

# 中国科学技术大学

# AI 期末报告



## 基于前后端分离的场景、目标、 人脸、人体以及关键点识别系统

作者姓名： 吴语港 SA19225404

苏成 SA19225321 盛黎明 SA19225313

报告内容： 人工智能期末课程设计

完成时间： 2019 年 12 月 2 日

## 摘 要

本设计通过客户端，服务端的方式实现了场景识别，目标识别，人脸识别，人体识别以及人体关键点识别的任务，操作流程就是通过客户端选择工作模式，然后打开本地图片，发送到服务端，然后告诉服务端识别模式，识别完成后将结果参数发送给客户端，并将识别结果在客户端所选择的图片上标注出来。

其中场景识别和目标识别依靠服务端本地已经训练完成的的 pd 模型文件进行处理，其他的三个功能通过服务端调用旷世科技 API 接口完成，并简化接口返回参数，发给客户端并在客户端选择的图片上用 opencv 标注出识别信息。

**关键词：**场景识别 目标识别 服务端 客户端 旷视科技 API 人体识别  
人脸识别 人体关键点识别 Inception-V3

### 任务分工：

#### 吴语港：

前后端的设计、本地模型的部署和 API 调用的部署

#### 苏成：

场景分类模型的设计训练与测试

#### 盛黎明：

目标识别模型的设计训练与测试

## 目 录

第 1 章 总体介绍 .....	4
1.1 资源地址 .....	4
1.2 使用介绍 .....	4
第 2 章 场景分类模型设计 .....	11
2.1 总体概述.....	11
2.2 测试程序.....	15
第 3 章 目标识别模型设计 .....	17
3.1 Inception Net-V3 概述.....	17
3.2 处理程序.....	20
第 4 章 客户端设计 .....	22
4.1 总体概述.....	22
4.2 场景识别类.....	22
4.3 目标识别类.....	25
4.4 人脸识别类.....	26
4.5 人体识别类.....	27
4.6 人体关键点识别类.....	28
4.7 客户端类.....	29
第 5 章 服务端设计 .....	31
5.1 总体概述.....	31
5.2 场景识别部分.....	32
5.3 目标识别部分.....	33
5.4 人脸识别部分.....	34
5.5 人体识别部分.....	35
5.6 人体关键点识别部分.....	36
5.7 接受图片部分.....	38
5.8 其他部分.....	38
第 6 章 其他技术分析 .....	40
6.1 旷视科技 API 接口分析.....	40
6.2 python 的 socket 库.....	56
6.3 python 的 requests 库.....	58
参考资料.....	61

# 第 1 章 总体介绍

## 1.1 资源地址

1. 程序文件：（包括模型训练文件夹和报告 PPT 以及演示视频）

百度网盘链接：[https://pan.baidu.com/s/1KbfqMkBc0niUB\\_dFkqQesQ](https://pan.baidu.com/s/1KbfqMkBc0niUB_dFkqQesQ)

提取码：7sry

2. 演示视频：

视频地址：<https://b23.tv/av77904661>

## 1.2 使用介绍

本设计一共有 5 个功能，分别是场景识别，目标识别，人脸识别，人体识别，人体关键点识别，下面来一一介绍：（注，为了截图方便起见，服务端和客户端在同一台电脑上运行，在不同电脑上运行的情况已经实测过，可以正常运行）

### 运行方式

在 py 文件的路径下打开 cmd 窗口分别输入下面内容，回车即运行

```
C:\Windows\System32\cmd.exe - python server.py
```

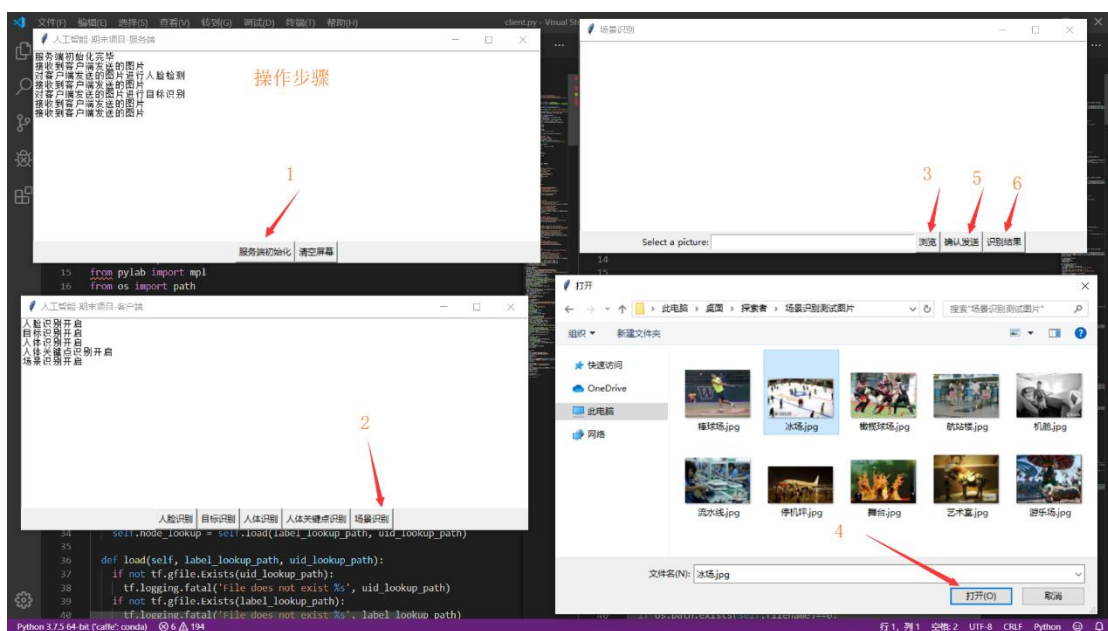
```
D:\Users\WYG\Desktop\探索者>python server.py
```

和

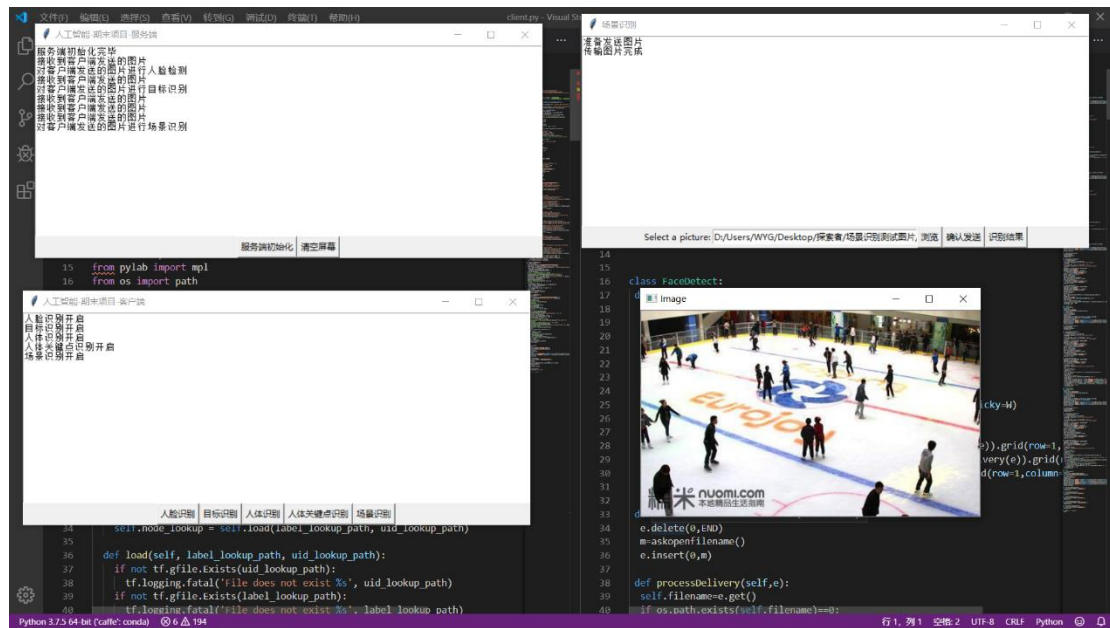
```
D:\Users\WYG\Desktop\探索者>python client.py
```

