**import** random  
  
**import** torch  
**import** torchvision  
**import** torchvision.models  
**from** torch.autograd **import** Variable  
**from** torchvision **import** transforms  
**from** matplotlib **import** pyplot **as** plt  
**from** tqdm **import** tqdm  
**from** torch **import** nn  
**from** torch.utils.data **import** DataLoader  
**from** torchvision.transforms **import** transforms, ToPILImage  
  
**from** PIL **import** ImageFile  
  
data\_transform = {  
 **"train"**: transforms.Compose([transforms.Resize((120,120)),  
 transforms.ToTensor(),  
 transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))]),  
 **"val"**: transforms.Compose([transforms.Resize((120, 120)), *# cannot 224, must (224, 224)* transforms.ToTensor(),  
 transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])}  
  
  
*# train\_data = torchvision.datasets.CIFAR10(root = "./data" , train = True ,download = True,  
# transform = trans)***class** alexnet(nn.Module):  
 **def** \_\_init\_\_(self):  
 super(alexnet, self).\_\_init\_\_()  
 self.model = nn.Sequential(  
  
 nn.Conv2d(3, 48, kernel\_size=11, stride=4, padding=2), *# input[3, 120, 120] output[48, 55, 55]* nn.ReLU(inplace=**True**),  
 nn.MaxPool2d(kernel\_size=3, stride=2), *# output[48, 27, 27]* nn.Conv2d(48, 128, kernel\_size=5, padding=2), *# output[128, 27, 27]* nn.ReLU(inplace=**True**),  
 nn.MaxPool2d(kernel\_size=3, stride=2), *# output[128, 13, 13]* nn.Conv2d(128, 192, kernel\_size=3, padding=1), *# output[192, 13, 13]* nn.ReLU(inplace=**True**),  
 nn.Conv2d(192, 192, kernel\_size=3, padding=1), *# output[192, 13, 13]* nn.ReLU(inplace=**True**),  
 nn.Conv2d(192, 128, kernel\_size=3, padding=1), *# output[128, 13, 13]* nn.ReLU(inplace=**True**),  
 nn.MaxPool2d(kernel\_size=3, stride=2), *# output[128, 6, 6]* nn.Flatten(),  
 nn.Dropout(p=0.5),  
 nn.Linear(512, 2048),  
 nn.ReLU(inplace=**True**),  
 nn.Dropout(p=0.5),  
 nn.Linear(2048, 1024),  
 nn.ReLU(inplace=**True**),  
 nn.Linear(1024, 5), *# 自己的数据是几种，这里的7就改成几* )  
  
 **def** forward(self, x):  
 x = self.model(x)  
 **return** x  
  
  
alexnet1 = alexnet()  
**def** main():  
 train\_data = torchvision.datasets.ImageFolder(root=**"DDR\\train"**, transform=data\_transform[**"train"**])  
  
 traindata = DataLoader(dataset=train\_data, batch\_size=64, shuffle=**True**, num\_workers=0)  
  
 *# test\_data = torchvision.datasets.CIFAR10(root = "./data" , train = False ,download = False,  
 # transform = trans)* test\_data = torchvision.datasets.ImageFolder(root=**"DDR\\valid"**, transform=data\_transform[**"val"**])  
  
 train\_size = len(train\_data)  
 test\_size = len(test\_data)  
 print(train\_size)  
 print(test\_size)  
 testdata = DataLoader(dataset=test\_data, batch\_size=64, shuffle=**True**, num\_workers=0)  
  
 device = torch.device(**"cuda:0" if** torch.cuda.is\_available() **else "cpu"**)  
 print(**"using {} device."**.format(device))  
  
 **class** alexnet(nn.Module):  
 **def** \_\_init\_\_(self):  
 super(alexnet, self).\_\_init\_\_()  
 self.model = nn.Sequential(  
  
 nn.Conv2d(3, 48, kernel\_size=11, stride=4, padding=2), *# input[3, 120, 120] output[48, 55, 55]* nn.ReLU(inplace=**True**),  
 nn.MaxPool2d(kernel\_size=3, stride=2), *# output[48, 27, 27]* nn.Conv2d(48, 128, kernel\_size=5, padding=2), *# output[128, 27, 27]* nn.ReLU(inplace=**True**),  
 nn.MaxPool2d(kernel\_size=3, stride=2), *# output[128, 13, 13]* nn.Conv2d(128, 192, kernel\_size=3, padding=1), *# output[192, 13, 13]* nn.ReLU(inplace=**True**),  
 nn.Conv2d(192, 192, kernel\_size=3, padding=1), *# output[192, 13, 13]* nn.ReLU(inplace=**True**),  
 nn.Conv2d(192, 128, kernel\_size=3, padding=1), *# output[128, 13, 13]* nn.ReLU(inplace=**True**),  
 nn.MaxPool2d(kernel\_size=3, stride=2), *# output[128, 6, 6]* nn.Flatten(),  
 nn.Dropout(p=0.5),  
 nn.Linear(512, 2048),  
 nn.ReLU(inplace=**True**),  
 nn.Dropout(p=0.5),  
 nn.Linear(2048, 1024),  
 nn.ReLU(inplace=**True**),  
 nn.Linear(1024, 5), *# 自己的数据是几种，这里的7就改成几* )  
  
