```
### 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 torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, TensorDataset

class MLP_torch(nn.Module):
    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,regulatisation_constant =
0.0001):
        super(MLP_torch, self).__init__()
        self.regularisation_constant = regulatisation_constant
        self.first_layer = nn.Linear(input_layer_size,
first_hidden_layer_size)
        self.second_layer = nn.Linear(first_hidden_layer_size,
second_hidden_layer_size)
        self.third_layer = nn.Linear(second_hidden_layer_size,
third_hidden_layer_size)
        self.output_layer = nn.Linear(third_hidden_layer_size,
output_layer_size)

    def forward(self, x):
        x = torch.sigmoid(self.first_layer(x))
        x = torch.sigmoid(self.second_layer(x))
        x = torch.sigmoid(self.third_layer(x))
        x = self.output_layer(x)
        x = torch.softmax(x, dim=1)
        return x
```

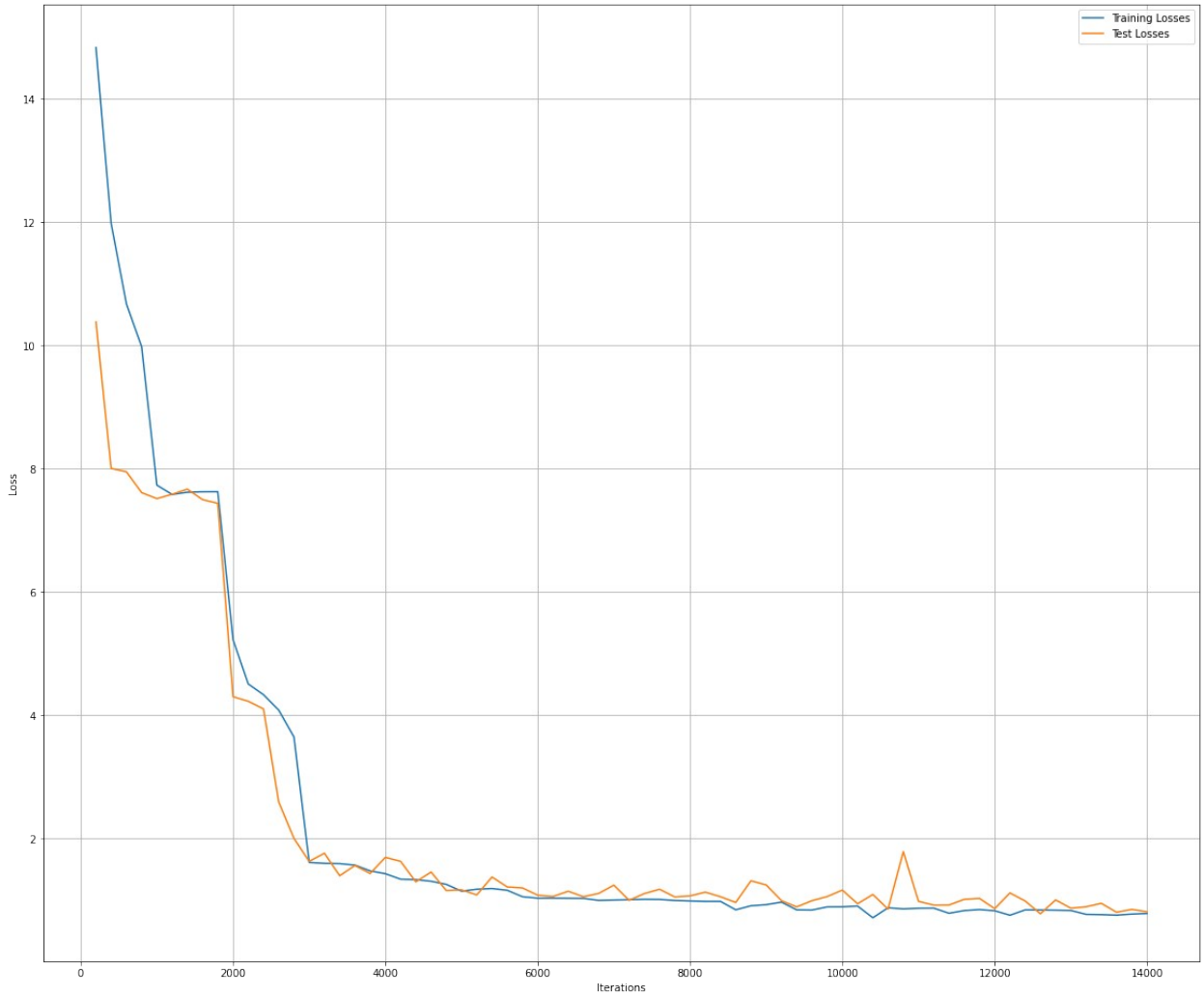
```
    def entropy_class_loss(self, Y, Y_hat):
        epsilon = 1e-10
        Y_hat = torch.clamp(Y_hat, epsilon, 1 - epsilon)

        loss = -torch.sum(Y * torch.log(Y_hat)) / Y.size(0)


        l2_norm_sum = 0
        for param in self.parameters():
            param_norm = torch.norm(param, p=2)
            l2_norm_sum += param_norm

        return loss + self.regularisation_constant*l2_norm_sum

import torch
import torch.nn as nn
import matplotlib.pyplot as plt
from torch.optim import Adam
import torch.nn.functional as F
import torch.nn.functional as F

def train(model, optimizer, train_loader, test_loader, epochs,
plot_interval=200, device='cpu'):
    training_loss_list = []
    test_loss_list = []
    iteration_counts = []
    current_iteration = 0

    for epoch in range(epochs):
        model.train()
        running_loss = 0
        for batch_idx, (images, labels) in enumerate(train_loader):
            images, labels = images.view(images.size(0), -
1).to(device), labels.to(device)
            labels_one_hot = F.one_hot(labels,
num_classes=10).float().to(device)
            optimizer.zero_grad()
            outputs = model(images)
            loss = model.entropy_class_loss(outputs, labels_one_hot)
            loss.backward()
            optimizer.step()