### 1. 场景识别

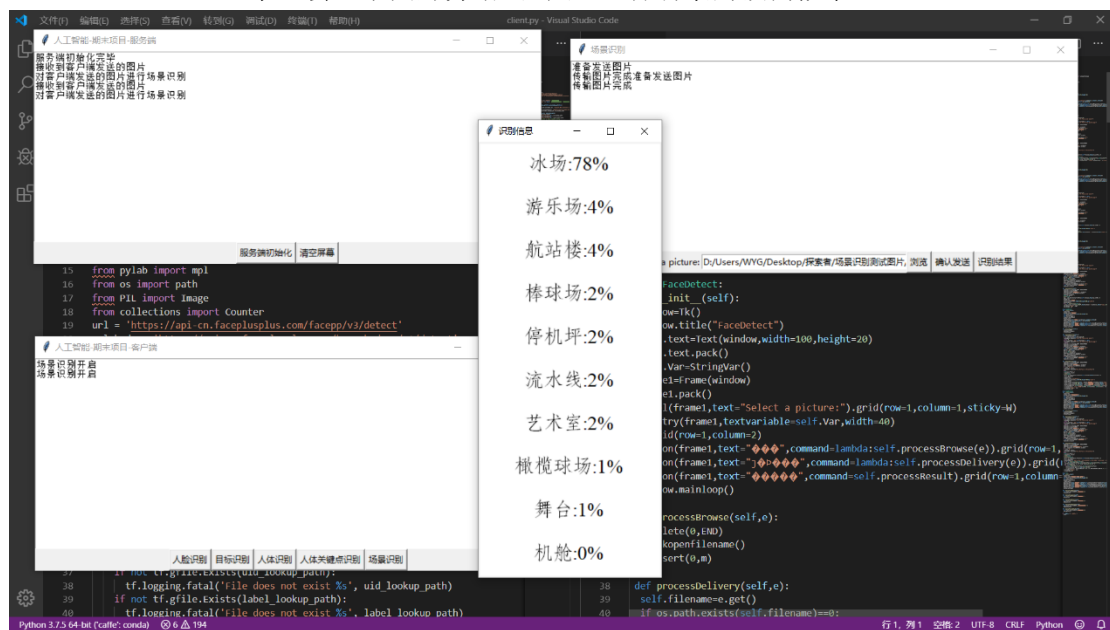
本设计设定了 10 个场景分类，下面我们对冰场进行测试



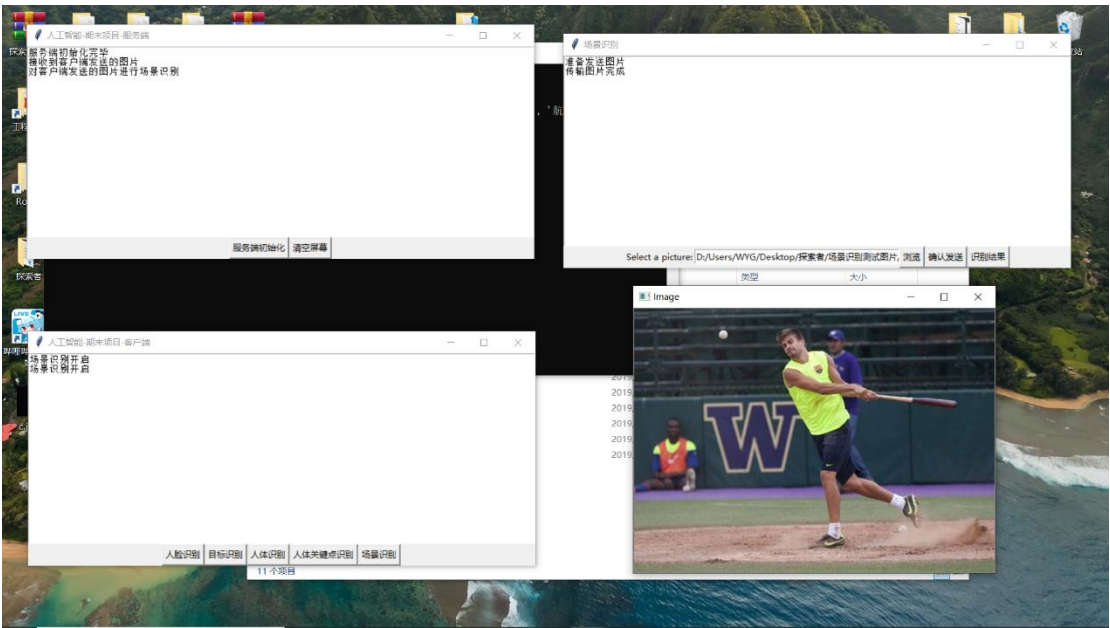
## 选择完图片后确认发送



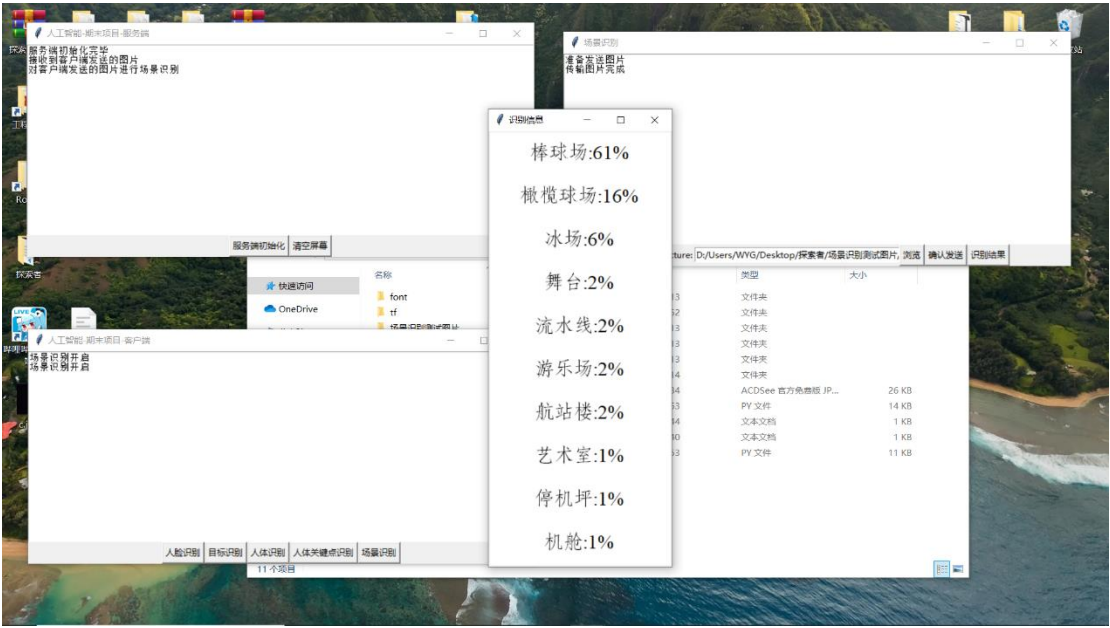
本地会显示图片并给出对于 10 种场景的预测概率



# 对棒球场测试

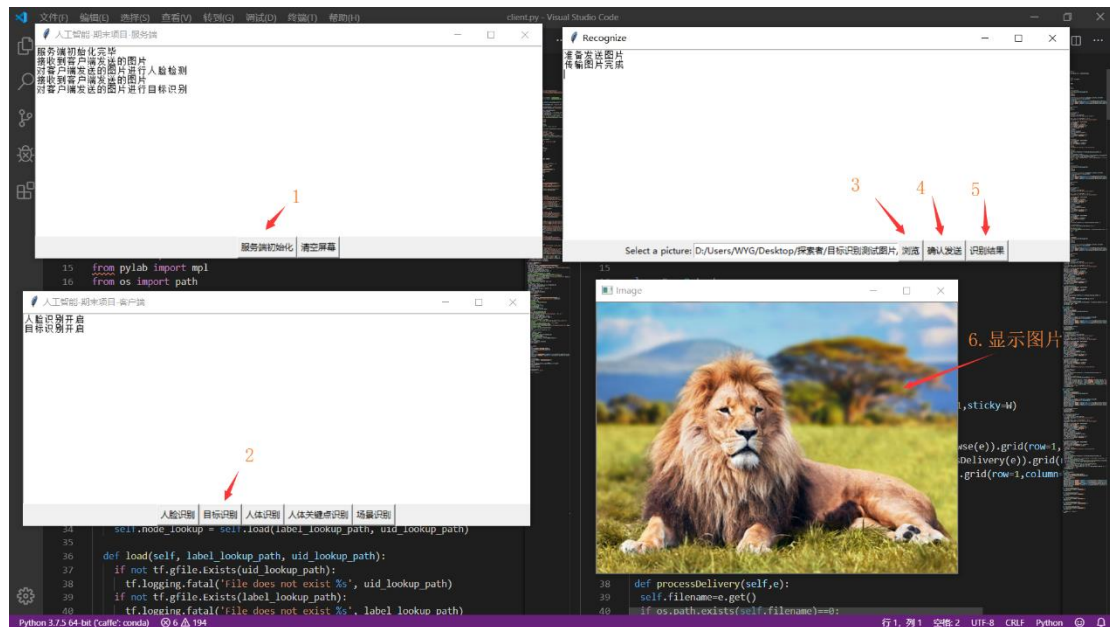


相关性比较高的两个场地，正常的输出结果

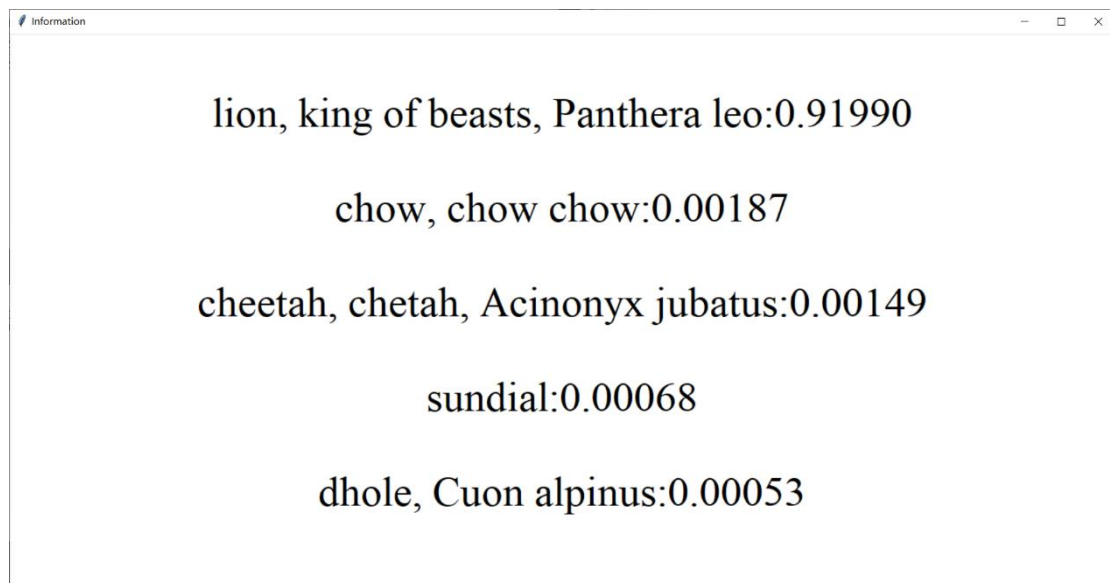


## 2. 目标识别

选择狮子图片进行识别



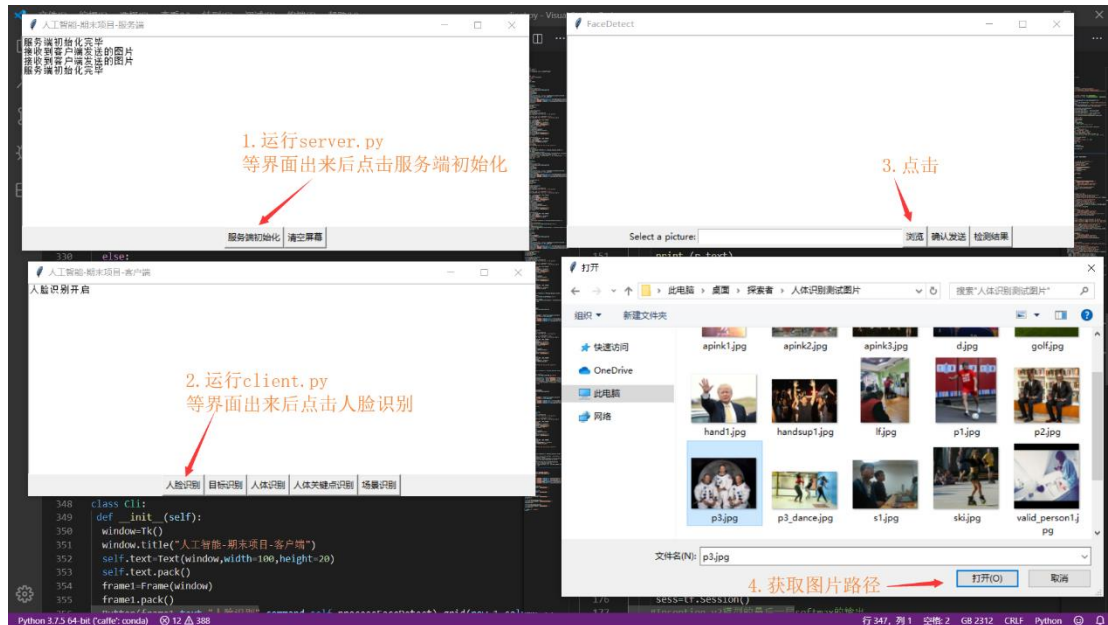
各出概率最高的 5 种分类



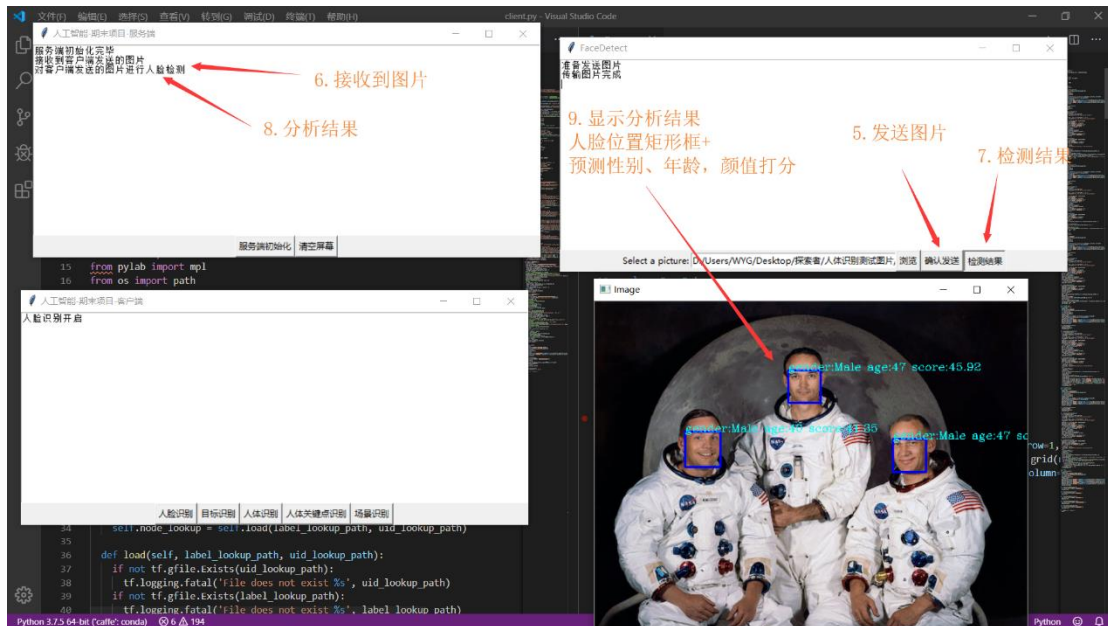


### 3. 人脸识别

选择人脸图片进行识别



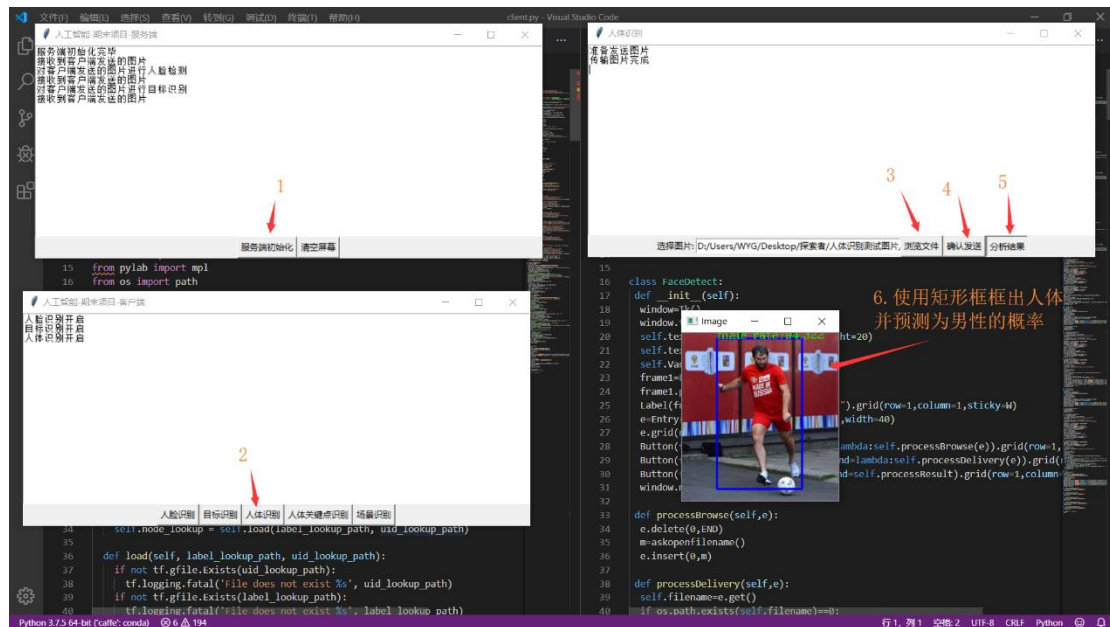
对识别出来的所有脸进行标注



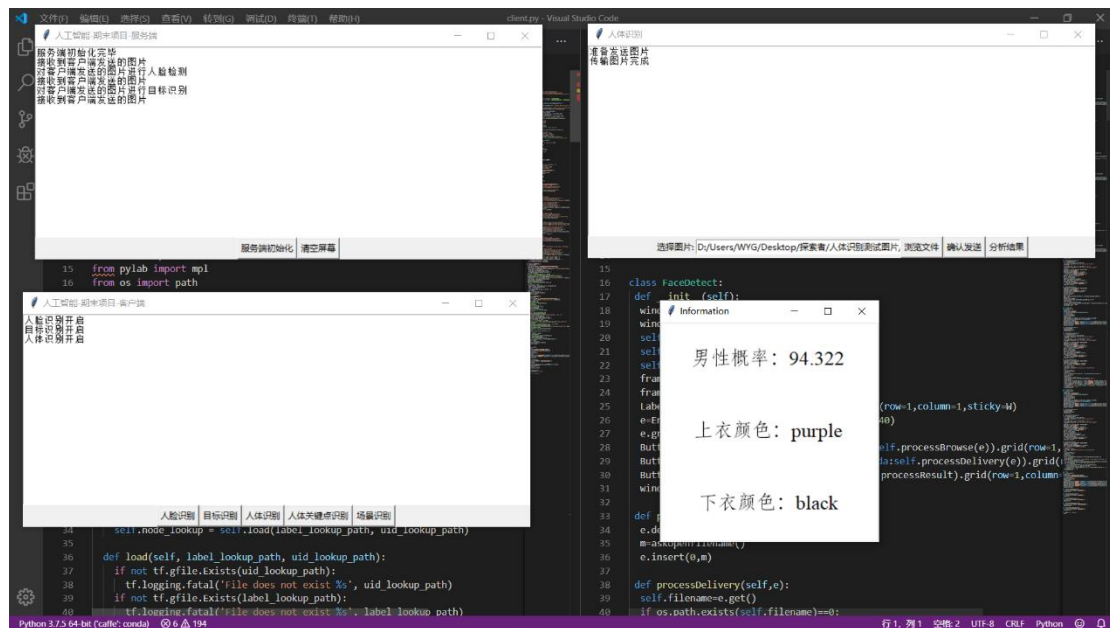


## 4. 人体识别

选择人体图片进行识别



矩形框住人体并各出预测信息

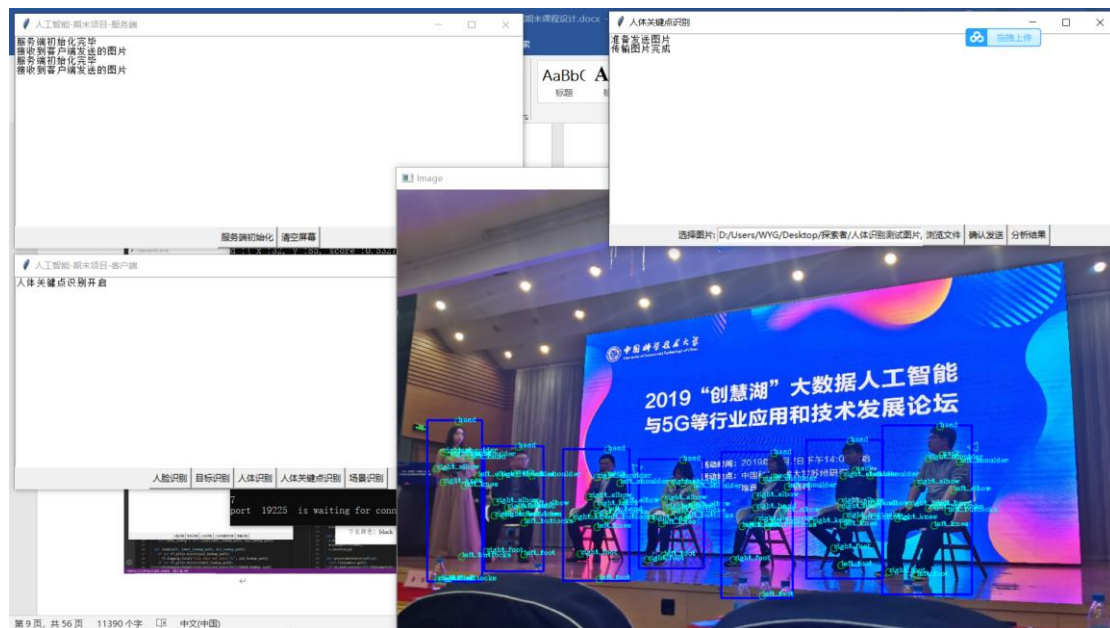


## 5. 人体关键点识别

选择人体图片进行识别



识别信息已经在图片上标注出来了



## 第 2 章 场景分类模型设计






### 2.1 总体概述

#### 1. 场景识别

##### 场景分类

场景分类是指根据图片内容，判断内容是属于哪种场景，由于训练准确的场景分类需要大量的训练集以及复杂的网络结构以及算力，为了更加方便训练我们自己的需要的模型，我们采用迁移学习的方式去训练模型，即用一个训练好的模型去训练另一个需要的模型。我们采用的是 Google 官方提供的 inception v3 模型为原模型，inception v3 模型用 Image Net 数据集训练而来，用于区分 1000 个图像分类，经过大量训练，可以很好地提取图片特征。

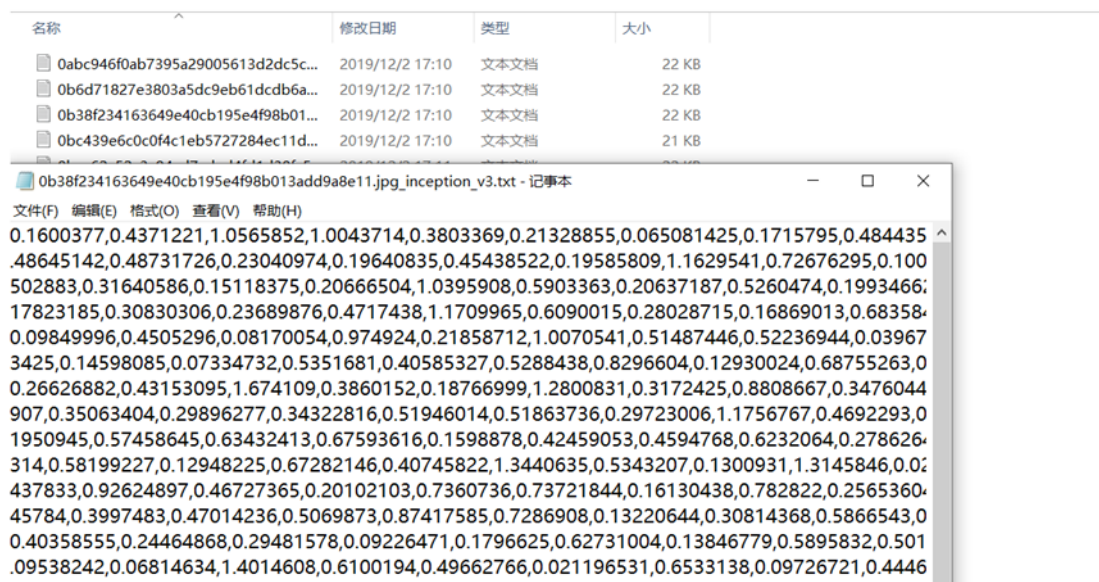
以下为下载的模型解压之后的样子，一个训练好的 pb 文件，不是 ckpt，这个已经冻结，不能训练，只能用来识别。

名称	修改日期	类型	大小
 classify_image_graph_def.pb	2015/12/5 10:14	PB 文件	93,432 KB
 cropped_panda.jpg	2015/12/1 12:28	JPG 文件	3 KB
 imagenet_2012_challenge_label_map...	2015/11/19 6:39	PBTXT 文件	64 KB
 imagenet_synset_to_human_label_m...	2015/11/19 6:39	文本文档	725 KB
 LICENSE	2015/12/5 7:07	文件	12 KB

打开两个文件 imagenet\_2012\_challenge\_label\_map\_proto.pbtxt 和 imagenet\_synset\_to\_human\_label\_map.txt。这是原 inception v3 的检测的情况，就像下图的 384 是一个类别，对应的字符串编号是 n01514859，应该英文描述为 hen，母鸡，存储标签内容。

target_class: 445 target_class_string: "n01496331"	n01501641 n01501777 n01501948 n01502101 n01503061 n01503976 n01504179 n01504344 n01514668 n01514752 n01514859 n01514926 n01515078 n01515217 n01515303 n01516212 n01517389 n01517565 n01517966 n01518878 n01519563 n01519873 n01520576 n01521399 n01521756 n01522450	grey skate, gray skate, Raja batis little skate, Raja erinacea thorny skate, Raja radiata barndoor skate, Raja laevis bird dickeybird, dicky-bird, dickybird, dicky-bird fledgling, fledgeling nestling, baby bird cock gamecock, fighting cock hen nester night bird night raven bird of passage archaeopteryx, archeopteryx, Archaeopteryx lithographica archaeornis ratite, ratite bird, flightless bird carinate, carinate bird, flying bird ostrich, Struthio camelus cassowary emu, Dromaius novaehollandiae, Emu novaehollandiae kiwi, apteryx rhea, Rhea americana rhea, nandu, Pterocnemia pennata elephant bird, aepyornis
