**1、load\_data.py**

import pandas as pd #数据分析包

import numpy as np #支持高维数组和矩阵运算

import os #用于实现操作系统的多种功能

from PIL import Image #图像处理库

from torch.utils.data import Dataset #Dataset代表自定义数据集方法的抽象类，可以定义自己的数据类继承这个抽象类

#只需要定义\_\_len\_\_和\_\_getitem\_\_这两个方法就可以

# 数据集类,这个是对数据集进行处理，image\_path是数据集路径，transforms对数据进行预处理，数据增强

#phrase：训练集，验证集，测试集

class MyDataSet(Dataset):

def \_\_init\_\_(self, image\_path, csv\_path, transforms, phrase):

# 读取csv#header指定第几行作为列名(忽略注解行)若没有指定列名，默认header=0; 若指定了列名header=None

csv = pd.read\_csv(csv\_path,header=None)

# 读取第一列,组合成完整的图片地址

self.imgs = [image\_path+str(k)+".jpg" for k in csv[0].values]

self.phrase = phrase

if self.phrase != "test":

self.labels = np.asarray([k for k in csv[1].values]) #对元数据进行了复制

self.transforms = transforms

##返回与所给键对应的值

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

img\_path = self.imgs[index]

pil\_img = Image.open(img\_path).convert("RGB") ##将图片格式转换为RGB格式，一般模型只识别该模式

if self.transforms:

data = self.transforms(pil\_img)

else:

pil\_img = np.asarray(pil\_img) #将PIL image转换为array

data = torch.from\_numpy(pil\_img)

if self.phrase != "test":

label = self.labels[index]

sample = (data, label)

else:

sample = data

return sample

##返回集合中所含项目的数量

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

return len(self.imgs)

**2、model.py**

from torchvision import models

from torch import nn

import sys

import torch

import torchvision.models as models

# 冻结最初几层网络 训练过程中权重不会更新

def set\_parameter\_requires\_grad(model, feature\_extracting): #feature\_extracting特征提取

if feature\_extracting: #迭代打印model.parameters()将会打印每一次迭代元素的param而不是打印名字

for i, para in enumerate(model.parameters()):#enumerate()函数将list变为类似于[(0,"Lisa"),(1,"Bob"),(2,"Tom")]

if i < 50:

para.requires\_grad = False #冻结参数 不要求梯度 不计算不需要更新的参数的梯度，节省时间

else:

para.requires\_grad = True #更新参数 要求梯度

# 模型初始化，修改输出神经网络，

def initialize\_model(model\_name, num\_classes, feature\_extract, use\_pretrained=True):

model\_ft = models.resnet34(pretrained=False) # 使用预训练模型pretrained默认是false ##ft（feature transform）

model\_ft.load\_state\_dict(torch.load('/model/Polly/acm/resnet34-333f7ec4.pth')) #加载预训练模型（加载本地模型）

set\_parameter\_requires\_grad(model\_ft, feature\_extract)

num\_ftrs = model\_ft.fc.in\_features #提取fc层中固定的参数

model\_ft.fc = nn.Sequential(nn.Linear(num\_ftrs, num\_classes)) # 修改类别，分类num\_classes=5，肺炎有5类 fc（fully connected layer全连接层）

#输入图片大小

input\_size = 224

return model\_ft, input\_size

**3、train.py**

# 训练

import copy

import time

import torch

import matplotlib.pyplot as plt

# 这里参考pytorch官方训练文档，输入训练集训练 #criterion标准，optimizer 优化器

def train\_model(model, device, dataloaders, criterion, optimizer, num\_epochs, is\_inception=False):

since = time.time()

val\_acc\_history = []

train\_acc\_history = []

val\_loss\_history = []

train\_loss\_history = []

best\_model\_wts = copy.deepcopy(model.state\_dict())

best\_acc = 0.0

for epoch in range(num\_epochs):

print('Epoch {}/{}'.format(epoch, num\_epochs - 1))

print('-' \* 10)

