Artificial Intelligence Lab Work (4)

レポート解答用紙（Report Answer Sheet）

学生証番号（Student ID): 20521150

名前(Name): Pham Quoc Cuong

問題 1.

（プログラム）

Q1. Implement the MNIST learning and inference program by following the 10th lecture's slides (copy the program on the slide), and submit the program (.py) and the execution results, i.e., loss at each epoch during training and accuracy against test data, displayed on the console in a word file.

```
[55] import numpy as np
     import matplotlib.pyplot as plt
     import torch
     import torch.nn.functional as F
     import torchvision as tv
     import seaborn as sns
     from sklearn.metrics import confusion_matrix

     train_dataset=tv.datasets.MNIST(root="./",train=True,transform=tv.transforms.ToTensor(),download=True)
     test_dataset=tv.datasets.MNIST(root="./",train=False,transform=tv.transforms.ToTensor(),download=True)

     train_loader=torch.utils.data.DataLoader(dataset=train_dataset,batch_size=100,shuffle=True)
     test_loader=torch.utils.data.DataLoader(dataset=test_dataset,batch_size=100,shuffle=False)
```

```
[56] MODELNAME = 'mnist.model'
     EPOCH = 10
     DEVICE = 'cuda' if torch.cuda.is_available() else 'cpu'
```

```
[57] class MNIST(torch.nn.Module):
         losses = []

         def __init__(self):
             super(MNIST,self).__init__()
             self.l1 = torch.nn.Linear(784,300)
             self.l2 = torch.nn.Linear(300,300)
             self.l3 = torch.nn.Linear(300,10)

         def forward(self,x):
             h = F.relu(self.l1(x))
             h = F.relu(self.l2(h))
             y = self.l3(h)
             return y
```

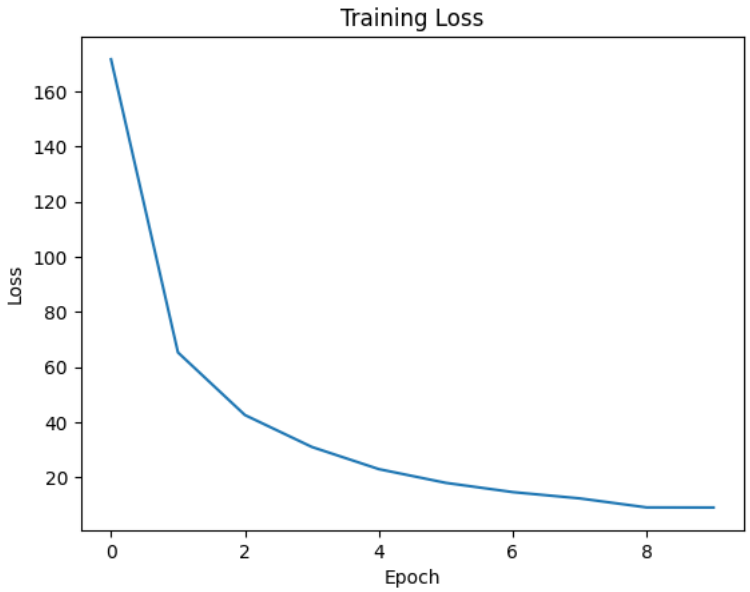
```
[58] def train():
         model = MNIST().to(DEVICE)
         optimizer = torch.optim.Adam(model.parameters())
         for epoch in range(EPOCH):
             loss = 0
             for images, labels in train_loader:
                 images = images.view(-1,28*28).to(DEVICE)
                 labels = labels.to(DEVICE)
                 optimizer.zero_grad()
                 y = model(images)
                 batchloss = F.cross_entropy(y, labels)
                 batchloss.backward()
                 optimizer.step()
                 loss = loss + batchloss.item()
             print('epoch: ', epoch,', loss: ',loss)
             model.losses.append(loss)
         torch.save(model.state_dict(), MODELNAME)

         plt.plot(model.losses)
         plt.xlabel('Epoch')
         plt.ylabel('Loss')
         plt.title('Training Loss')
         plt.show()
```

