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


# Allow loading of truncated images
ImageFile.LOAD_TRUNCATED_IMAGES = True

from google.colab import drive

drive.mount('/content/drive')

# Define transformations
transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize([0.5], [0.5], [0.5])
])
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
# Paths
train_data_dir = '/content/drive/MyDrive/edgeai_dataset/train'
test_data_dir = '/content/drive/MyDrive/edgeai_dataset/test'

# Datasets and Dataloaders
train_dataset = datasets.ImageFolder(train_data_dir, transform=transform)
test_dataset = datasets.ImageFolder(test_data_dir, transform=transform)

train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True, num_workers=2)
test_loader = DataLoader(test_dataset, batch_size=32, shuffle=False, num_workers=2)

num_classes = len(train_dataset.classes)
print(f"Training Samples: {len(train_dataset)}, Testing Samples: {len(test_dataset)}, Classes: {num_classes}")
```

```
Training Samples: 4570, Testing Samples: 1131, Classes: 19
```

```
# Define CNN model
class MyCNN(nn.Module):
    def __init__(self, num_classes):
        super(MyCNN, self).__init__()
        self.features = nn.Sequential(
            nn.Conv2d(3, 64, 3, 1, 1),
            nn.ReLU(),
            nn.MaxPool2d(2),

            nn.Conv2d(64, 128, 3, 1, 1),
            nn.ReLU(),
            nn.MaxPool2d(2),

            nn.Conv2d(128, 256, 3, 1, 1),
            nn.ReLU(),
            nn.MaxPool2d(2),

            nn.Conv2d(256, 512, 3, 1, 1),
            nn.ReLU(),
            nn.MaxPool2d(2),
        )
        with torch.no_grad():
            dummy = torch.zeros(1, 3, 224, 224)
            flat_dim = self.features(dummy).view(1, -1).shape[1]

        self.classifier = nn.Sequential(
            nn.Linear(flat_dim, 512),
            nn.ReLU(),
            nn.Dropout(0.5),
            nn.Linear(512, num_classes)
        )

    def forward(self, x):
        x = self.features(x)
        x = x.view(x.size(0), -1)
        return self.classifier(x)
```

```
# Set device and initialize model
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = MyCNN(num_classes).to(device)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.0001)

# Evaluation function
def evaluate(model, loader, device):
    model.eval()
    correct, total = 0, 0
    with torch.no_grad():
        for images, labels in loader:
            images, labels = images.to(device), labels.to(device)
            outputs = model(images)
            _, predicted = torch.max(outputs, 1)
            correct += (predicted == labels).sum().item()
            total += labels.size(0)
    return 100 * correct / total
```

```
# Training function
def train_model(model, train_loader, criterion, optimizer, device):
    model.train()
    running_loss, correct, total = 0.0, 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() * images.size(0)
        _, predicted = torch.max(outputs, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

    epoch_loss = running_loss / total
    epoch_acc = 100 * correct / total
    return epoch_loss, epoch_acc

# Training loop
epochs = 10
for epoch in range(epochs):
    train_loss, train_acc = train_model(model, train_loader, criterion, optimizer, device)
    test_acc = evaluate(model, test_loader, device)
    print(f"Epoch [{epoch+1}/{epochs}] | Loss: {train_loss:.4f} | Train Acc: {train_acc:.2f}% | Test Acc: {test_acc:.2f}%")