---	--	---

Inception v3 是用来物体识别的，而我们这次模型训练为场景识别，虽然有所不同，但都需要对图片特征进行提取，而 inception v3 我们迁移学习所用的模型，是让图像先输入 inception v3 模型，经过一系列的卷积池化后，也就是特征提取工作，在最后通过全连接层连接到 1000 个分类之前，会得到图像的特征，我们将这些特征保存到本地文件位置，也就是 bottleneck 这个文件夹下面，每个分类下面会有一个个 txt 文件存储这些特征，以下为 txt 文件存储特征信息内容。



我们的模型是用这些特征信息为输入，10 个不同场景类别为输出，通过训练这个模型的全连接层，而不用重新训练整个模型，节约训练时间，同时由于 inception v3 很好的特征提取功能使模型的预测性能很好，训练模型采用 Tensorflow 官方提供程序，根据自己场景模型需要修改部分内容，得到我们自己的模型，模型为 pb 格式保存。以下为文件内容展示。







































以下展示训练图片集，为 10 个不同场景，以及游乐场的具体图片展示。最后一张是测试集图片。













名称	修改日期	类型
棒球场	2019/12/1 20:08	文件夹
冰场	2019/12/1 20:08	文件夹
橄榄球场	2019/12/1 20:08	文件夹
航站楼	2019/12/1 20:08	文件夹
机舱	2019/12/1 20:08	文件夹
流水线	2019/12/1 20:08	文件夹
停机坪	2019/12/1 20:08	文件夹
舞台	2019/12/1 20:08	文件夹
艺术室	2019/12/1 20:08	文件夹
游乐场	2019/12/1 20:08	文件夹

 0abd1645c54cc 8ca8574e8c68d 01d58f4aa9db2 0.jpg	 0ae7dbf7c3838 f60781fd30bf2 b84236d8a94b ac.jpg	 0b2ec392219c7 d114eba3275b ae6b6f1fcb638 24.jpg	 0b84e822cf9a3 38d7d1e3e940 34993ab54c53f a7.jpg	 0b40257814cfb 2596d2644615 d8828a86c00ab 67.jpg	 0ba9ada395e1a 261cda8965b7 049579e1ee04 876.jpg	 0be7d35b5bf8 5f357ca284bbd fe2757233c69e 6e.jpg	 0c7bebe3fdddc 71aa1b39c75a3 8d5f833494644 b.jpg	 0c7d033d9277 d2eafd3d0dd0 49d6457af894f cd0.jpg
 0e0e615b4edd 684e3049648ae d875b7adaf3eb ba.jpg	 0e37eb21169a e143e9e09dda cbc06dea049fe fee.jpg	 0e5293738679c bb67529bc66a 4fbed72de7e2a fa.jpg	 0ee5232c2459e 9b8a14ab5f095 369f7c1e8df92 3.jpg	 0f1d5170a8b06 e13c60fa90799 10bc851dd823f 4.jpg	 0f86b898cd2fd 2198a5f458883 f35c79b6c3a75 5.jpg	 01f0fc691c28fd 906337e6bd7b 2109965dc87d bc.jpg	 1a0bf89e3fd28 7811d3c67e55 15e22d5b52f2d d9.jpg	 1ac4d44d4a22 d0a85c347b0e efc34304994e6 2b3.jpg
 1bd4d7850390 7debc0a30c230 b90aef101aa1 9e.jpg	 1bde6535046b c7479ab447ffd 42bead6ed387 123.jpg	 1c537dd13840 bdd6a01a257a 2f85417e7aed3 d76.jpg	 1d34bedc2480 3e3fe633c515a 058a2718d676 dc1.jpg	 1df726762cbea e90c9eafdb776 43cb7ca0569bc 1.jpg	 1e2e4e7076f5b 87b3c21486d5 1fdd4b599f5ae 19.jpg	 1e4bdca449aa4 9321f27f18d4e 86e0843c2adf0 5.jpg	 1ebf041be2a3e ee1cc16d22f6f 77489874729c1 1.jpg	 1f2a1afb732e4 7689dbe4ab4b 246e22890b71 23b.jpg
 1f627803dc8e9	 1fc35aa8edd32	 1fcb1bba6cb3c	 002af7017528c	 2a3f0f64d74cac	 2a10aa2567e8a	 2a078a84c3df3f	 2ac85dac1c5a9	 2b5c641ec2c6d

