今日学习任务

1.研究如何使用web端

3.用已经训练的模型同文件夹下的文件遍历并分类

2.研究如何打包python文件为api

Web框架之Flask

摘自知乎@下班去看书 与 百度百科

flask是一个轻量级的基于Python的web框架。它没有太多复杂的功能，flask没有默认使用的数据库等。

其他Python web框架：

- Django：它比flask重的多。包含了web开发的常用功能，orm、session、form、admin、分页、中间件、信号、缓存、contenttype等等，Django最著名的是它的全自动化的管理后台：只要使用ORM，做简单的对象定义，就能自动生成数据库结构和管理后台。
- Tornado：特性是异步非阻塞，和nodejs很像、原生支持websocket。

请求解析

- GET请求
  导入request，从*request*对象获取请求数据：
  浏览器请求：http://127.0.0.1:3000?user=john&age=18

```
from flask import Flask, request

app = Flask(__name__) # 程序名称

@app.route("/", methods=["GET"])
# 定义路由，指定哪些请求可以响应
def index():
    request.args.__str__() # 获取所有的参数
    request.args.get("name") # 获取单个参数，输出：john
    requets.args.get("age") # 输出 18
    return "hello world"

if __name__ == "__main__": # 在代码不是模块的情况下运行下面的代码
    app.run(port=3000, debug=True) # port指定端口；debug为True时，修改代码，程序会自动重启
```

- POST请求

```
from flask import Flask

app = Flask(__name__) # 程序名称

@app.route("/", methods=["POST"])
```

```
# 定义路由，指定哪些请求可以响应
def index():
    request.form # 获取form请求数据
    request.form["name"] # 获取form中的name
    request.form["age"] # 获取form中的age

    request.json
    # 获取json格式请求数据，会自动将json数据转换为Python的dict或者list
    request.json["name"] # 获取json数据中的name
    request.json["age"] # 获取json数据中age
    return "hello world"

if __name__ == "__main__": # 在代码不是模块的情况下运行下面的代码
    app.run(port=3000, debug=True) # port指定端口；debug为True时，修改代码，程序会自动重启
```

请求响应

使用Response设置响应头

```
from flask import Flask, Response
import json

class Person:
    def __init__(self, name, age):
        self.__name__=name
        self.__age__=age

    def as_dict(self):
        return {
            "name": self.name,
            "age": self.age
        }

app = Flask(__name__) # 程序名称

@app.route("/", methods=["POST"])
# 定义路由，指定哪些请求可以响应
def index():
    person = Person("John", 18)
    return Response(json.dumps(person, default=Person.as_dict), mimetype="application/json")
# 使用json.dumps转换为json格式。第一个参数为要转换的数据，第二个参数default为转换为json需要执行的方法。如果第一个参数是list，json.dumps会自动进行遍历。

if __name__ == "__main__": # 在代码不是模块的情况下运行下面的代码
    app.run(port=3000, debug=True) # port指定端口；debug为True时，修改代码，程序会自动重启
```

另外，使用flask中的jsonify函数也可以进行转换。

---

cookie和session

flask中的session借助session实现，cookie由Response实现。

```
from flask import Flask, session, Response

app = Flask(__name__) # 程序名称

@app.route("/login", methods=["GET", "POST"])
# 定义路由，指定哪些请求可以响应
def login():
    session["user_name"]=request.json[username]
    # 取出请求中的用户标识，存储在session中
    res = Response("add cookie") # 使用cookie
    res.set_cookie(key="cookie_demo", value="hello",
expires=time.time()+6*60) # 6分钟失效
    return "hello world"

@app.route("/logout", methods=["POST"])
def logout():
    session.pop("username", None)
    return redirect(url_for('login'))
# 重定向到login

@app.route("/logout/del_cookie", methods=["POST"])
def logout0():
    res = Response("del cookie")
    # 删除cookie
    res.set_cookie("cookie_demo", "", expires=0)
    # 删除cookie
    return res

if __name__ == "__main__": # 在代码不是模块的情况下运行下面的代码
    app.run(port=3000, debug=True)
    # port指定端口；debug为True时，修改代码，程序会自动重启
```

API的作用：

-对于软件提供商来说，留出API，让别的应用程序来调用，形成生态，软件才能发挥最大的价值，才能更有生命力。（同时别人也看不见代码，不伤害商业机密。）（可以用顺丰京东举例）