# Final evaluation
final_test_acc = evaluate(model, test_loader, device)
print("\nFinal Evaluation on Test Set:")
print(f"  Test Accuracy: {final_test_acc:.2f}%")
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
Training Samples: 4570, Testing Samples: 1131, Classes: 19
/usr/local/lib/python3.11/dist-packages/PIL/Image.py:1047: UserWarning: Palette images with Transparency expressed in bytes should b
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/PIL/Image.py:1047: UserWarning: Palette images with Transparency expressed in bytes should b
  warnings.warn(
Epoch [1/10] | Loss: 1.9903 | Train Acc: 39.87% | Test Acc: 53.32%
/usr/local/lib/python3.11/dist-packages/PIL/Image.py:1047: UserWarning: Palette images with Transparency expressed in bytes should b
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/PIL/Image.py:1047: UserWarning: Palette images with Transparency expressed in bytes should b
  warnings.warn(
Epoch [2/10] | Loss: 1.2142 | Train Acc: 62.91% | Test Acc: 62.51%
/usr/local/lib/python3.11/dist-packages/PIL/Image.py:1047: UserWarning: Palette images with Transparency expressed in bytes should b
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/PIL/Image.py:1047: UserWarning: Palette images with Transparency expressed in bytes should b
  warnings.warn(
Epoch [3/10] | Loss: 0.8139 | Train Acc: 74.86% | Test Acc: 66.31%
/usr/local/lib/python3.11/dist-packages/PIL/Image.py:1047: UserWarning: Palette images with Transparency expressed in bytes should b
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/PIL/Image.py:1047: UserWarning: Palette images with Transparency expressed in bytes should b
  warnings.warn(
Epoch [4/10] | Loss: 0.5240 | Train Acc: 83.52% | Test Acc: 70.20%
/usr/local/lib/python3.11/dist-packages/PIL/Image.py:1047: UserWarning: Palette images with Transparency expressed in bytes should b
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/PIL/Image.py:1047: UserWarning: Palette images with Transparency expressed in bytes should b
  warnings.warn(
Epoch [5/10] | Loss: 0.3262 | Train Acc: 89.98% | Test Acc: 69.94%
```

```
/usr/local/lib/python3.11/dist-packages/PIL/Image.py:1047: UserWarning: Palette images with Transparency expressed in bytes should b
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/PIL/Image.py:1047: UserWarning: Palette images with Transparency expressed in bytes should b
  warnings.warn(
Epoch [6/10] | Loss: 0.2214 | Train Acc: 92.95% | Test Acc: 72.59%
/usr/local/lib/python3.11/dist-packages/PIL/Image.py:1047: UserWarning: Palette images with Transparency expressed in bytes should b
  warnings.warn(
Epoch [7/10] | Loss: 0.1635 | Train Acc: 94.95% | Test Acc: 71.88%
/usr/local/lib/python3.11/dist-packages/PIL/Image.py:1047: UserWarning: Palette images with Transparency expressed in bytes should b
  warnings.warn(
Epoch [8/10] | Loss: 0.1136 | Train Acc: 96.35% | Test Acc: 72.50%
/usr/local/lib/python3.11/dist-packages/PIL/Image.py:1047: UserWarning: Palette images with Transparency expressed in bytes should b
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/PIL/Image.py:1047: UserWarning: Palette images with Transparency expressed in bytes should b
  warnings.warn(
Epoch [9/10] | Loss: 0.0857 | Train Acc: 97.42% | Test Acc: 73.30%
/usr/local/lib/python3.11/dist-packages/PIL/Image.py:1047: UserWarning: Palette images with Transparency expressed in bytes should b
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/PIL/Image.py:1047: UserWarning: Palette images with Transparency expressed in bytes should b
  warnings.warn(
Epoch [10/10] | Loss: 0.0603 | Train Acc: 98.03% | Test Acc: 73.03%

Final Evaluation on Test Set:
  Test Accuracy: 73.03%
```

```
from PIL import Image
import matplotlib.pyplot as plt
import torch
from torchvision import transforms

# Define transform used during testing
predict_transform = transforms.Compose([
    transforms.Resize((224, 224)),  # match training size
    transforms.ToTensor(),
    transforms.Normalize([0.5, 0.5, 0.5], [0.5, 0.5, 0.5])
])

# Prediction function
def predict_from_path(image_path, model, class_names):
    img = Image.open(image_path)

    # Convert image to RGB if needed
    if img.mode in ['P', 'RGBA']:
        img = img.convert("RGBA").convert("RGB")
    else:
        img = img.convert("RGB")