 棒球场.jpg	 冰场.jpg	 橄榄球场.jpg	 航站楼.jpg	 机舱.jpg	 流水线.jpg	 停机坪.jpg	 舞台.jpg	 艺术室.jpg
 游乐场.jpg								

以下为测试结果展示，显示图片内容，并给出了对应场景的得分置信度。对于测试结果来看效果不错，而且训练模型时间大大压缩，总共花费大约半小时，相对于 inception v3 的训练周期来说，时间的成本大大降低。

images/停机坪. jpg



停机坪 (score = 0.89911)  
机舱 (score = 0.02417)  
游乐场 (score = 0.01591)  
航站楼 (score = 0.01387)  
流水线 (score = 0.00900)



流水线 (score = 0.71104)  
艺术室 (score = 0.15505)  
航站楼 (score = 0.04694)  
游乐场 (score = 0.03025)  
机舱 (score = 0.01274)  
舞台 (score = 0.01131)

---

## 2.2 测试程序

```
import tensorflow as tf
import os
import numpy as np
import re
from PIL import Image
import matplotlib.pyplot as plt
lines = tf.gfile.GFile('output_labels.txt').readlines()
uid_to_human = {}

# 一行一行读取数据
for uid,line in enumerate(lines) :
    #去掉换行符
    line=line.strip('\n')
    uid_to_human[uid] = line

# 分类编号变成描述
def id_to_string(node_id):
    if node_id not in uid_to_human:
        return ''
    return uid_to_human[node_id]

# 创建一个图来存放训练好的模型
with tf.gfile.GFile('output_graph.pb', 'rb') as f:
    graph_def = tf.GraphDef()
    graph_def.ParseFromString(f.read())
    tf.import_graph_def(graph_def, name='')

with tf.Session() as sess:
    # final_result 为输出 tensor 的名字
    softmax_tensor = sess.graph.get_tensor_by_name('final_result:0')
    # 遍历目录
    for root,dirs,files in os.walk('images/'):
        for file in files:
            # 载入图片
            image_data = tf.gfile.GFile(os.path.join(root,file), 'rb').
read()

            # 把图像数据传入模型获得模型输出结果
            predictions = sess.run(softmax_tensor,{ 'DecodeJpeg/contents
:0': image_data})
            # 把结果转为 1 维数据
            predictions = np.squeeze(predictions)
```



---

```
# 打印图片路径及名称
image_path = os.path.join(root, file)
print(image_path)
# 显示图片
img=Image.open(image_path)
plt.imshow(img)
plt.axis('off')
plt.show()

# 排序
top_k = predictions.argsort()[::-1]
for node_id in top_k:
    # 获取分类名称
    human_string = id_to_string(node_id)
    # 获取该分类的置信度
    score = predictions[node_id]
    print('%s (score = %.5f)' % (human_string, score))
print()
```

---

## 第 3 章 目标识别模型设计

### 3.1 Inception Net-V3 概述

#### 1. 概述

Google Inception Net 在 2014 年的 ImageNet Large Scale Visual Recognition Competition (ILSVRC) 中取得第一名, 该网络以结构上的创新取胜, 通过采用全局平均池化层取代全连接层, 极大的降低了参数量, 是非常实用的模型, 一般称该网络模型为 Inception V1。随后的 Inception V2 中, 引入了 Batch Normalization 方法, 加快了训练的收敛速度。在 Inception V3 模型中, 通过将二维卷积层拆分成两个一维卷积层, 不仅降低了参数数量, 同时减轻了过拟合现象。

#### 2. 特性

v3 一个最重要的改进是分解 (Factorization), 将  $7 \times 7$  分解成两个一维的卷积 ( $1 \times 7, 7 \times 1$ ),  $3 \times 3$  也是一样 ( $1 \times 3, 3 \times 1$ ), 这样的好处, 既可以加速计算 (多余的计算能力可以用来加深网络), 又可以将 1 个 conv 拆成 2 个 conv, 使得网络深度进一步增加, 增加了网络的非线性, 还有值得注意的地方是网络输入从  $224 \times 224$  变为了  $299 \times 299$ , 更加精细设计了  $35 \times 35 / 17 \times 17 / 8 \times 8$  的模块。

2015 年 12 月, 该团队发布 Inception 模块和类似架构的一个新版本 V3。该论文更好地解释了原始的 GoogLeNet 架构, 在设计选择上给出了更多的细节。原始思路如下:

通过谨慎建筑网络, 平衡深度与宽度, 从而最大化进入网络的信息流。在每次池化之前, 增加特征映射。

当深度增加时, 网络层的深度或者特征的数量也系统性的增加。使用每一层深度增加在下一层之前增加特征的结合。

而 Inception V3 网络则主要有两方面的改造:

一、引入了 Factorization into small convolutions 的思想, 将一个较大的二维卷积拆成两个较小的一维卷积, 比如将  $7'7$  卷积拆成  $1'7$  卷积和  $7'1$  卷积, 或者将  $3'3$  卷积拆成  $1'3$  卷积和  $3'1$  卷积, 如图 1 所示。

一方面节约了大量参数, 加速运算并减轻了过拟合 (比将  $7'7$  卷积拆成  $1'7$  卷积和  $7'1$  卷积, 比拆成 3 个  $3'3$  卷积更节约参数),

一方面增加了一层非线性扩展模型表达能力。论文中指出, 这种非对称的卷积结构拆分, 其结果比对称地拆为几个相同的小卷积核效果更明显, 可以处理更多、更丰富的空间特征, 增加特征多样性。

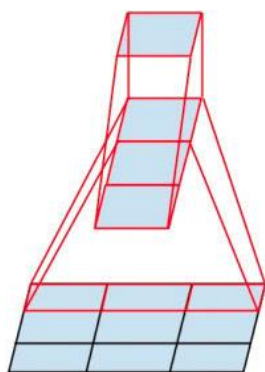


图 1 将一个  $3 \times 3$  卷积拆成  $1 \times 3$  卷积和  $3 \times 1$  卷积

只使用  $3 \times 3$  的卷积，可能的情况下给定的  $5 \times 5$  和  $7 \times 7$  过滤器能分成多个  $3 \times 3$ 。

二、Inception V3 优化了 Inception Module 的结构，现在 Inception Module 有  $35 \times 35$ 、 $17 \times 17$  和  $8 \times 8$  三种不同结构，如图 3 所示。这些 Inception Module 只在网络的后部出现，前部还是普通的卷积层。并且 Inception V3 除了在 Inception Module 中使用分支，还在分支中使用了分支（ $8 \times 8$  的结构中），可以说是 Network In Network In Network。

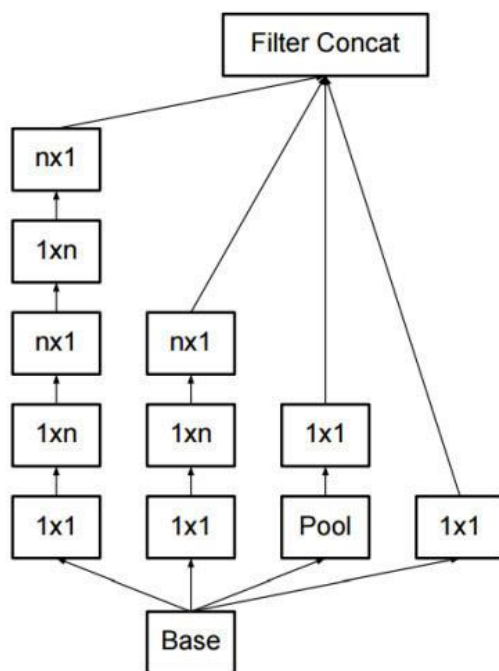
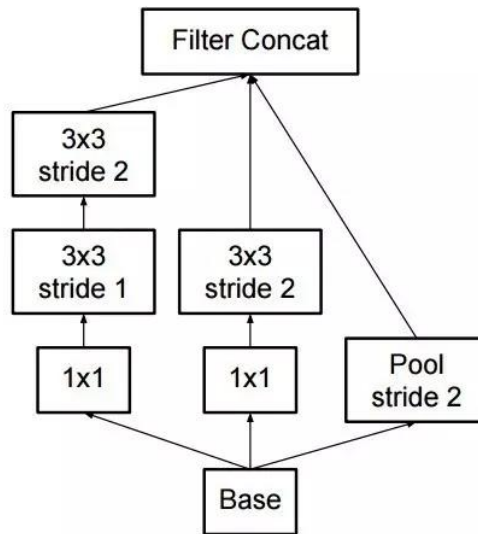


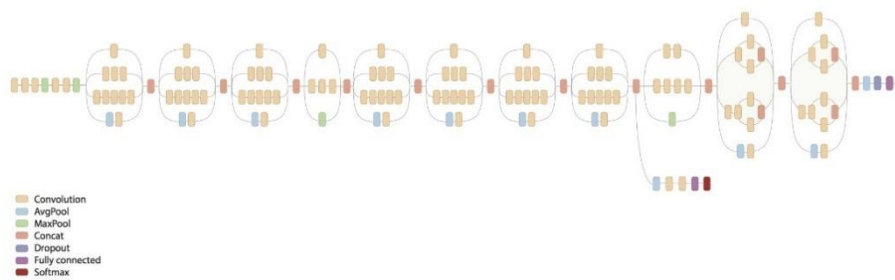
图 2 Inception V3 中三种结构的 Inception Module

在进行 inception 计算的同时，Inception 模块也能通过提供池化降低数据的大小。这基本类似于在运行一个卷积的时候并行一个简单的池化层：



Inception 也使用一个池化层和 softmax 作为最后的分类器。

### 3. 网络结构



类型	kernel 尺寸/步长（或注释）	输入尺寸
卷积	3×3 / 2	299×299×3
卷积	3×3 / 1	149×149×32
卷积	3×3 / 1	147×147×32
池化	3×3 / 2	147×147×64
卷积	3×3 / 1	73×73×64
卷积	3×3 / 2	71×71×80
卷积	3×3 / 1	35×35×192
Inception 模块组	3 个 Inception Module	35×35×288
Inception 模块组	5 个 Inception Module	17×17×768
Inception 模块组	3 个 Inception Module	8×8×1280
池化	8×8	8×8×2048
线性	logits	1×1×2048
Softmax	分类输出	1×1×1000

表 1 Inception V3 网络结构

## 3.2 处理程序

```
class NodeLookup(object):
    def __init__(self, label_lookup_path=None, uid_lookup_path=None):
        if not label_lookup_path:
            label_lookup_path = os.path.join(model_dir, 'imagenet_2012_challenge_label_map_proto.pbtxt')
        if not uid_lookup_path:
            uid_lookup_path = os.path.join(model_dir, 'imagenet_synset_to_human_label_map.txt')
        self.node_lookup = self.load(label_lookup_path, uid_lookup_path)

    def load(self, label_lookup_path, uid_lookup_path):
        if not tf.gfile.Exists(uid_lookup_path):
            tf.logging.fatal('File does not exist %s', uid_lookup_path)
        if not tf.gfile.Exists(label_lookup_path):
            tf.logging.fatal('File does not exist %s', label_lookup_path)
        # Loads mapping from string UID to human-readable string
        proto_as_ascii_lines = tf.gfile.GFile(uid_lookup_path).readlines()
        uid_to_human = {}
        p = re.compile(r'[n\d]*[ \S,]*')
        for line in proto_as_ascii_lines:
            parsed_items = p.findall(line)
            uid = parsed_items[0]
            human_string = parsed_items[2]
            uid_to_human[uid] = human_string
        # Loads mapping from string UID to integer node ID.
        node_id_to_uid = {}
        proto_as_ascii = tf.gfile.GFile(label_lookup_path).readlines()
        for line in proto_as_ascii:
            if line.startswith(' target_class:'):
                target_class = int(line.split(':')[1])
            if line.startswith(' target_class_string:'):
                target_class_string = line.split(':')[1]
                node_id_to_uid[target_class] = target_class_string[1:-2]
        # Loads the final mapping of integer node ID to human-readable string
        node_id_to_name = {}
        for key, val in node_id_to_uid.items():
            if val not in uid_to_human:
                tf.logging.fatal('Failed to locate: %s', val)
            name = uid_to_human[val]
            node_id_to_name[key] = name
        return node_id_to_name
```

