```
import torch
import torchvision
import torchvision.transforms as transforms
import torch.nn as nn
import torch.optim as optim
from torchvision.models import densenet121, DenseNet121_Weights


# Set device to GPU if available, else use CPU
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(device)
```

```
cuda
```

```
# Define transforms for data preprocessing
normMean = [0.49139968, 0.48215827, 0.44653124]
normStd = [0.24703233, 0.24348505, 0.26158768]
normTransform = transforms.Normalize(normMean, normStd)

trainTransform = transforms.Compose([
      transforms.RandomCrop(32, padding=4),
      transforms.RandomHorizontalFlip(),
      transforms.ToTensor(),
      normTransform
  ])
testTransform = transforms.Compose([
    transforms.ToTensor(),
    normTransform
])


# Load CIFAR-10 dataset
trainset = torchvision.datasets.CIFAR10(root='./data', train=True,
                                        download=True, transform=trainTransform)
trainloader = torch.utils.data.DataLoader(trainset, batch_size=64,
                                          shuffle=True, num_workers=2)

testset = torchvision.datasets.CIFAR10(root='./data', train=False,
                                       download=True, transform=testTransform)
testloader = torch.utils.data.DataLoader(testset, batch_size=64,
                                         shuffle=False, num_workers=2)

# Load pre-trained DenseNet model
model = densenet121(weights=DenseNet121_Weights.DEFAULT)
model.to(device)

# Define loss function and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(), lr=1e-1, momentum=0.9, weight_decay=1e-4)

def adjust_opt(optAlg, optimizer, epoch):
    if optAlg == 'sgd':
        if epoch < 150:
            lr = 1e-1
        elif epoch == 150:
            lr = 1e-2
        elif epoch == 225:
            lr = 1e-3
        else:
            return

        for param_group in optimizer.param_groups:
            param_group['lr'] = lr
    return lr
# Train the model
for epoch in range(300):
    lr = adjust_opt('sgd', optimizer, epoch)  # Adjust learning rate
    running_loss = 0.0
    correct = 0
    total = 0
    for i, data in enumerate(trainloader, 0):
        inputs, labels = data[0].to(device), data[1].to(device)
        optimizer.zero_grad()

        outputs = model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        # Calculate accuracy
        _, predicted = torch.max(outputs.data, 1)
```

```
            total += labels.size(0)
            correct += (predicted == labels).sum().item()

            running_loss += loss.item()
        # print every epoch
        print(f'Epoch: {epoch + 1}, Loss: {running_loss / i:.4f}, Accuracy: {100 * correct / total:.2f}%, lr: {lr:.4f}')
        running_loss = 0.0

    print("Training finished!")

    # Test the model
    correct = 0
    total = 0
    with torch.no_grad():
        for data in testloader:
            images, labels = data[0].to(device), data[1].to(device)
            outputs = model(images)
            _, predicted = torch.max(outputs.data, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()

    accuracy = 100 * correct / total
    print(f'Test Accuracy: {accuracy:.2f}%')
```