[60]
```python
def test():
    total = len(test_loader.dataset)
    correct = 0
    model = MNIST().to(DEVICE)
    model.load_state_dict(torch.load(MODELNAME))
    model.eval()

    true_labels = []
    pred_labels = []

    for images, labels in test_loader:
        images = images.view(-1, 28 * 28).to(DEVICE)
        y = model(images)
        labels = labels.to(DEVICE)
        pred = y.max(dim=1)[1]

        true_labels.extend(labels.cpu().numpy())
        pred_labels.extend(pred.cpu().numpy())

        correct += (pred == labels).sum()

    print('Correct: ', correct.item())
    print('Total: ', total)
    print('Accuracy: ', correct.item() / float(total))

    # Generate confusion matrix
    cm = confusion_matrix(true_labels, pred_labels)

    # Plot the confusion matrix as an image
    plt.figure(figsize=(10, 8))
    sns.heatmap(cm, annot=True, fmt="d", cmap="Blues")
    plt.xlabel('Predicted Labels')
    plt.ylabel('True Labels')
    plt.title('Confusion Matrix')
    plt.show()
```

（実行結果）

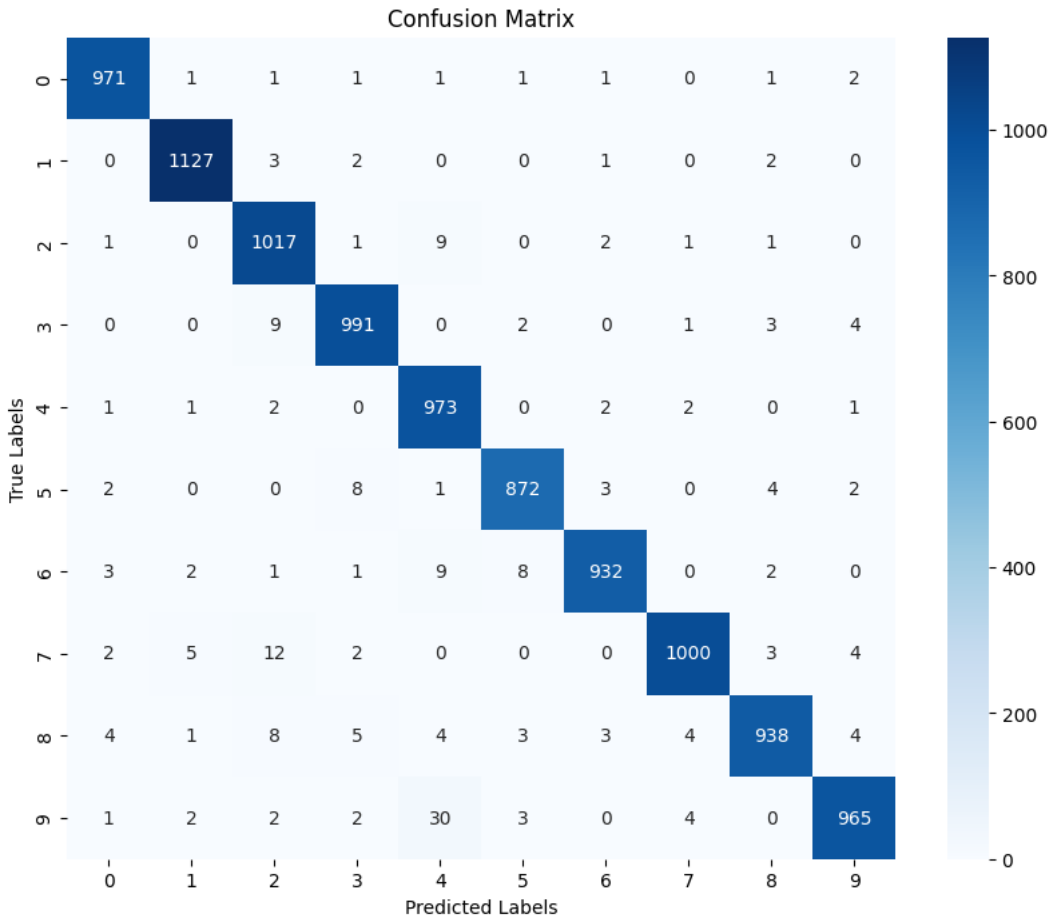
```
[59] train()
```

```
epoch:  0 , loss:  171.65008279308677
epoch:  1 , loss:  65.32018205523491
epoch:  2 , loss:  42.61820778762922
epoch:  3 , loss:  31.041309498250484
epoch:  4 , loss:  23.00104690855369
epoch:  5 , loss:  18.023108201567084
epoch:  6 , loss:  14.666266271553468
epoch:  7 , loss:  12.384661977353971
epoch:  8 , loss:  9.112057573875063
epoch:  9 , loss:  9.07755283745064
```




```
[61] test()
```

```
Correct:  9786
Total:  10000
Accuracy:  0.9786
```


