Python

jupyter 中安装

1. 在jupyter 中通过!pip install h5py可以安装

os.path.basename(path)

1.返回path最后的文件名。如何path以 / 或\结尾，那么就会返回空值。即os.path.split(path)的第二个元素。

```
>>> os.path.basename('c:\\test.csv')
'test.csv'
>>> os.path.basename('c:\\csv')
'csv'（这里csv被当作文件名处理了）
>>> os.path.basename('c:\\csv\\')
''
```

glob.glob

glob.glob

返回所有匹配的文件路径列表。它只有一个参数pathname，定义了文件路径匹配规则，这里可以是绝对路径，也可以是相对路径。下面是使用glob.glob的例子：

```python
import glob

#获取指定目录下的所有图片
print glob.glob(r"E:/Picture/*/*.jpg")

#获取上级目录的所有.py文件
print glob.glob(r'../*.py') #相对路径
```

os.walk

os.walk()

函数声明：walk(top,topdown=True,onerror=None)

1>参数top表示需要遍历的目录树的路径

2>参数topdown的默认值是"True",表示首先返回目录树下的文件，然后在遍历目录树的子目录.Topdown的值为"False"时，则表示先遍历目录树的子目录，返回子目录下的文件，最后返回根目录下的文件

3>参数onerror的默认值是"None",表示忽略文件遍历时产生的错误.如果不为空，则提供一个自定义函数提示错误信息后继续遍历或抛出异常中止遍历

4>该函数返回一个元组，该元组有3个元素，这3个元素分别表示每次遍历的路径名，目录列表和文件列表

os,walk()实例:

```
import os
def VisitDir(path):
  for root,dirs,files in os.walk(path):
    for filespath in files:
      print os.path.join(root,filespath)
if __name__=="__main__":
path="/root"
```

os.path.walk()

函数声明：walk(top,func,arg)

1>参数top表示需要遍历的目录树的路径

2>参数func表示回调函数，对遍历路径进行处理.所谓回调函数，是作为某个函数的参数使用，当某个时间触发时，程序将调用定义好的回调函数处理某个任务.回调函数必须提供3个参数：第1个参数为walk()的参数tag，第2个参数表示目录列表，第3个参数表示文件列表

3>参数arg是传递给回调参数func的元组.回调函数的一个参数必须是arg，为回调函数提供处理参数.参数arg可以为空

os.path.walk()实例：

```
import os,os.path
def VisitDir(arg,dirname,names):
  for filespath in name:
    print os.path.join(dirname,filespath)
if __name__=="__main__":
path="/root"
```

1. os.path.walk()与os.walk()产生的文件名列表并不相同.os.path.walk()产生目录树下的目录路径和文件路径，而os.walk()只产生文件路径

参数

1. 有位置参数时，位置参数必须在关键字参数的前面，但关键字参数之间不存在先后顺序的

Python命令行解析argparse常用语法使用简介

1. add_argument()方法，用来指定程序需要接受的命令参数

```
import argparse
parse = argparse.ArgumentParser()
parse.add_argument("a", help="params means")
parse.add_argument("-C", "--gc", default="count")
parse.add_argument("--ga", help="params means ga",dest='simple_value',choices=['A', 'B', 'C', 0])
parse.add_argument("--gb", help="params means gb",action="store_const",const='value-to-store')
args = parse.parse_args()
print args.simple_value,args.gb,args.gc

### add_argument 说明
不带'--'的参数
   调用脚本时必须输入值
   参数输入的顺序与程序中定义的顺序一致
'-'的参数
   可不输入    add_argument("-a")
   类似有'--'的shortname，但程序中的变量名为定义的参数名
'--'参数
   参数别名: 只能是1个字符，区分大小写
      add_argument("-shortname","--name", help="params means")，但代码中不能使用shortname
   dest: 参数在程序中对应的变量名称 add_argument("a",dest='code_name')
   default: 参数默认值
   help: 参数作用解释  add_argument("a", help="params means")
   type : 默认string  add_argument("c", type=int)

action:
      store：默认action模式，存储值到指定变量。
      store_const：存储值在参数的const部分指定，多用于实现非布尔的命令行flag。
      store_true / store_false：布尔开关。 store_true.默认为False，输入则为true。 store_flase 相反
      append：存储值到列表，该参数可以重复使用。
      append_const：存储值到列表，存储值在参数的const部分指定。
      count: 统计参数简写输入的个数  add_argument("-c", "--gc", action="count")
      version 输出版本信息然后退出。
const:配合action="store_const|append_const"使用，默认值
choices:输入值的范围 add_argument("--gb", choices=['A', 'B', 'C', 0])
required : 默认False, 若为 True, 表示必须输入该参数
```