 **def** forward(self, x):  
 x = self.model(x)  
 **return** x  
  
 alexnet1 = alexnet()  
 checkpoint = torch.load(**'alexnet.pth'**)  
 alexnet1.load\_state\_dict(checkpoint)  
 print(alexnet1)  
 alexnet1.to(device)  
 test1 = torch.ones(64, 3, 120, 120)  
  
 test1 = alexnet1(test1.to(device))  
 print(test1.shape)  
  
 epoch = 30  
 learning = 0.0001  
 optimizer = torch.optim.Adam(alexnet1.parameters(), lr=learning)  
 loss = nn.CrossEntropyLoss()  
  
 train\_loss\_all = []  
 train\_accur\_all = []  
 test\_loss\_all = []  
 test\_accur\_all = []  
 **for** i **in** range(epoch):  
 train\_loss = 0  
 train\_num = 0.0  
 train\_accuracy = 0.0  
 alexnet1.train()  
 train\_bar = tqdm(traindata)  
 **for** step, data **in** enumerate(train\_bar):  
 img, target = data  
 optimizer.zero\_grad()  
 outputs = alexnet1(img.to(device))  
  
 loss1 = loss(outputs, target.to(device))  
 outputs = torch.argmax(outputs, 1)  
 loss1.backward()  
 optimizer.step()  
 train\_loss += abs(loss1.item()) \* img.size(0)  
 accuracy = torch.sum(outputs == target.to(device))  
 train\_accuracy = train\_accuracy + accuracy  
 train\_num += img.size(0)  
  
 print(**"epoch：{} ， train-Loss：{} , train-accuracy：{}"**.format(i + 1, train\_loss / train\_num,  
 train\_accuracy / train\_num))  
 train\_loss\_all.append(train\_loss / train\_num)  
 train\_accur\_all.append(train\_accuracy.double().item() / train\_num)  
 test\_loss = 0  
 test\_accuracy = 0.0  
 test\_num = 0  
 alexnet1.eval()  
 **with** torch.no\_grad():  
 test\_bar = tqdm(testdata)  
 **for** data **in** test\_bar:  
 img, target = data  
  
 outputs = alexnet1(img.to(device))  
  
 loss2 = loss(outputs, target.to(device))  
 outputs = torch.argmax(outputs, 1)  
 test\_loss = test\_loss + abs(loss2.item()) \* img.size(0)  
 accuracy = torch.sum(outputs == target.to(device))  
 test\_accuracy = test\_accuracy + accuracy  
 test\_num += img.size(0)  
  
 print(**"test-Loss：{} , test-accuracy：{}"**.format(test\_loss / test\_num, test\_accuracy / test\_num))  
 test\_loss\_all.append(test\_loss / test\_num)  
 test\_accur\_all.append(test\_accuracy.double().item() / test\_num)  
 plt.figure(figsize=(12, 4))  
 plt.subplot(1, 2, 1)  
 plt.plot(range(epoch), train\_loss\_all,  
 **"ro-"**, label=**"Train loss"**)  
 plt.plot(range(epoch), test\_loss\_all,  
 **"bs-"**, label=**"test loss"**)  
 plt.legend()  
 plt.xlabel(**"epoch"**)  
 plt.ylabel(**"Loss"**)  
 plt.subplot(1, 2, 2)  
 plt.plot(range(epoch), train\_accur\_all,  
 **"ro-"**, label=**"Train accur"**)  
 plt.plot(range(epoch), test\_accur\_all,  
 **"bs-"**, label=**"test accur"**)  
 plt.xlabel(**"epoch"**)  
 plt.ylabel(**"acc"**)  
 plt.legend()  
 plt.show()  
  
 torch.save(alexnet1.state\_dict(), **"alexnet.pth"**)  
  
 print(**"模型已保存"**)  
  
  
  
**if** \_\_name\_\_ == **'\_\_main\_\_'**:  
 main()

1

Epoch 1 Train Loss 1.5938268249685115 Train Accuracy 0.28789198606271776 Teat Loss 1.5533986250559488 Test Accuracy 0.5012522163120567