            running_loss += loss.item()
            current_iteration += 1

            if current_iteration % plot_interval == 0:
                iteration_counts.append(current_iteration)
                avg_train_loss = running_loss / (batch_idx + 1)
                training_loss_list.append(avg_train_loss)
```

```
                model.eval()
                total_test_loss = 0
                total_test_samples = 0
                with torch.no_grad():
                    for test_images, test_labels in test_loader:
                        test_images, test_labels =
test_images.view(test_images.size(0), -1).to(device),
test_labels.to(device)

                        test_outputs = model(test_images)
                        test_labels_one_hot = F.one_hot(test_labels,
num_classes=10).float().to(device)
                        test_loss =
model.entropy_class_loss(test_outputs, test_labels_one_hot)
                        total_test_loss += test_loss.item() *
test_images.size(0)
                        total_test_samples += test_images.size(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, 17))
    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_torch()
optimizer = Adam(model.parameters())
train(model, optimizer, train_loader, test_loader, epochs=15)
```

```
Current Iteration 200, Training Loss: 14.8370, Test Loss: 10.3845
Current Iteration 400, Training Loss: 11.9816, Test Loss: 8.0058
Current Iteration 600, Training Loss: 10.6772, Test Loss: 7.9521
Current Iteration 800, Training Loss: 9.9824, Test Loss: 7.6136
Epoch 1/15
Current Iteration 1000, Training Loss: 7.7351, Test Loss: 7.5165
Current Iteration 1200, Training Loss: 7.5839, Test Loss: 7.5872
Current Iteration 1400, Training Loss: 7.6201, Test Loss: 7.6701
Current Iteration 1600, Training Loss: 7.6275, Test Loss: 7.5003
Current Iteration 1800, Training Loss: 7.6283, Test Loss: 7.4399
Epoch 2/15
```

```
Current Iteration 2000, Training Loss: 5.2271, Test Loss: 4.3022
Current Iteration 2200, Training Loss: 4.5080, Test Loss: 4.2277
Current Iteration 2400, Training Loss: 4.3342, Test Loss: 4.1028
Current Iteration 2600, Training Loss: 4.0838, Test Loss: 2.5926
Current Iteration 2800, Training Loss: 3.6474, Test Loss: 2.0018
Epoch 3/15
Current Iteration 3000, Training Loss: 1.6120, Test Loss: 1.6286