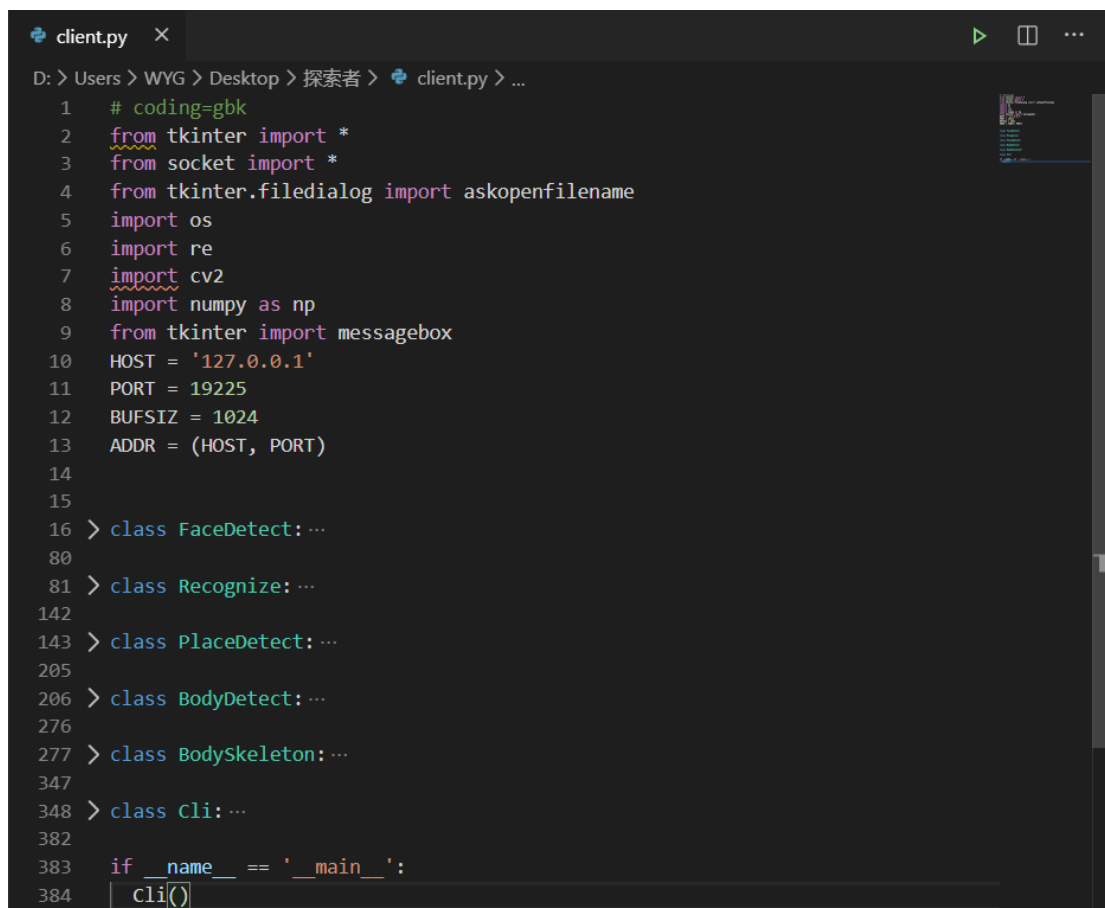
---

```
def id_to_string(self, node_id):  
    if node_id not in self.node_lookup:  
        return ''  
    return self.node_lookup[node_id]
```

## 第4章 客户端设计

### 4.1 总体概述

客户端的总体设计如下所示



```
client.py x
D: > Users > WYG > Desktop > 探索者 > client.py > ...
1  # coding=gbk
2  from tkinter import *
3  from socket import *
4  from tkinter.filedialog import askopenfilename
5  import os
6  import re
7  import cv2
8  import numpy as np
9  from tkinter import messagebox
10 HOST = '127.0.0.1'
11 PORT = 19225
12 BUFSIZ = 1024
13 ADDR = (HOST, PORT)
14
15
16 > class FaceDetect: ...
80
81 > class Recognize: ...
142
143 > class PlaceDetect: ...
205
206 > class BodyDetect: ...
276
277 > class BodySkeleton: ...
347
348 > class Cli: ...
382
383 if __name__ == '__main__':
384     Cli()
```

本设计使用到了 tkinter 来完成 GUI 设计，使用 socket 进行客户端与服务端的交互，使用 tkinter.filedialog 来打开文件选择对话框，使用 opencv 为客户端准备输出的图片进行处理。

一共设计了 6 个类，分别是场景识别、目标识别、人脸识别、人体识别、人体关键点识别、以及客户端类，分别对应场景识别、目标识别、人脸识别、人体识别、人体关键点识别的客户端任务、其中客户端类实现了基础界面生成的一些任务。

### 4.2 场景识别类

场景识别类主要有这样四个成员函数，对其他的类前三个也是类似的，第四个函数每个类都有自己的处理方法，如下图所示，下面我来一一介绍，介绍以注释形式给出



```
143 class PlaceDetect:
144 > def __init__(self): ...
159
160 > def processBrowse(self,e): ...
164
165 > def processDelivery(self,e): ...
187
188 > def processResult(self): ...
205
```

## 1. 初始化

```
def __init__(self):
    # 建立一个 tkinter 窗口
    window=Tk()
    window.title("场景识别")
    # 在窗口上建立文本框
    self.text=Text(window,width=100,height=20)
    self.text.pack()
    self.Var=StringVar()
    frame1=Frame(window)
    frame1.pack()
    # 标签提示
    Label(frame1,text="Select a picture:").grid(row=1,column=1,sticky=W)
    # 建立了一个输入框
    e=Entry(frame1,textvariable=self.Var,width=40)
    e.grid(row=1,column=2)
    # 生成三个按钮 在 frame1 上生成 点击时调用对应函数 生成位置
    Button(frame1,text="浏览",
    ",command=lambda:self.processBrowse(e)).grid(row=1,column=3)
    Button(frame1,text="确认发送",
    ",command=lambda:self.processDelivery(e)).grid(row=1,column=4)
    Button(frame1,text="识别结果",
    ",command=self.processResult).grid(row=1,column=5)
    # 窗口循环
    window.mainloop()
```

## 2. 文件浏览

```
def processBrowse(self,e):
    # 清空输入框
    e.delete(0,END)
    # 调用打开文件窗口
    m=askopenfilename()
    # 将文件名插入输入框
    e.insert(0,m)
```

### 3. 文件传输

```
def processDelivery(self,e):
    # 获取文件输入框的文件路径
    self.filename=e.get()
    # 判断文件路径是否存在
    if os.path.exists(self.filename)==0:
        messagebox.showinfo("Tip","The file is not exist!")
        return 0
    # 判断是否有 jpg 图片文件
    elif re.search('\.jpg',self.filename) is None:
        messagebox.showinfo("Tip","The file is not picture.jpg!")
        return
    tcpCliSock = socket(AF_INET, SOCK_STREAM)
    # 连接到服务器地址
    tcpCliSock.connect(ADDR)
    # 发送字符串'发送图片 mode'让服务端知道工作模式
    tcpCliSock.send('发送图片 mode'.encode())
    # 如果接受到服务端的确认就在文本框显示准备发送图片
    if tcpCliSock.recv(BUFSIZ).decode()=='OK':
        self.text.insert(INSERT,'准备发送图片\n')
    myfile = open(self.filename, 'rb')
    # 将文件读入 data 中
    data = myfile.read()
    # 获取文件长度
    size = str(len(data))
    tcpCliSock.send(size.encode())
    rec = tcpCliSock.recv(BUFSIZ).decode()
    # 发送图片
    tcpCliSock.send(data)
    if tcpCliSock.recv(BUFSIZ).decode()=='OK':
        self.text.insert(INSERT,'传输图片完成\n')
    # 关闭连接
    tcpCliSock.close()
```

### 4. 分析结果

```
def processResult(self):
    # 发送工作模式给服务端
    tcpCliSock = socket(AF_INET, SOCK_STREAM)
    tcpCliSock.connect(ADDR)
    tcpCliSock.send('场景识别 mode'.encode())
    # 接受到服务端发来的信息
    rec=tcpCliSock.recv(BUFSIZ).decode()
    # 关闭连接
```

```

tcpCliSock.close()
# 服务端通过'!'区分隔开为字符串，客户端通过'!'去除隔开，转换为列表
list=rec.split('!')
# 查看列表信息，用来调试程序
print(list)
# 使用 opencv 打开图片
img = cv2.imdecode(np.fromfile(self.filename,dtype=np.uint8),-1)
# 新建 tkinter 窗口
window=Tk()
window.title("识别信息")
canvas=Canvas(window,width=250,height=600,bg='white')
canvas.pack()
# 输出列表的信息
for i in range(0,len(list),2):
    canvas.create_text(125,30+30*i,text=list[i]+'-'+list[i+1],font=('Times',20))
# 显示图片
cv2.imshow("Image", img)
cv2.waitKey (0)

```

### 4.3 目标识别类

由于前三个函数基本上都一样，这里就不再赘述

下面我们来讲分析结果的函数

```

def processResult(self):
    # 发送工作模式给服务端
    tcpCliSock = socket(AF_INET, SOCK_STREAM)
    tcpCliSock.connect(ADDR)
    tcpCliSock.send('目标识别 mode'.encode())
    # 接受到服务端发来的信息
    rec=tcpCliSock.recv(BUFSIZ).decode()
    tcpCliSock.close()
    # 服务端通过'!'区分隔开为字符串，客户端通过'!'去除隔开，转换为列表
    list=rec.split('!')
    # 使用 opencv 打开图片
    img = cv2.imdecode(np.fromfile(self.filename,dtype=np.uint8),-1)
    window=Tk()
    window.title("Information")
    canvas=Canvas(window,width=1400,height=700,bg='white')
    canvas.pack()
    # 输出列表的信息
    for i in range(0,len(list),2):
        canvas.create_text(700,100+60*i,text=list[i]+'-'+list[i+1],font=('Times',40))
    # 显示图片

```

```
cv2.imshow("Image", img)
cv2.waitKey (0)
```

可以看出来操作和前面的过程差不多

#### 4.4 人脸识别类

```
def processResult(self):
    # 发送工作模式给服务端
    tcpCliSock = socket(AF_INET, SOCK_STREAM)
    tcpCliSock.connect(ADDR)
    tcpCliSock.send('人脸识别 mode'.encode())
    # 接受到服务端发来的信息
    rec = tcpCliSock.recv(BUFSIZ).decode()
    # 判断没有脸的情况
    if rec=='No face':
        tcpCliSock.close()
        messagebox.showinfo("Tip","No face in the picture!")
    # 有脸的情况
    else:
        # 去除','形成列表
        list=rec.split(',')
        tcpCliSock.close()
        # 使用 opencv 解码图片
        img = cv2.imdecode(np.fromfile(self.filename,dtype=np.uint8),-1)
        vis = img.copy()
        # 对于每张脸有 7 个参数，分别是矩形框左上角的坐标(l,t)，矩形框的高 h 和宽 w，
        # 脸所对应的性别、年龄、颜值得分
        # for 循环可以自动根据 len(list)/7 确定脸数
        for i in range(int(len(list)/7)):
            # 获取人脸矩形框参数
            t,l,w,h=int(list[0+7*i]),int(list[1+7*i]),int(list[2+7*i]),int(list
[3+7*i])
            # 输入矩形框左上角以及右下角的坐标参数
            cv2.rectangle(vis, (l, t), (l+w, t+h),(255, 0, 0), 2)
            # 在每一张脸矩形框的左上角标注性别年龄颜值打分信息
            cv2.putText(vis, 'gender: '+list[4+7*i]+' '+'age: '+list[5+7*i]+' '+'
score: '+list[6+7*i], (l, t), cv2.FONT_HERSHEY_COMPLEX, 0.5, (255, 255,
0), 1)
            # 显示处理过的图片
            cv2.imshow("Image", vis)
            cv2.waitKey (0)
```

## 4.5 人体识别类

```
def processResult(self):
    # 发送工作模式给服务端
    tcpCliSock = socket(AF_INET, SOCK_STREAM)
    tcpCliSock.connect(ADDR)
    tcpCliSock.send('人体识别 mode'.encode())
    # 接受到服务端发来的信息
    rec = tcpCliSock.recv(BUFSIZ).decode()
    # 判断没有脸的情况
    if rec=='No':
        tcpCliSock.close()
        messagebox.showinfo("Tip","No face in the picture!")
    else:
        # 去除','形成列表
        list=rec.split(',')
        tcpCliSock.close()
        print(list)
        # 使用 opencv 解码图片
        img = cv2.imdecode(np.fromfile(self.filename,dtype=np.uint8),-1)
        vis = img.copy()
        # 新建 tk 窗口
        window=Tk()
        window.title("Information")
        # 建一个画布
        canvas=Canvas(window,width=300,height=300,bg='white')
        canvas.pack()
        # 输出三条信息，分别是男性概率，上衣颜色，下衣颜色
        canvas.create_text(150,50,text=''.join(['男性概率:',list[0]]),font=('Times',20))
        canvas.create_text(150,150,text=''.join(['上衣颜色:',list[1]]),font=('Times',20))
        canvas.create_text(150,250,text=''.join(['下衣颜色:',list[2]]),font=('Times',20))
        # 输入矩形框左上角以及右下角的坐标参数
        cv2.rectangle(vis, (int(list[6]), int(list[4])), (int(list[6])+int(list[3]), int(list[4])+int(list[5])),(255, 0, 0), 2)
        # 标注性别概率信息
        cv2.putText(vis, 'male rate: '+list[0], (int(list[6]), int(list[4])), cv2.FONT_HERSHEY_COMPLEX, 0.5, (0, 255, 0), 1)
        # 显示处理过的图片
        cv2.imshow("Image", vis)
        cv2.waitKey (0)
```