# Each epoch has a training and validation phase

for phase in ['train', 'val']:

if phase == 'train':

model.train() # Set model to training mode##设置模型为训练模式

else:

model.eval() # Set model to evaluate mode##设置模型为评估模式

running\_loss = 0.0

running\_corrects = 0

# Iterate over data.

for inputs, labels in dataloaders[phase]:

inputs = inputs.to(device)

labels = labels.to(device)

# zero the parameter gradients

optimizer.zero\_grad()

# forward

# track history if only in train

with torch.set\_grad\_enabled(phase == 'train'):

outputs = model(inputs)

loss = criterion(outputs, labels)

\_, preds = torch.max(outputs, 1)

# backward + optimize only if in training phase

if phase == 'train':

loss.backward()

optimizer.step()

# statistics

running\_loss += loss.item() \* inputs.size(0)

running\_corrects += torch.sum(preds == labels.data)

epoch\_loss = running\_loss / len(dataloaders[phase].dataset)

epoch\_acc = running\_corrects.double() / len(dataloaders[phase].dataset)

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

# deep copy the model

if phase == 'val' and epoch\_acc > best\_acc:

best\_acc = epoch\_acc

best\_model\_wts = copy.deepcopy(model.state\_dict())

if phase == 'val':

val\_acc\_history.append(epoch\_acc)

val\_loss\_history.append(epoch\_loss)

if phase == "train":

train\_acc\_history.append(epoch\_acc)

train\_loss\_history.append(epoch\_loss)

print()

his = [train\_acc\_history, val\_acc\_history, train\_loss\_history, val\_loss\_history]

time\_elapsed = time.time() - since

print('Training complete in {:.0f}m {:.0f}s'.format(time\_elapsed // 60, time\_elapsed % 60))

print('Best val Acc: {:4f}'.format(best\_acc))

# load best model weights

model.load\_state\_dict(best\_model\_wts)

torch.save(model, "model\_{}.pkl".format(best\_acc))

return model, his

**4、train\_model.py**

from torchvision import transforms

from train import train\_model #train\_model是train.py中的一个类

from torch import nn,optim

from torch.utils.data import DataLoader,SubsetRandomSampler

from load\_data import MyDataSet #MyDataSet是load\_data.py中的一个类

from model import initialize\_model #initialize\_model是model中的一个类

import numpy as np

import torch

# 数据增强

data\_transforms = {

"train":

transforms.Compose([ #transforms是常用的数据变换方式，Compose将各个变换串联起来

transforms.RandomResizedCrop(size=256, scale=(0.8, 1.0)),#随机长宽比裁剪 size是输出的分辨率

transforms.CenterCrop(size=224), #从中心裁剪图像，大小为224\*224

transforms.RandomHorizontalFlip(),# 随机水平翻转给定的PIL.Image，概率为0.5，即一半的概率翻转，一半的概率不翻转

transforms.ColorJitter(brightness=0.2, contrast=0.2),#随机改变图片的亮度、对比度

transforms.RandomRotation(15),#按角度旋转对象

transforms.ToTensor(),#将PIL图片或者numpy.ndarray转成Tensor类型的，并且归一化至[0-1]，tensor可以在GPU上运行，numpy仅cpu

transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])#用均值和标准差对张量图像进行归一化

# 给定均值(R,G,B)，方差(R,G,B)，将会把tensor正则化，即 Normalized\_image = (image-mean)/std

]),

"val":

transforms.Compose([

transforms.RandomResizedCrop(size=256, scale=(0.8, 1.0)),

transforms.CenterCrop(size=224),

transforms.ToTensor(),

transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])

]),

"test":

transforms.Compose([

transforms.RandomResizedCrop(size=256, scale=(0.8, 1.0)),

transforms.CenterCrop(size=224),

transforms.ToTensor(),

transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])