Current Iteration 3200, Training Loss: 1.5974, Test Loss: 1.7613
Current Iteration 3400, Training Loss: 1.5930, Test Loss: 1.3964
Current Iteration 3600, Training Loss: 1.5685, Test Loss: 1.5625
Epoch 4/15
Current Iteration 3800, Training Loss: 1.4752, Test Loss: 1.4325
Current Iteration 4000, Training Loss: 1.4293, Test Loss: 1.6935
Current Iteration 4200, Training Loss: 1.3398, Test Loss: 1.6296
Current Iteration 4400, Training Loss: 1.3324, Test Loss: 1.2966
Current Iteration 4600, Training Loss: 1.3060, Test Loss: 1.4562
Epoch 5/15
Current Iteration 4800, Training Loss: 1.2556, Test Loss: 1.1559
Current Iteration 5000, Training Loss: 1.1453, Test Loss: 1.1667
Current Iteration 5200, Training Loss: 1.1771, Test Loss: 1.0848
Current Iteration 5400, Training Loss: 1.1882, Test Loss: 1.3754
Current Iteration 5600, Training Loss: 1.1598, Test Loss: 1.2138
Epoch 6/15
Current Iteration 5800, Training Loss: 1.0559, Test Loss: 1.1979
Current Iteration 6000, Training Loss: 1.0320, Test Loss: 1.0801
Current Iteration 6200, Training Loss: 1.0326, Test Loss: 1.0596
Current Iteration 6400, Training Loss: 1.0306, Test Loss: 1.1444
Epoch 7/15
Current Iteration 6600, Training Loss: 1.0287, Test Loss: 1.0559
Current Iteration 6800, Training Loss: 0.9960, Test Loss: 1.1091
Current Iteration 7000, Training Loss: 1.0020, Test Loss: 1.2430
Current Iteration 7200, Training Loss: 1.0069, Test Loss: 0.9973
Current Iteration 7400, Training Loss: 1.0136, Test Loss: 1.1068
Epoch 8/15
Current Iteration 7600, Training Loss: 1.0113, Test Loss: 1.1766
Current Iteration 7800, Training Loss: 0.9947, Test Loss: 1.0521
Current Iteration 8000, Training Loss: 0.9866, Test Loss: 1.0694
Current Iteration 8200, Training Loss: 0.9803, Test Loss: 1.1300
Current Iteration 8400, Training Loss: 0.9800, Test Loss: 1.0557
Epoch 9/15
Current Iteration 8600, Training Loss: 0.8429, Test Loss: 0.9627
Current Iteration 8800, Training Loss: 0.9082, Test Loss: 1.3135
Current Iteration 9000, Training Loss: 0.9263, Test Loss: 1.2462
Current Iteration 9200, Training Loss: 0.9679, Test Loss: 0.9925
Epoch 10/15
Current Iteration 9400, Training Loss: 0.8419, Test Loss: 0.8898
Current Iteration 9600, Training Loss: 0.8386, Test Loss: 0.9911
