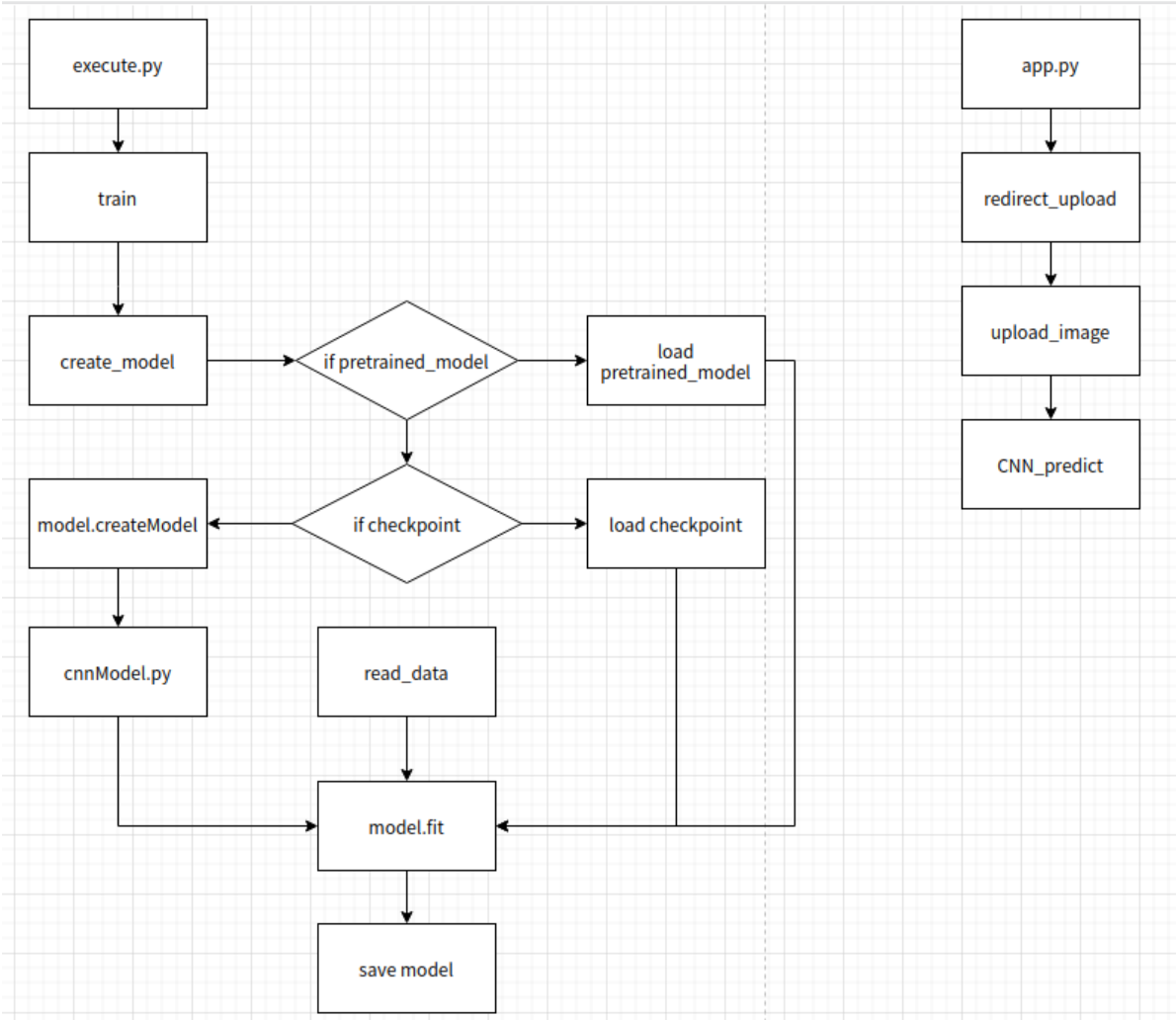
有关werkzeug的bug https://blog.csdn.net/ispeasant/article/details/105627677

项目结构



从train函数开始

create_model 是创建一个模型

先看下面那句，model.summary

api上的描述是这样的 Prints a string summary of the network.

效果如下，就是显示出神经网络的结构

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1 (Conv2D)               (None, 32, 32, 32)        896
_________________________________________________________________
pool1 (MaxPooling2D)         (None, 32, 32, 32)        0
_________________________________________________________________
batch_normalization (BatchNo (None, 32, 32, 32)        128
_________________________________________________________________
conv2 (Conv2D)               (None, 32, 32, 64)        18496
_________________________________________________________________
pool2 (MaxPooling2D)         (None, 32, 32, 64)        0
_________________________________________________________________
batch_normalization_1 (Batch (None, 32, 32, 64)        256
_________________________________________________________________
conv3 (Conv2D)               (None, 32, 32, 128)       73856
_________________________________________________________________
pool3 (MaxPooling2D)         (None, 32, 32, 128)       0
_________________________________________________________________
batch_normalization_2 (Batch (None, 32, 32, 128)       512
_________________________________________________________________
flatten (Flatten)            (None, 131072)            0
_________________________________________________________________
d3 (Dropout)                 (None, 131072)            0
_________________________________________________________________
dense (Dense)                (None, 10)                1310730
=================================================================
Total params: 1,404,874
Trainable params: 1,404,426
Non-trainable params: 448
```

然后是到create_model里面

这里我们没有pretrained_model 所以第一个if先不看，他的意思就是如果有预先训练好的模型，我们就直接load那个训练好的模型

tensorflow中高阶的api就是keras，基本上后面用到的都是与keras有关的

有关这一行的gConfig之后会提到

下面tf.io.gfile表示的是tensorflow的文件操作

而这行中用到的listdir的作用就是列出目录中包含的条目列表

我们的目的就是判断存放模型的文件中有没有之前的model

