```
# Import Libraries
import torch
import torch.nn as nn
import torch.optim as optim
from torchvision import datasets, transforms, models
from torch.utils.data import DataLoader
from sklearn.metrics import classification_report, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns

# Device setup
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
```

```
# 1. Data Preprocessing
transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
])

train_data = datasets.CIFAR10(root='./data', train=True, transform=transform, download=True)
test_data = datasets.CIFAR10(root='./data', train=False, transform=transform, download=True)

train_loader = DataLoader(train_data, batch_size=64, shuffle=True)
test_loader = DataLoader(test_data, batch_size=64, shuffle=False)
```

```
100%|██████████| 170M/170M [00:13<00:00, 12.3MB/s]
```

```
# 2. Load Pre-trained Model
model = models.resnet18(pretrained=True)
```

```
/usr/local/lib/python3.12/dist-packages/torchvision/models/_utils.py:208: UserWarning: The parameter 'pretrained' is depreca
  warnings.warn(
/usr/local/lib/python3.12/dist-packages/torchvision/models/_utils.py:223: UserWarning: Arguments other than a weight enum or
  warnings.warn(msg)
Downloading: "https://download.pytorch.org/models/resnet18-f37072fd.pth" to /root/.cache/torch/hub/checkpoints/resnet18-f37(
100%|██████████| 44.7M/44.7M [00:00<00:00, 211MB/s]
```

```
# 3. Freeze feature extractor layers
for param in model.parameters():
    param.requires_grad = False
```

```
# 4. Modify final layer for CIFAR10 (10 classes)
num_features = model.fc.in_features
model.fc = nn.Linear(num_features, 10)
model = model.to(device)
```

```
# 5. Define Loss and Optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.fc.parameters(), lr=0.001)
```

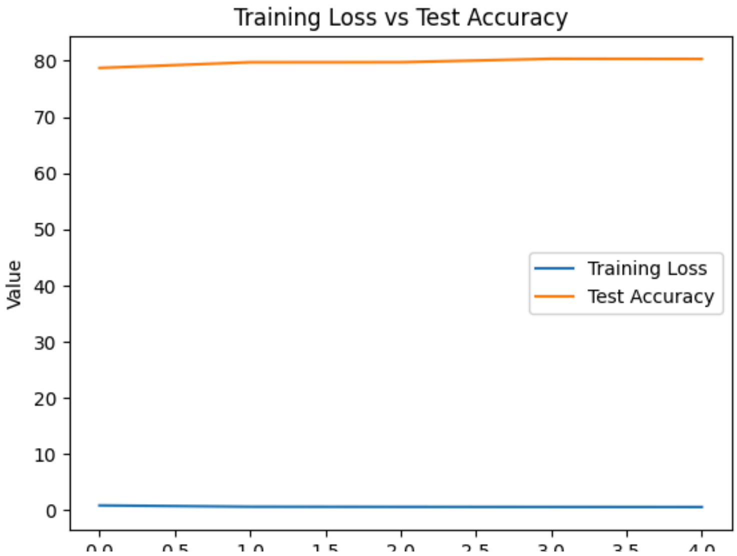
```
# 6. Training Loop
train_losses, test_accuracies = [], []
for epoch in range(5):  # small epochs for simplicity
    model.train()
    running_loss = 0.0
    for images, labels in train_loader:
        images, labels = images.to(device), labels.to(device)
        optimizer.zero_grad()
        outputs = model(images)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        running_loss += loss.item()

    train_losses.append(running_loss / len(train_loader))

    # Evaluate accuracy
    model.eval()
    correct, total = 0, 0
    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)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()
    acc = 100 * correct / total
    test_accuracies.append(acc)
    print(f"Epoch [{epoch+1}/5] - Loss: {train_losses[-1]:.4f} - Test Accuracy: {acc:.2f}%")
```

```
Epoch [1/5] - Loss: 0.8472 - Test Accuracy: 78.73%
Epoch [2/5] - Loss: 0.6233 - Test Accuracy: 79.74%
Epoch [3/5] - Loss: 0.5949 - Test Accuracy: 79.75%
Epoch [4/5] - Loss: 0.5790 - Test Accuracy: 80.36%
Epoch [5/5] - Loss: 0.5685 - Test Accuracy: 80.33%
```

```
# 7. Classification Report
y_true, y_pred = [], []
model.eval()
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)
        y_true.extend(labels.cpu().numpy())
        y_pred.extend(predicted.cpu().numpy())

print("\nClassification Report:\n")
print(classification_report(y_true, y_pred, target_names=train_data.classes))
```

```

Classification Report:

              precision    recall  f1-score   support

    airplane       0.80      0.82      0.81      1000
  automobile       0.94      0.83      0.88      1000
        bird       0.82      0.68      0.74      1000
         cat       0.73      0.66      0.69      1000
        deer       0.71      0.81      0.76      1000
         dog       0.78      0.77      0.77      1000
        frog       0.86      0.81      0.83      1000
       horse       0.75      0.88      0.81      1000
        ship       0.84      0.88      0.86      1000
       truck       0.84      0.91      0.88      1000

    accuracy                           0.80     10000
   macro avg       0.81      0.80      0.80     10000
weighted avg       0.81      0.80      0.80     10000
```

```
# 8. Accuracy/Loss Graph
plt.figure()
plt.plot(train_losses, label='Training Loss')
plt.plot(test_accuracies, label='Test Accuracy')
plt.legend()
plt.title('Training Loss vs Test Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Value')
plt.show()
```


Training Loss vs Test Accuracy
Value
80
70
60
50
40
30
20
10
0
Training Loss
Test Accuracy
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0