Confusion Matrix
True Labels
Predicted Labels
0 971 1 1 1 1 1 1 0 1 2
1 0 1127 3 2 0 0 1 0 2 0
2 1 0 1017 1 9 0 2 1 1 0
3 0 0 9 991 0 2 0 1 3 4
4 1 1 2 0 973 0 2 2 0 1
5 2 0 0 8 1 872 3 0 4 2
6 3 2 1 1 9 8 932 0 2 0
7 2 5 12 2 0 0 0 1000 3 4
8 4 1 8 5 4 3 3 4 938 4
9 1 2 2 2 30 3 0 4 0 965
0 1 2 3 4 5 6 7 8 9
1000
800
600
400
200
0

問題 2

（プログラム）

Q2. Rewrite the program you wrote in Q1 to train and infer on the image recognition dataset CIFAR-10, and submit the program (.py) and the execution results, i.e., loss at each epoch during training and accuracy against test data, displayed on the console in a word file. CIFAR-10 is a 10-class image classification data, and can be downloaded by the following program.

```
[62] train_dataset = tv.datasets.CIFAR10(root="./", train=True, transform=tv.transforms.ToTensor(), download=True)
     test_dataset = tv.datasets.CIFAR10(root="./", train=False, transform=tv.transforms.ToTensor(), download=True)

     train_loader=torch.utils.data.DataLoader(dataset=train_dataset,batch_size=100,shuffle=True)
     test_loader=torch.utils.data.DataLoader(dataset=test_dataset,batch_size=100,shuffle=False)
```

```
     Files already downloaded and verified
     Files already downloaded and verified
```

```
[63] MODELNAME = 'cifar-10.model'
     EPOCH = 10
     DEVICE = 'cuda' if torch.cuda.is_available() else 'cpu'
```

```
[64] class CIFAR(torch.nn.Module):
         losses = [] # List to store the loss values

         def __init__(self):
             super(CIFAR,self).__init__()
             self.l1 = torch.nn.Linear(32*32*3,300)
             self.l2 = torch.nn.Linear(300,300)
             self.l3 = torch.nn.Linear(300,10)

         def forward(self,x):
             h = F.relu(self.l1(x))
             h = F.relu(self.l2(h))
             y = self.l3(h)
             return y
```

