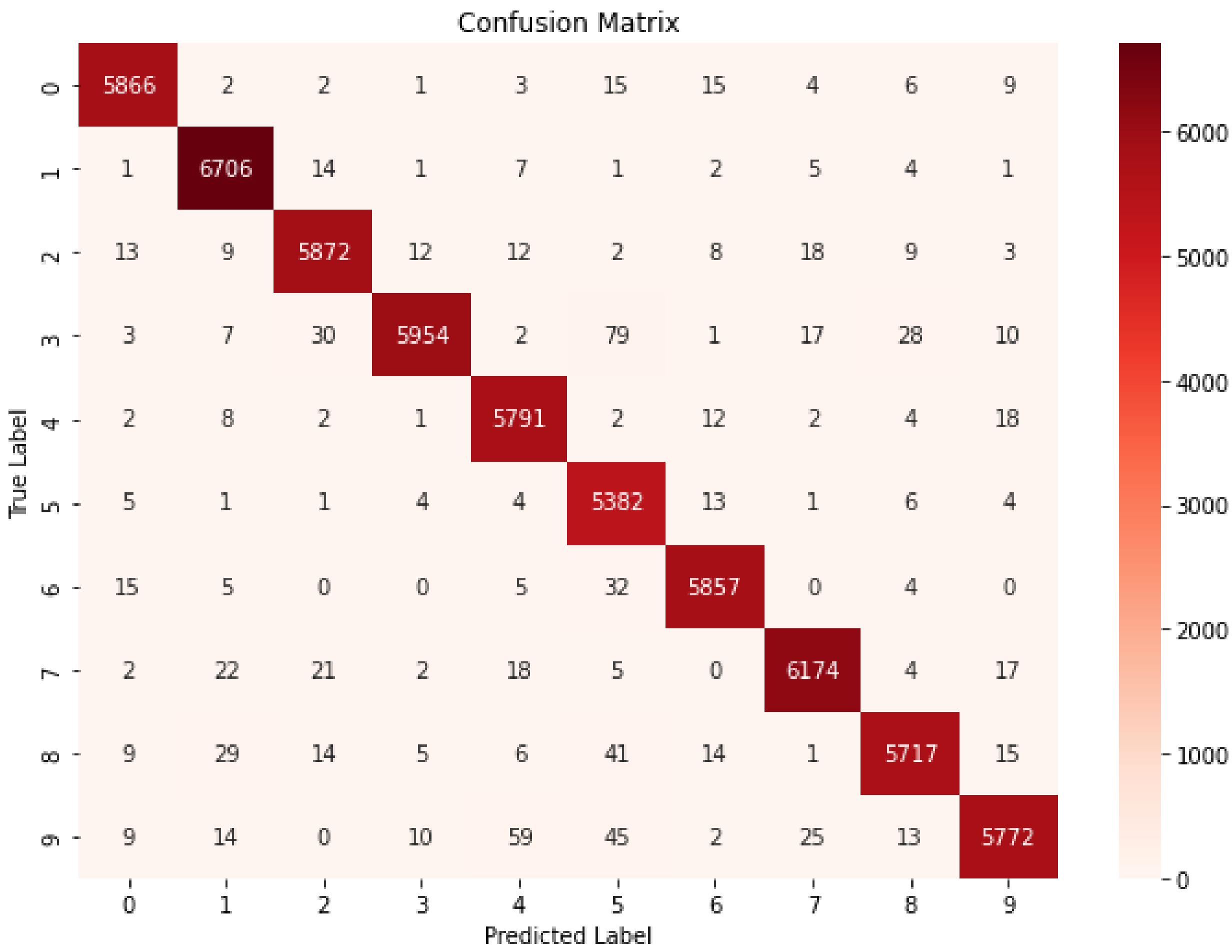


```python
from sklearn.metrics import confusion_matrix, accuracy_score
import seaborn as sns
def final_test(model, test_loader):
    all_preds = []
    all_labels = []
    for images, labels in test_loader:
        X = images.view(-1, 28*28).numpy()
        Y_hat = model.forward(X)
        preds = np.argmax(Y_hat, 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=(10, 7))
    sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Reds')
    plt.xlabel('Predicted Label')
    plt.ylabel('True Label')
    plt.title('Confusion Matrix')
    plt.show()

final_test(model,test_loader)
```

```
Final Test Accuracy: 97.26%
Confusion Matrix:
 [[ 965    0    0    1    0    6    2    2    2    2]
 [    0 1126    2    1    0    1    3    0    2    0]
 [    6    2 1001    3    4    2    5    4    5    0]
 [    0    0    3  977    1   16    0    7    4    2]
 [    1    0    4    0  966    1    2    2    1    5]
 [    3    1    0    4    1  871    7    1    2    2]
 [    5    3    0    1    8   14  926    0    1    0]
 [    1   11   10    2    3    1    0  990    3    7]
 [    4    1    2    4    4    8    5    3  942    1]
 [    3    7    1    5   14   12    0    3    2  962]]
```




```
final_test(model,train_loader)

Final Test Accuracy: 98.48%
Confusion Matrix:
 [[5866    2    2    1    3   15   15    4    6    9]
 [   1 6706   14    1    7    1    2    5    4    1]
 [  13    9 5872   12   12    2    8   18    9    3]
 [   3    7   30 5954    2   79    1   17   28   10]
 [   2    8    2    1 5791    2   12    2    4   18]
 [   5    1    1    4    4 5382   13    1    6    4]
 [  15    5    0    0    5   32 5857    0    4    0]
 [   2   22   21    2   18    5    0 6174    4   17]
 [   9   29   14    5    6   41   14    1 5717   15]
 [   9   14    0   10   59   45    2   25   13 5772]]
```


Confusion Matrix
True Label
Predicted Label
0 1 2 3 4 5 6 7 8 9
5866 2 2 1 3 15 15 4 6 9
1 6706 14 1 7 1 2 5 4 1
13 9 5872 12 12 2 8 18 9 3
3 7 30 5954 2 79 1 17 28 10
2 8 2 1 5791 2 12 2 4 18
5 1 1 4 4 5382 13 1 6 4
15 5 0 0 5 32 5857 0 4 0
2 22 21 2 18 5 0 6174 4 17
9 29 14 5 6 41 14 1 5717 15
9 14 0 10 59 45 2 25 13 5772
6000
5000
4000
3000
2000
1000
0