]),

}

#读取数据集,划分数据集

dataset = MyDataSet("/data/Polly/ready\_for\_train/train/0/", "train.csv", data\_transforms["train"], "train")

validation\_split = .2 # 验证集比例

shuffle\_dataset = True #是否打乱数据集

random\_seed = 42 #随机生成种子数

batch\_size = 128 #一次训练序所选取的样本数

dataset\_size = len(dataset)

indices = list(range(dataset\_size)) #构造索引

split = int(np.floor(validation\_split \* dataset\_size)) #floor()函数向下取整

if shuffle\_dataset:

np.random.seed(random\_seed) #设置随机seed，使每次生成的随机数都一样

np.random.shuffle(indices) #生成随机列表

train\_indices, val\_indices = indices[split:], indices[:split] #将数据集划分成了训练集和验证集

# 创建sampler 对数据集采样 ，这里可以无视

train\_sampler = SubsetRandomSampler(train\_indices)

valid\_sampler = SubsetRandomSampler(val\_indices)

# 构建训练集和验证集

train\_loader = DataLoader(dataset, batch\_size=batch\_size, shuffle=shuffle\_dataset, num\_workers=4)#num\_workers通过多个进程来导入数据

val\_loader = DataLoader(dataset, batch\_size=batch\_size, shuffle=shuffle\_dataset, num\_workers=4)#加快数据的导入

data\_loader1 = {'train': train\_loader, 'val': val\_loader}

# 同样构建数据集验证集，两种方式，参考这个

image\_datasets = {x: MyDataSet("/data/Polly/ready\_for\_train/train/0/","./train.csv",

data\_transforms[x],x)

for x in ['train', 'val']}

dataloaders = {x: DataLoader(image\_datasets[x], batch\_size=128,

shuffle=True, num\_workers=4)

for x in ['train', 'val']}

# 建立模型

# resnet34,imagenet预训练模型迁移学习

num\_classes = 5 #分5类

model\_name = "resnet34"

feature\_extract = True #仅更新最后一层参数，其余参数保持固定

#初始化模型

model, input\_size = initialize\_model(model\_name=model\_name, num\_classes=num\_classes, feature\_extract=feature\_extract, use\_pretrained=True)

# 20个epoch，也就是训练20轮

epoch = 20

# 使用GPU训练

device = torch.device("cuda:0" if torch.cuda.is\_available() else "cpu") #定义device，cuda:0表示起始的device\_id为0

model = model.to(device) #将模型加载到指定设备上

# 损失函数交叉熵

criterion = nn.CrossEntropyLoss() #criterion（标准）

# 优化器adam，学习率0.001

optimizer = optim.Adam(model.parameters(), lr=0.001,) #optimizer（优化器）

# 训练

model,his=train\_model(model, device, dataloaders, criterion, optimizer, num\_epochs=epoch, is\_inception=False) #is\_inception 标志用于容纳 Inception v3 模型

**5、predict.py**

## 建立提交csv

import torch

from load\_data import MyDataSet

from torch.utils.data import DataLoader

from torchvision import transforms

from train import train\_model #貌似没用上

from torch import nn,optim

from load\_data import MyDataSet

from model import initialize\_model

import numpy as np

# 数据增强

data\_transforms = {

"test":

transforms.Compose([

transforms.RandomResizedCrop(size=256, scale=(0.8, 1.0)),

transforms.CenterCrop(size=224),

transforms.ToTensor(),

transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])

]),

}

model = torch.load("/model/Polly/predict/model\_0.846150002498376.pkl")

# 读取测试集

dataset = MyDataSet("/data/Polly/ready\_for\_train/test/", "/output/predict.csv", data\_transforms["test"], "test")

testdata = DataLoader(dataset, batch\_size=32, shuffle=False, num\_workers=4)

print(dataset.\_\_len\_\_())

# 预测测试集

import pandas as pd

model.eval() #dropout层和BN层 dropout是神经网络中防止模型过拟合化重要的正则化方式。BN（batch normallize）批规范化，正则化的一个重要手段

predicted\_labels\_list = []

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

for batch\_id, (features) in enumerate(testdata):

features = features.to(device)

probas = model(features)

\_, predicted\_labels = torch.max(probas, 1)

for label in predicted\_labels:

predicted\_labels\_list.append(label.cpu().item())

dict = {'label': predicted\_labels\_list}

df = pd.DataFrame(dict)

df.to\_csv('/output/predict.csv',header=False,index=True)