## 4.6 人体关键点识别类

```
def processResult(self):
    # 发送工作模式给服务端
    tcpCliSock = socket(AF_INET, SOCK_STREAM)
    tcpCliSock.connect(ADDR)
    tcpCliSock.send('人体关键点识别 mode'.encode())
    # 接收到服务端发来的信息
    rec = tcpCliSock.recv(BUFSIZ).decode()
    # 判断没有人体的情况
    if rec=='No':
        tcpCliSock.close()
        messagebox.showinfo("Tip","No!")
    # 有人体的情况
    else:
        # 去除','形成列表
        list=rec.split(',')
        # 关闭 tcp 连接
        tcpCliSock.close()
        # 输出 list 供调试使用
        print(list)
        # 输出 list 长度
        print(len(list))
        # 使用 opencv 解码图片
        img = cv2.imdecode(np.fromfile(self.filename,dtype=np.uint8),-1)
        vis = img.copy()
        # 关键点名称集合（列表）
        skeletonName = ['head','neck','left_shoulder','left_elbow','left_hand',
            'right_shoulder','right_elbow','right_hand','left_buttocks','left_knee',
            'left_foot','right_buttocks','right_knee','right_foot']
        # 对于每个人体有 32 个参数，分别是矩形框左上角的坐标(l,t)，矩形框的高 h 和宽 w
        # 以及十四个关键点所对应的 x, y 坐标 32=4+2*14
        # 这里的坐标是相对于矩形框的坐标
        # for 循环可以自动根据 len(list)/32 确定人体数
        for i in range(int(len(list)/32)):
            # 获取人体矩形框参数
            t,l,w,h=int(list[0+32*i]),int(list[1+32*i]),int(list[2+32*i]),int(list[3+32*i])
            # 输入矩形框左上角以及右下角的坐标参数
            cv2.rectangle(vis, (l, t), (l+w, t+h),(255, 0, 0), 2)
            # 通过循环来标注 14 个关键点的位置以及名称
            for j in range(14):
                # 通过关键点的相对矩形框坐标生成相当于图片的坐标
                pos=(l+int(list[4+2*j+32*i]),t+int(list[5+2*j+32*i]))
```

```
    # 用小圈标注出关键点的位置
    cv2.circle(vis, pos, 5, color=(0, 255, 0))
    # 标注名称信息
    cv2.putText(vis, skeletonName[j], pos, cv2.FONT_HERSHEY_COMPLEX,
0.3, (255, 255, 0), 1)
    # 显示标注完的图片
    cv2.imshow("Image", vis)
    cv2.waitKey (0)
```

## 4.7 客户端类

```
class Cli:
    def __init__(self):
        # 新建窗口（客户端的基础窗口）
        window=Tk()
        window.title("人工智能-期末项目-客户端")
        # 新建文本框
        self.text=Text(window,width=100,height=20)
        self.text.pack()
        # 按钮的一排
        frame1=Frame(window)
        frame1.pack()
        # 5 个按钮
        Button(frame1,text="人脸识别",
",command=self.processFaceDetect).grid(row=1,column=1)
        Button(frame1,text="目标识别",
",command=self.processRecognize).grid(row=1,column=3)
        Button(frame1,text="人体识别",
",command=self.processBodyDetect).grid(row=1,column=5)
        Button(frame1,text="人体关键点识别",
",command=self.processBodySkeleton).grid(row=1,column=7)
        Button(frame1,text="场景识别",
",command=self.processPlaceDetect).grid(row=1,column=9)
        window.mainloop()

    def processFaceDetect(self):
        self.text.insert(INSERT,'人脸识别开启\n')
        # FaceDetect 对象实例化
        FaceDetect()

    def processRecognize(self):
        self.text.insert(INSERT,'目标识别开启\n')
        # Recognize 对象实例化
        Recognize()
```



---

```
def processBodyDetect(self):
    self.text.insert(INSERT, '人体识别开启\n')
    # BodyDetect 对象实例化
    BodyDetect()

def processBodySkeleton(self):
    self.text.insert(INSERT, '人体关键点识别开启\n')
    # BodySkeleton 对象实例化
    BodySkeleton()

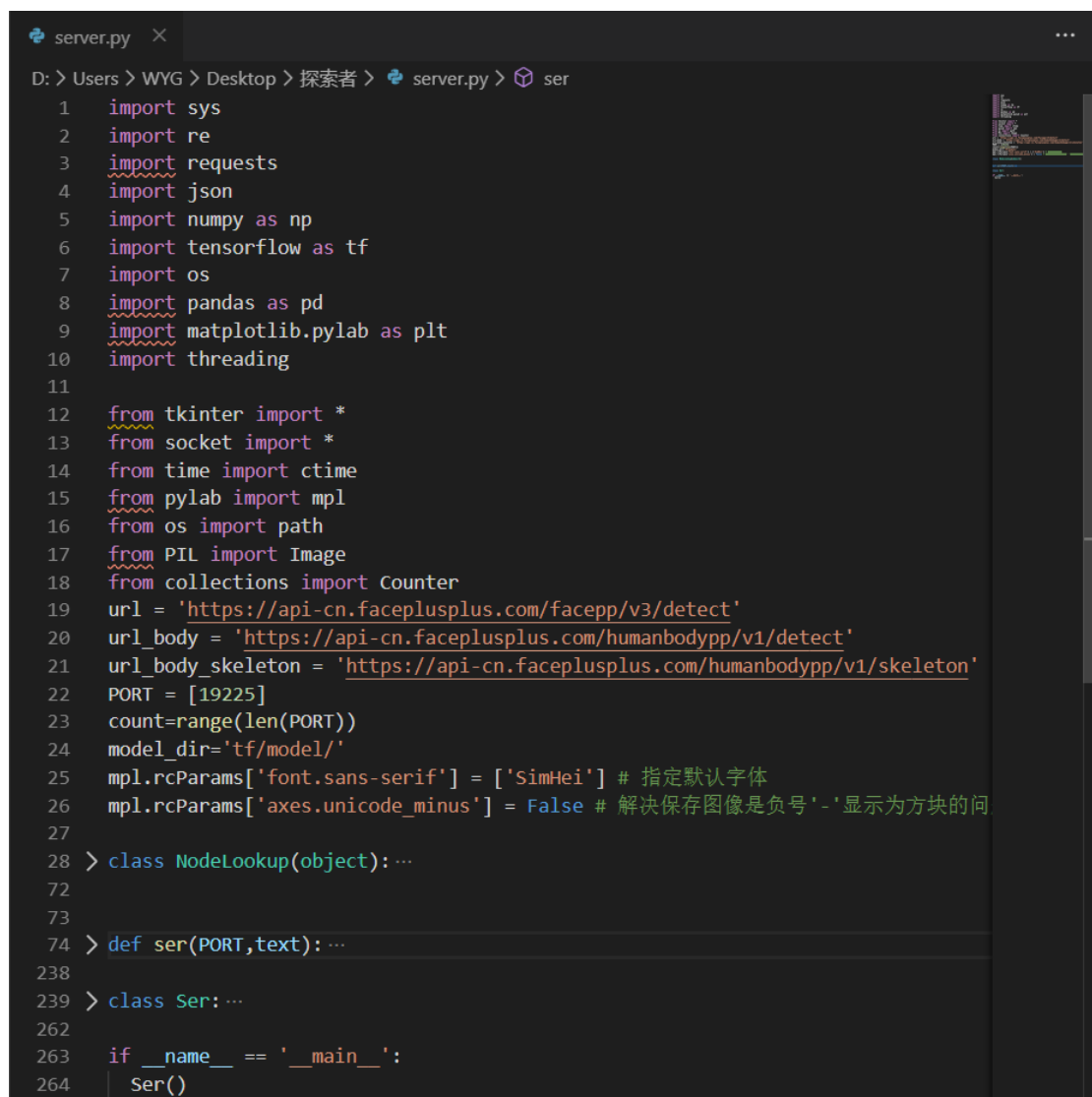
def processPlaceDetect(self):
    self.text.insert(INSERT, '场景识别开启\n')
    # PlaceDetect 对象实例化
    PlaceDetect()

# 如果运行本文件，而不是当包导入，那么就执行下面的程序
if __name__ == '__main__':
    Cli()
```

## 第5章 服务端设计

### 5.1 总体概述

服务端的总体设计如下所示



```
server.py
D: > Users > WYG > Desktop > 探索者 > server.py > ser
1  import sys
2  import re
3  import requests
4  import json
5  import numpy as np
6  import tensorflow as tf
7  import os
8  import pandas as pd
9  import matplotlib.pyplot as plt
10 import threading
11
12 from tkinter import *
13 from socket import *
14 from time import ctime
15 from pylab import mpl
16 from os import path
17 from PIL import Image
18 from collections import Counter
19 url = 'https://api-cn.faceplusplus.com/facepp/v3/detect'
20 url_body = 'https://api-cn.faceplusplus.com/humanbodypp/v1/detect'
21 url_body_skeleton = 'https://api-cn.faceplusplus.com/humanbodypp/v1/skeleton'
22 PORT = [19225]
23 count=range(len(PORT))
24 model_dir='tf/model/'
25 mpl.rcParams['font.sans-serif'] = ['SimHei'] # 指定默认字体
26 mpl.rcParams['axes.unicode_minus'] = False # 解决保存图像是负号 '-' 显示为方块的问题
27
28 > class NodeLookup(object):...
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74 > def ser(PORT,text):...
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139 > class Ser:...
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163 if __name__ == '__main__':
164     Ser()
```

本设计使用到了 tkinter 来完成 GUI 设计，使用 socket 进行客户端与服务端的交互，使用 json 来处理 api 返回的 json 数据，使用 mpl 解决字体问题，还列出了一些 api 地址，模型存放的相对路径，端口号 19225

主要的功能在 ser 函数里面实现，通过一共分支判断语句来处理，分别是发送图片、场景识别、目标识别、人脸识别、人体识别、人体关键点识别模式，分别对应场景识别、目标识别、人脸识别、人体识别、人体关键点识别的服务端任务。

```

74 def ser(PORT,text):
75     # 使用本地环回ip进行测试，正式工作使用服务端的IP地址
76     HOST = '127.0.0.1'
77     BUFSIZ = 1024
78     ADDR = (HOST, PORT)
79     tcpSerSock = socket(AF_INET, SOCK_STREAM)
80     tcpSerSock.bind(ADDR)
81     tcpSerSock.listen(5)
82     while True:
83         # 等待连接
84         print('port ',PORT,' is waiting for connection...')
85         tcpCliSock, addr = tcpSerSock.accept()
86         # 传入工作模式信息
87         print('port ',PORT,' connected from:', addr)
88         data = tcpCliSock.recv(BUFSIZ).decode()
89 > if data=='发送图片mode': ...
102
103 > elif data=='人脸识别mode': ...
127
128 > elif data=='人体识别mode': ...
148
149 > elif data=='人体关键点识别mode': ...
172
173 > elif data=='目标识别mode': ...
200
201 > elif data=='场景识别mode': ...
243

```

下面就不同分支的处理展开分析

## 5.2 场景识别部分

```

elif data=='场景识别 mode':
    # 确认端口号
    if PORT==19225:
        text.insert(INSERT,'对客户端发送的图片进行场景识别\n')
        # 获取标签名称，按行读入
        lines = tf.gfile.GFile('output_labels.txt').readlines()
        uid_to_human = {}
        # 一行一行读取数据
        for uid,line in enumerate(lines) :
            #去掉换行符
            line=line.strip('\n')
            uid_to_human[uid] = line
        # 分类编号变成描述
        def id_to_string(node_id):
            if node_id not in uid_to_human:
                return ''
            return uid_to_human[node_id]
        # 创建一个图来存放训练好的模型
        with tf.gfile.GFile('tf/model/output_graph.pb', 'rb') as f:
            graph_def = tf.GraphDef()

