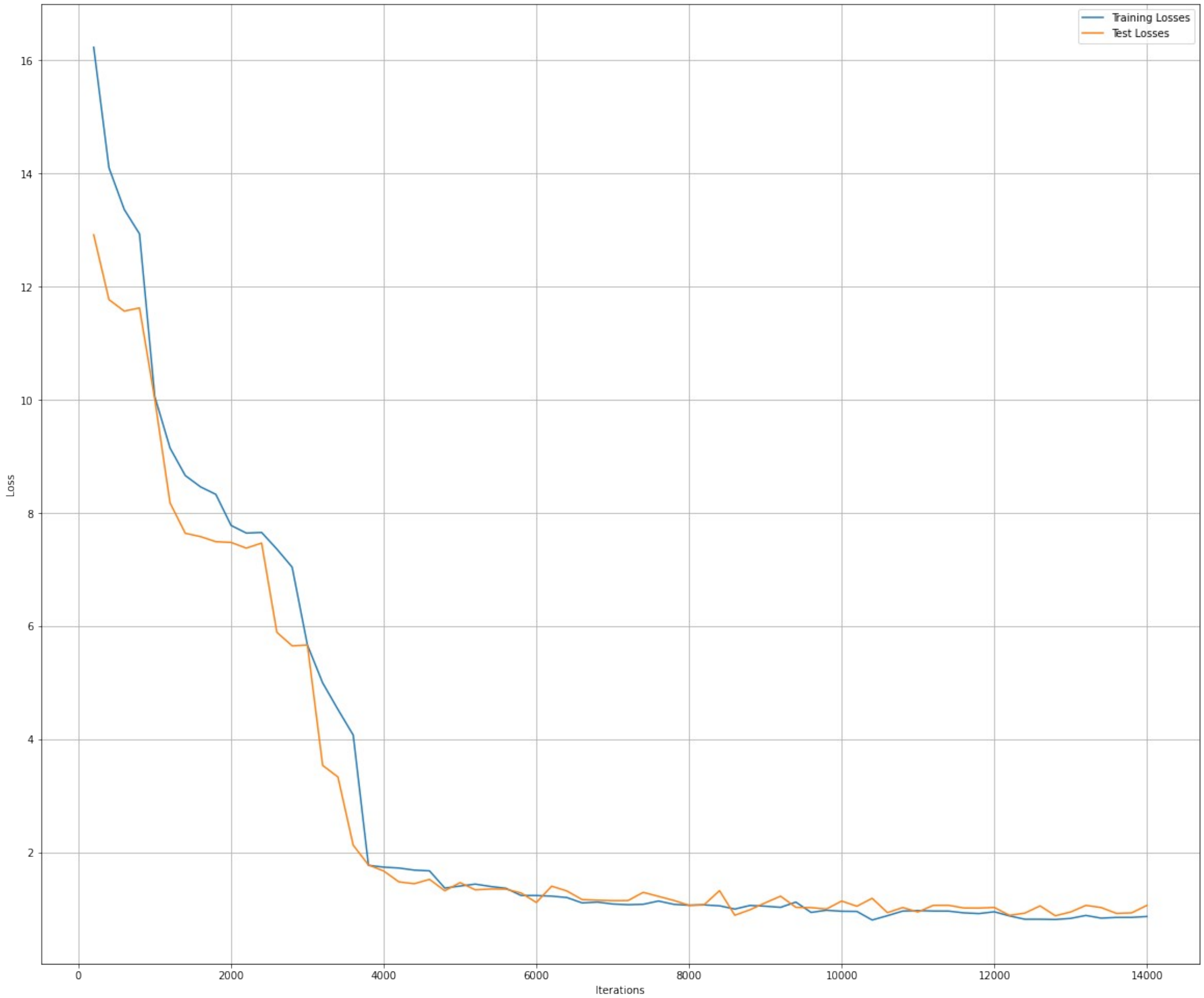
```
### Importing the dataset - in the form of batches - using dataloader
package - mini batch descent - size =64
import torch
from torchvision import datasets, transforms
from torch.utils.data import DataLoader

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):
        super(MLP_torch, self).__init__()
        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)
        return loss

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):
    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), labels
            labels_one_hot = F.one_hot(labels, num_classes=10).float()

            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), test_labels

                        test_outputs = model(test_images)
                        test_labels_one_hot = F.one_hot(test_labels,
num_classes=10).float()
```

```
                        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: 16.2288, Test Loss: 12.9149
Current Iteration [400], Training Loss: 14.0975, Test Loss: 11.7689
Current Iteration [600], Training Loss: 13.3621, Test Loss: 11.5662
Current Iteration [800], Training Loss: 12.9302, Test Loss: 11.6226
Epoch [1/15]
Current Iteration [1000], Training Loss: 10.0574, Test Loss: 10.0100
Current Iteration [1200], Training Loss: 9.1474, Test Loss: 8.1779
Current Iteration [1400], Training Loss: 8.6605, Test Loss: 7.6411
Current Iteration [1600], Training Loss: 8.4607, Test Loss: 7.5820
Current Iteration [1800], Training Loss: 8.3287, Test Loss: 7.4925
Epoch [2/15]
Current Iteration [2000], Training Loss: 7.7794, Test Loss: 7.4789
Current Iteration [2200], Training Loss: 7.6441, Test Loss: 7.3788
Current Iteration [2400], Training Loss: 7.6558, Test Loss: 7.4683
