```
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
```

```
## Step 1: Load and Preprocess Data
# Define transformations for images
transform = transforms.Compose([
    transforms.ToTensor(),          # Convert images to tensors
    transforms.Normalize((0.5,), (0.5,))  # Normalize images
])
```

```
# Load Fashion-MNIST dataset
train_dataset = torchvision.datasets.FashionMNIST(root="./data", train=True, transform=transform, downl
test_dataset = torchvision.datasets.FashionMNIST(root="./data", train=False, transform=transform, downl
```

```
# Get the shape of the first image in the training dataset
image, label = train_dataset[0]
print(image.shape)
print(len(train_dataset))
```

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

```
# Create DataLoader for batch processing
train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=32, shuffle=False)
```

```
## Step 1: Load and Preprocess Data
# Define transformations for images
transform = transforms.Compose([
    transforms.ToTensor(),          # Convert images to tensors
    transforms.Normalize((0.5,), (0.5,))  # Normalize images
])
```

```
# Load Fashion-MNIST dataset
train_dataset = torchvision.datasets.FashionMNIST(root="./data", train=True, transform=transform, dow
test_dataset = torchvision.datasets.FashionMNIST(root="./data", train=False, transform=transform, dow
```

```
# Get the shape of the first image in the training dataset
image, label = train_dataset[0]
print(image.shape)
print(len(train_dataset))
```

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

```
# Create DataLoader for batch processing
train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=32, shuffle=False)
```

```
# Load Fashion-MNIST dataset
train_dataset = torchvision.datasets.FashionMNIST(root="./data", train=True, transform=transform, dow
test_dataset = torchvision.datasets.FashionMNIST(root="./data", train=False, transform=transform, dow
```

```
# Get the shape of the first image in the training dataset
image, label = train_dataset[0]
print(image.shape)
print(len(train_dataset))
```

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

```
# Create DataLoader for batch processing
train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=32, shuffle=False)
```

```
class CNNClassifier (nn.Module):
  def __init__(self):
    super (CNNClassifier, self).__init__()
    self.conv1 = nn.Conv2d(in_channels=1, out_channels=32, kernel_size=3, padding=1)
    self.conv2 = nn.Conv2d(in_channels=32, out_channels=64, kernel_size=3, padding=1)
    self.conv3 = nn.Conv2d(in_channels=64, out_channels=128, kernel_size=3, padding=1)
    self.pool = nn.MaxPool2d(kernel_size=2, stride=2)
    self.fc1 = nn.Linear(128 * 3 * 3, 128)
    self.fc2 = nn.Linear (128, 64)
    self.fc3 = nn.Linear (64, 10)
  def forward(self, x):
    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 = x.view(x.size(0), -1) # Flatten the image
    x = torch.relu(self.fc1(x))
    x = torch.relu(self.fc2(x))
    x = self.fc3(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:CHANDRU')
print('Register Number:212223110007')
summary(model, input_size=(1, 28, 28))
```

```
Name:CHANDRU
Register Number:212223110007
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1          [-1, 32, 28, 28]             320
         MaxPool2d-2          [-1, 32, 14, 14]               0
            Conv2d-3          [-1, 64, 14, 14]          18,496
         MaxPool2d-4            [-1, 64, 7, 7]               0
            Conv2d-5           [-1, 128, 7, 7]          73,856
         MaxPool2d-6           [-1, 128, 3, 3]               0
            Linear-7                 [-1, 128]         147,584
            Linear-8                  [-1, 64]           8,256
            Linear-9                  [-1, 10]             650
================================================================
Total params: 249,162
Trainable params: 249,162
Non-trainable params: 0
----------------------------------------------------------------
```

```
Input size (MB): 0.00
Forward/backward pass size (MB): 0.42
Params size (MB): 0.95
Estimated Total Size (MB): 1.37
----------------------------------------------------------------
```

```
# Initialize the Model, Loss Function, and Optimizer
model = CNNClassifier()
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)
```

```
# Train the Model
def train_model(model, train_loader, num_epochs=3):
    # Move model to GPU if available, inside the function
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    model.to(device)

    for epoch in range(num_epochs):
        running_loss = 0.0
        for images, labels in train_loader:
            # Move images and labels to the same device as the model
            images = images.to(device)
            labels = labels.to(device)

            # Forward pass
            outputs = model(images)
            loss = criterion(outputs, labels)

            # Backward and optimize
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()

            running_loss += loss.item()

        print('Name:CHANDRU.P')
        print('Register Number:212223110007')
        print(f'Epoch [{epoch+1}/{num_epochs}], Loss: {running_loss/len(train_loader):.4f}')
```