```

```

graph_def.ParseFromString(f.read())
tf.import_graph_def(graph_def, name='')
with tf.Session() as sess:
    # final_result 为输出 tensor 的名字
    softmax_tensor = sess.graph.get_tensor_by_name('final_result:0'
)

    # 载入图片
    image_data = tf.gfile.GFile('c.jpg', 'rb').read()
    # 把图像数据传入模型获得模型输出结果
    predictions = sess.run(softmax_tensor, {'DecodeJpeg/contents:0':
image_data})
    # 把结果转为 1 维数据
    predictions = np.squeeze(predictions)
    # 排序
    top_k = predictions.argsort()[::-1]
    list = []
    for node_id in top_k:
        # 获取分类名称
        human_string = id_to_string(node_id)
        # 获取该分类的置信度
        score = predictions[node_id]
        # 通过列表存储置信度信息
        list+=(human_string, str(int(100*score))+'%')
    sess.close()
    # 列表转字符串
    data='!'.join(list)
    # 发送字符串
    tcpCliSock.send(data.encode())
    # 关闭连接
    tcpCliSock.close()

```

### 5.3 目标识别部分

```

elif data=='目标识别 mode':
    # 确认端口号
    if PORT==19225:
        text.insert(INSERT, '对客户端发送的图片进行目标识别\n')
        image_data = tf.gfile.FastGFile('c.jpg', 'rb').read()
        # 创建图
        with tf.gfile.FastGFile(os.path.join(model_dir, 'classify_image_gr
aph_def.pb'), 'rb') as f:
            graph_def = tf.GraphDef()
            graph_def.ParseFromString(f.read())
            tf.import_graph_def(graph_def, name='')
        #启动会话

```

```

sess=tf.Session()
#Inception-v3 模型的最后一层 softmax 的输出
softmax_tensor= sess.graph.get_tensor_by_name('softmax:0')
#输入图像数据, 得到 softmax 概率值 (一个 shape=(1,1008)的向量)
predictions = sess.run(softmax_tensor,{'DecodeJpeg/contents:0': image_data})
predictions = np.squeeze(predictions)
node_lookup = NodeLookup()
#取出前 5 个概率最大的值 (top-5)
top_5 = predictions.argsort()[-5:][::-1]
list=[]
for node_id in top_5:
    human_string = node_lookup.id_to_string(node_id)
    score = predictions[node_id]
    score = '%.5f'%score
    list.append(human_string+'!'+score+'!')
sess.close()
# 列表转字符串通过空格隔开
data=' '.join(list)[: -1]
tcpCliSock.send(data.encode())
tcpCliSock.close()

```

## 5.4 人脸识别部分

```

elif data=='人脸识别 mode':
    text.insert(INSERT,'对客户端发送的图片进行人脸检测\n')
    # 字典方式对打开本地文件操作录入
    files = {'image_file':open('c.jpg', 'rb')}
    # 调用 api 所需要的参数
    payload = {'api_key': 'aiyUZ7KnALYTZgI8ekWI-jnpxCCI2s5z',
               'api_secret': 'jxdfCWIcFP8fZQmVQRhI-fnSFa10_iwp',
               'return_landmark': 0,
               'return_attributes': 'gender,age,glass,beauty'}
    # 通过本地文件调用接口
    r = requests.post(url,files=files,data=payload)
    # 将返回的 json 文件转化为 python 的字典
    data=json.loads(r.text)
    # 输出 r.text 供调试使用
    print (r.text)
    # 新建字符串 sdata
    sdata = ""
    # 如果至少有一张脸
    if data["faces"]:
        # 对所有的脸迭代一次
        for i in range(len(data["faces"])):

```

```

        # 获取脸的性别信息
        gender=data['faces'][i]['attributes']['gender']['value']
        # 获取脸的年龄信息
        age=data['faces'][i]['attributes']['age']['value']
        # 脸的评分通过对男女评分的平均值表示
        score=(data['faces'][i]['attributes']['beauty']['female_score']
+data['faces'][0]['attributes']['beauty']['male_score'])/2
        # 保留小数点后两位
        score='%.2f'%score
        # 脸的矩形框信息分别是矩形框左上角的坐标(l,t)，矩形框的高 h 和宽 w
        width= data['faces'][i]['face_rectangle']['width']
        top= data['faces'][i]['face_rectangle']['top']
        height= data['faces'][i]['face_rectangle']['height']
        left= data['faces'][i]['face_rectangle']['left']
        # 将上述信息通过','隔开成字符串，存入 sdata
        sdata=sdata+", ".join([str(top),str(left),str(width),str(height)
,gender,str(age),score])+', '
        # 通过 tcp 发送出去
        tcpCliSock.send(sdata.encode())
    else: tcpCliSock.send('No face'.encode())

```

## 5.5 人体识别部分

```

elif data=='人体识别 mode':
    # 字典方式对打开本地文件操作录入
    files = {'image_file':open('c.jpg', 'rb')}
    # 调用 api 所需要的参数
    payload = {'api_key': 'aiyUZ7KnALYTZgI8ekWI-jnpxCCI2s5z',
               'api_secret': 'jxdfCWicFP8fZQmVQRhI-fnSFa10_iwp',
               'return_attributes':'gender,upper_body_cloth,lower_body_cloth'}
    # 通过本地文件调用接口
    r = requests.post(url_body,files=files,data=payload)
    # 将返回的 json 文件转化为 python 的字典
    data=json.loads(r.text)
    # 输出 r.text 供调试使用
    print (r.text)
    # 如果至少识别出一个人体
    if data["humanbodies"]:
        # 性别信息
        gender=data['humanbodies'][0]['attributes']['gender']['male']
        # 上衣颜色
        upper_body_cloth=data['humanbodies'][0]['attributes']['upper_body_cloth']['upper_body_cloth_color']
        # 下衣颜色

```

```

        lower_body_cloth=data['humanbodies'][0]['attributes']['lower_body_cloth']['lower_body_cloth_color']
        # 人体的矩形框信息分别是矩形框左上角的坐标(l,t)，矩形框的高 h 和宽 w
        width= data['humanbodies'][0]['humanbody_rectangle']['width']
        top= data['humanbodies'][0]['humanbody_rectangle']['top']
        height= data['humanbodies'][0]['humanbody_rectangle']['height']
        left= data['humanbodies'][0]['humanbody_rectangle']['left']
        # 将上述信息通过', '隔开成字符串，存入 sdata
        data=",".join([str(gender),str(upper_body_cloth),str(lower_body_cloth),str(width),str(top),str(height),str(left)])
        # 通过 tcp 发送出去
        tcpCliSock.send(data.encode())
    else: tcpCliSock.send('No'.encode())

```

## 5.6 人体关键点识别部分

```

elif data=='人体关键点识别 mode':
    # 字典方式对打开本地文件操作录入
    files = {'image_file':open('c.jpg', 'rb')}
    # 调用 api 所需要的参数
    payload = {'api_key': 'aiyUZ7KnALYTZgI8ekWI-jnpxCCI2s5z',
               'api_secret': 'jxdfCWicFP8fZQmVQRhI-fnSFa10_iwp',}
    # 通过本地文件调用接口
    r = requests.post(url_body_skeleton,files=files,data=payload)
    # 将返回的 json 文件转化为 python 的字典
    data=json.loads(r.text)
    # 新建字符串 sdata
    sdata = ""
    # 输出 r.text 供调试使用
    print (r.text)
    # 输出人体数目供测试使用
    print (len(data["skeletons"]))
    # 人体关键点列表
    skeletonName = ['head','neck','left_shoulder','left_elbow','left_hand','right_shoulder','right_elbow','right_hand','left_buttocks','left_knee','left_foot','right_buttocks','right_knee','right_foot']
    if data["skeletons"]:
        # 遍历所有的人体
        for i in range(len(data["skeletons"])):
            # 人体的矩形框信息分别是矩形框左上角的坐标(l,t)，矩形框的高 h 和宽 w
            width= data['skeletons'][i]['body_rectangle']['width']
            top= data['skeletons'][i]['body_rectangle']['top']
            height= data['skeletons'][i]['body_rectangle']['height']
            left= data['skeletons'][i]['body_rectangle']['left']
            # 将上述信息通过', '隔开成字符串，存入 sdata

```





---

可以看出服务端对 api 返回的参数进行了简化，可以减小前后端之间的信息量，这是我对前后端思想的第一个想法，而且通过 api 的调用，让我熟悉了人体关键点识别任务的参数。

## 5.7 接受图片部分

```
def ser(PORT,text):
    # 使用本地环回 ip 进行测试，正式工作使用服务端的 IP 地址
    HOST = '127.0.0.1'
    BUFSIZ = 1024
    ADDR = (HOST, PORT)
    tcpSerSock = socket(AF_INET, SOCK_STREAM)
    tcpSerSock.bind(ADDR)
    tcpSerSock.listen(5)
    while True:
        # 等待连接
        print('port ',PORT,' is waiting for connection...')
        tcpCliSock, addr = tcpSerSock.accept()
        # 传入工作模式信息
        print('port ',PORT,' connected from:', addr)
        data = tcpCliSock.recv(BUFSIZ).decode()
        if data=='发送图片 mode':
            text.insert(INSERT,'接收到客户端发送的图片\n')
            # 发送 OK 字符
            tcpCliSock.send('OK'.encode())
            # 获取客户端发来的尺寸信息
            size= tcpCliSock.recv(BUFSIZ).decode()
            # 发送 OK 字符
            tcpCliSock.send('OK'.encode())
            # 将接收到的尺寸信息字符串转换为整型
            size=int(size)
            # 接受图片信息，尺寸信息上文已经给出
            data = tcpCliSock.recv(size)
            # 将接受到的图片写入 c.jpg
            myfile = open('c.jpg', 'wb')
            myfile.write(data)
            myfile.close()
            # 发送 OK 字符
            tcpCliSock.send('OK'.encode())
```

## 5.8 其他部分

```
class Ser:
    def __init__(self):
        # 新建窗口
```

```
window=Tk()
window.title("人工智能-期末项目-服务端")
# 新建文本框
self.text=Text(window,width=100,height=20)
self.text.pack()
frame1=Frame(window)
frame1.pack()
# 按钮
Button(frame1,text="服务端初始化
",command=self.processInit).grid(row=1,column=1)
Button(frame1,text="清空屏幕
",command=self.processClear).grid(row=1,column=2)
window.mainloop()

def processInit(self):
    self.text.insert(INSERT,'服务端初始化完毕\n')
    # python 多线程，根据客户端的预设数量启动线程数
    # 这里只有一个客户端，所以只有启动一个线程
    self.threads=[]
    for i in count:
        t=threading.Thread(target=ser,args=(PORT[i],self.text))
        self.threads.append(t)
    # 启动线程
    for i in count:
        self.threads[i].start()

def processClear(self):
    # 对文本框进行清屏操作
    self.text.delete('1.0','end')

if __name__ == '__main__':
    Ser()
```

---

## 第6章 其他技术分析

### 6.1 旷视科技 API 接口分析

#### API 调用——原理

Face++ 人工智能开放平台 API 是 HTTP API。常用的编程语言都能发起 HTTP 请求（通过第三方库或自带 API），使用者向我们的服务器发起 HTTP 请求，并加上合适的参数，服务器将会对请求进行处理，得到结果将会返回给使用者。

#### API 调用——鉴权

帐号下每创建一个应用就会生成一组对应的 `api_key` 和 `api_secret`，用以识别用户是否有权限调用 API，所有的 API 调用必须提供对应的一组 `api_key` 和 `api_secret` 参数。

#### API 调用——参数

调用每个 API 根据需求传不同的参数，每个 API 参数的详细定义请查看 人脸识别 。所有 API 的调用都要使用 POST 请求，用户可以以 Query String 的形式将参数写进请求体中，传图片文件参数时则需要在请求体中使用 multipart/form-data 格式来编码。

#### API 调用——提示

为了避免因网络问题而造成的阻塞，建议将 API 调用放进异步线程里执行。

### 1. 人脸识别 API

#### 描述

传入图片进行人脸检测和人脸分析。

可以检测图片内的所有人脸，对于每个检测出的人脸，会给出其唯一标识 `face_token`，可用于后续的人脸分析、人脸比对等操作。对于正式 API Key，支持指定图片的某一区域进行人脸检测。

本 API 支持对检测到的人脸直接进行分析，获得人脸的关键点和各类属性信息。对于试用 API Key，最多只对人脸框面积最大的 5 个人脸进行分析，其他检测到的人脸可以使用 Face Analyze API 进行分析。对于正式 API Key，支持分析所有检测到的人脸。

#### 关于 `face_token`

如果您需要将检测出的人脸用于后续的分析、比对等操作，建议将对应的 face\_token 添加到 FaceSet 中。如果一个 face\_token 在 72 小时内没有存放在任一 FaceSet 中，则该 face\_token 将会失效。如果对同一张图片进行多次人脸检测，同一个人脸得到的 face\_token 是不同的。

图片要求

图片格式：JPG(JPEG)，PNG  
图片像素尺寸：最小 48\*48 像素，最大 4096\*4096 像素  
图片文件大小：2 MB  
最小人脸像素尺寸：系统能够检测到的人脸框为一个正方形，正方形边长的最小值为图像短边长度的 48 分之一，最小值不低于 48 像素。例如图片为 4096\*3200 像素，则最小人脸像素尺寸为 66\*66 像素。