    # Apply transform
    input_tensor = predict_transform(img).unsqueeze(0).to(device)

    # Predict
    model.eval()
    with torch.no_grad():
        output = model(input_tensor)
        _, predicted = torch.max(output, 1)
        predicted_label = class_names[predicted.item()]

    # Display result
    print(f"Predicted class: {predicted_label}")
    plt.imshow(img)
    plt.title(f"Prediction: {predicted_label}")
    plt.axis("off")
    plt.show()

# Provide image path
image_path = "/content/drive/MyDrive/edgeai_dataset/test/furniture/Coffee-Table-Transparent-Background.png"  # Change as needed
predict_from_path(image_path, model, train_dataset.classes)
```

```
Predicted class: furniture
```

Prediction: furniture

```
from PIL import Image
import matplotlib.pyplot as plt
import torch
from torchvision import transforms

# Define transform used during testing
predict_transform = transforms.Compose([
    transforms.Resize((224, 224)),  # match training size
    transforms.ToTensor(),
    transforms.Normalize([0.5, 0.5, 0.5], [0.5, 0.5, 0.5])
])

# Prediction function
def predict_from_path(image_path, model, class_names):
    img = Image.open(image_path)

    # Convert image to RGB if needed
    if img.mode in ['P', 'RGBA']:
        img = img.convert("RGBA").convert("RGB")
    else:
        img = img.convert("RGB")

    # Apply transform
    input_tensor = predict_transform(img).unsqueeze(0).to(device)

    # Predict
    model.eval()
    with torch.no_grad():
        output = model(input_tensor)
        _, predicted = torch.max(output, 1)
        predicted_label = class_names[predicted.item()]

    # Display result
    print(f" Predicted class: {predicted_label}")
    plt.imshow(img)
    plt.title(f"Prediction: {predicted_label}")
    plt.axis("off")
    plt.show()

# Provide image path
image_path = "/content/drive/MyDrive/edgeai_dataset/test/signboard/road204.png"  # Change as needed
predict_from_path(image_path, model, train_dataset.classes)
```

```
Predicted class: furniture
```

Prediction: furniture

```
import torch
import torch.nn as nn
import torch.optim as optim
from torchvision import models
from torch.utils.data import DataLoader
from torchvision import datasets, transforms
import os
from PIL import ImageFile
from google.colab import drive

# Fixes for dataset and image loading
ImageFile.LOAD_TRUNCATED_IMAGES = True
drive.mount('/content/drive')

# ================= SETUP =================
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
data_dir = '/content/drive/MyDrive/edgeai_dataset'
train_dir = os.path.join(data_dir, 'train')
test_dir = os.path.join(data_dir, 'test')
batch_size = 32
num_epochs = 10

# ================= TRANSFORMS =================
transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize([0.5]*3, [0.5]*3)
])

# ================= DATA =================
train_dataset = datasets.ImageFolder(train_dir, transform=transform)
test_dataset = datasets.ImageFolder(test_dir, transform=transform)
train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True, num_workers=2)
test_loader = DataLoader(test_dataset, batch_size=batch_size, shuffle=False, num_workers=2)
num_classes = len(train_dataset.classes)
print(f"Classes: {train_dataset.classes}")

# ================= MODEL =================
vgg16 = models.vgg16(weights=models.VGG16_Weights.DEFAULT)  # use pretrained weights

# Freeze all feature layers
for param in vgg16.features.parameters():
    param.requires_grad = False

# Replace the last classifier layer
vgg16.classifier[6] = nn.Linear(4096, num_classes)

model = vgg16.to(device)

# ================= LOSS & OPTIMIZER =================
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.classifier.parameters(), lr=1e-4)