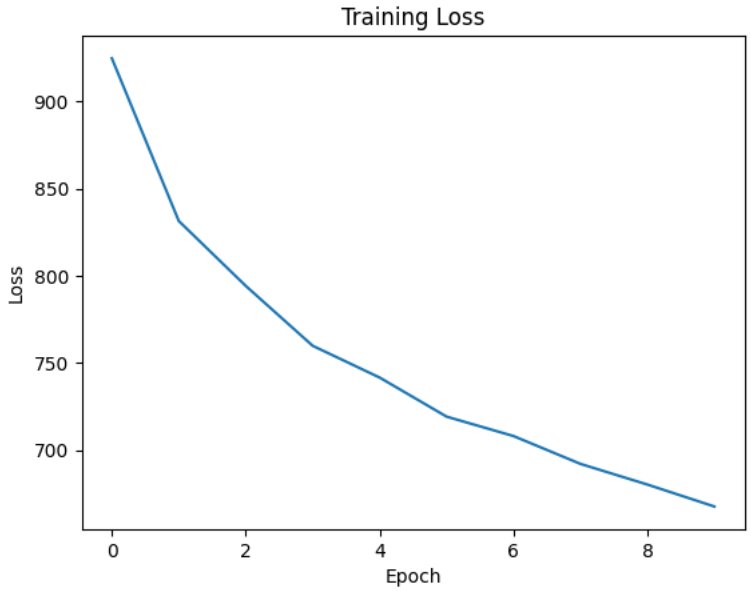
```
[65] def train():
         model = CIFAR().to(DEVICE)
         optimizer = torch.optim.Adam(model.parameters())
         for epoch in range(EPOCH):
             loss =0
             for images, labels in train_loader:
                 images = images.view(-1,32*32*3).to(DEVICE)
                 labels = labels.to(DEVICE)
                 optimizer.zero_grad()
                 y = model(images)
                 batchloss = F.cross_entropy(y, labels)
                 batchloss.backward()
                 optimizer.step()
                 loss = loss + batchloss.item()
             print('epoch: ', epoch,', loss: ',loss)
             model.losses.append(loss)
         torch.save(model.state_dict(), MODELNAME)
         # Plot the losses result
         plt.plot(model.losses)
         plt.xlabel('Epoch')
         plt.ylabel('Loss')
         plt.title('Training Loss')
         plt.show()
```

```
[67] def test():
         total = len(test_loader.dataset)
         correct = 0
         model = CIFAR().to(DEVICE)
         model.load_state_dict(torch.load(MODELNAME))
         model.eval()

         true_labels = []
         pred_labels = []

         for images, labels in test_loader:
             images = images.view(-1, 32 * 32 * 3).to(DEVICE)
             y = model(images)
             labels = labels.to(DEVICE)
             pred = y.max(dim=1)[1]

             true_labels.extend(labels.cpu().numpy())
             pred_labels.extend(pred.cpu().numpy())

             correct += (pred == labels).sum()

         print('Correct: ', correct.item())
         print('Total: ', total)
         print('Accuracy: ', correct.item() / float(total))

         # Generate confusion matrix
         cm = confusion_matrix(true_labels, pred_labels)

         # Plot the confusion matrix as an image
         plt.figure(figsize=(10, 8))
         sns.heatmap(cm, annot=True, fmt="d", cmap="Blues")
         plt.xlabel('Predicted Labels')
         plt.ylabel('True Labels')
         plt.title('Confusion Matrix')
         plt.show()
```

(実行結果)