-对于应用开发者来说，有了开放的API，就可以直接调用多家公司做好的功能来做自己的应用，不需要所有的事情都自己操刀，节省精力。

API (Application Programming Interface)：

翻译成中文就是"应用程式介面"，其实这样翻译不好，应该说是"程式沟通介面"。

翻译为介面，顾名思义就要沟通两个不同的东西用的，通常由一组函式库所组成。

在一个 同一个平台 下的 两个不同东西(程式 or 系统)，为了能取用对方的功能等等，

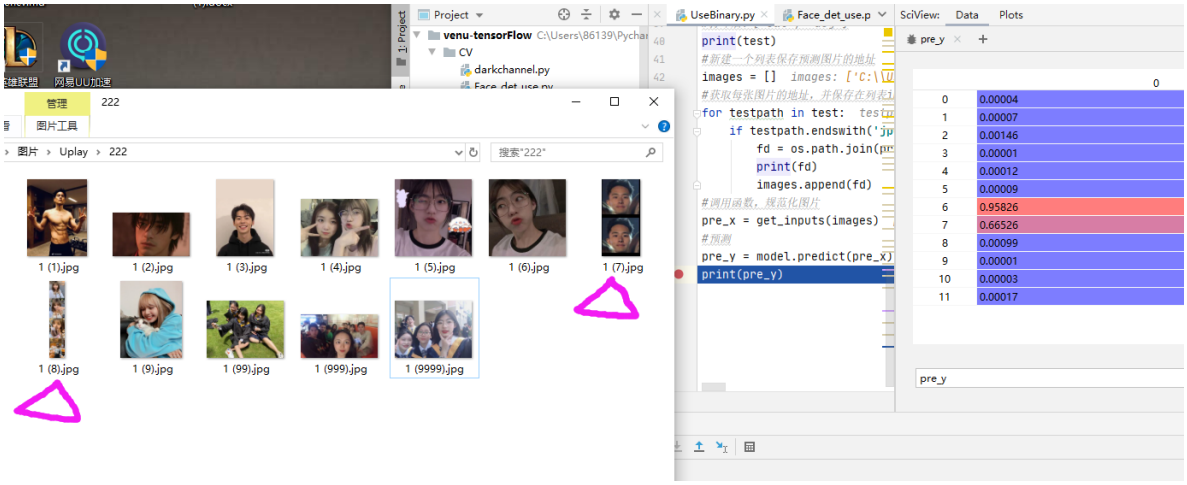
所以一个 X程式 写了一组函式，让 同一平台的其他程式 取用 X程式 的功能，

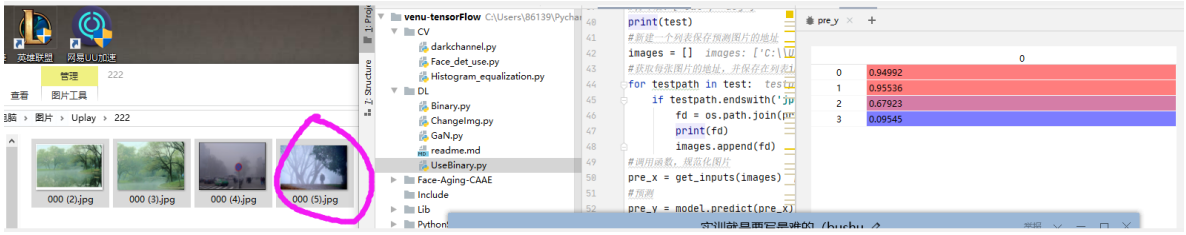
那组函式就可以说是那个 X程式 对外开放的 API

对单一图片进行预测

predict函数：返回的是一个'识别率'的数组

结果





可以看出，对于长图片以及低清晰度的图片，有一定的识别错误，还可以改进，但是已经可以进行简单的预测。

接下来需要进行分类，新建目录，这一系列工作比较简单，所以不进行赘述。

今日重点：输出为API

```
@app.route('/predict', methods=['get', 'post'])
```

首先定义了该api的基本思路

```
if __name__ == '__main__':
    model = load_model("D:\\数据集\\animals_else.h5")
    print('Model loaded')
    app.run(debug=True, port=7777, host='0.0.0.0')

#http://localhost:7777/predict?url=文件夹路径
#https://www.jianshu.com/p/cafbeec49674
```

目前还是使用本电脑端的localhost进行简单的操作，输入的图片是上述branch的结果中的一个文件夹（222）

```
return jsonify({'prediction': str(prediction)})
```

最后返回的是一个str(数组)

结果:

{
  "prediction": "[array([0.05555016], dtype=float32), array([0.00019738],
dtype=float32), array([0.9977445], dtype=float32), array([0.99673927],
dtype=float32), array([0.9462516], dtype=float32), array([0.99999684],
dtype=float32), array([0.00010607], dtype=float32), array([0.00100729],
dtype=float32), array([0.00019329], dtype=float32), array([0.99998975],
dtype=float32), array([0.9944793], dtype=float32)]"
}