# ================= TRAINING =================
def train(model, loader, optimizer, criterion):
    model.train()
    total_loss, correct, total = 0.0, 0, 0
    for images, labels in loader:
```

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

        total_loss += loss.item() * images.size(0)
        _, predicted = torch.max(outputs, 1)
        correct += (predicted == labels).sum().item()
        total += labels.size(0)

    avg_loss = total_loss / total
    accuracy = 100 * correct / total
    return avg_loss, accuracy

# ================= EVALUATION =================
def evaluate(model, loader):
    model.eval()
    correct, total = 0, 0
    with torch.no_grad():
        for images, labels in loader:
            images, labels = images.to(device), labels.to(device)
            outputs = model(images)
            _, predicted = torch.max(outputs, 1)
            correct += (predicted == labels).sum().item()
            total += labels.size(0)
    return 100 * correct / total

# ================= TRAIN LOOP =================
for epoch in range(num_epochs):
    train_loss, train_acc = train(model, train_loader, optimizer, criterion)
    test_acc = evaluate(model, test_loader)
    print(f"Epoch [{epoch+1}/{num_epochs}] | Train Loss: {train_loss:.4f} | Train Acc: {train_acc:.2f}% | Test Acc: {test_acc:.2f}%")

# Final Test Accuracy
final_acc = evaluate(model, test_loader)
print(f"\n Final Test Accuracy with VGG-16: {final_acc:.2f}%")
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
Classes: ['bed', 'bottles', 'car', 'cellphone', 'chair', 'college', 'furniture', 'indoor', 'monitor', 'nameboard', 'office', 'railwa
/usr/local/lib/python3.11/dist-packages/PIL/Image.py:1047: UserWarning: Palette images with Transparency expressed in bytes should b
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/PIL/Image.py:1047: UserWarning: Palette images with Transparency expressed in bytes should b
  warnings.warn(
Epoch [1/10] | Train Loss: 0.7152 | Train Acc: 78.07% | Test Acc: 87.09%
/usr/local/lib/python3.11/dist-packages/PIL/Image.py:1047: UserWarning: Palette images with Transparency expressed in bytes should b
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/PIL/Image.py:1047: UserWarning: Palette images with Transparency expressed in bytes should b
  warnings.warn(
Epoch [2/10] | Train Loss: 0.1547 | Train Acc: 95.14% | Test Acc: 89.39%
/usr/local/lib/python3.11/dist-packages/PIL/Image.py:1047: UserWarning: Palette images with Transparency expressed in bytes should b
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/PIL/Image.py:1047: UserWarning: Palette images with Transparency expressed in bytes should b
  warnings.warn(
Epoch [3/10] | Train Loss: 0.0543 | Train Acc: 98.60% | Test Acc: 88.24%
/usr/local/lib/python3.11/dist-packages/PIL/Image.py:1047: UserWarning: Palette images with Transparency expressed in bytes should b
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/PIL/Image.py:1047: UserWarning: Palette images with Transparency expressed in bytes should b
  warnings.warn(
Epoch [4/10] | Train Loss: 0.0159 | Train Acc: 99.65% | Test Acc: 88.77%
/usr/local/lib/python3.11/dist-packages/PIL/Image.py:1047: UserWarning: Palette images with Transparency expressed in bytes should b
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/PIL/Image.py:1047: UserWarning: Palette images with Transparency expressed in bytes should b
  warnings.warn(
Epoch [5/10] | Train Loss: 0.0110 | Train Acc: 99.74% | Test Acc: 89.48%
/usr/local/lib/python3.11/dist-packages/PIL/Image.py:1047: UserWarning: Palette images with Transparency expressed in bytes should b
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/PIL/Image.py:1047: UserWarning: Palette images with Transparency expressed in bytes should b
  warnings.warn(
Epoch [6/10] | Train Loss: 0.0044 | Train Acc: 99.96% | Test Acc: 89.57%
/usr/local/lib/python3.11/dist-packages/PIL/Image.py:1047: UserWarning: Palette images with Transparency expressed in bytes should b
  warnings.warn(
Epoch [7/10] | Train Loss: 0.0035 | Train Acc: 99.93% | Test Acc: 89.48%
/usr/local/lib/python3.11/dist-packages/PIL/Image.py:1047: UserWarning: Palette images with Transparency expressed in bytes should b
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/PIL/Image.py:1047: UserWarning: Palette images with Transparency expressed in bytes should b
  warnings.warn(
Epoch [8/10] | Train Loss: 0.0041 | Train Acc: 99.96% | Test Acc: 89.12%
/usr/local/lib/python3.11/dist-packages/PIL/Image.py:1047: UserWarning: Palette images with Transparency expressed in bytes should b
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/PIL/Image.py:1047: UserWarning: Palette images with Transparency expressed in bytes should b
  warnings.warn(
Epoch [9/10] | Train Loss: 0.0028 | Train Acc: 99.98% | Test Acc: 89.21%