```
Epoch: 85, Loss: 0.4880, Accuracy: 83.12%, lr: 0.1000
Epoch: 86, Loss: 0.4765, Accuracy: 83.69%, lr: 0.1000
Epoch: 87, Loss: 0.4777, Accuracy: 83.42%, lr: 0.1000
Epoch: 88, Loss: 0.4748, Accuracy: 83.47%, lr: 0.1000
Epoch: 89, Loss: 0.4738, Accuracy: 83.54%, lr: 0.1000
Epoch: 90, Loss: 0.4776, Accuracy: 83.62%, lr: 0.1000
Epoch: 91, Loss: 0.4805, Accuracy: 83.23%, lr: 0.1000
Epoch: 92, Loss: 0.4751, Accuracy: 83.57%, lr: 0.1000
Epoch: 93, Loss: 0.4734, Accuracy: 83.53%, lr: 0.1000
Epoch: 94, Loss: 0.4783, Accuracy: 83.39%, lr: 0.1000
Epoch: 95, Loss: 0.4726, Accuracy: 83.58%, lr: 0.1000
Epoch: 96, Loss: 0.4715, Accuracy: 83.67%, lr: 0.1000
Epoch: 97, Loss: 0.4623, Accuracy: 83.97%, lr: 0.1000
Epoch: 98, Loss: 0.4673, Accuracy: 83.85%, lr: 0.1000
Epoch: 99, Loss: 0.4639, Accuracy: 83.89%, lr: 0.1000
Epoch: 100, Loss: 0.4711, Accuracy: 83.61%, lr: 0.1000
Epoch: 101, Loss: 0.4701, Accuracy: 83.81%, lr: 0.1000
Epoch: 102, Loss: 0.4704, Accuracy: 83.78%, lr: 0.1000
Epoch: 103, Loss: 0.4567, Accuracy: 84.28%, lr: 0.1000
Epoch: 104, Loss: 0.4645, Accuracy: 83.96%, lr: 0.1000
Epoch: 105, Loss: 0.4609, Accuracy: 84.05%, lr: 0.1000
Epoch: 106, Loss: 0.4644, Accuracy: 83.99%, lr: 0.1000
Epoch: 107, Loss: 0.4578, Accuracy: 84.27%, lr: 0.1000
Epoch: 108, Loss: 0.4620, Accuracy: 83.88%, lr: 0.1000
Epoch: 109, Loss: 0.4592, Accuracy: 84.13%, lr: 0.1000
Epoch: 110, Loss: 0.4626, Accuracy: 83.98%, lr: 0.1000
Epoch: 111, Loss: 0.4645, Accuracy: 83.97%, lr: 0.1000
Epoch: 112, Loss: 0.4575, Accuracy: 84.15%, lr: 0.1000
Epoch: 113, Loss: 0.4589, Accuracy: 83.93%, lr: 0.1000
Epoch: 114, Loss: 0.4509, Accuracy: 84.42%, lr: 0.1000
Epoch: 115, Loss: 0.4565, Accuracy: 84.26%, lr: 0.1000
Epoch: 116, Loss: 0.4535, Accuracy: 84.26%, lr: 0.1000
Epoch: 117, Loss: 0.4572, Accuracy: 84.16%, lr: 0.1000
Epoch: 118, Loss: 0.4530, Accuracy: 84.25%, lr: 0.1000
Epoch: 119, Loss: 0.4536, Accuracy: 84.32%, lr: 0.1000
Epoch: 120, Loss: 0.4555, Accuracy: 84.12%, lr: 0.1000
Epoch: 121, Loss: 0.4548, Accuracy: 84.24%, lr: 0.1000
Epoch: 122, Loss: 0.4476, Accuracy: 84.52%, lr: 0.1000
Epoch: 123, Loss: 0.4489, Accuracy: 84.42%, lr: 0.1000
Epoch: 124, Loss: 0.4482, Accuracy: 84.52%, lr: 0.1000
Epoch: 125, Loss: 0.4439, Accuracy: 84.71%, lr: 0.1000
Epoch: 126, Loss: 0.4501, Accuracy: 84.48%, lr: 0.1000
Epoch: 127, Loss: 0.4516, Accuracy: 84.34%, lr: 0.1000
Epoch: 128, Loss: 0.4432, Accuracy: 84.63%, lr: 0.1000
Epoch: 129, Loss: 0.4526, Accuracy: 84.36%, lr: 0.1000
Epoch: 130, Loss: 0.4438, Accuracy: 84.64%, lr: 0.1000
Epoch: 131, Loss: 0.4482, Accuracy: 84.55%, lr: 0.1000
Epoch: 132, Loss: 0.4520, Accuracy: 84.36%, lr: 0.1000
Epoch: 133, Loss: 0.4444, Accuracy: 84.42%, lr: 0.1000
Epoch: 134, Loss: 0.4474, Accuracy: 84.69%, lr: 0.1000
Epoch: 135, Loss: 0.4479, Accuracy: 84.58%, lr: 0.1000
Epoch: 136, Loss: 0.4414, Accuracy: 84.84%, lr: 0.1000
Epoch: 137, Loss: 0.4457, Accuracy: 84.52%, lr: 0.1000
Epoch: 138, Loss: 0.4355, Accuracy: 84.93%, lr: 0.1000
Epoch: 139, Loss: 0.4552, Accuracy: 84.15%, lr: 0.1000
Epoch: 140, Loss: 0.4519, Accuracy: 84.38%, lr: 0.1000
Epoch: 141, Loss: 0.4499, Accuracy: 84.29%, lr: 0.1000
Epoch: 142, Loss: 0.4542, Accuracy: 84.32%, lr: 0.1000
```