Pratham Nagar - 64 Experiment 06 ~ Design and implement a CNN model for digit recognition application.

```
import torch
import torch.nn as nn
import torch.optim as optim
import torchvision
import torchvision.transforms as transforms
from torch.utils.data import DataLoader
from sklearn.metrics import accuracy_score

# Transform: Convert to tensor and normalize to [0, 1]
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5,), (0.5,))
])

# Download and load Fashion-MNIST dataset
train_dataset = torchvision.datasets.FashionMNIST(root='./data', train=True, download=True, transform=transform)
test_dataset = torchvision.datasets.FashionMNIST(root='./data', train=False, download=True, transform=transform)

train_loader = DataLoader(train_dataset, batch_size=64, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=64, shuffle=False)

class FashionCNN(nn.Module):
    def __init__(self):
        super(FashionCNN, self).__init__()

        # Convolutional Layers
        self.conv1 = nn.Conv2d(in_channels=1, out_channels=32, kernel_size=3, padding=1)  # Output: 32x28x28
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2)  # Halves dimensions
        self.conv2 = nn.Conv2d(32, 64, 3, padding=1)       # Output: 64x14x14

        # Fully Connected Layers
        self.fc1 = nn.Linear(64 * 7 * 7, 128)  # After pooling twice
        self.fc2 = nn.Linear(128, 10)  # 10 classes for fashion items

        # Activation
        self.relu = nn.ReLU()
        self.dropout = nn.Dropout(0.3)

    def forward(self, x):
        x = self.pool(self.relu(self.conv1(x)))  # Conv1 -> ReLU -> Pool
        x = self.pool(self.relu(self.conv2(x)))  # Conv2 -> ReLU -> Pool
        x = x.view(-1, 64 * 7 * 7)               # Flatten
        x = self.dropout(self.relu(self.fc1(x))) # FC1 -> ReLU -> Dropout
        x = self.fc2(x)                          # Output layer
        return x

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model = FashionCNN().to(device)

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:
        images, labels = images.to(device), labels.to(device)

        optimizer.zero_grad()
        outputs = model(images)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        running_loss += loss.item()

    print(f"Epoch [{epoch+1}/{epochs}], Loss: {running_loss/len(train_loader):.4f}")

model.eval()
all_preds = []
all_labels = []
```

```
with torch.no_grad():
    for images, labels in test_loader:
        images, labels = images.to(device), labels.to(device)
        outputs = model(images)
        _, predicted = torch.max(outputs, 1)

        all_preds.extend(predicted.cpu().numpy())
        all_labels.extend(labels.cpu().numpy())

acc = accuracy_score(all_labels, all_preds)
print(f"Test Accuracy: {acc * 100:.2f}%")
```

```
100%|██████████| 26.4M/26.4M [00:01<00:00, 16.1MB/s]
100%|██████████| 29.5k/29.5k [00:00<00:00, 271kB/s]
100%|██████████| 4.42M/4.42M [00:00<00:00, 5.09MB/s]
100%|██████████| 5.15k/5.15k [00:00<00:00, 21.3MB/s]
Epoch [1/5], Loss: 0.4990
Epoch [2/5], Loss: 0.3244
Epoch [3/5], Loss: 0.2755
Epoch [4/5], Loss: 0.2439
Epoch [5/5], Loss: 0.2227
Test Accuracy: 90.93%
```