**实验4 迁移resnet50猫狗图像分类**

**一、实验目的**

1. 掌握迁移学习方法快速训练resnet50的分类模型；

2. 了解使用pytorch对resnet50进行部分参数进行微调的方法。

**二、实验内容**

1. 通过代码自动下载模型并直接调用；

2. 对当前迁移过来的模型进行全连接层的调整；

3. 模型训练及测试结果。

**三、实验指导**

**1.通过代码自动下载模型并直接调用**

首先需要下载已经具备最优参数的模型，这需要对我们之前使用的 model = Models()代码部分进行替换，此时不需要自己搭建和定义训练模型了，而是通过代码自动下载模型并直接调用（此时干脆把前面对数据集进行的处理再重新做一遍）：

import torch

import torchvision

from torchvision import datasets, models, transforms

import os

from torch.autograd import Variable

import matplotlib.pyplot as plt

import time

model\_path = 'transferResNet50/model\_name.pth'

model\_params\_path = 'transferResNet50/params\_name.pth'

transform = transforms.Compose(

[transforms.CenterCrop(224),

transforms.ToTensor(),

transforms.Normalize([0.5,0.5,0.5],[0.5,0.5,0.5])

]

)

data\_dir = "C:/Users/xinyu/Desktop/data/DogsVSCats/"

data\_transform = {

x:transforms.Compose(

[

transforms.Scale([224,224]), #Scale类将原始图片的大小统一缩放至64×64

transforms.ToTensor(),

transforms.Normalize(

mean=[0.5,0.5,0.5],

std=[0.5,0.5,0.5]

)

]

)

for x in ["train","valid"]

}

image\_datasets = {

x:datasets.ImageFolder(

root=os.path.join(data\_dir,x), #将输入参数中的两个名字拼接成一个完整的文件路径

transform=data\_transform[x]

)

for x in ["train","valid"]

}

dataloader = {

#注意：标签0/1自动根据子目录顺序以及目录名生成

#如：{'cat': 0, 'dog': 1} #{'狗dog': 0, '猫cat': 1}

#如：['cat', 'dog'] #['狗dog', '猫cat']

x:torch.utils.data.DataLoader(

dataset=image\_datasets[x],

batch\_size=16,

shuffle=True

)

for x in ["train","valid"]

}

X\_example, y\_example = next(iter(dataloader["train"]))

example\_classes = image\_datasets["train"].classes #['cat', 'dog'] #['狗dog', '猫cat']

index\_classes = image\_datasets["train"].class\_to\_idx #{'cat': 0, 'dog': 1} #{'狗dog': 0, '猫cat': 1}

Use\_gpu = torch.cuda.is\_available()

model = models.resnet50(pretrained=True)

print(model)

**2. 对当前迁移过来的模型进行全连接层的调整**

尽管迁移学习要求我们需要解决的问题之间最好具有很强的相似性，但是每个问题对最后输出的结果会有不一样的要求，而承担整个模型输出分类工作的是卷积神经网络模型中的全连接层，所以在迁移学习的过程中调整最多的也是全连接层部分。

其基本思路是冻结卷积神经网络中全连接层之前的全部网络层次，让这些被冻结的网络层次中的参数在模型的训练过程中不进行梯度更新，能够被优化的参数仅仅是没有被冻结的全连接层的全部参数。

首先，迁移过来的 resnet50 架构模型在最后输出的结果是 1000 个，在我们的问题中只需两个输出结果，所以全连接层必须进行调整：

for param in model.parameters():

param.requires\_grad = False

'''重写model的classifier属性，重新设计分类器的结构'''

model.fc = torch.nn.Linear(2048,2)

print(model)

**3.模型训练及结果**

if Use\_gpu:

model = model.cuda()

loss\_f = torch.nn.CrossEntropyLoss()

optimizer = torch.optim.Adam(model.fc.parameters(),lr=0.00001)

has\_been\_trained = os.path.isfile(model\_path)

if has\_been\_trained:

epoch\_n = 0

else:

epoch\_n = 1

time\_open = time.time()

for epoch in range(epoch\_n):

print("Epoch {}/{}".format(epoch,epoch\_n -1))

print("-"\*10)

for phase in ["train","valid"]:

if phase == "train":

print("Training...")

model.train(True) #model.train(),启用 BatchNormalization 和 Dropout

else:

print("Validing...")

model.train(False) #model.eval(),不启用 BatchNormalization 和 Dropout

running\_loss = 0.0

running\_corrects = 0

#cxq = 1

for batch, data in enumerate(dataloader[phase],1):

X, y = data

#print("$$$$$$",cxq)

#cxq+=1

if Use\_gpu:

X, y = Variable(X.cuda()), Variable(y.cuda())

else:

X, y = Variable(X), Variable(y)

y\_pred = model(X)

\_, pred = torch.max(y\_pred.data,1)

optimizer.zero\_grad()

loss = loss\_f(y\_pred,y)

if phase == "train":

loss.backward()

optimizer.step()

running\_loss += loss.item()

running\_corrects += torch.sum(pred == y.data)

if batch%500 == 0 and phase == "train":

print("Batch {}, Train Loss:{:.4f},Train ACC:{:.4f}%".format(

batch, running\_loss/batch, 100.0\*running\_corrects/(16\*batch)

)

)

epoch\_loss = running\_loss\*16/len(image\_datasets[phase])

epoch\_acc = 100.0 \* running\_corrects/len(image\_datasets[phase])

print("{} Loss:{:.4f} Acc:{:.4f}%".format(phase,epoch\_loss,epoch\_acc))

time\_end = time.time() - time\_open

print("程序运行时间:{}分钟...".format(int(time\_end/60)))

**4.举例说明**

X\_example, Y\_example = next(iter(dataloader['train']))

#print('X\_example个数{}'.format(len(X\_example))) #X\_example个数16 torch.Size([16, 3, 64, 64])

#print('Y\_example个数{}'.format(len(Y\_example))) #Y\_example个数16 torch.Size([16]

#X, y = data #torch.Size([16, 3, 64, 64]) torch.Size([16]

if Use\_gpu:

X\_example, Y\_example = Variable(X\_example.cuda()), Variable(Y\_example.cuda())

else:

X\_example, Y\_example = Variable(X\_example), Variable(Y\_example)

y\_pred = model(X\_example)

index\_classes = image\_datasets['train'].class\_to\_idx # 显示类别对应的独热编码

#print(index\_classes) #{'cat': 0, 'dog': 1}

example\_classes = image\_datasets['train'].classes # 将原始图像的类别保存起来

#print(example\_classes) #['cat', 'dog']

img = torchvision.utils.make\_grid(X\_example)

img = img.cpu().numpy().transpose([1,2,0])

print("实际:",[example\_classes[i] for i in Y\_example])

#['cat', 'cat', 'cat', 'cat', 'dog', 'cat', 'cat', 'dog', 'cat', 'cat', 'dog', 'dog', 'cat', 'dog', 'dog', 'cat']

\_, y\_pred = torch.max(y\_pred,1)

print("预测:",[example\_classes[i] for i in y\_pred])

plt.imshow(img)

plt.show()

Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

实际: ['cat', 'dog', 'dog', 'cat', 'cat', 'dog', 'cat', 'cat', 'dog', 'dog', 'dog', 'cat', 'cat', 'dog', 'cat', 'dog']

预测: ['cat', 'dog', 'dog', 'cat', 'cat', 'dog', 'cat', 'cat', 'dog', 'dog', 'dog', 'cat', 'cat', 'dog', 'cat', 'dog']