```
['cnn_model.h5']
```

这里我的文件夹中已经有这个模型了，所以返回了这个文件名称，否则返回的就是空

下面则判断，如果存在了checkpoint，也就是ckpt变量，那么我们就加载这个model

下面的os.path.join是python中的路径拼接函数，他可以把若干个路径拼接到一起

这里我们将我们的工作目录，也就是working_directory 与 ckpt中的文件拼接到一起

注意ckpt存的是文件中的所有的文件名，网上搜到的资料表示tf.io.gfile.listdir返回的是无顺序的

但是这里作者认为是有顺序的，即最后一个是最新的，暂时不考虑这个，因为目前只处理一个模型文件

那么拼接好以后，model_file就是模型相对我们工程文件的一个相对路径，这时我们只需要用之前的方法加载这个模型即可

看到这里请先移步到getConfig.py中，先讲一下配置文件

configparser包是python中用于读取配置文件的包

他是一个section一个section配置的，[]中表示的就是一个section

看到config.ini中，我们就有三个section，这个名字是可以随意变换的

从这里就可以看到我们之前使用的working_directory等变量，实际上就是在这里预先配置好了

回到getConfig中，定义了函数用于在config.ini中找到配置信息，同时配置了默认参数

创建了parser并读取文件，parser为对应的解析器

parser.items表示以键值对的形式返回该section下的所有的option

这里输出代码中使用的三个变量用于理解

```
[('steps_per_checkpoint', 2), ('num_dataset_classes', 10), ('dataset_size', 50000), ('im_dim', 32), ('num_channels', 3), ('num_files', 6), ('images_per_file', 10000), ('epochs', 2)]
[('mode', 'train'), ('working_directory', 'model_dir'), ('dataset_path', 'train_data/'), ('test_path', 'test_data/')]
[('keeps', 0.5)]
```

其中每个变量都是一个list类型，里面储存的是一个tuple的二元组

最后我们将这些二元组转化成dict用于之后的使用

那么最后一个分支，就是如果我们没有model，那就要自己从头创建一个了

其中keep的含义表示的是dropout层神经元失效的概率

那么从这里我们就要来到cnnModel.py中了，这里的注释给的较为详细，所以对于每一层的解释就不再多加赘述了

可以发现我们每个文件的开头都有一部初始化gConfig的字典，并调用get_config来获得配置信息

这里再提一个点，python中的代码首先执行的是非函数定义，非类定义的无缩进的代码

当python程序以py文件运行时，文件属性`__name__`为main，否则就是文件名

还有一点就是模块导入的时候，全局变量是不会冲突的 使用module.var来访问对应模块的变量，或者使用from module import var 来导入到当前文件

但是对于使用来说，module.var是会改变原本的变量的值的，而from import不会

为神经网络添加完神经元层以后，使用model.compile对神经网络进行编译，同时选择对应的损失函数，优化器以及模型衡量指标

最后返回编译完成的model

创建完model后回到之前的train函数，下一步就是使用model.fit对模型进行训练，第一个参数是数据x，第二个参数就是结果（标签）y，verbose=1表示以进度条形式显示进度，epochs是训练模型迭代次数，也就是要训练多少次，validation_data指的是测试数据，用元组的格式传入，即每轮训练结束时的测试指标，这个数据不会被用来训练，还有一个选项就是validation_split，模型将根据这个比例选择一部分用作验证，batch_size参数为每次梯度更新的样本数，默认值为32

说道了训练模型，就要先开始说一下数据的导入了

来到execute.py的read_data中，首先看到五个参数，分别表示的是数据集路径，输入的图片大小，通道数，文件数以及每个文件的图片数

解释一下，图片的大小就是指分辨率，这里我们的im_dim就是32，表示图片是32*32的，通道数就是RGB通道，也就是三个通道。

这里我们的数据包括标签都是二进制文件，所以需要特定的读取方法

np.zeros返回一个给定形状和类型的用0填充的数组，对于我们的数据来说，我们就需要50000*32*32*3这么大的数组来存放数据，因为一共50000张图片，每个是32*32*3的，同理对于dataset_labels也一样，只不过存的是标签

for循环中遍历了每个文件，并找出数据集对应的文件，并使用unpickle_patch函数读取二进制文件

其中pickle.load就是将层级序列化的数据进行反序列化，得到原本的数据，并返回

下一行`b'data'`表示字符串是bytes类型的字符串，这一点可以通过输出字典的keys来看到

```
dict_keys([b'batch_label', b'labels', b'data', b'filenames'])
```

我们取到其中的data，输出数据类型可以发现是numpy的数组，输出他的shape看一下数组的大小

```
(10000, 3072)
```

10000张照片，每一张照片都有3072的数据，也就是32*32*3的数据，我们把他转化一下，用np.reshape转化一下

```
(10000, 32, 32, 3)
```

获得数据后，因为我们有5组数据，所以要把数据按照第一维拼接

返回读取到的数据集和标签

使用astype也就是numpy中的类型转换将数据转换成float型并除以255，变成01区间的小数

最后再使用to_categorical将标签转换为onehot编码，对于onehot编码的理解，可以用主析取范式来想

我们就完成了数据处理的部分

最后的网页端，可以参考我写的flask的一个简单的教程，结合w3cschool的教程，看完关键部分后理解app.py中的代码应该比较容易