Current Iteration [2600], Training Loss: 7.3617, Test Loss: 5.8912
Current Iteration [2800], Training Loss: 7.0421, Test Loss: 5.6524
Epoch [3/15]
Current Iteration [3000], Training Loss: 5.6680, Test Loss: 5.6672
Current Iteration [3200], Training Loss: 4.9975, Test Loss: 3.5398
Current Iteration [3400], Training Loss: 4.5287, Test Loss: 3.3356
Current Iteration [3600], Training Loss: 4.0769, Test Loss: 2.1312
Epoch [4/15]
```

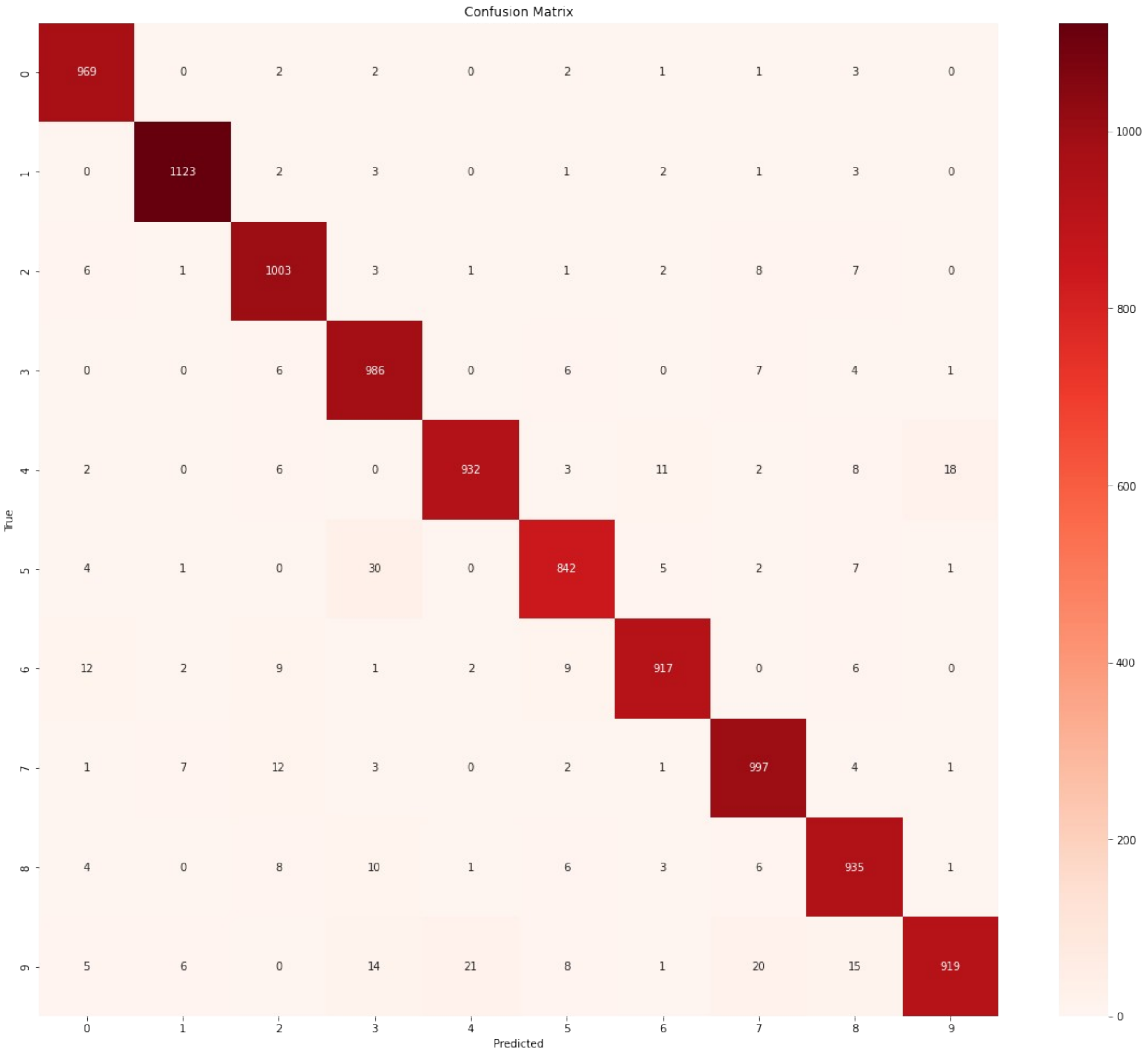
```
Current Iteration [3800], Training Loss: 1.7744, Test Loss: 1.7805
Current Iteration [4000], Training Loss: 1.7427, Test Loss: 1.6741
Current Iteration [4200], Training Loss: 1.7269, Test Loss: 1.4815
Current Iteration [4400], Training Loss: 1.6901, Test Loss: 1.4493
Current Iteration [4600], Training Loss: 1.6770, Test Loss: 1.5266
Epoch [5/15]
Current Iteration [4800], Training Loss: 1.3730, Test Loss: 1.3253
Current Iteration [5000], Training Loss: 1.4087, Test Loss: 1.4684
Current Iteration [5200], Training Loss: 1.4427, Test Loss: 1.3422
Current Iteration [5400], Training Loss: 1.4011, Test Loss: 1.3566
Current Iteration [5600], Training Loss: 1.3695, Test Loss: 1.3542
Epoch [6/15]
Current Iteration [5800], Training Loss: 1.2421, Test Loss: 1.2868
Current Iteration [6000], Training Loss: 1.2413, Test Loss: 1.1178
Current Iteration [6200], Training Loss: 1.2303, Test Loss: 1.4063
Current Iteration [6400], Training Loss: 1.2054, Test Loss: 1.3215
Epoch [7/15]
Current Iteration [6600], Training Loss: 1.1108, Test Loss: 1.1693
Current Iteration [6800], Training Loss: 1.1253, Test Loss: 1.1583
Current Iteration [7000], Training Loss: 1.0912, Test Loss: 1.1507
Current Iteration [7200], Training Loss: 1.0791, Test Loss: 1.1518