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

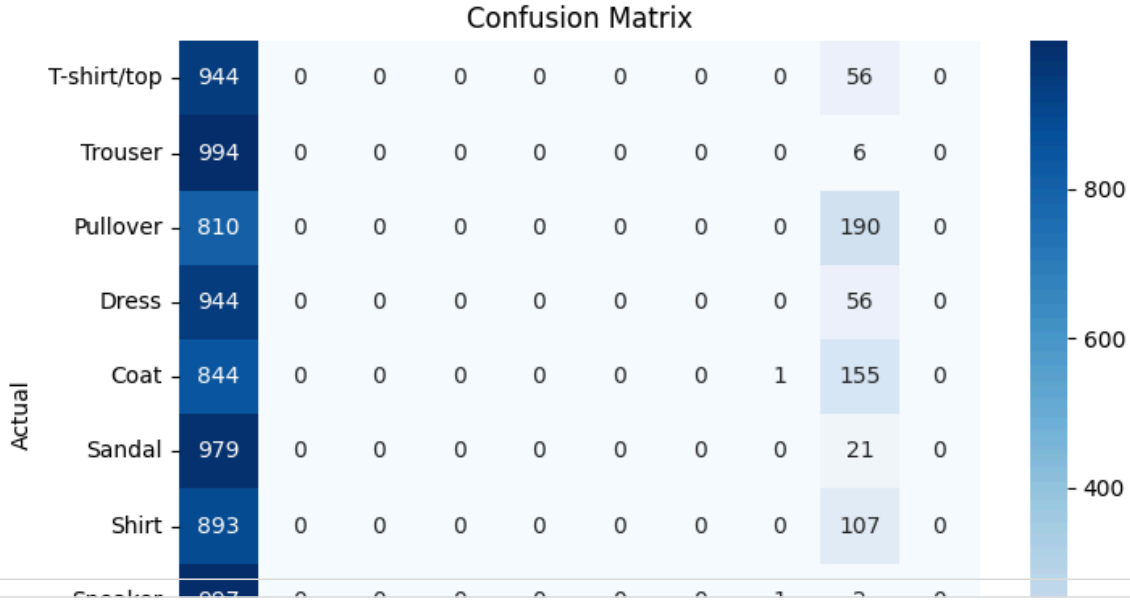
```
Name:CHANDRU.P
Register Number:212223110007
Epoch [1/3], Loss: 0.2053
Name:CHANDRU.P
Register Number:212223110007
Epoch [2/3], Loss: 0.1796
Name:CHANDRU.P
Register Number:212223110007
Epoch [3/3], Loss: 0.1574
```

```
## Step 4: Test the Model
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:        ')
    print('Register Number:       ')
    print(f'Test Accuracy: {accuracy:.4f}')

    # Compute confusion matrix
    cm = confusion_matrix(all_labels, all_preds)
    plt.figure(figsize=(8, 6))
    print('Name:        ')
    print('Register Number:       ')
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=test_dataset.classes, yticklabels=
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    plt.title('Confusion Matrix')
    plt.show()

    # Print classification report
    print('Name:CHANDRU.P        ')
    print('Register Number:212223110007       ')
    print("Classification Report:")
    print(classification_report(all_labels, all_preds, target_names=test_dataset.classes))
```

```
# Evaluate the model
test_model(model, test_loader)
```

```
Name:
Register Number:
Test Accuracy: 0.0993
Name:
Register Number:
```

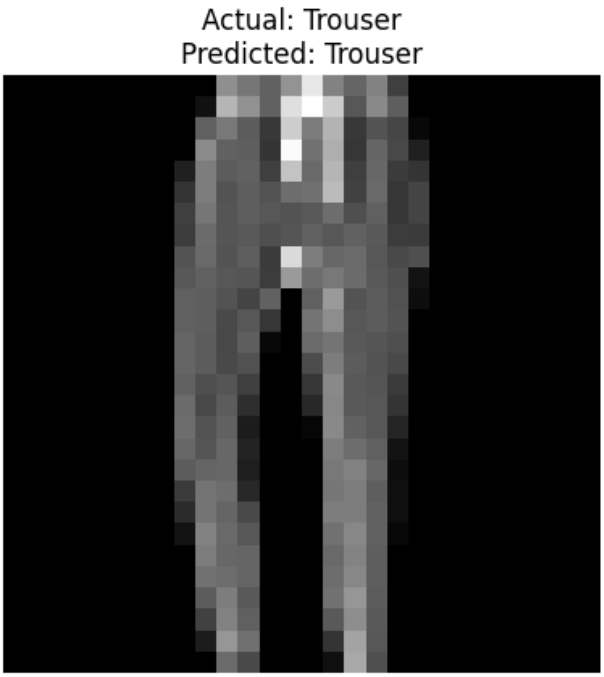



```
## Step 5: Predict on a Single Image
import matplotlib.pyplot as plt
def predict_image(model, image_index, dataset):
    model.eval()
    image, label = dataset[image_index]
    with torch.no_grad():
        output = model(image.unsqueeze(0))  # Add batch dimension
        _, predicted = torch.max(output, 1)
    class_names = dataset.classes

    # Display the image
    print('Name:CHANDRU.P        ')
    print('Register Number:212223110007       ')
    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()]}')
```

```
# Example Prediction
predict_image(model, image_index=80, dataset=test_dataset)
```

```
     Sneaker       0.50      0.00      0.00      1000
         Bag       0.07      0.05      0.06      1000
  Ankle boot       0.00      0.00      0.00      1000

    accuracy                           0.10     10000
   macro avg       0.07      0.10      0.02     10000
weighted avg       0.07      0.10      0.02     10000

/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarnin
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarnin
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarnin
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

```
Name:CHANDRU.P
Register Number:212223110007
```




```
Actual: Trouser, Predicted: Trouser
```