可视化模型

```
from IPython.display import SVG, Image
from keras.utils.vis_utils import plot_model

plot_model(model,to_file='DenseNet.png' ,show_shapes=True)
Image('DenseNet.png')
```

Keras的one_hot

```
from keras.utils import np_utils
X_train = X_train.reshape(60000, 784)
X_test = X_test.reshape(10000, 784)
X_train = X_train.astype("float32")
X_test = X_test.astype("float32")

# Put everything on grayscale
X_train /= 255
X_test /= 255

# convert class vectors to binary class matrices
Y_train = np_utils.to_categorical(y_train, 10)
Y_test = np_utils.to_categorical(y_test, 10)

from sklearn.model_selection import train_test_split
#划分数据集
X_train, X_val, Y_train, Y_val = train_test_split(X_train, Y_train)
```

Keras的History

1. 其History.history属性记录了损失函数和其他指标的数值随epoch变化的情况，如果有验证集的话，也包含了验证集的这些指标变化情况。
2. fit函数返回的是一个History对象

Python的hasattr() getattr() setattr() 函数使用方法详解

hasattr(object, name)

判断一个对象里面是否有name属性或者name方法，返回BOOL值，有name特性返回True， 否则返回False。
需要注意的是name要用括号括起来

```
>>> class test():
...     name="xiaohua"
...     def run(self):
...             return "HelloWord"
...
>>> t=test()
>>> hasattr(t, "name") #判断对象有name属性
True
>>> hasattr(t, "run")  #判断对象有run方法
True
>>>
```

getattr(object, name[,default])

获取对象object的属性或者方法，如果存在打印出来，如果不存在，打印出默认值，默认值可选。
需要注意的是，如果是返回的对象的方法，返回的是方法的内存地址，如果需要运行这个方法，
可以在后面添加一对括号。

```
>>> class test():
...     name="xiaohua"
...     def run(self):
...             return "HelloWord"
...
>>> t=test()
>>> getattr(t, "name") #获取name属性，存在就打印出来。
'xiaohua'
>>> getattr(t, "run")  #获取run方法，存在就打印出方法的内存地址。
<bound method test.run of <__main__.test instance at 0x0269C878>>
>>> getattr(t, "run")()  #获取run方法，后面加括号可以将这个方法运行。
'HelloWord'
>>> getattr(t, "age")  #获取一个不存在的属性。
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: test instance has no attribute 'age'
>>> getattr(t, "age","18")  #若属性不存在，返回一个默认值。
'18'
>>>
```

setattr(object, name, values)

给对象的属性赋值，若属性不存在，先创建再赋值。

```
>>> class test():
...     name="xiaohua"
...     def run(self):
...             return "HelloWord"
...
>>> t=test()
>>> hasattr(t, "age")    #判断属性是否存在
False
>>> setattr(t, "age", "18")    #为属相赋值，并没有返回值
>>> hasattr(t, "age")    #属性存在了
True
>>>
```

字典中根据value获取key

```
def get_keys(d, value):
    return [k for k,v in d.items() if v == value]

get_keys({'a':'001', 'b':'002'}, '001') # => ['a']
```

```
dicxx = {'a':'001', 'b':'002'}
new_dict = {v:k for k,v in dicxx.items()}  # {'001': 'a', '002':
'b'}
new_dict['001']  # 'a'
```