Current Iteration [7400], Training Loss: 1.0862, Test Loss: 1.2971
Epoch [8/15]
Current Iteration [7600], Training Loss: 1.1443, Test Loss: 1.2267
Current Iteration [7800], Training Loss: 1.0860, Test Loss: 1.1547
Current Iteration [8000], Training Loss: 1.0688, Test Loss: 1.0689
Current Iteration [8200], Training Loss: 1.0747, Test Loss: 1.0805
Current Iteration [8400], Training Loss: 1.0592, Test Loss: 1.3271
Epoch [9/15]
Current Iteration [8600], Training Loss: 1.0019, Test Loss: 0.8924
Current Iteration [8800], Training Loss: 1.0656, Test Loss: 0.9898
Current Iteration [9000], Training Loss: 1.0523, Test Loss: 1.1128
Current Iteration [9200], Training Loss: 1.0323, Test Loss: 1.2300
Epoch [10/15]
Current Iteration [9400], Training Loss: 1.1250, Test Loss: 1.0324
Current Iteration [9600], Training Loss: 0.9422, Test Loss: 1.0272
Current Iteration [9800], Training Loss: 0.9809, Test Loss: 1.0020
Current Iteration [10000], Training Loss: 0.9629, Test Loss: 1.1437
Current Iteration [10200], Training Loss: 0.9596, Test Loss: 1.0513
Epoch [11/15]
Current Iteration [10400], Training Loss: 0.8088, Test Loss: 1.1928
Current Iteration [10600], Training Loss: 0.8851, Test Loss: 0.9382
Current Iteration [10800], Training Loss: 0.9659, Test Loss: 1.0283
Current Iteration [11000], Training Loss: 0.9750, Test Loss: 0.9502
Current Iteration [11200], Training Loss: 0.9667, Test Loss: 1.0673
Epoch [12/15]
Current Iteration [11400], Training Loss: 0.9656, Test Loss: 1.0686
Current Iteration [11600], Training Loss: 0.9346, Test Loss: 1.0207
Current Iteration [11800], Training Loss: 0.9222, Test Loss: 1.0191
Current Iteration [12000], Training Loss: 0.9533, Test Loss: 1.0300
Epoch [13/15]
Current Iteration [12200], Training Loss: 0.8811, Test Loss: 0.8911
