写在教程前面

首先,本教程是给**python**和**tensorflow**有一定基础知识,但不是很了解**keras**的朋友写作的
目的是快速理解这个**INRIADATA PD **项目是如何构建,运行的.并且指导如何在本项目基础上,
修改项目,以达到活用项目的最终用途.

[TOC]

INRIADATA 训练集行人检测项目

一个简单的机器学习实例

快速上手

事先声明,本程序是在**window64**位系统下**ANACONDA 3.0**,**keras2.0**,**python 3.6.3**

的环境下运行的.如果涉及到环境相关问题,请到**stackoverflow**阅读相关文档解决.

本项目的主程序可以按照主程序涉及的module分为三个部分:

1.数据的预处理 包含在**preloading.py**内

2.模型的建立 包含在**myModel.py**

3.模型的训练,保存和评估 即主文件**KERAS TRAIN.py**主函数内部

试运行

先用文本编辑器打开config.py,修改其中的path 变量:

```
BATCH_SIZE = 32
EPOCHS = 1000
DATA_PATH = 'this is your inriadata path'#在这里修改你的数据路径
```

同样的,你也可以在这里修改之后训练的batch大小,和总训练的epochs数量.

我们通过module的方式,把这些通用常量都定义在文件的外部.

接下来,程序就会在 **\INRIA\INRIADATA\original_images**\pos**目录下,分别生成*x_train.trian.npy*和*y_train.trian.npy*文件作为训练时的输入.如果你有别的训练的程序,请参见 如何修改训练集

如果嫌一次生成4个数据集合的*npy*文件很麻烦的话,可以使用*preloading.py*中的*_build_train_data*函数来创建训练数据.方法是这样

```
from preloading import _build_train_data as build_data
build_data(DATA_PATH)#输入训练集合,程序会自动匹配annotation目录
```

当程序运行结束之后,就可以直接运行*KERAS_TRAIN.py*了

这是运行的画面

*keras*在运行开始时,会显示其所用后端的名称.然后当模型被编译运行之后,就能看到这些参数:

- ETA:每个Batch所用时间#这里我不是很理解哈
- loss:平均损失
- accuracy:准确率
- **ms/step :每执行一个操作所花费的毫秒数.可以通过这个数值来评估工作的效率,和预估工作总时长.

最终训练结束

如何制作自己的网络

简单介绍Keras

keras是一个基于世面上常见深度学习框架,如*tensorflow*,*theano*,*caffe*的高级api.用于快速的搭建网络

快速入门keras

使用keras进行机器学习的流程通常分为两个部分.

- 构建网络
- 训练模型

下面是一个简短的例子

```
import numpy as np
import keras

>>>Using TensorFlow backend.
```

```
from keras import Sequential
from keras.layers import Dense, Dropout, Activation

#生成测试数据
x_train = np.random.random([1000,5])
y_train = np.random.random([1000,5])
#x_train.shape =(1000,5)

#实例化一个模型/即开始构建网络
model = Sequential()
model.add(Dense(32,input_shape = x_train.shape[1:]))#增加一个全连接层
model.add(Activation('relu'))#在全连接层上增加一个relu激活函数
model.add(Dense(5))#增加一个全连接层
model.add(Activation('softmax'))#在全连接层上增加一个softmax激活函数
model.add(Dropout(0.05))#增加dropout防止过拟合,rate = 0.05

#完成网络的构建,调用summary方法来获得网络信息
print(model.summary())

#得到网络模型的文本
>>>_________________________________________________________________
>>>Layer (type)                 Output Shape              Param #
>>>=================================================================
>>>dense_1 (Dense)              (None, 32)                192
>>>_________________________________________________________________
>>>activation_1 (Activation)    (None, 32)                0
>>>_________________________________________________________________
>>>dense_2 (Dense)              (None, 5)                 165
>>>_________________________________________________________________
>>>activation_2 (Activation)    (None, 5)                 0
>>>_________________________________________________________________
>>>dropout_1 (Dropout)          (None, 5)                 0
>>>=================================================================
>>>Total params: 357
>>>Trainable params: 357
>>>Non-trainable params: 0
>>>_________________________________________________________________
>>>None
#编译网络
model.compile(optimizer = 'sgd',
              loss = 'mae',
               metrics = ['accuracy'])
#模型训练
```

```
model.fit(x_train,y_train,
                        epochs = 10,
                        batch_size = 10)
```

构建网络

我们先来看看网络搭建的部分

```
model = Sequential()
model.add(Dense(32,input_shape = x_train.shape[1:]))#增加一个全连接层
model.add(Activation('relu'))#在全连接层上增加一个relu激活函数
model.add(Dense(5))#增加一个全连接层
model.add(Activation('softmax'))#在全连接层上增加一个softmax激活函数
model.add(Dropout(0.05))#增加dropout防止过拟合,rate = 0.05
```

Sequential,即序贯模型,是*keras*中实现网络的一种方式,是说数据的流动从网络输入到输出单向流动的这么一种网络模型.与此相对应的式被称作函数式或称作*Functional*的网络模型.

可以把序贯模型,看做是一个list-like的对象.其子对象是对应的网络层.我们可以通过add方法,来讲层添加到序贯模型中.或者,我们也可以用pop方法,将最后添加的网络层从序贯模型中取出.

我们先把Sequential实例化给model,然后使用Sequential的add方法,将keras.layers模块中的Dense层,添加到我们的网络中.

在keras里,数据通常是由numpy.ndarray类型来传递的.

