代码

import torch  
import torch.nn as nn  
import torch.optim as optim  
import torchvision  
import torchvision.transforms as transforms  
  
# 定义ResNet18模型  
class ResNet18(nn.Module):  
 def \_\_init\_\_(self, num\_classes=10):  
 super(ResNet18, self).\_\_init\_\_()  
 self.conv1 = nn.Conv2d(3, 64, kernel\_size=3, padding=1)  
 self.bn1 = nn.BatchNorm2d(64)  
 self.relu = nn.ReLU(inplace=True)  
 self.maxpool = nn.MaxPool2d(kernel\_size=3, stride=2, padding=1)  
 self.layer1 = self.\_make\_layer(64, 3, 1)  
 self.layer2 = self.\_make\_layer(128, 4, 2)  
 self.layer3 = self.\_make\_layer(256, 6, 2)  
 self.layer4 = self.\_make\_layer(512, 3, 2)  
 self.avgpool = nn.AdaptiveAvgPool2d((1, 1))  
 self.fc = nn.Linear(512 \* 1 \* 1, num\_classes)  
  
 def \_make\_layer(self, inplanes, planes, blocks, stride=1):  
 downsample = None  
 if stride != 1 or inplanes != planes:  
 downsample = nn.Sequential(  
 nn.Conv2d(inplanes, planes, kernel\_size=1, stride=stride, bias=False),  
 nn.BatchNorm2d(planes),  
 )  
  
 layers = []  
 layers.append(nn.Conv2d(inplanes, planes, kernel\_size=3, padding=1, bias=False))  
 layers.append(nn.BatchNorm2d(planes))  
 layers.append(nn.ReLU(inplace=True))  
 layers.append(nn.Conv2d(planes, planes, kernel\_size=3, stride=stride, padding=1, bias=False))  
 layers.append(nn.BatchNorm2d(planes))  
 layers.append(nn.ReLU(inplace=True))  
  
 inplanes = planes  
  
 for i in range(1, blocks):  
 layers.append(nn.Conv2d(inplanes, planes, kernel\_size=3, padding=1, bias=False))  
 layers.append(nn.BatchNorm2d(planes))  
 layers.append(nn.ReLU(inplace=True))  
  
 return nn.Sequential(\*layers), downsample  
  
 def forward(self, x):  
 x = self.conv1(x)  
 x = self.bn1(x)  
 x = self.relu(x)  
 x = self.maxpool(x)  
  
 x = self.layer1(x)  
 x = self.layer2(x)  
 x = self.layer3(x)  
 x = self.layer4(x)  
  
 x = self.avgpool(x)  
 x = torch.flatten(x, 1)  
 x = self.fc(x)  
  
 return x  
  
# 准备数据  
transform = transforms.Compose(  
 [transforms.Resize(32),  
 transforms.CenterCrop(32),  
 transforms.ToTensor(),  
 transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))]  
)  
  
trainset = torchvision.datasets.CIFAR10(root='./data', train=True,  
 download=True, transform=transform)  
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=transform)  
testloader = torch.utils.data.DataLoader(testset, batch\_size=64,  
 shuffle=False, num\_workers=2)  
  
# 定义损失函数和优化器  
criterion = nn.CrossEntropyLoss()  
optimizer = optim.SGD(net.parameters(), lr=0.001, momentum=0.9)  
  
# 训练模型  
num\_epochs = 5  
for epoch in range(num\_epochs):  
 running\_loss = 0.0  
 for i, data in enumerate(trainloader, 0):  
 inputs, labels = data  
  
 optimizer.zero\_grad()  
  
 outputs = net(inputs)  
 loss = criterion(outputs, labels)  
 loss.backward()  
 optimizer.step()  
  
 running\_loss += loss.item()  
 print(f'Epoch {epoch + 1}, Loss: {running\_loss / (i + 1)}')  
  
# 评估模型  
correct = 0  
total = 0  
with torch.no\_grad():  
 for data in testloader:  
 images, labels = data  
 outputs = net(images)  
 \_, predicted = torch.max(outputs.data, 1)  
 total += labels.size(0)  
 correct += (predicted == labels).sum().item()  
 print(f'Accuracy: {100 \* correct / total}%')