Current Iteration [12400], Training Loss: 0.8225, Test Loss: 0.9300
Current Iteration [12600], Training Loss: 0.8224, Test Loss: 1.0582
Current Iteration [12800], Training Loss: 0.8180, Test Loss: 0.8838
Current Iteration [13000], Training Loss: 0.8374, Test Loss: 0.9495
Epoch [14/15]
Current Iteration [13200], Training Loss: 0.8896, Test Loss: 1.0667
Current Iteration [13400], Training Loss: 0.8419, Test Loss: 1.0289
Current Iteration [13600], Training Loss: 0.8547, Test Loss: 0.9252
Current Iteration [13800], Training Loss: 0.8552, Test Loss: 0.9345
Current Iteration [14000], Training Loss: 0.8711, Test Loss: 1.0675
Epoch [15/15]
```

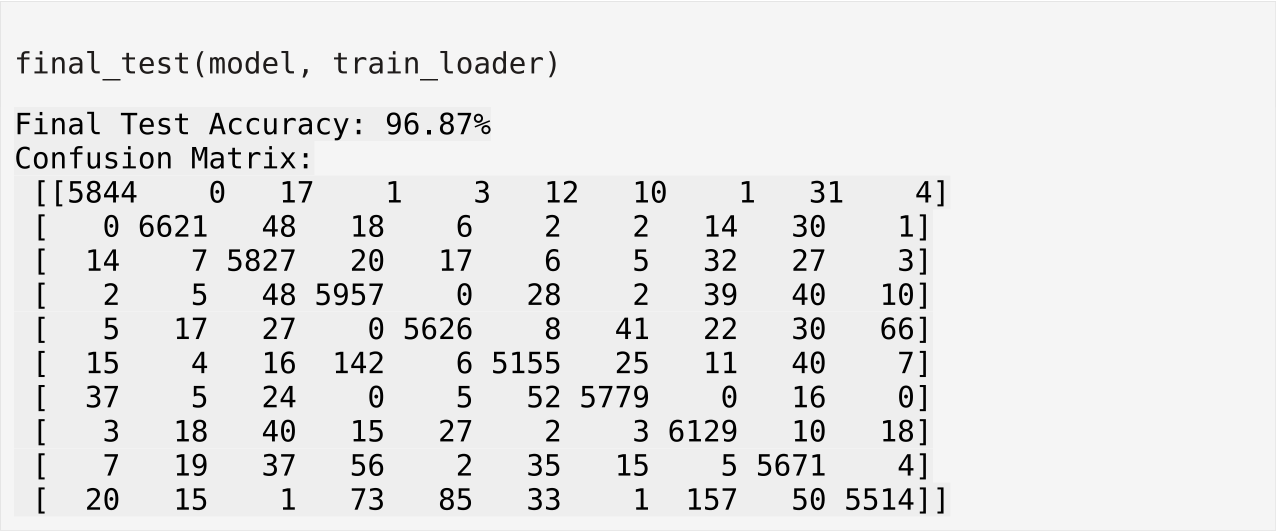


```python
import torch
import numpy as np
from sklearn.metrics import confusion_matrix, accuracy_score
```

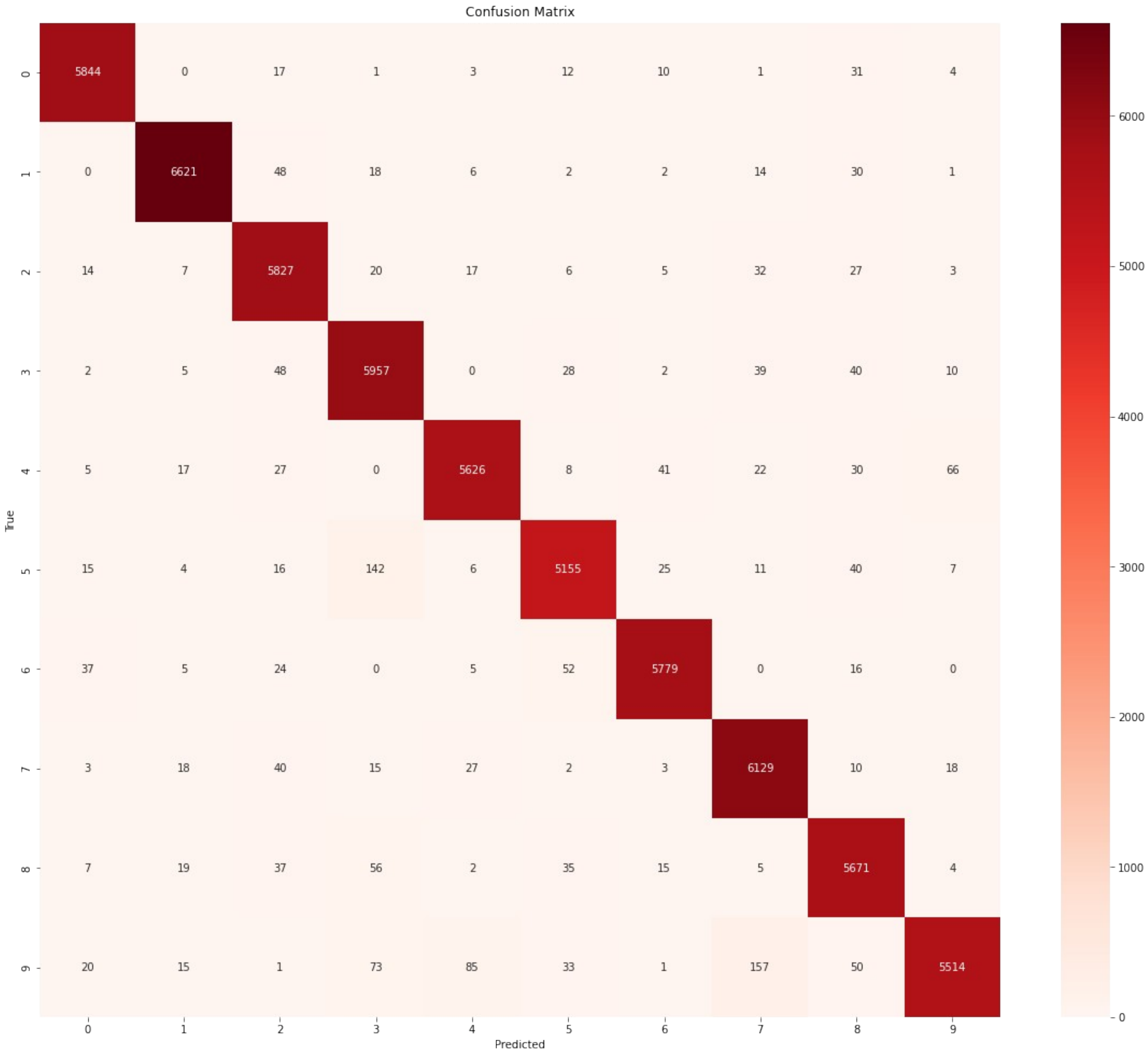
```python
import seaborn as sns
import matplotlib.pyplot as plt

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)
            labels = labels

            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)
    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: 96.23%
Confusion Matrix:
 [[ 969    0    2    2    0    2    1    1    3    0]
 [   0 1123    2    3    0    1    2    1    3    0]
 [   6    1 1003    3    1    1    2    8    7    0]
 [   0    0    6  986    0    6    0    7    4    1]
 [   2    0    6    0  932    3   11    2    8   18]
 [   4    1    0   30    0  842    5    2    7    1]
 [  12    2    9    1    2    9  917    0    6    0]
 [   1    7   12    3    0    2    1  997    4    1]
 [   4    0    8   10    1    6    3    6  935    1]
 [   5    6    0   14   21    8    1   20   15  919]]
```





```
final_test(model, train_loader)