此时,由于Dense是我们的Sequential的第一个输入,所以需要额外指定输入的input_shape,input_shape以元组的方式传递.
在之后的网络层中不用再额外指定input_shape.keras将完成剩下的输入计算
我们通过加入Activation,和Dropout,最后得到了我们的网络模型.

但,我们要如何确认我们的模型是按照我们所想的那样正常运作呢?
那就对model使用summary方法.
将model.summary()打印,我们可以得到如下输出.

```
print(model.summary())

>>>_________________________________________________________________
>>>Layer (type)                 Output Shape              Param #
>>>=================================================================
>>>dense_1 (Dense)              (None, 32)                192
>>>_________________________________________________________________
>>>activation_1 (Activation)    (None, 32)                0
```

```
>>>_________________________________________________________________
>>>dense_2 (Dense)              (None, 5)                 165
>>>_________________________________________________________________
>>>activation_2 (Activation)    (None, 5)                 0
>>>_________________________________________________________________
>>>dropout_1 (Dropout)          (None, 5)                 0
>>>=================================================================
>>>Total params: 357
>>>Trainable params: 357
>>>Non-trainable params: 0
>>>_________________________________________________________________
>>>None
```

对以上参量做出解释:

- layer:网络层
- param:层中参数
- Total params:网络中总共的参数
- Trainable params:可训练的参数
- Non-trainable params:不可训练的参数

不可训练参数是考虑到对于某些已经训练的网络进行增添新的层,或者是其他需要冻结参数的情况下,增添的可选项.想要让某个网络层不可训练,就需要在添加层的时候,显示的传入trainable=False
像这样

```
model.add(Dense(32,trainable=False))
```

模型训练

keras对模型的训练设置了大量的函数,并且,会把训练集合的一部分作为测试集来测试准确程度.所以在快速测试网络的时候,可以不用特地准备测试集.

```
model.compile(optimizer = 'sgd',
              loss = 'mae',
               metrics = ['accuracy'])
#模型训练
model.fit(x_train,y_train,
                        epochs = 10,
                        batch_size = 10)
```

在模型训练之前,需要先让keras将模型编译.把之前搭建好的网络传入后端.所以要注意网络搭建的时候是否和对应的后端契合.如tensorflow 和 caffe 之间对图像处理格式的不同等,需要小心.

当对model使用fit方法时,模型就将开始训练.我们只需对fit传入训练集合x_train,目标集合y_trian和batch大小以及epoch数就可以快速开始训练.对于fit的更多用法,请参看这里.

好的现在已经了解了keras了,就可以尝试编写自己的网络啦!

活用模型

主要介绍两个方面,图片预处理的部分和如何添加自己的网络两个部分.

图片预处理

_turn_a_pic_to_tensor

在*preloading.py*中,函数*_turn_a_pic_to_tensor *被作为图像预处理的函数来使用.

```
def _turn_a_pic_to_tensor(img_path):
        img = tf.gfile.FastGFile(img_path,'rb').read()
        #turn's into 4-D tensor,and resize it
        with tf.Session() as sess:
                img_tensor = tf.image.decode_png(img)
                img_tensor= tf.image.resize_images(img_tensor,
[256,256],method = 0)
                img_tensor = img_tensor.eval()
        return img_tensor
```

该函数先读取图片对象,调用tf.image.resize_images函数,将原图像正规成256x256的图片
最后传出一个numpy.ndarray的对象.
只要满足输出为ndarray即可,固可以修改该函数来达到不同预处理的目的
在修改完该函数后,需要运行*_build_train_data*或*get_input*来修改输入

get_input

在*preloading.py*中,函数*get_input*被作为数据的输入口,传出一个元祖(x_train,y_train,x_test,y_test),其中的元素为ndarray
若自己有其他的数据集需要传入,只需修改get_input,将输出的数据集改为你需要的数据集即可.

添加自定义网络

网络被定义在*myModel.py*中,定义自己的网络的通常的方式是,写一个传入数据集合的函数,返回一个构建好的网络.
如下图所示:

```
def CNN(X_train,Y_train,X_test,Y_test):
        #build model
        model = Sequential()
        #add conv1 3x3
        model.add(Conv2D(32,(3,3),input_shape = X_train.shape[1:]))
        #add relu1
        model.add(Activation('relu'))
        #add maxpooling1
        model.add(Maxpool(pool_size=(2, 2)))
        #add conv2
        model.add(Conv2D(16,(2,2)))
        #add flatten1
        model.add(Flatten())
        #add full_connected layer1
        model.add(Dense(24))
        #add full_connected layer2
        model.add(Dense(16))
        #add softmax
        model.add(Activation('softmax'))
        model.add(Reshape((4,4)))
        #return the model to call
        return model
```

构建好之后,随便给这个网络起个关键字加入 myModel.py的model函数中,方便调用即可.如:

```
def Model(modelname,X_train,Y_train,X_test,Y_test):
        if modelname == 'CNN' or modelname == 'MODELNAME':
                return CNN(X_train,Y_train,X_test,Y_test)
```

修改参数

参数被定义在config.py中

```
BATCH_SIZE = 32
EPOCHS = 1000
```

```
DATA_PATH = 'Your filepath'
MODELNAME = 'Your nextworkname'
```

可以直接通过config.py调整参数,来得到训练数据或者是定制batch大小和epochs总数.

可以参阅的网站

这些网站是对更深层次的定义网络相对重要的,或者是对机器学习有益的,我都列在下面,方便的时候就看一下.

- python学习--廖雪峰的python网站
- tensorflow 官方 python api
- tensorflow 的 其他具体实现
- keras 的 中文文档(并不完善)
- keras 官方文档
- 机器学习的一些小技巧

希望能有所脾益.一起学习吧!