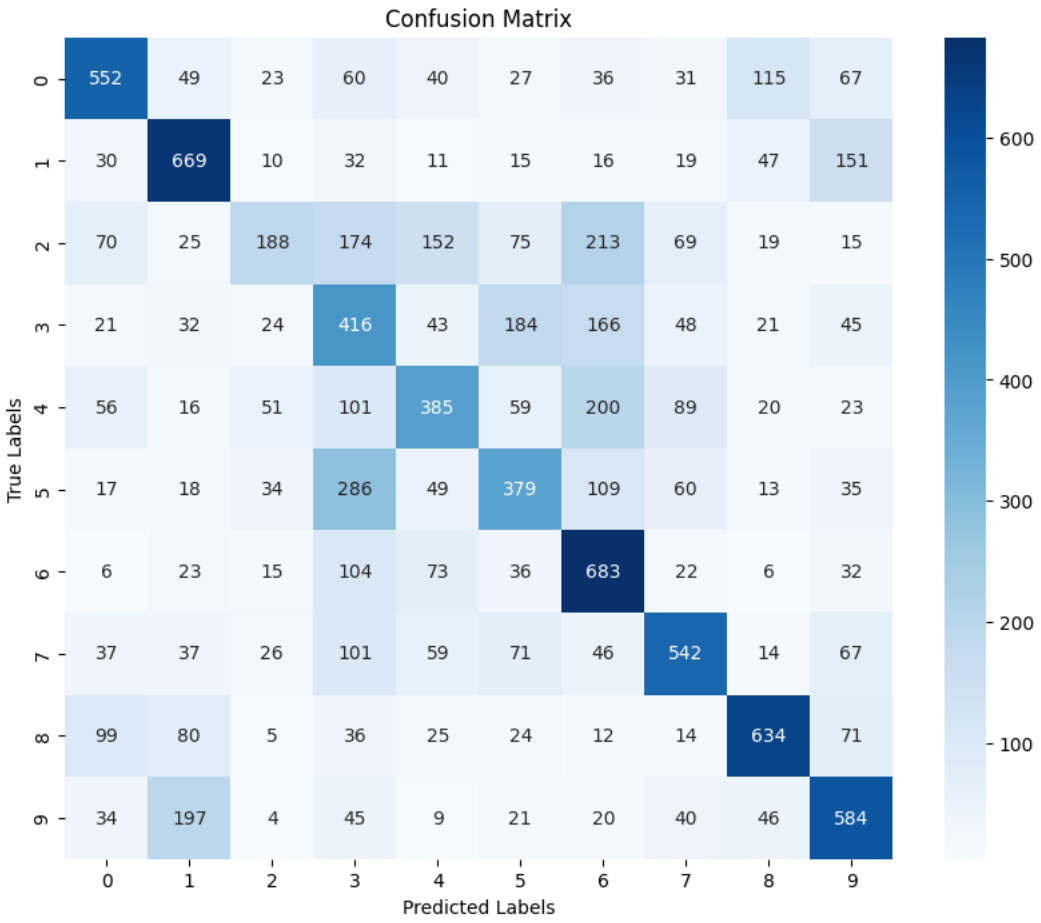
```
[66] train()
```

```
epoch:  0 , loss:  924.9251463413239
epoch:  1 , loss:  831.4743949174881
epoch:  2 , loss:  794.2123394012451
epoch:  3 , loss:  759.8124611377716
epoch:  4 , loss:  741.5656607151031
epoch:  5 , loss:  719.1269598007202
epoch:  6 , loss:  707.9702979326248
epoch:  7 , loss:  691.9842377901077
epoch:  8 , loss:  680.1186082363129
epoch:  9 , loss:  667.5125402212143
```




```
[68] test()
```

```
Correct:  5032
Total:  10000
Accuracy:  0.5032
```


Confusion Matrix
True Labels
Predicted Labels
0 552 49 23 60 40 27 36 31 115 67
1 30 669 10 32 11 15 16 19 47 151
2 70 25 188 174 152 75 213 69 19 15
3 21 32 24 416 43 184 166 48 21 45
4 56 16 51 101 385 59 200 89 20 23
5 17 18 34 286 49 379 109 60 13 35
6 6 23 15 104 73 36 683 22 6 32
7 37 37 26 101 59 71 46 542 14 67
8 99 80 5 36 25 24 12 14 634 71
9 34 197 4 45 9 21 20 40 46 584
0 1 2 3 4 5 6 7 8 9
600
500
400
300
200
100

問題 3

（プログラム）

Q3. Rewrite the program you wrote in Q2 to create an NN with one intermediate layer as a convolutional layer, and submit the program (.py) and the execution results, i.e., loss at each epoch during training and accuracy against test data, displayed on the console in a word file. The convolutional layer can be obtained by the following program

nn.Conv2d(in_channel, out_channel, filtersize)

where in_channel is the number of input channels, out_channel is the number of output channels, and filtersize is the size of the filter.

```
[69] MODELNAME = 'cifar-10-conv2d.model'
     EPOCH = 10
     DEVICE = 'cuda' if torch.cuda.is_available() else 'cpu'
```

```
[70] class CIFAR_Conv2D(torch.nn.Module):
         losses = [] # List to store the loss values

         def __init__(self):
             super(CIFAR_Conv2D,self).__init__()
             self.l1 = torch.nn.Conv2d(3,100,4)
             self.l2 = torch.nn.Linear(29*29*100,300)
             self.l3 = torch.nn.Linear(300,10)

         def forward(self,x):
             h = F.relu(self.l1(x))
             h = torch.flatten(h, start_dim=1)
             h = F.relu(self.l2(h))
             y = self.l3(h)
             return y
```