Final Test Accuracy: 96.87%
Confusion Matrix:
 [[5844    0   17    1    3   12   10    1   31    4]
 [   0 6621   48   18    6    2    2   14   30    1]
 [  14    7 5827   20   17    6    5   32   27    3]
 [   2    5   48 5957    0   28    2   39   40   10]
 [   5   17   27    0 5626    8   41   22   30   66]
 [  15    4   16  142    6 5155   25   11   40    7]
 [  37    5   24    0    5   52 5779    0   16    0]
 [   3   18   40   15   27    2    3 6129   10   18]
 [   7   19   37   56    2   35   15    5 5671    4]
 [  20   15    1   73   85   33    1  157   50 5514]]
```

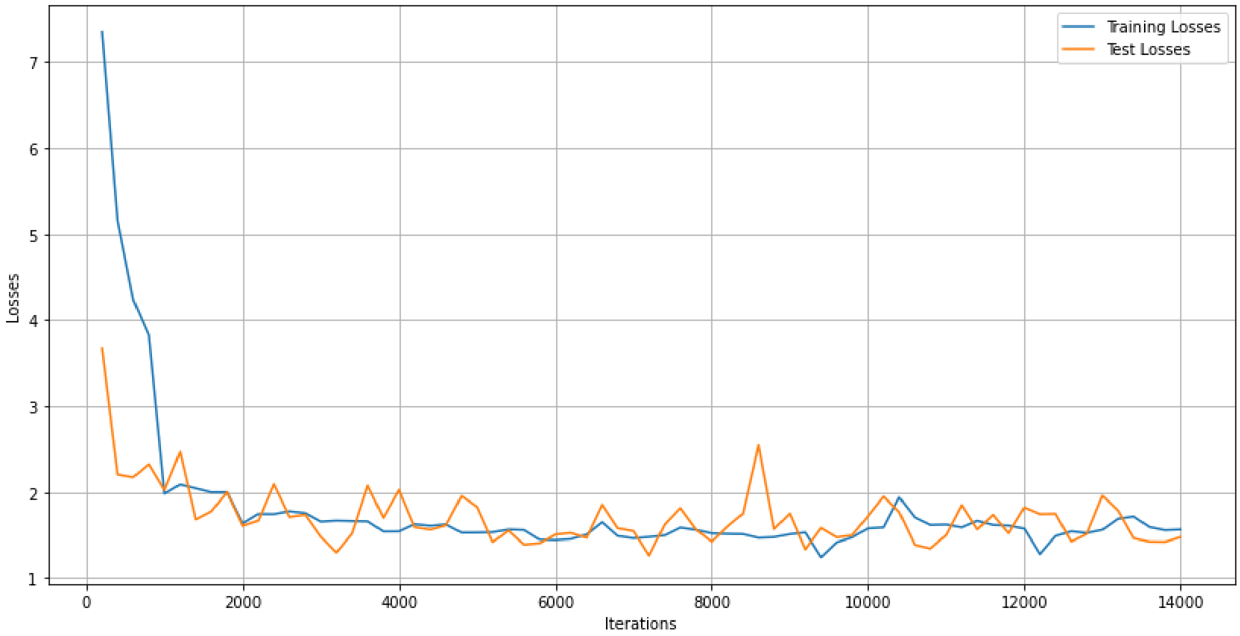


	0	1	2	3	4	5	6	7	8	9
0	5844	0	17	1	3	12	10	1	31	4
1	0	6621	48	18	6	2	2	14	30	1
2	14	7	5827	20	17	6	5	32	27	3
3	2	5	48	5957	0	28	2	39	40	10
4	5	17	27	0	5626	8	41	22	30	66
5	15	4	16	142	6	5155	25	11	40	7
6	37	5	24	0	5	52	5779	0	16	0
7	3	18	40	15	27	2	3	6129	10	18
8	7	19	37	56	2	35	15	5	5671	4
9	20	15	1	73	85	33	1	157	50	5514

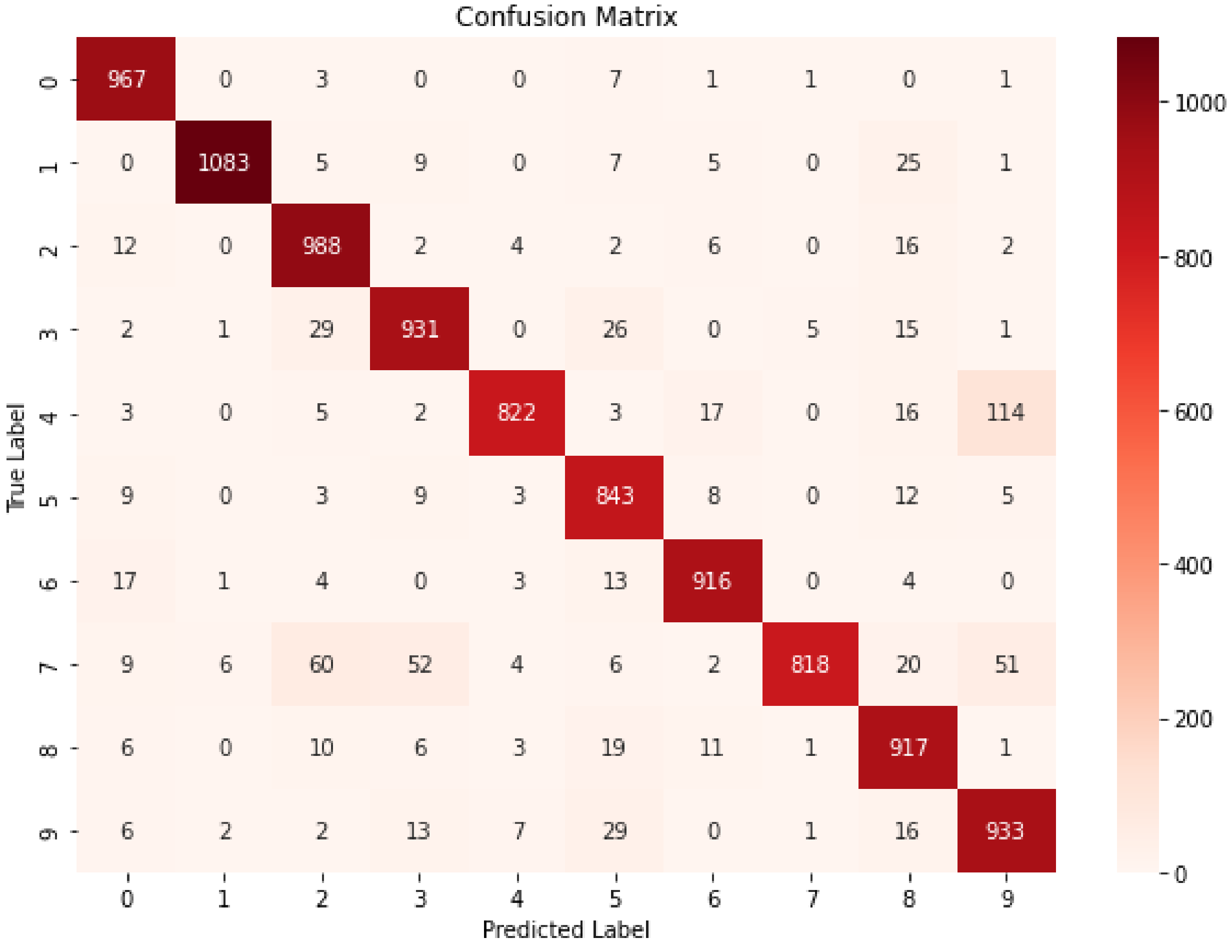
TANH

```
### 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):
        super(MLP_torch, self).__init__()
        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.tanh(self.first_layer(x))
        x = torch.tanh(self.second_layer(x))
        x = torch.tanh(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)
        return loss

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('Losses')
    plt.legend()
    plt.grid(True)
    plt.show()

model = MLP_torch()
optimizer = Adam(model.parameters())
train(model, optimizer, train_loader, test_loader, epochs=15)
```

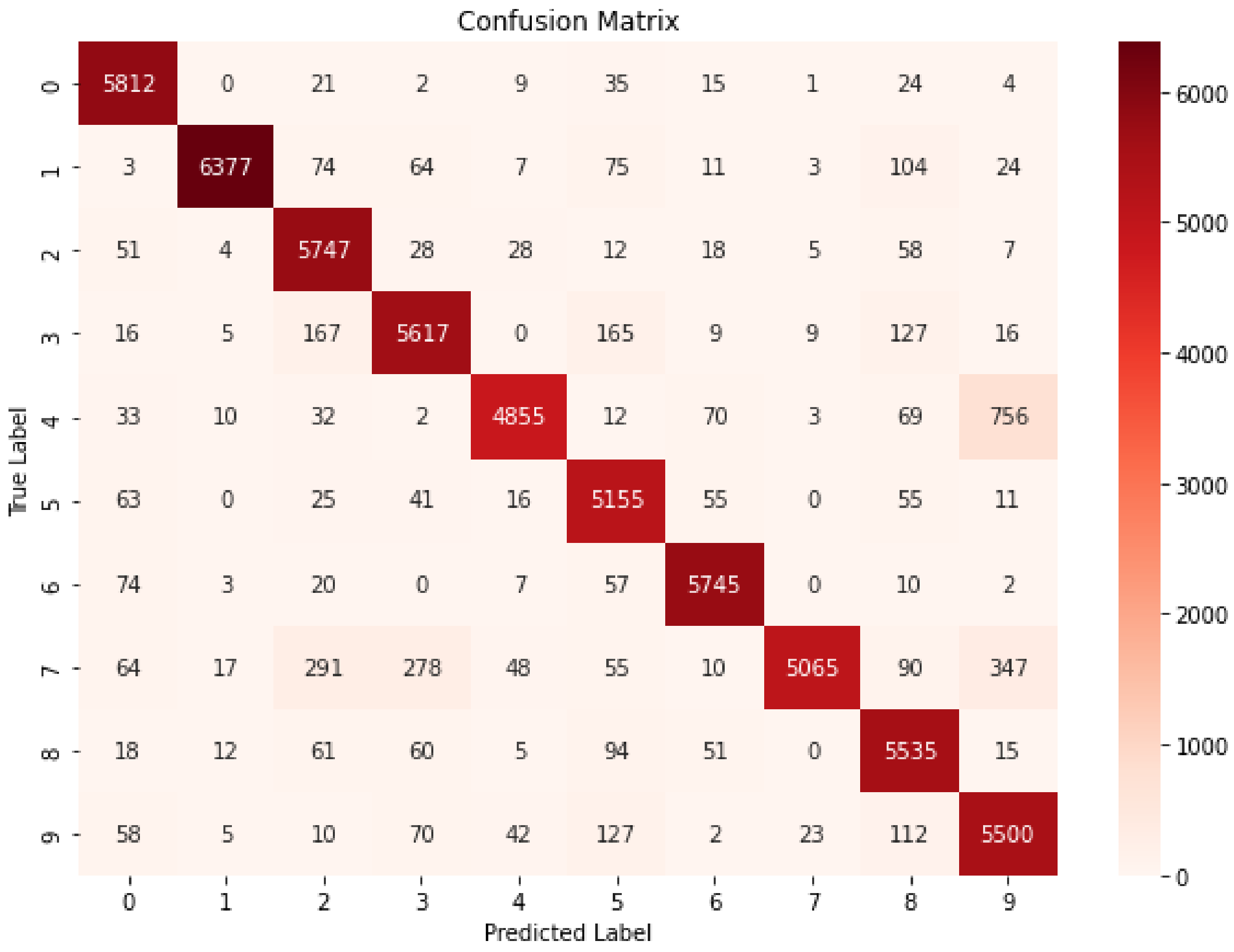
```
Current Iteration 200, Training Loss: 7.3515, Test Loss: 3.6744
Current Iteration 400, Training Loss: 5.1583, Test Loss: 2.2069
