# author: baiCai

import os

from PIL import Image

import numpy as np

import torch

from torch import nn

from torch.nn.functional import interpolate

from torch.utils.data import DataLoader,Dataset

from torch import optim

from torchvision import transforms

from torchvision.models import vgg16

# VGG16：自己的模型

class My\_VGG16(nn.Module):

def \_\_init\_\_(self,num\_classes=5,init\_weight=True):

super(My\_VGG16, self).\_\_init\_\_()

# 特征提取层

self.features = nn.Sequential(

nn.Conv2d(in\_channels=3,out\_channels=64,kernel\_size=3,stride=1,padding=1),

nn.Conv2d(in\_channels=64,out\_channels=64,kernel\_size=3,stride=1,padding=1),

nn.MaxPool2d(kernel\_size=2,stride=2),

nn.Conv2d(in\_channels=64, out\_channels=128, kernel\_size=3, stride=1, padding=1),

nn.Conv2d(in\_channels=128, out\_channels=128, kernel\_size=3, stride=1, padding=1),

nn.MaxPool2d(kernel\_size=2, stride=2),

nn.Conv2d(in\_channels=128, out\_channels=256, kernel\_size=3, stride=1, padding=1),

nn.Conv2d(in\_channels=256, out\_channels=256, kernel\_size=3, stride=1, padding=1),

nn.Conv2d(in\_channels=256, out\_channels=256, kernel\_size=3, stride=1, padding=1),

nn.MaxPool2d(kernel\_size=2,stride=2),

nn.Conv2d(in\_channels=256, out\_channels=512, kernel\_size=3, stride=1, padding=1),

nn.Conv2d(in\_channels=512, out\_channels=512, kernel\_size=3, stride=1, padding=1),

nn.Conv2d(in\_channels=512, out\_channels=512, kernel\_size=3, stride=1, padding=1),

nn.MaxPool2d(kernel\_size=2, stride=2),

nn.Conv2d(in\_channels=512, out\_channels=512, kernel\_size=3, stride=1, padding=1),

nn.Conv2d(in\_channels=512, out\_channels=512, kernel\_size=3, stride=1, padding=1),

nn.Conv2d(in\_channels=512, out\_channels=512, kernel\_size=3, stride=1, padding=1),

nn.MaxPool2d(kernel\_size=2, stride=2),

)

# 分类层

self.classifier = nn.Sequential(

nn.Linear(in\_features=7\*7\*512,out\_features=4096),

nn.ReLU(),

nn.Dropout(0.5),

nn.Linear(in\_features=4096,out\_features=4096),

nn.ReLU(),

nn.Dropout(0.5),

nn.Linear(in\_features=4096,out\_features=num\_classes)

)

# 参数初始化

if init\_weight: # 如果进行参数初始化

for m in self.modules(): # 对于模型的每一层

if isinstance(m, nn.Conv2d): # 如果是卷积层

# 使用kaiming初始化

nn.init.kaiming\_normal\_(m.weight, mode="fan\_out", nonlinearity="relu")

# 如果bias不为空，固定为0

if m.bias is not None:

nn.init.constant\_(m.bias, 0)

elif isinstance(m, nn.Linear):# 如果是线性层

# 正态初始化

nn.init.normal\_(m.weight, 0, 0.01)

# bias则固定为0

nn.init.constant\_(m.bias, 0)

def forward(self,x):

x = self.features(x)

x = torch.flatten(x,1)

result = self.classifier(x)

return result

# 模型输入：224\*224\*3

# 训练集数据加载

class My\_Dataset(Dataset):

def \_\_init\_\_(self,filename,transform=None):

self.filename = filename # 文件路径

self.transform = transform # 是否对图片进行变化

self.image\_name,self.label\_image = self.operate\_file()

def \_\_len\_\_(self):

return len(self.image\_name)

def \_\_getitem\_\_(self,idx):

# 由路径打开图片

image = Image.open(self.image\_name[idx])

# 下采样： 因为图片大小不同，需要下采样为224\*224

trans = transforms.RandomResizedCrop(224)

image = trans(image)

# 获取标签值

label = self.label\_image[idx]

# 是否需要处理

if self.transform:

image = self.transform(image)

# image = image.reshape(1,image.size(0),image.size(1),image.size(2))

# print('变换前',image.size())

# image = interpolate(image, size=(227, 227))

# image = image.reshape(image.size(1),image.size(2),image.size(3))

# print('变换后', image.size())

# 转为tensor对象

label = torch.from\_numpy(np.array(label))

return image,label

def operate\_file(self):

# 获取所有的文件夹路径 '../data/net\_train\_images'的文件夹

dir\_list = os.listdir(self.filename)

# 拼凑出图片完整路径 '../data/net\_train\_images' + '/' + 'xxx.jpg'

full\_path = [self.filename+'/'+name for name in dir\_list]