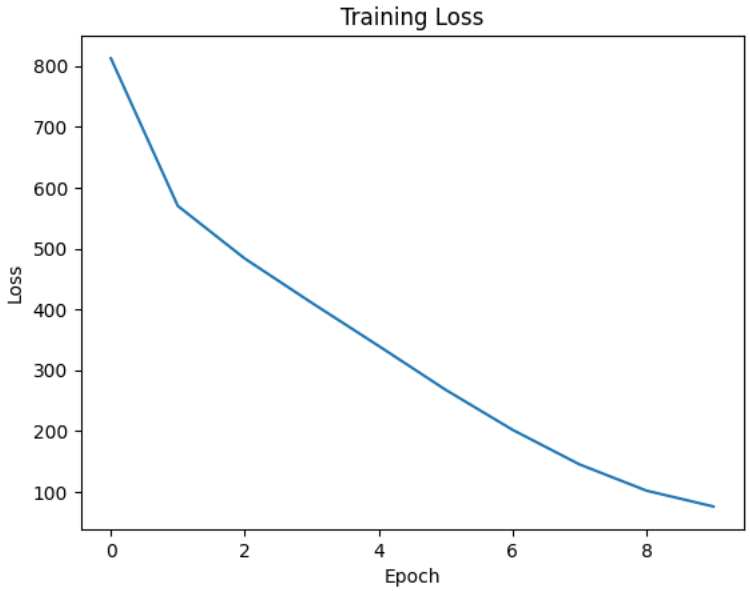
```
[71] def train():
         model = CIFAR_Conv2D().to(DEVICE)
         optimizer = torch.optim.Adam(model.parameters())

         for epoch in range(EPOCH):
             loss = 0
             for images, labels in train_loader:
                 images = images.view(-1,3,32,32).to(DEVICE)
                 labels = labels.to(DEVICE)
                 optimizer.zero_grad()
                 y = model(images)
                 batchloss = F.cross_entropy(y, labels)
                 batchloss.backward()
                 optimizer.step()
                 loss = loss + batchloss.item()

             print('epoch: ', epoch,', loss: ',loss)
             model.losses.append(loss)  # Append the loss to the list

         torch.save(model.state_dict(), MODELNAME)

         plt.plot(model.losses)
         plt.xlabel('Epoch')
         plt.ylabel('Loss')
         plt.title('Training Loss')
         plt.show()
```

```
[73] def test():
         total = len(test_loader.dataset)
         correct = 0
         model = CIFAR_Conv2D().to(DEVICE)
         model.load_state_dict(torch.load(MODELNAME))
         model.eval()

         true_labels = []
         pred_labels = []

         for images, labels in test_loader:
             images = images.view(-1, 3, 32, 32).to(DEVICE)
             y = model(images)
             labels = labels.to(DEVICE)
             pred = y.max(dim=1)[1]

             true_labels.extend(labels.cpu().numpy())
             pred_labels.extend(pred.cpu().numpy())

             correct += (pred == labels).sum()

         print('Correct: ', correct.item())
         print('Total: ', total)
         print('Accuracy: ', correct.item() / float(total))

         # Generate confusion matrix
         cm = confusion_matrix(true_labels, pred_labels, labels=range(10))

         # Plot the confusion matrix as an image
         plt.figure(figsize=(10, 8))
         sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=range(10), yticklabels=range(10))
         plt.xlabel('Predicted Labels')
         plt.ylabel('True Labels')
         plt.title('Confusion Matrix')
         plt.show()
```