Current Iteration 600, Training Loss: 4.2415, Test Loss: 2.1752
Current Iteration 800, Training Loss: 3.8239, Test Loss: 2.3268
Epoch 1/15
Current Iteration 1000, Training Loss: 1.9836, Test Loss: 2.0245
Current Iteration 1200, Training Loss: 2.0903, Test Loss: 2.4702
Current Iteration 1400, Training Loss: 2.0462, Test Loss: 1.6834
Current Iteration 1600, Training Loss: 2.0011, Test Loss: 1.7775
Current Iteration 1800, Training Loss: 2.0016, Test Loss: 2.0025
Epoch 2/15
Current Iteration 2000, Training Loss: 1.6379, Test Loss: 1.6093
Current Iteration 2200, Training Loss: 1.7474, Test Loss: 1.6699
Current Iteration 2400, Training Loss: 1.7463, Test Loss: 2.0954
Current Iteration 2600, Training Loss: 1.7792, Test Loss: 1.7126
Current Iteration 2800, Training Loss: 1.7581, Test Loss: 1.7391
Epoch 3/15
Current Iteration 3000, Training Loss: 1.6594, Test Loss: 1.4838
Current Iteration 3200, Training Loss: 1.6705, Test Loss: 1.2993
Current Iteration 3400, Training Loss: 1.6656, Test Loss: 1.5223
Current Iteration 3600, Training Loss: 1.6621, Test Loss: 2.0784
Epoch 4/15
Current Iteration 3800, Training Loss: 1.5449, Test Loss: 1.7026
Current Iteration 4000, Training Loss: 1.5462, Test Loss: 2.0287
Current Iteration 4200, Training Loss: 1.6291, Test Loss: 1.5962
Current Iteration 4400, Training Loss: 1.6104, Test Loss: 1.5663
Current Iteration 4600, Training Loss: 1.6269, Test Loss: 1.6153
Epoch 5/15
Current Iteration 4800, Training Loss: 1.5318, Test Loss: 1.9575
Current Iteration 5000, Training Loss: 1.5326, Test Loss: 1.8284
Current Iteration 5200, Training Loss: 1.5368, Test Loss: 1.4199
Current Iteration 5400, Training Loss: 1.5686, Test Loss: 1.5570
Current Iteration 5600, Training Loss: 1.5625, Test Loss: 1.3905
