```
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
import matplotlib.pyplot as plt
import numpy as np
from sklearn.metrics import confusion_matrix, classification_report
import seaborn as sns
```

```
transform = transforms.Compose([
    transforms.ToTensor(),          # Convert images to tensors
    transforms.Normalize((0.5,), (0.5,))  # Normalize images
])
```

```
train_dataset = torchvision.datasets.FashionMNIST(root="./data", train=True, transform=transform, download=True)
test_dataset = torchvision.datasets.FashionMNIST(root="./data", train=False, transform=transform, download=True)
```

```
100%|██████████| 26.4M/26.4M [00:01<00:00, 15.1MB/s]
100%|██████████| 29.5k/29.5k [00:00<00:00, 272kB/s]
100%|██████████| 4.42M/4.42M [00:00<00:00, 5.05MB/s]
100%|██████████| 5.15k/5.15k [00:00<00:00, 6.43MB/s]
```

```
image, label = train_dataset[0]
print(image.shape)
print(len(train_dataset))
```

```
torch.Size([1, 28, 28])
60000
```

```
image, label = test_dataset[0]
print(image.shape)
print(len(test_dataset))
```

```
torch.Size([1, 28, 28])
10000
```

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

```
class CNNClassifier(nn.Module):
    def __init__(self):
        super(CNNClassifier, self).__init__()
        # write your code here
        self.conv1 = nn.Conv2d(in_channels=1, out_channels=16, kernel_size=3, padding=1)
        self.conv2 = nn.Conv2d(in_channels=16, out_channels=32, kernel_size=3, padding=1)
        self.conv3 = nn.Conv2d(in_channels=32, out_channels=64, kernel_size=3, padding=1)
        self.conv4 = nn.Conv2d(in_channels=64, out_channels=128, kernel_size=3, padding=1)
        self.pool = nn.AvgPool2d(kernel_size=2, stride=2)
        self.fc1 = nn.Linear(128*3*3, 256)
        self.fc2 = nn.Linear(256,128)
        self.fc3 = nn.Linear(128,64)
        self.fc4 = nn.Linear(64,10)

    def forward(self, x):
        # write your code here
        x = self.pool(torch.relu(self.conv1(x)))
        x = self.pool(torch.relu(self.conv2(x)))
        x = self.pool(torch.relu(self.conv3(x)))
        x = torch.relu(self.conv4(x)) # Removed the last pooling layer
        x = x.view(x.size(0), -1)
        x = torch.relu(self.fc1(x))
        x = torch.relu(self.fc2(x))
        x = torch.relu(self.fc3(x))
        x = self.fc4(x)
        return x
```

```
from torchsummary import summary

# Initialize model
model = CNNClassifier()

# Move model to GPU if available
if torch.cuda.is_available():
```

```
    device = torch.device("cuda")
    model.to(device)

# Print model summary
print('Name: A.Lahari')
print('Register Number: 212223230111')
summary(model, input_size=(1, 28, 28))
```

```
Name: A.Lahari
Register Number: 212223230111
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1          [-1, 16, 28, 28]             160
         AvgPool2d-2          [-1, 16, 14, 14]               0
            Conv2d-3          [-1, 32, 14, 14]           4,640
         AvgPool2d-4            [-1, 32, 7, 7]               0
            Conv2d-5            [-1, 64, 7, 7]          18,496
         AvgPool2d-6            [-1, 64, 3, 3]               0
            Conv2d-7           [-1, 128, 3, 3]          73,856
            Linear-8                 [-1, 256]         295,168
            Linear-9                 [-1, 128]          32,896
           Linear-10                  [-1, 64]           8,256
           Linear-11                  [-1, 10]             650
================================================================
Total params: 434,122
Trainable params: 434,122
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 0.22
Params size (MB): 1.66
Estimated Total Size (MB): 1.88
----------------------------------------------------------------
```

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

```
def train_model(model, train_loader, num_epochs=3):

    # write your code here
    print('Name: A.LAHARI')
    print('Register Number: 212223230111')
    for epoch in range(num_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}/{num_epochs}], Loss: {running_loss/len(train_loader):.4f}')
```

```
train_model(model, train_loader)
```

```
Name: A.LAHARI
Register Number: 212223230111
Epoch [1/3], Loss: 0.7208
Epoch [2/3], Loss: 0.4659
Epoch [3/3], Loss: 0.3711
```

```
def test_model(model, test_loader):
    model.eval()
    correct = 0
    total = 0
    all_preds = []
    all_labels = []

    with torch.no_grad():
        for images, labels in test_loader:
            outputs = model(images)
            _, predicted = torch.max(outputs, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()
            all_preds.extend(predicted.cpu().numpy())
```

```
            all_labels.extend(labels.cpu().numpy())

    accuracy = correct / total
    print('Name: A.LAHARI')
    print('Register Number: 212223230111')
    print(f'Test Accuracy: {accuracy:.4f}')
    # Compute confusion matrix
    cm = confusion_matrix(all_labels, all_preds)
    plt.figure(figsize=(8, 6))
    print('Name: A.LAHARI')
    print('Register Number: 212223230111')
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=test_dataset.classes, yticklabels=test_dataset.classes)
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    plt.title('Confusion Matrix')
    plt.show()

    # Print classification report
    print('Name: A.LAHARI')
    print('Register Number: 212223230111')
    print("Classification Report:")
    print(classification_report(all_labels, all_preds, target_names=test_dataset.classes))
```

```
test_model(model, test_loader)
```

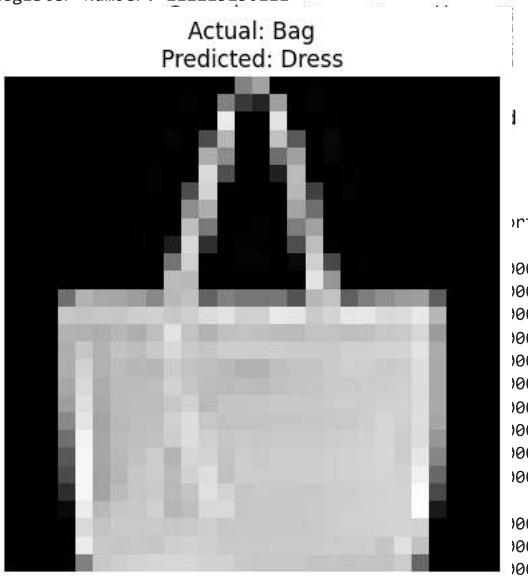
```
Name: A.LAHARI
Register Number: 212223230111
Test Accuracy: 0.1000
Name: A.LAHARI
Register Number: 212223230111
```


```
import matplotlib.pyplot as plt
def predict_image(model, image_index, dataset):
    model.eval()
    image, label = dataset[image_index]
    with torch.no_grad():
        output = model(image.unsqueeze(0))  # Add batch dimension
        _, predicted = torch.max(output, 1)
    class_names = dataset.classes

    # Display the image
    print('Name: A.LAHARI')
    print('Register Number: 212223230111')
    plt.imshow(image.squeeze(), cmap="gray")
    plt.title(f'Actual: {class_names[label]}\nPredicted: {class_names[predicted.item()]}')
    plt.axis("off")
    plt.show()
    print(f'Actual: {class_names[label]}, Predicted: {class_names[predicted.item()]}')
```

```
predict_image(model, image_index=900, dataset=test_dataset)
```

```
Name: A.LAHARI
Register Number: 212223230111
```




```
Actual: Bag, Predicted: Dress
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defir
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defir
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defir
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```