（実行結果）

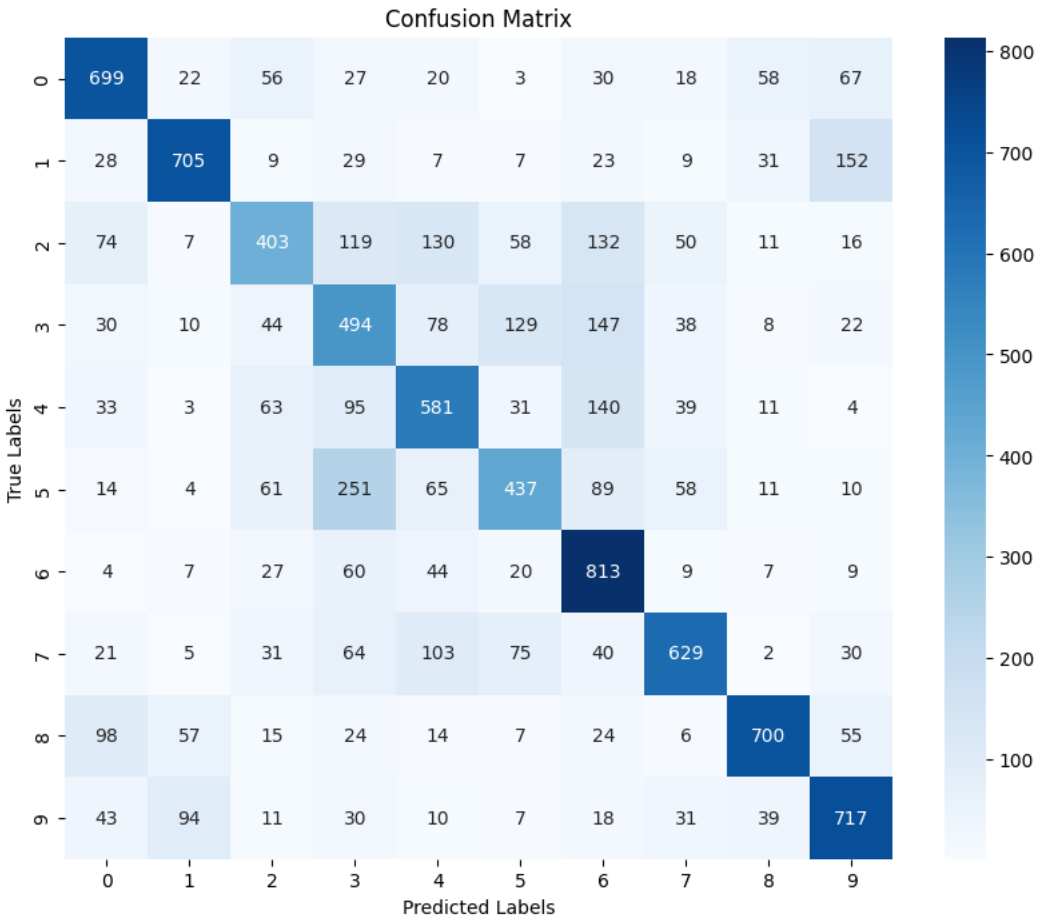
```
[72] train()
```

```
epoch:  0 , loss:  812.6806900501251
epoch:  1 , loss:  570.0182233452797
epoch:  2 , loss:  483.50918436050415
epoch:  3 , loss:  410.78025180101395
epoch:  4 , loss:  339.84813117980957
epoch:  5 , loss:  267.78175631165504
epoch:  6 , loss:  201.96683129668236
epoch:  7 , loss:  145.0935261696577
epoch:  8 , loss:  101.98463297635317
epoch:  9 , loss:  75.8632026873529
```




```
[74] test()
```

```
Correct:  6178
Total:  10000
Accuracy:  0.6178
```


Confusion Matrix
True Labels
Predicted Labels
0 699 22 56 27 20 3 30 18 58 67
1 28 705 9 29 7 7 23 9 31 152
2 74 7 403 119 130 58 132 50 11 16
3 30 10 44 494 78 129 147 38 8 22
4 33 3 63 95 581 31 140 39 11 4
5 14 4 61 251 65 437 89 58 11 10
6 4 7 27 60 44 20 813 9 7 9
7 21 5 31 64 103 75 40 629 2 30
8 98 57 15 24 14 7 24 6 700 55
9 43 94 11 30 10 7 18 31 39 717
0 1 2 3 4 5 6 7 8 9
800 700 600 500 400 300 200 100