Epoch 6/15
Current Iteration 5800, Training Loss: 1.4508, Test Loss: 1.4073
Current Iteration 6000, Training Loss: 1.4454, Test Loss: 1.5103
Current Iteration 6200, Training Loss: 1.4581, Test Loss: 1.5290
Current Iteration 6400, Training Loss: 1.5102, Test Loss: 1.4721
Epoch 7/15
Current Iteration 6600, Training Loss: 1.6533, Test Loss: 1.8559
Current Iteration 6800, Training Loss: 1.4937, Test Loss: 1.5825
Current Iteration 7000, Training Loss: 1.4673, Test Loss: 1.5501
Current Iteration 7200, Training Loss: 1.4816, Test Loss: 1.2628
Current Iteration 7400, Training Loss: 1.5008, Test Loss: 1.6275
Epoch 8/15
Current Iteration 7600, Training Loss: 1.5901, Test Loss: 1.8171
Current Iteration 7800, Training Loss: 1.5639, Test Loss: 1.5695
Current Iteration 8000, Training Loss: 1.5239, Test Loss: 1.4265
Current Iteration 8200, Training Loss: 1.5174, Test Loss: 1.6004
Current Iteration 8400, Training Loss: 1.5145, Test Loss: 1.7507
Epoch 9/15
Current Iteration 8600, Training Loss: 1.4719, Test Loss: 2.5488
Current Iteration 8800, Training Loss: 1.4801, Test Loss: 1.5729
Current Iteration 9000, Training Loss: 1.5142, Test Loss: 1.7528
Current Iteration 9200, Training Loss: 1.5336, Test Loss: 1.3346
Epoch 10/15
Current Iteration 9400, Training Loss: 1.2453, Test Loss: 1.5852
Current Iteration 9600, Training Loss: 1.4172, Test Loss: 1.4780
Current Iteration 9800, Training Loss: 1.4777, Test Loss: 1.5022
Current Iteration 10000, Training Loss: 1.5813, Test Loss: 1.7233
Current Iteration 10200, Training Loss: 1.5919, Test Loss: 1.9513
Epoch 11/15
Current Iteration 10400, Training Loss: 1.9416, Test Loss: 1.7701
Current Iteration 10600, Training Loss: 1.7081, Test Loss: 1.3875