/usr/local/lib/python3.11/dist-packages/PIL/Image.py:1047: UserWarning: Palette images with Transparency expressed in bytes should b
```

```
  warnings.warn(
Epoch [10/10] | Train Loss: 0.0018 | Train Acc: 99.96% | Test Acc: 90.72%

✅ Final Test Accuracy with VGG-16: 90.72%
```

```
# 5. Load Pretrained VGG-16 and Modify Classifier

# Set device
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# Define the number of classes
num_classes = len(train_dataset.classes)

vgg16 = models.vgg16(pretrained=True)

# Freeze convolutional layers if you only want to train the classifier
for param in vgg16.features.parameters():
    param.requires_grad = False

# Replace classifier to match your number of classes
vgg16.classifier[6] = nn.Linear(4096, num_classes)

model = vgg16.to(device)


# 6. Define Loss and Optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.classifier.parameters(), lr=0.0001)


# 7. Train VGG-16
num_epochs = 10

for epoch in range(num_epochs):
    model.train()
    running_loss = 0.0
    correct, total = 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()
        _, predicted = torch.max(outputs, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

    acc = 100 * correct / total
    print(f"Epoch [{epoch+1}/{num_epochs}], Loss: {running_loss:.4f}, Accuracy: {acc:.2f}%")
```

```
/usr/local/lib/python3.11/dist-packages/PIL/Image.py:1047: UserWarning: Palette images with Transparency expressed in bytes should b
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/PIL/Image.py:1047: UserWarning: Palette images with Transparency expressed in bytes should b
  warnings.warn(
Epoch [1/10], Loss: 138.1430, Accuracy: 69.17%
/usr/local/lib/python3.11/dist-packages/PIL/Image.py:1047: UserWarning: Palette images with Transparency expressed in bytes should b
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/PIL/Image.py:1047: UserWarning: Palette images with Transparency expressed in bytes should b
  warnings.warn(
Epoch [2/10], Loss: 58.0008, Accuracy: 86.74%
/usr/local/lib/python3.11/dist-packages/PIL/Image.py:1047: UserWarning: Palette images with Transparency expressed in bytes should b
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/PIL/Image.py:1047: UserWarning: Palette images with Transparency expressed in bytes should b
  warnings.warn(
Epoch [3/10], Loss: 39.6811, Accuracy: 91.25%
/usr/local/lib/python3.11/dist-packages/PIL/Image.py:1047: UserWarning: Palette images with Transparency expressed in bytes should b
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/PIL/Image.py:1047: UserWarning: Palette images with Transparency expressed in bytes should b
  warnings.warn(
Epoch [4/10], Loss: 29.6578, Accuracy: 92.58%
/usr/local/lib/python3.11/dist-packages/PIL/Image.py:1047: UserWarning: Palette images with Transparency expressed in bytes should b
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/PIL/Image.py:1047: UserWarning: Palette images with Transparency expressed in bytes should b
  warnings.warn(
Epoch [5/10], Loss: 23.9685, Accuracy: 94.51%
/usr/local/lib/python3.11/dist-packages/PIL/Image.py:1047: UserWarning: Palette images with Transparency expressed in bytes should b
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/PIL/Image.py:1047: UserWarning: Palette images with Transparency expressed in bytes should b
  warnings.warn(
```

```
Epoch [6/10], Loss: 19.1989, Accuracy: 95.78%
/usr/local/lib/python3.11/dist-packages/PIL/Image.py:1047: UserWarning: Palette images with Transparency expressed in bytes should b
  warnings.warn(
Epoch [7/10], Loss: 15.1965, Accuracy: 96.08%
/usr/local/lib/python3.11/dist-packages/PIL/Image.py:1047: UserWarning: Palette images with Transparency expressed in bytes should b
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/PIL/Image.py:1047: UserWarning: Palette images with Transparency expressed in bytes should b
  warnings.warn(
Epoch [8/10], Loss: 13.1364, Accuracy: 96.78%
/usr/local/lib/python3.11/dist-packages/PIL/Image.py:1047: UserWarning: Palette images with Transparency expressed in bytes should b
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/PIL/Image.py:1047: UserWarning: Palette images with Transparency expressed in bytes should b
  warnings.warn(
Epoch [9/10], Loss: 13.2793, Accuracy: 96.85%
/usr/local/lib/python3.11/dist-packages/PIL/Image.py:1047: UserWarning: Palette images with Transparency expressed in bytes should b
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/PIL/Image.py:1047: UserWarning: Palette images with Transparency expressed in bytes should b
  warnings.warn(
Epoch [10/10], Loss: 10.3550, Accuracy: 97.57%
```

```
# 8. Evaluate on Test Set
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.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

print(f"\nTest Accuracy with VGG-16: {100 * correct / total:.2f}%")
```

```
Test Accuracy with VGG-16: 90.19%
```