# 获取里面的图片名字

name\_list = []

for i,v in enumerate(full\_path):

temp = os.listdir(v)

temp\_list = [v+'/'+j for j in temp]

name\_list.extend(temp\_list)

# 由于一个文件夹的所有标签都是同一个值，而字符值必须转为数字值，因此我们使用数字0-4代替标签值

label\_list = []

temp\_list = np.array([0,1,2,3,4],dtype=np.int64) # 用数字代表不同类别

# 将标签每个复制200个

for j in range(5):

for i in range(200):

label\_list.append(temp\_list[j])

return name\_list,label\_list

# 测试集数据加载器

class My\_Dataset\_test(My\_Dataset):

def operate\_file(self):

# 获取所有的文件夹路径

dir\_list = os.listdir(self.filename)

full\_path = [self.filename+'/'+name for name in dir\_list]

# 获取里面的图片名字

name\_list = []

for i,v in enumerate(full\_path):

temp = os.listdir(v)

temp\_list = [v+'/'+j for j in temp]

name\_list.extend(temp\_list)

# 将标签每个复制一百个

label\_list = []

temp\_list = np.array([0,1,2,3,4],dtype=np.int64) # 用数字代表不同类别

for j in range(5):

for i in range(100): # 只修改了这里

label\_list.append(temp\_list[j])

return name\_list,label\_list

# 调整学习率

loss\_save = []

flag = 0

lr = 0.0002

def adjust\_lr(loss):

global flag,lr

loss\_save.append(loss)

if len(loss\_save) >= 2:

# 如果已经训练了2次，可以判断是否收敛或波动

if abs(loss\_save[-1] - loss\_save[-2]) <= 0.0005:

# 如果变化范围小于0.0005，说明可能收敛了

flag += 1

if loss\_save[-1] - loss\_save[-2] >= 0:

# 如果损失值增加，也记一次

flag += 1

if flag >= 3:

# 如果出现3次这样的情况，需要调整学习率

lr /= 10

print('学习率已改变，变为了%s' % (lr))

# 并将flag清为0

flag = 0

# 加载预训练模型

def load\_pretrained():

path = 'F:/官方\_预训练模型/vgg16-397923af.pth' # 需要改为自己的路径

model = vgg16()

model.load\_state\_dict(torch.load(path))

return model

# 训练过程

def train():

batch\_size = 10 # 批量训练大小

model = My\_VGG16() # 创建模型

# 加载预训练vgg

# model = load\_pretrained()

# 定义优化器

optimizer = optim.SGD(params=model.parameters(), lr=lr)

# 将模型放入GPU中

device = torch.device("cuda:0" if torch.cuda.is\_available() else "cpu")

model.to(device)

# 定义损失函数

loss\_func = nn.CrossEntropyLoss()

# 加载数据

train\_set = My\_Dataset('../data/net\_train\_images',transform=transforms.ToTensor())

train\_loader = DataLoader(train\_set, batch\_size, shuffle=True)

# 训练20次

for i in range(20):

loss\_temp = 0 # 临时变量

for j,(batch\_data,batch\_label) in enumerate(train\_loader):

# 数据放入GPU中

batch\_data,batch\_label = batch\_data.cuda(),batch\_label.cuda()

# 梯度清零

optimizer.zero\_grad()

# 模型训练

prediction = model(batch\_data)

# 损失值

loss = loss\_func(prediction,batch\_label)

loss\_temp += loss.item()

# 反向传播

loss.backward()

# 梯度更新

optimizer.step()

# 每25个批次打印一次损失值

print('[%d] loss: %.4f' % (i+1,loss\_temp/len(train\_loader)))

# 是否调整学习率，如果调整的话，需要把优化器也移动到循环内部

# adjust\_lr(loss\_temp/len(train\_loader))

# torch.save(model,'VGG16.pkl')

test(model)

def test(model):

# 批量数目

batch\_size = 10

# 预测正确个数

correct = 0

# 加载数据

test\_set = My\_Dataset\_test('../data/net\_test\_images', transform=transforms.ToTensor())

test\_loader = DataLoader(test\_set, batch\_size, shuffle=False)

# 开始

for batch\_data,batch\_label in test\_loader:

# 放入GPU中

batch\_data, batch\_label = batch\_data.cuda(), batch\_label.cuda()

# 预测

prediction = model(batch\_data)

# 将预测值中最大的索引取出，其对应了不同类别值

predicted = torch.max(prediction.data, 1)[1]

# 获取准确个数

correct += (predicted == batch\_label).sum()

print('准确率: %.2f %%' % (100 \* correct / 500)) # 因为总共500个测试数据

if \_\_name\_\_ == '\_\_main\_\_':

train()