Current Iteration 10800, Training Loss: 1.6206, Test Loss: 1.3455
Current Iteration 11000, Training Loss: 1.6252, Test Loss: 1.5040
Current Iteration 11200, Training Loss: 1.5928, Test Loss: 1.8514
Epoch 12/15
Current Iteration 11400, Training Loss: 1.6690, Test Loss: 1.5689
Current Iteration 11600, Training Loss: 1.6206, Test Loss: 1.7379
Current Iteration 11800, Training Loss: 1.6134, Test Loss: 1.5243
Current Iteration 12000, Training Loss: 1.5783, Test Loss: 1.8225
Epoch 13/15
Current Iteration 12200, Training Loss: 1.2799, Test Loss: 1.7457
Current Iteration 12400, Training Loss: 1.4925, Test Loss: 1.7505
Current Iteration 12600, Training Loss: 1.5474, Test Loss: 1.4273
Current Iteration 12800, Training Loss: 1.5275, Test Loss: 1.5176
Current Iteration 13000, Training Loss: 1.5649, Test Loss: 1.9603
Epoch 14/15
Current Iteration 13200, Training Loss: 1.6928, Test Loss: 1.7866
Current Iteration 13400, Training Loss: 1.7188, Test Loss: 1.4679
Current Iteration 13600, Training Loss: 1.5966, Test Loss: 1.4243
Current Iteration 13800, Training Loss: 1.5606, Test Loss: 1.4218
Current Iteration 14000, Training Loss: 1.5678, Test Loss: 1.4803
Epoch 15/15
```



```python
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=(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: 92.18%
```



```
final_test(model, train_loader)

Final Test Accuracy: 92.35%
```



Confusion Matrix

	0	1	2	3	4	5	6	7	8	9
0	5812	0	21	2	9	35	15	1	24	4
1	3	6377	74	64	7	75	11	3	104	24
2	51	4	5747	28	28	12	18	5	58	7
3	16	5	167	5617	0	165	9	9	127	16
4	33	10	32	2	4855	12	70	3	69	756
5	63	0	25	41	16	5155	55	0	55	11
6	74	3	20	0	7	57	5745	0	10	2
7	64	17	291	278	48	55	10	5065	90	347
8	18	12	61	60	5	94	51	0	5535	15
9	58	5	10	70	42	127	2	23	112	5500