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

# Assuming train_loader, device, num_classes are already defined

# Load and modify AlexNet
alexnet_model = models.alexnet(pretrained=True)
alexnet_model.classifier[6] = nn.Linear(alexnet_model.classifier[6].in_features, num_classes)

# Define Loss and Optimizer
criterion_alexnet = nn.CrossEntropyLoss()
optimizer_alexnet = optim.Adam(alexnet_model.parameters(), lr=0.0001) # Using the learning rate defined earlier

# Define training function (if not already defined)
def train_model(model, dataloader, criterion, optimizer, device):
    model.train()
    running_loss = 0.0
    correct, total = 0, 0
    for images, labels in dataloader:
        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()
        _, predicted = torch.max(outputs, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()
    accuracy = 100 * correct / total
    return running_loss / len(dataloader), accuracy

# Train AlexNet
print("\nTraining AlexNet...")
alexnet_model.to(device)
num_epochs = 10 # Assuming num_epochs is defined

start_time_alexnet = time.time()

for epoch in range(num_epochs):
    train_loss, train_accuracy = train_model(alexnet_model, train_loader, criterion_alexnet, optimizer_alexnet, device)
```

```
    print(f"AlexNet - Epoch {epoch+1}/{num_epochs}: Train Loss: {train_loss:.4f}, Train Accuracy: {train_accuracy:.4f}"

end_time_alexnet = time.time()
alexnet_training_time = end_time_alexnet - start_time_alexnet

print(f"\nAlexNet Training Time: {alexnet_training_time:.2f}s")

# You would typically follow this with evaluation code
# test_loss_alexnet, test_accuracy_alexnet = evaluate_model(alexnet_model, test_loader, criterion_alexnet, device)
# print(f"AlexNet - Test Loss: {test_loss_alexnet:.4f}, Test Accuracy: {test_accuracy_alexnet:.4f}")
```

```
/usr/local/lib/python3.11/dist-packages/torchvision/models/_utils.py:208: UserWarning: The parameter 'pretrained' is deprecated sinc
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/torchvision/models/_utils.py:223: UserWarning: Arguments other than a weight enum or `None`
  warnings.warn(msg)

