# Class5_NN

February 4, 2025

[1]:
```
import numpy as np

import torch
from torch import nn
from torch import optim
from torch.utils.data import Dataset, DataLoader
from torchvision import datasets, transforms
```

[2]:
```
device = 'cuda' if torch.cuda.is_available() else 'cpu'
print(device)
```

```
cuda
```

### 0.0.1 Dataset preparation

- Import Dataset and Get Properties

[12]:
```
training_data = datasets.FashionMNIST(root="./dataset", train=True,␣
  ↪download=False, transform=transforms.ToTensor())
testing_data = datasets.FashionMNIST(root="./dataset", train=False,␣
  ↪download=False, transform=transforms.ToTensor())
print(training_data, "\n############################\n", testing_data)
```

```
Dataset FashionMNIST
    Number of datapoints: 60000
    Root location: ./dataset
    Split: Train
    StandardTransform
Transform: ToTensor()
############################
 Dataset FashionMNIST
    Number of datapoints: 10000
    Root location: ./dataset
    Split: Test
    StandardTransform
Transform: ToTensor()
```

- Inspect dataset

[14]:

```
print("training:", training_data.data.size())
print("testing:", testing_data.data.size())
```

```
training: torch.Size([60000, 28, 28])
testing: torch.Size([10000, 28, 28])
```

[17]:

```
training_data.targets
```

[17]:

```
tensor([9, 0, 0,  …, 3, 0, 5])
```

### 0.0.2 Pipeline setup

[69]:

```
# Must Preprocess first
class ClassificationDataset(Dataset):
    def __init__(self, x, y):
        self.features = x.to(torch.float)
        self.target = y.to(torch.float)

    def __len__(self):
        return len(self.features)

    def __getitem__(self, idx):
        return self.features[idx].to(device), self.target[idx].to(device)

train_dataset = ClassificationDataset(training_data.data, training_data.targets)
test_dataset = ClassificationDataset(testing_data.data, testing_data.targets)

train_loader = DataLoader(train_dataset, batch_size=256)
test_loader = DataLoader(test_dataset, batch_size=256)
```

### 0.0.3 Model Setup

- Define Model

[64]:

```
class ClassificationNN(nn.Module):
    def __init__(self, out_dim, dtype=torch.float32):
        super().__init__()
        self.extract_layer = nn.Conv2d(1, 16, 1)
        self.bn1 = nn.BatchNorm2d(16)
        self.hidden_layer = nn.Conv2d(16, 16, 3)
        self.bn2 = nn.BatchNorm2d(16)
        self.expose_layer = nn.Conv2d(16, 1, 1)
        self.bn3 = nn.BatchNorm2d(1)

        self.linear_layer = nn.Linear(1* 26* 26, out_dim)

    def forward(self, x):
        x = self.bn1(self.extract_layer(x))
        x = self.bn2(self.hidden_layer(x))
        x = self.bn3(self.expose_layer(x))

        # Logit
        y = self.linear_layer(torch.flatten(x, start_dim=1, end_dim=-1))

        return y

    def prediction(self, logit):
        with torch.no_grad():
            prob = nn.functional.softmax(logit, dim=1)
            outputs = torch.argmax(prob, dim=1)
            return outputs
```

[70]:

```
torch.manual_seed(42)

model = ClassificationNN(len(training_data.classes)).to(device)

loss_fn = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), 1e-3)
```

### 0.0.4 Training Loop

[71]:

```
model.train()
for i in range(1, 101):
    last_iter_loss = 0
    for batch in train_loader:
        input_batch, target_batch = batch

        optimizer.zero_grad()

        outputs = model(input_batch.unsqueeze(1))

        loss = loss_fn(outputs, target_batch.to(torch.long))
        loss.backward()

        optimizer.step()

        last_iter_loss = loss

    if i%20 == 0:
        print(f"loss at iter {i} = {last_iter_loss}")
```

```
loss at iter 20 = 0.30945494771003723
loss at iter 40 = 0.2895567715167999
loss at iter 60 = 0.28571146726608276
loss at iter 80 = 0.2837536931037903
loss at iter 100 = 0.28216496109962463
```

### 0.0.5 Eval Model

[76]:

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

[77]:

```
model.eval()

total_loss = 0

y_true = []
y_pred = []

for i, batch in enumerate(test_loader):
    input_batch, target_batch = batch

    with torch.no_grad():
        outputs = model(input_batch.unsqueeze(1))
        loss = loss_fn(outputs, target_batch.to(torch.long))
        total_loss += loss

        y_true.extend(target_batch.cpu().numpy())
        y_pred.extend(model.prediction(outputs).detach().cpu().numpy())

print(f"loss of testset = {total_loss/len(test_loader)}")
print(f"accuracy = {accuracy_score(y_true, y_pred)}")
```

