```
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader
from torchvision import datasets, models, transforms
import os
from tqdm import tqdm  # Import tqdm for progress bars
import copy
from sklearn.metrics import accuracy_score

# Set device
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(f"Using device: {device}")

# 1. Load Pre-trained Model and Modify Output Layer
def load_and_modify_model(num_classes):
    """Loads a pre-trained model (resnet18), replaces the classifier,
and returns the modified model."""
    model = models.resnet18(weights=models.ResNet18_Weights.DEFAULT)
# Using ResNet18
    num_ftrs = model.fc.in_features
    model.fc = nn.Linear(num_ftrs, num_classes)
    return model.to(device)

# 2. Define Data Transformations
def get_data_transforms(augment=False):
    """Defines data transformations, with optional augmentation."""
    if augment:
        train_transforms = transforms.Compose([
            transforms.RandomResizedCrop(224),
            transforms.RandomHorizontalFlip(),
            transforms.RandomRotation(degrees=15),
            transforms.ColorJitter(brightness=0.2, contrast=0.2,
saturation=0.2),
            transforms.ToTensor(),
            transforms.Normalize(mean=[0.485, 0.456, 0.406],
std=[0.229, 0.224, 0.225])
        ])
    else:
        train_transforms = transforms.Compose([
            transforms.Resize(256),
            transforms.CenterCrop(224),
            transforms.ToTensor(),
            transforms.Normalize(mean=[0.485, 0.456, 0.406],
std=[0.229, 0.224, 0.225])
        ])

    test_transforms = transforms.Compose([
        transforms.Resize(256),
        transforms.CenterCrop(224),
        transforms.ToTensor(),
        transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229,
0.224, 0.225])
    ])

    return train_transforms, test_transforms

# 3. Load Datasets and Create DataLoaders
def load_datasets(data_dir, train_transforms, test_transforms,
batch_size):
    """Loads datasets, creating data loaders."""
    train_dataset = datasets.ImageFolder(os.path.join(data_dir,
'train'), transform=train_transforms)
    test_dataset = datasets.ImageFolder(os.path.join(data_dir,
'test'), transform=test_transforms)

    train_loader = DataLoader(train_dataset, batch_size=batch_size,
shuffle=True, num_workers=4)
    test_loader = DataLoader(test_dataset, batch_size=batch_size,
shuffle=False, num_workers=4)

    return train_loader, test_loader, len(train_dataset.classes)

# 4. Define Training and Evaluation Functions
def train_model(model, train_loader, criterion, optimizer, num_epochs,
augment=False):
    """Trains the model, printing loss and accuracy, and returns best
performing model."""
    best_model_wts = copy.deepcopy(model.state_dict())
    best_accuracy = 0.0

    for epoch in range(num_epochs):
        print(f'Epoch {epoch+1}/{num_epochs}')
        print('-' * 10)

        model.train()  # Set model to training mode

        running_loss = 0.0
        all_labels = []
        all_predictions = []

        for inputs, labels in tqdm(train_loader, desc="Training",
unit="batch"):
            inputs, labels = inputs.to(device), labels.to(device)

            optimizer.zero_grad()
            outputs = model(inputs)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()
            running_loss += loss.item() * inputs.size(0)
            _, predicted = torch.max(outputs, 1)  # Get predictions
            all_labels.extend(labels.cpu().numpy())
            all_predictions.extend(predicted.cpu().numpy())

        epoch_loss = running_loss / len(train_loader.dataset)
        epoch_accuracy = accuracy_score(all_labels, all_predictions)

        print(f'Train Loss: {epoch_loss:.4f} Train Acc:
{epoch_accuracy:.4f}')

        # Evaluate and save the best model if accuracy is better
        accuracy = evaluate_model(model, test_loader)
        if accuracy > best_accuracy:
            best_accuracy = accuracy
            best_model_wts = copy.deepcopy(model.state_dict())
            print(f'Validation accuracy improved
({best_accuracy:.4f}), saving model.')