Training AlexNet...
/usr/local/lib/python3.11/dist-packages/PIL/Image.py:1047: UserWarning: Palette images with Transparency expressed in bytes should b
  warnings.warn(
AlexNet - Epoch 1/10: Train Loss: 0.8876, Train Accuracy: 72.5383
/usr/local/lib/python3.11/dist-packages/PIL/Image.py:1047: UserWarning: Palette images with Transparency expressed in bytes should b
  warnings.warn(
AlexNet - Epoch 2/10: Train Loss: 0.3470, Train Accuracy: 88.9059
/usr/local/lib/python3.11/dist-packages/PIL/Image.py:1047: UserWarning: Palette images with Transparency expressed in bytes should b
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/PIL/Image.py:1047: UserWarning: Palette images with Transparency expressed in bytes should b
  warnings.warn(
AlexNet - Epoch 3/10: Train Loss: 0.2356, Train Accuracy: 92.0569
/usr/local/lib/python3.11/dist-packages/PIL/Image.py:1047: UserWarning: Palette images with Transparency expressed in bytes should b
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/PIL/Image.py:1047: UserWarning: Palette images with Transparency expressed in bytes should b
  warnings.warn(
AlexNet - Epoch 4/10: Train Loss: 0.1666, Train Accuracy: 94.2888
/usr/local/lib/python3.11/dist-packages/PIL/Image.py:1047: UserWarning: Palette images with Transparency expressed in bytes should b
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/PIL/Image.py:1047: UserWarning: Palette images with Transparency expressed in bytes should b
  warnings.warn(
AlexNet - Epoch 5/10: Train Loss: 0.1192, Train Accuracy: 95.8425
/usr/local/lib/python3.11/dist-packages/PIL/Image.py:1047: UserWarning: Palette images with Transparency expressed in bytes should b
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/PIL/Image.py:1047: UserWarning: Palette images with Transparency expressed in bytes should b
  warnings.warn(
AlexNet - Epoch 6/10: Train Loss: 0.1003, Train Accuracy: 96.6302
/usr/local/lib/python3.11/dist-packages/PIL/Image.py:1047: UserWarning: Palette images with Transparency expressed in bytes should b
  warnings.warn(
AlexNet - Epoch 7/10: Train Loss: 0.0791, Train Accuracy: 97.2429
/usr/local/lib/python3.11/dist-packages/PIL/Image.py:1047: UserWarning: Palette images with Transparency expressed in bytes should b
  warnings.warn(
AlexNet - Epoch 8/10: Train Loss: 0.0654, Train Accuracy: 97.8118
/usr/local/lib/python3.11/dist-packages/PIL/Image.py:1047: UserWarning: Palette images with Transparency expressed in bytes should b
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/PIL/Image.py:1047: UserWarning: Palette images with Transparency expressed in bytes should b
  warnings.warn(
AlexNet - Epoch 9/10: Train Loss: 0.0677, Train Accuracy: 97.6368
/usr/local/lib/python3.11/dist-packages/PIL/Image.py:1047: UserWarning: Palette images with Transparency expressed in bytes should b
  warnings.warn(
AlexNet - Epoch 10/10: Train Loss: 0.0473, Train Accuracy: 98.4245

AlexNet Training Time: 1303.07s
```

```
# Evaluate AlexNet
print("\nEvaluating AlexNet...")
alexnet_model.eval()
correct, total = 0, 0
running_loss = 0.0
criterion_alexnet = nn.CrossEntropyLoss() # Define criterion for evaluation

with torch.no_grad():
    for images, labels in test_loader:
        images, labels = images.to(device), labels.to(device)
        outputs = alexnet_model(images)
        loss = criterion_alexnet(outputs, labels)
        running_loss += loss.item()
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

test_accuracy_alexnet = 100 * correct / total
test_loss_alexnet = running_loss / len(test_loader)
```