**一、如何食用（重要！）**

1. 请新建一个**yangben**的目录 → 将数据集16个目录放在其中 → 将yangben目录与提交的相关代码、文件放在同一目录下，如下图所示：



2. 先运行build\_input\_index\_file.py：**python build\_input\_index\_file.py**，其会生成PyTorch加载数据集时所需的信息train.txt、test.txt → 再运行Plant\_VGG.py即可：**python Plant\_VGG.py**，运行结束后，各类别准确率、总准确率会显示在终端上。

3. 代码自动加载训练好的Plant\_vgg\_12itr.pt模型并开始测试集测试，大约mins便能得出结果；删除Plant\_vgg\_12itr.pt，代码将重新训练生成Plant\_vgg\_12itr.pt模型，然后得出测试结果，在Intel Core i7-4700MQ的CPU下，大约要运行5个半小时。



**二、运行环境**

操作系统：Win10

程序语言：Anaconda - Python3.7

框架：PyTorch

CNN模型：VGG11

**三、实验步骤&代码**

**！！！代码注释已阐明了实验步骤&内容！！！**

1. 安装Anaconda、PyTorch：

去官网即可下载Anaconda3。

执行如下2命令即可安装带CUDA9.0的PyTorch0.4.1：

conda install pytorch -c pytorch pip3 install torchvision

2. 写build\_input\_index\_file.py：

build\_input\_index\_file.py中，我们实现了**1.随机划分数据集为80%训练集、20%测试集。2.数据乱序。**（PyTorch根据train.txt、test.txt依次读取数据）

*# split the dataset into 0.8 & 0.2 randomly*

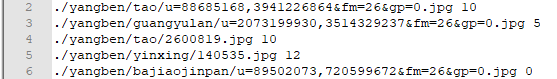
random.shuffle(instances)

spoint = int(len(instances) \* 0.8)

*# shuffling dataset is necessary*

random.shuffle(trainset)

random.shuffle(testset)



3. 写Plant\_VGG.py中的数据预处理：

将图片统一处理，变成符合网络输入的形式：**去均值、缩小为64\*64、转为RGB格式**。

*# 1.Load and normalizing the training and test dataset*

transform = transforms.Compose([

transforms.Resize((64, 64), interpolation=2),

transforms.ToTensor(),

transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))

])

4. 写Plant\_VGG.py中的网络结构：

使用流行的VGG模型训练并测试，选择VGG11而非更深层但更好的VGG模型。更深的VGG需要更多的训练数据、更多的训练轮数。由于我们的**训练集少，使用更浅层的VGG11便能得到较合适的模型**。网络结构为：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 各层 | 输入特征通道 | 输出特征通道=卷积核数 | 卷积核尺寸 | 图像尺寸 |
| Conv1 | 3 | 64 | 3\*3 | 64\*64 |
| MaxPool | - | - | 2\*2 | 32\*32 |
| Conv2 | 64 | 128 | 3\*3 | 32\*32 |
| MaxPool | - | - | 2\*2 | 16\*16 |
| Conv3 | 128 | 256 | 3\*3 | 16\*16 |
| Conv4 | 256 | 256 | 3\*3 | 16\*16 |
| MaxPool | - | - | 2\*2 | 8\*8 |
| Conv5 | 256 | 512 | 3\*3 | 8\*8 |
| Conv6 | 512 | 512 | 3\*3 | 8\*8 |
| MaxPool | - | - | 2\*2 | 4\*4 |
| Conv7 | 512 | 512 | 3\*3 | 4\*4 |
| Conv8 | 512 | 512 | 3\*3 | 4\*4 |
| MaxPool | - | - | 2\*2 | 2\*2 |
| Resize | 512\*2\*2 | 1024\*1 |  | 1024\*1 |
| Fc | 1024 | 16 | 1\*1 | 16\*1 |

*# 2.Define a Convolutional Neural Network*

'VGG11':[64, 'M', 128, 'M', 256, 256, 'M', 512, 512, 'M', 512, 512, 'M']

for x in cfg:

if x == 'M':

layers += [nn.MaxPool2d(kernel\_size=2, stride=2)]

else:

layers += [nn.Conv2d(in\_channels, x, kernel\_size=3, padding=1),

nn.BatchNorm2d(x), *# BatchNorm2d() is helpful*

nn.ReLU(inplace=True)]

in\_channels = x

return nn.Sequential(\*layers)

5. 写Plant\_VGG.py中的VGG模型训练：

**前向传播、后向传播、更新权重**

*# 3.Define a loss function*

criterion = nn.CrossEntropyLoss()

optimizer = optim.SGD(net.parameters(), lr=0.0001, momentum=0.9)

*# 4.Train the network on the training data*

optimizer.zero\_grad()

*# forward + backward + optimize*

outputs = net(inputs)

loss = criterion(outputs, labels)

loss.backward()

optimizer.step()

6. 写Plant\_VGG.py中的VGG模型预测，得出准确率：

outputs = net(images.to(device))

\_, predicted = torch.max(outputs.data, 1)

c = (predicted == labels).squeeze().numpy()

*# calculate every batch*

for i in range(1):

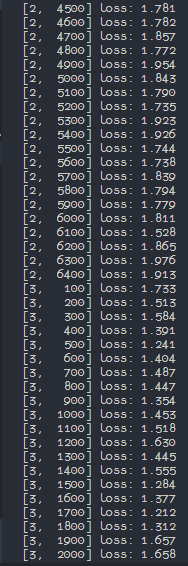
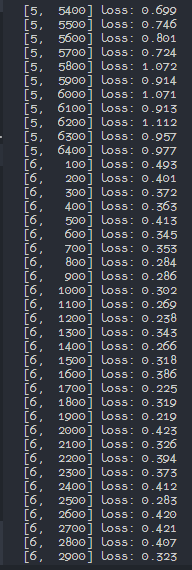
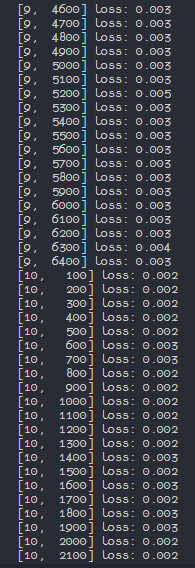
label = labels[i]

class\_correct[label] += c[i]

class\_total[label] += 1

**四、实验结果**

迭代12轮训练VGG。可以看出loss在逐渐降低，VGG模型在逐渐学习。12轮后达到一个较稳定状态。



12轮后各类别的准确率、总准确率为：



**祝每天开心、幸福！**