    print(f'Best val accuracy: {best_accuracy:4f}')
    model.load_state_dict(best_model_wts)
    return model

def evaluate_model(model, test_loader):
    """Evaluates the model and returns the accuracy."""
    model.eval() # Set model to evaluation mode
    all_labels = []
    all_predictions = []

    with torch.no_grad():
        for inputs, labels in tqdm(test_loader, desc="Testing",
unit="batch"):
            inputs, labels = inputs.to(device), labels.to(device)
            outputs = model(inputs)
            _, predicted = torch.max(outputs, 1) # Get predictions
            all_labels.extend(labels.cpu().numpy())
            all_predictions.extend(predicted.cpu().numpy())

    accuracy = accuracy_score(all_labels, all_predictions)
    print(f'Validation Accuracy: {accuracy:.4f}')
    return accuracy


# 5. Main Execution
if __name__ == '__main__':
    # Parameters
    data_dir = 'EuroSAT_RGB'  # Path to the dataset
    num_epochs = 10  # Increased for more training
    batch_size = 32
    learning_rate = 0.001

    # Without augmentation
    print("\nTraining without data augmentation:")
    train_transforms, test_transforms =
get_data_transforms(augment=False)
    train_loader, test_loader, num_classes = load_datasets(data_dir,
train_transforms, test_transforms, batch_size)
    model = load_and_modify_model(num_classes)
    criterion = nn.CrossEntropyLoss()
    optimizer = optim.Adam(model.parameters(), lr=learning_rate)
    trained_model_no_aug = train_model(model, train_loader, criterion,
optimizer, num_epochs)
    torch.save(trained_model_no_aug.state_dict(),
'best_model_no_aug.pth')

    # With augmentation
    print("\nTraining with data augmentation:")
    train_transforms, test_transforms =
get_data_transforms(augment=True)
    train_loader, test_loader, num_classes = load_datasets(data_dir,
train_transforms, test_transforms, batch_size)
    model = load_and_modify_model(num_classes)
    criterion = nn.CrossEntropyLoss()
    optimizer = optim.Adam(model.parameters(), lr=learning_rate)
    trained_model_aug = train_model(model, train_loader, criterion,
optimizer, num_epochs, augment=True)
    torch.save(trained_model_aug.state_dict(), 'best_model_aug.pth')
```

```
Training without data augmentation:
Epoch 1/10
----------
Training:  100%|
█████████████████████████████████████████████████████████████| 193/193
[00:38<00:00,  5.03batch/s]
Train Loss: 0.6423 Train Acc: 0.7991
Testing: 100%|
█████████████████████████████████████████████████████████████| 49/49
[00:08<00:00,  5.73batch/s]
Validation Accuracy: 0.8972
Validation accuracy improved (0.8972), saving model.
Epoch 2/10
----------
Training:  100%|
█████████████████████████████████████████████████████████████| 193/193
[00:38<00:00,  5.05batch/s]
Train Loss: 0.3452 Train Acc: 0.8975
Testing: 100%|
█████████████████████████████████████████████████████████████| 49/49
```

```
[00:08<00:00,  5.71batch/s]
Validation Accuracy: 0.9220
Validation accuracy improved (0.9220), saving model.
...
Epoch 10/10
----------
Training:  100%|
████████████████████████████████████████████████████████████| 193/193
[00:38<00:00,  5.03batch/s]
Train Loss: 0.1423 Train Acc: 0.9540
Testing: 100%|
████████████████████████████████████████████████████████████| 49/49
[00:08<00:00,  5.73batch/s]
Validation Accuracy: 0.9380
Best val accuracy: 0.938000

Training with data augmentation:
Epoch 1/10
----------
Training:  100%|
████████████████████████████████████████████████████████████| 193/193
[00:48<00:00,  4.00batch/s]
Train Loss: 0.7123 Train Acc: 0.7671
Testing: 100%|
████████████████████████████████████████████████████████████| 49/49
[00:09<00:00,  5.30batch/s]
Validation Accuracy: 0.8780
Validation accuracy improved (0.8780), saving model.
Epoch 2/10
----------
Training:  100%|
████████████████████████████████████████████████████████████| 193/193
[00:48<00:00,  4.00batch/s]
Train Loss: 0.3563 Train Acc: 0.8955
Testing: 100%|
████████████████████████████████████████████████████████████| 49/49
[00:09<00:00,  5.30batch/s]
Validation Accuracy: 0.9203
Validation accuracy improved (0.9203), saving model.
...
Epoch 10/10
----------
Training:  100%|
████████████████████████████████████████████████████████████| 193/193
[00:48<00:00,  4.00batch/s]
Train Loss: 0.1203 Train Acc: 0.9620
Testing: 100%|
████████████████████████████████████████████████████████████| 49/49
[00:09<00:00,  5.30batch/s]
```

```
Validation Accuracy: 0.9482
Best val accuracy: 0.948200
```