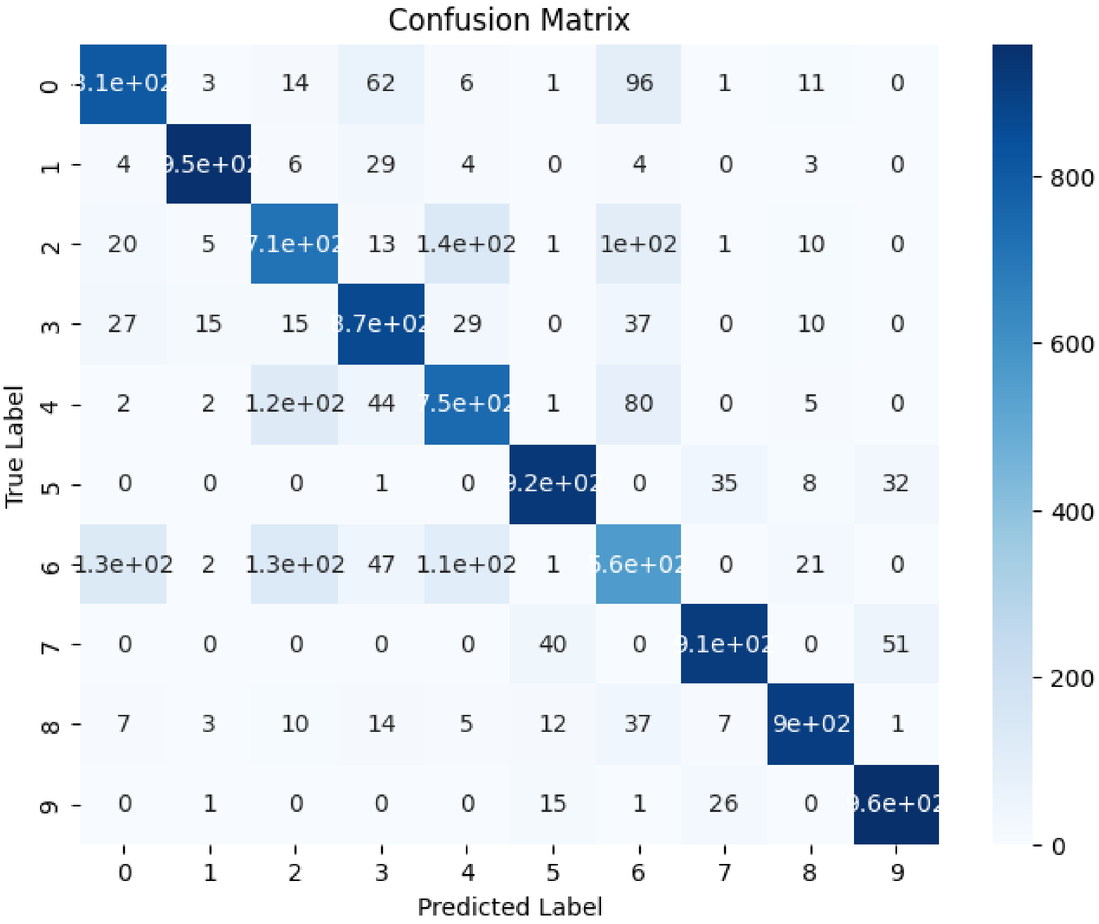
```
loss of testset = 0.4844067692756653
accuracy = 0.8335
```

[79]:

```
import matplotlib.pyplot as plt
```

[82]:

```
conf_matrix = confusion_matrix(y_true, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, xticklabels=range(10),␣
 ↪yticklabels=range(10), cmap="Blues")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.title("Confusion Matrix")
plt.show()
```



|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 8.1e+02 | 3 | 14 | 62 | 6 | 1 | 96 | 1 | 11 | 0 |
| 1 | 4 | 9.5e+02 | 6 | 29 | 4 | 0 | 4 | 0 | 3 | 0 |
| 2 | 20 | 5 | 7.1e+02 | 13 | 1.4e+02 | 1 | 1e+02 | 1 | 10 | 0 |
| 3 | 27 | 15 | 15 | 8.7e+02 | 29 | 0 | 37 | 0 | 10 | 0 |
| 4 | 2 | 2 | 1.2e+02 | 44 | 7.5e+02 | 1 | 80 | 0 | 5 | 0 |
| 5 | 0 | 0 | 0 | 1 | 0 | 9.2e+02 | 0 | 35 | 8 | 32 |
| 6 | 1.3e+02 | 2 | 1.3e+02 | 47 | 1.1e+02 | 1 | 5.6e+02 | 0 | 21 | 0 |
| 7 | 0 | 0 | 0 | 0 | 0 | 40 | 0 | 9.1e+02 | 0 | 51 |
| 8 | 7 | 3 | 10 | 14 | 5 | 12 | 37 | 7 | 9e+02 | 1 |
| 9 | 0 | 1 | 0 | 0 | 0 | 15 | 1 | 26 | 0 | 9.6e+02 |

```
[ ]:
```