14-8-25

Page
Date

LAB - 4

Aim: To Build and train a Simple Feed Forward Network (FFNN) on the MNIST dataset.

Objective:

1. Load and Preprocess the MNIST dataset.
2. Design a feed-forward neural network with input, hidden and output layer.
3. Train the model on handwritten digits.
4. Evaluate the model using accuracy and loss metrics.

Pseudo code:

```
1. Import libraries.
2. Load MNIST dataset.
3. Normalize pixel values.
4. Define feed forward neural network.
5. Compile model.
6. Train model on training data.
7. Evaluate model on test data.
8. Print accuracy and loss.
```

Observation:

- The network quickly learned digit patterns from MNIST Images.
- Training accuracy steadily increased, and test accuracy

remained high, showing good generalization

Result:

- The feed forward neural network achieved about 97-98% accuracy on MNIST test set.

- The exp demonstrates that even a simple FFNN can effectively recognize handwritten digits.

Successfully implemented FFNN on MNIST dataset.

[illegible] 22/1/26

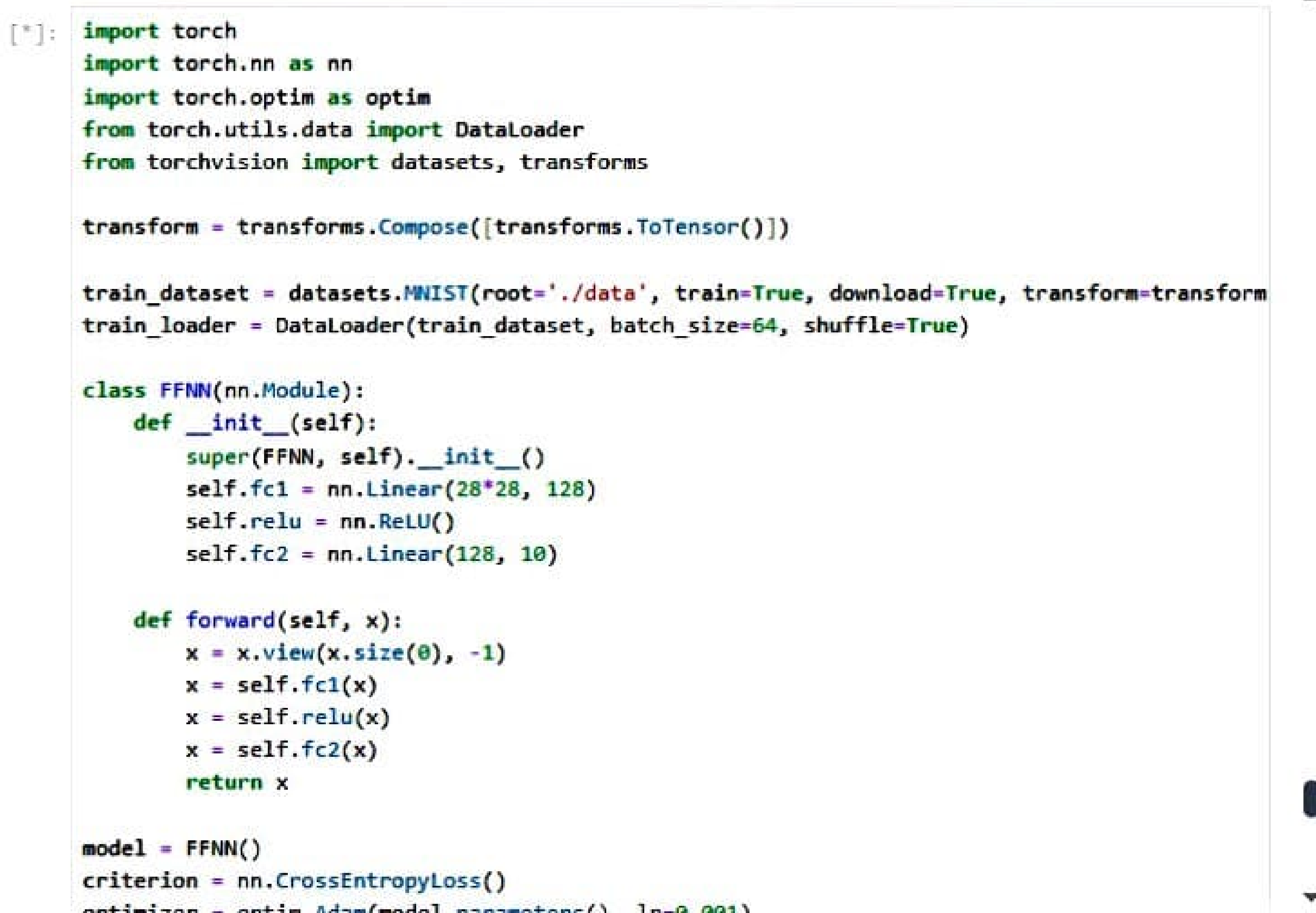
Observation:

Metric	Value
Accuracy	97.82

↓

This shows that:

- Model successfully learned patterns of handwritten characters from the dataset.
- Normalization of pixel values helped improve training performance.
- Increasing no of layers, neurons or training epochs may improve accuracy further.



[*]:

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

transform = transforms.Compose([transforms.ToTensor()])

train_dataset = datasets.MNIST(root='./data', train=True, download=True, transform=transform
train_loader = DataLoader(train_dataset, batch_size=64, shuffle=True)

class FFNN(nn.Module):
    def __init__(self):
        super(FFNN, self).__init__()
        self.fc1 = nn.Linear(28*28, 128)
        self.relu = nn.ReLU()
        self.fc2 = nn.Linear(128, 10)

    def forward(self, x):
        x = x.view(x.size(0), -1)
        x = self.fc1(x)
        x = self.relu(x)
        x = self.fc2(x)
        return x

model = FFNN()
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)
```

```python
        x = self.relu(x)
        x = self.fc2(x)
        return x

model = FFNN()
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)
epochs = 5

for epoch in range(epochs):
    model.train()
    running_loss = 0.0
    for images, labels in train_loader:
        optimizer.zero_grad()
        outputs = model(images)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        running_loss += loss.item()
    print(f"Epoch {epoch+1}, Loss: {running_loss/len(train_loader)}")
```