Current Iteration 9800, Training Loss: 0.8898, Test Loss: 1.0584
Current Iteration 10000, Training Loss: 0.8912, Test Loss: 1.1633
```

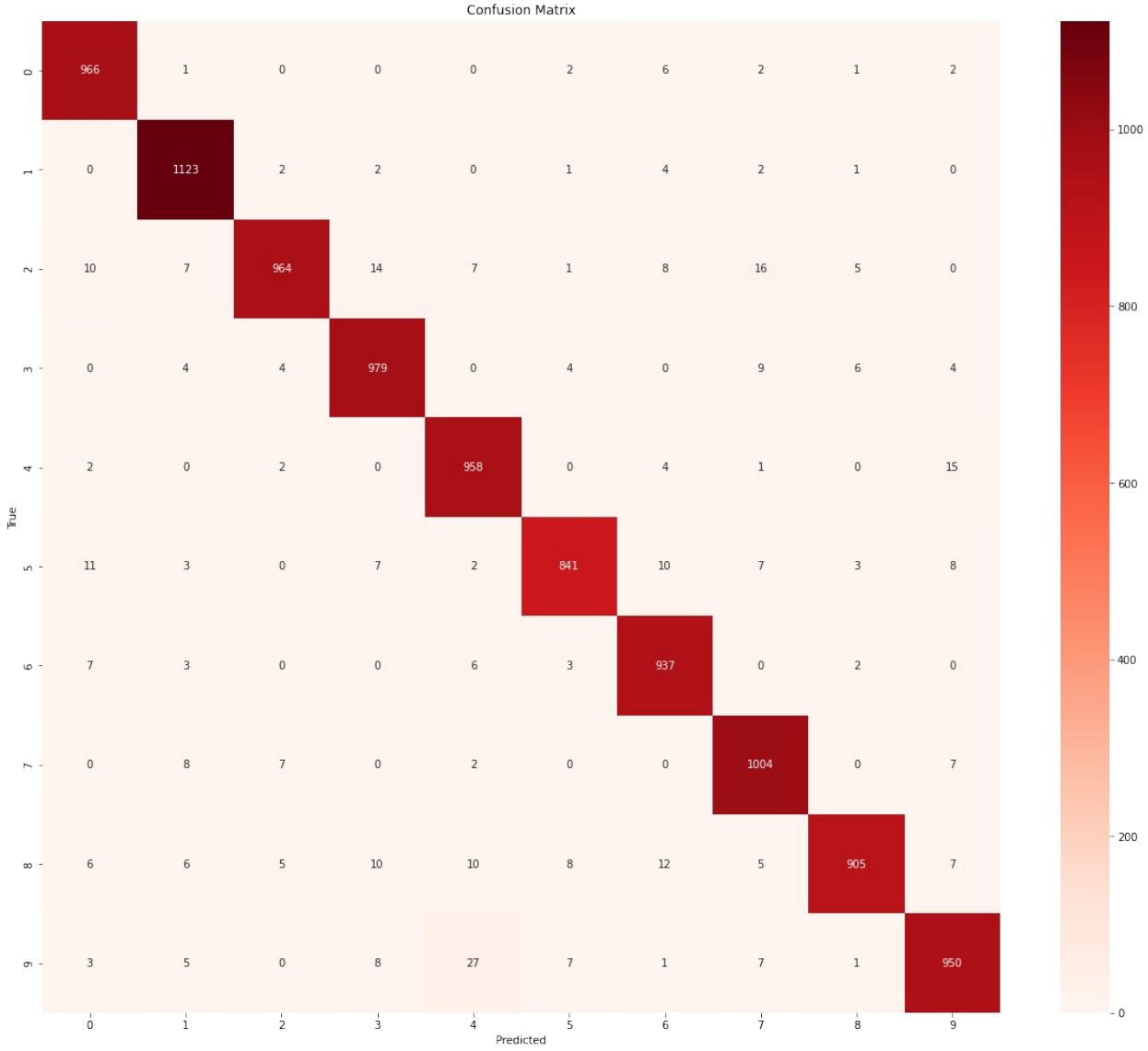
```
Current Iteration 10200, Training Loss: 0.9041, Test Loss: 0.9412
Epoch 11/15
Current Iteration 10400, Training Loss: 0.7138, Test Loss: 1.0924
Current Iteration 10600, Training Loss: 0.8758, Test Loss: 0.8545
Current Iteration 10800, Training Loss: 0.8573, Test Loss: 1.7867
Current Iteration 11000, Training Loss: 0.8682, Test Loss: 0.9807
Current Iteration 11200, Training Loss: 0.8700, Test Loss: 0.9192
Epoch 12/15
Current Iteration 11400, Training Loss: 0.7863, Test Loss: 0.9207
Current Iteration 11600, Training Loss: 0.8303, Test Loss: 1.0136
Current Iteration 11800, Training Loss: 0.8459, Test Loss: 1.0285
Current Iteration 12000, Training Loss: 0.8274, Test Loss: 0.8618
Epoch 13/15
Current Iteration 12200, Training Loss: 0.7547, Test Loss: 1.1169
Current Iteration 12400, Training Loss: 0.8412, Test Loss: 0.9867
Current Iteration 12600, Training Loss: 0.8395, Test Loss: 0.7782
Current Iteration 12800, Training Loss: 0.8365, Test Loss: 1.0042
Current Iteration 13000, Training Loss: 0.8315, Test Loss: 0.8685
Epoch 14/15
Current Iteration 13200, Training Loss: 0.7687, Test Loss: 0.8922
Current Iteration 13400, Training Loss: 0.7642, Test Loss: 0.9481
Current Iteration 13600, Training Loss: 0.7562, Test Loss: 0.8029
Current Iteration 13800, Training Loss: 0.7733, Test Loss: 0.8487
Current Iteration 14000, Training Loss: 0.7834, Test Loss: 0.8111
Epoch 15/15
```




```
import torch
import numpy as np
from sklearn.metrics import confusion_matrix, accuracy_score
import seaborn as sns
import matplotlib.pyplot as plt

def final_test(model, test_loader):
    model.eval()
    all_preds = []
    all_labels = []
    with torch.no_grad():
        for images, labels in test_loader:
            images = images.view(images.size(0), -1)
            outputs = model(images)
            _, preds = torch.max(outputs, 1)
            all_preds.append(preds.cpu().numpy())
            all_labels.append(labels.cpu().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)
    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: 96.27%
```



```
final_test(model, train_loader)
```

```
Final Test Accuracy: 97.17%
```