调用 URL

https://api-cn.faceplusplus.com/facepp/v3/detect

调用方法

POST

权限

所有 API Key 都可以调用本 API。其中 calculate\_all 和 face\_rectangle 参数只有正式 API Key 才能使用，试用 API Key 无法使用。

请求参数

是否 必选	参数名	类型	参数说明
必选	api_key	String	调用此 API 的 API Key
必选	api_secret	String	调用此 API 的 API Secret
必选 (三 选 一)	image_url	String	图片的 URL。 注：在下载图片时可能由于网络等原因导致下载图片时间过长，建议使用 image_file 或 image_base64 参数直接上传图片。
	image_file	File	一个图片，二进制文件，需要用 post multipart/form-data 的方式上传。

	image_base64	String	base64 编码的二进制图片数据 如果同时传入了 image_url、image_file 和 image_base64 参数，本 API 使用顺序为 image_file 优先，image_url 最低。							
可选	return_landmark	Int	是否检测并返回人脸关键点。合法值为： <table><tr><td>2</td><td>检测。返回 106 个人脸关键点。</td></tr><tr><td>1</td><td>检测。返回 83 个人脸关键点。</td></tr><tr><td>0</td><td>不检测</td></tr></table> 注：本参数默认值为 0		2	检测。返回 106 个人脸关键点。	1	检测。返回 83 个人脸关键点。	0	不检测
2	检测。返回 106 个人脸关键点。									
1	检测。返回 83 个人脸关键点。									
0	不检测									
可选	return_attributes	String	是否检测并返回根据人脸特征判断出的年龄、性别、情绪等属性。合法值为： <table><tr><td>none</td><td>不检测属性</td></tr><tr><td>gender age smiling headpose facequality blur eyestatus emotion ethnicity beauty mouthstatus eyegaze skinstatus</td><td>希望检测并返回的属性。 需要将属性组成一个用逗号分隔的字符串，属性之间的顺序没有要求。 关于各属性的详细描述，参见下文“返回值”说明的 "attributes" 部分。 <b>注：在此参数中的传入参数 smiling，对应在此返回值的 attributes 中参数名为 smile。在使用时请注意。</b></td></tr></table> 注：本参数默认值为 none		none	不检测属性	gender age smiling headpose facequality blur eyestatus emotion ethnicity beauty mouthstatus eyegaze skinstatus	希望检测并返回的属性。 需要将属性组成一个用逗号分隔的字符串，属性之间的顺序没有要求。 关于各属性的详细描述，参见下文“返回值”说明的 "attributes" 部分。 <b>注：在此参数中的传入参数 smiling，对应在此返回值的 attributes 中参数名为 smile。在使用时请注意。</b>		
none	不检测属性									
gender age smiling headpose facequality blur eyestatus emotion ethnicity beauty mouthstatus eyegaze skinstatus	希望检测并返回的属性。 需要将属性组成一个用逗号分隔的字符串，属性之间的顺序没有要求。 关于各属性的详细描述，参见下文“返回值”说明的 "attributes" 部分。 <b>注：在此参数中的传入参数 smiling，对应在此返回值的 attributes 中参数名为 smile。在使用时请注意。</b>									

返回值说明

字段	类型	说明
request_id	String	用于区分每一次请求的唯一的字符串。
faces	Array	被检测出的人脸数组，具体包含内容见下文。 注：如果没有检测出人脸则为空数组
image_id	String	被检测的图片在系统中的标识。

time_used	Int	整个请求所花费的时间，单位为毫秒。
error_message	String	当请求失败时才会返回此字符串，具体返回内容见后续错误信息章节。否则此字段不存在。

**faces 数组中单个元素的结构**

字段	类型	说明
face_token	String	人脸的标识
face_rectangle	Object	人脸矩形框的位置，包括以下属性。每个属性的值都是整数： <b>top</b> : 矩形框左上角像素点的纵坐标 <b>left</b> : 矩形框左上角像素点的横坐标 <b>width</b> : 矩形框的宽度 <b>height</b> : 矩形框的高度
landmark	Object	人脸的关键点坐标数组。 当传入的 landmark 参数值为 1 时，返回 83 个关键点坐标数组。 当传入的 landmark 参数值为 2 时，返回 106 个关键点坐标数组。 关于 83 个或 106 个关键点坐标的详细说明与图示，请分别参考文档：《 <a href="#">人脸关键点 Landmark 说明（83 点）</a> 》、《 <a href="#">人脸关键点 Landmark 说明（106 点）</a> 》
attributes	Object	人脸属性特征，具体包含的信息见下表。

**attributes 中包含（使用过的）的元素说明**

字段	类型	说明	
gender	String	性别分析结果。返回值为：	
		Male	男性
		Female	女性
age	Int	年龄分析结果。返回值为一个非负整数。	
beauty	Object	颜值识别结果。返回值包含以下两个字段。每个字段的值是一个浮点数，范围 [0,100]，小数点后 3 位有效数字。	
		male_score: 男性认为的此人脸颜值分数。值越大，颜值越高。	



		<b>female_score:</b> 女性认为的此人脸颜值分数。值越大，颜值越高。
--	--	---

请求成功返回示例

```
{
  "image_id": "Dd2xUw9S/7yjr0oDHHSL/Q==",
  "request_id": "1470472868,dacf2ff1-ea45-4842-9c07-6e8418cea78b",
  "time_used": 752,
  "faces": [{
    "landmark": {
      "mouth_upper_lip_left_contour2": {
        "y": 185,
        "x": 146
      },
      "contour_chin": {
        "y": 231,
        "x": 137
      },
      .....省略关键点信息
      "right_eye_pupil": {
        "y": 146,
        "x": 205
      },
      "mouth_upper_lip_bottom": {
        "y": 195,
        "x": 159
      }
    },
    "attributes": {
      "gender": {
        "value": "Female"
      },
      "age": {
        "value": 21
      },
      "glass": {
        "value": "None"
      },
      "headpose": {
        "yaw_angle": -26.625063,
        "pitch_angle": 12.921974,
        "roll_angle": 22.814377
      }
    }
  ]
}
```

```
        "smile": {
            "threshold": 30.1,
            "value": 2.566890001296997
        },
        "face_rectangle": {
            "width": 140,
            "top": 89,
            "left": 104,
            "height": 141
        },
        "face_token": "ed319e807e039ae669a4d1af0922a0c8"
    }
}
```

## 2. 人体识别 API

### 描述

传入图片进行人体检测和人体属性分析。

本 API 支持检测图片内的所有人体，并且支持对检测到的人体直接进行分析，获得每个人体的各类属性信息。

### 图片要求

图片格式：JPG(JPEG)，PNG

图片像素尺寸：最小 48\*48 像素，最大 1280\*1280 像素

图片文件大小：2 MB

### 调用 URL

<https://api-cn.faceplusplus.com/humanbodypp/v1/detect>

### 调用方法

POST

### 权限

所有 API Key 均可调用本 API。

### 请求参数

是否 必选	参数名	类型	参数说明
必选	api_key	String	调用此 API 的 API Key
必选	api_secret	String	调用此 API 的 API Secret

必选 (三选一)	image_url	String	图片的 URL 注：在下载图片时可能由于网络等原因导致下载图片时间过长，建议使用 image_file 或 image_base64 参数直接上传图片	
	image_file	File	一个图片，二进制文件，需要用 post multipart / form-data 的方式上传	
	image_base64	String	base64 编码的二进制图片数据 如果同时传入了 image_url、image_file 和 image_base64 参数，本 API 使用顺序为 image_file 优先，image_url 最低	
可选	return_attributes	String	是否检测并返回根据人体特征判断出的性别，服装颜色等属性。合法值为：	
			<table><tr><td>none</td><td>不检测属性 注：本参数默认值为 none</td></tr><tr><td>gender upper_body_cloth lower_body_cloth</td><td>希望检测并返回的属性 需要将属性组成一个用逗号分隔的字符串，属性之间的顺序没有要求 关于各属性的详细描述，参见下文“返回值”说明的 "attributes" 部分</td></tr></table>	none
none	不检测属性 注：本参数默认值为 none			
gender upper_body_cloth lower_body_cloth	希望检测并返回的属性 需要将属性组成一个用逗号分隔的字符串，属性之间的顺序没有要求 关于各属性的详细描述，参见下文“返回值”说明的 "attributes" 部分			

返回值说明

字段	类型	说明
request_id	String	用于区分每一次请求的唯一的字符串
humanbodies	Array	被检测出的人体数组，具体内容见下文 注：如果没有检测出的人体则为空数组
image_id	String	被检测的图片在系统中的标识

time_used	Int	整个请求所花费的时间，单位为毫秒
error_message	String	当请求失败时才会返回此字符串，具体返回内容见后续错误信息章节。否则此字段不存在

#### humanbodies 数组中单个元素的结构

字段	类型	说明
confidence	Float	人体检测的置信度，范围 [0,100]，小数点后 3 位有效数字，数字越大表示检测到的对象为人体的置信度越大
humanbody_rectangle	Object	人体矩形框的位置，包括以下属性。每个属性的值都是整数： top: 矩形框左上角像素点的纵坐标 left: 矩形框左上角像素点的横坐标 width: 矩形框的宽度 height: 矩形框的高度
attributes	Object	人体属性特征，具体包含的信息见下表

#### attributes 中包含的元素说明

字段	类型	说明
gender	Object	性别分析结果，返回值包含以下字段。每个字段的值都是一个浮点数，范围 [0,100]，小数点后 3 位有效数字。字段值的总和等于 100。 male: 性别为男性的置信度 female: 性别为女性的置信度
upper_body_cloth	Object	上身分析结果。返回值包含以下属性： upper_body_cloth_color: 上身衣物颜色，值为下方颜色列表中与上身衣物颜色最接近的颜色值 upper_body_cloth_color_rgb: 上身衣物颜色 RGB 值。 upper_body_cloth_color_rgb 属性包含以下字段： r : 红色通道值 g: 绿色通道值 b: 蓝色通道值
lower_body_cloth	Object	下身分析结果。返回值包含以下属性：

		<p><b>lower_body_cloth_color:</b> 下身衣物颜色，值为下方颜色列表中与下身衣物颜色最接近的颜色值</p> <p><b>lower_body_cloth_color_rgb:</b> 下身衣物颜色 RGB 值。</p> <p><b>lower_body_cloth_color_rgb</b> 属性包含以下字段：</p> <p><b>r :</b> 红色通道值</p> <p><b>g:</b> 绿色通道值</p> <p><b>b:</b> 蓝色通道值</p>
--	--	---

颜色列表

颜色值	颜色说明	R	G	B
black	黑色	0	0	0
white	白色	255	255	255
red	红色	255	0	0
green	绿色	0	255	0
blue	蓝色	0	0	255
yellow	黄色	255	255	0
magenta	洋红	255	0	255
cyan	青色	0	255	255
gray	灰色	128	128	128
purple	紫色	128	0	128
orange	橙色	255	128	0

请求成功返回示例：

```
{
  "image_id": "7007N1dYiJjszvV38oKVpw==",
  "request_id": "1491569448,de5a441f-6c6f-4955-896c-37b8bb2d4197",
  "time_used": 915,
  "humanbodies": [{
    "attributes": {
      "gender": {
        "male": 99.000,
        "female": 1.000
      }
    }
  ]
}
```

```
    },
    "upper_body_cloth": {
      "upper_body_cloth_color": "white",
      "upper_body_cloth_color_rgb": {
        "r": 255,
        "g": 255,
        "b": 255
      }
    },
    "lower_body_cloth": {
      "lower_body_cloth_color": "white",
      "lower_body_cloth_color_rgb": {
        "r": 255,
        "g": 255,
        "b": 255
      }
    }
  },
  "humanbody_rectangle": {
    "width": 456,
    "top": 0,
    "height": 500,
    "left": 0
  },
  "confidence": 99.905
}]
}
```

### 3. 人体关键点识别 API

#### 描述

传入图片进行人体检测和骨骼关键点检测，返回人体 14 个关键点

支持对图片中的所有人体进行骨骼检测

#### 图片要求

图片格式：JPG (JPEG)

图片像素尺寸：最小 100\*100 像素，最大 4096\*4096 像素

图片文件大小：2 MB

为了保证较好的识别结果，人体矩形框大小建议在 200\*200 像素及以上

#### 调用 RUL

<https://api-cn.faceplusplus.com/humanbodypp/v1/skeleton>

#### 调用方法

post

权限

所有 API key 均能调用此 API

请求参数

是否必选	参数名	类型	参数说明
必选	api_key	String	调用此 API 的 API Key
必选	api_secret	String	调用此 API 的 API Secret
必选 (三选一)	image_url	String	图片的 URL 注：在下载图片时可能由于网络等原因导致下载图片时间过长，建议使用 image_file 或 image_base64 参数直接上传图片
	image_file	File	一个图片，二进制文件，需要用 post multipart/form-data 的方式上传
	image_base64	String	base64 编码的二进制图片数据 如果同时传入了 image_url、image_file 和 image_base64 参数，本 API 使用顺序为 image_file 优先，image_url 最低

返回参数

返回字段说明	类型	说明
request_id	String	用于区分每一次请求的唯一的字符串
skeletons	Object	被检测出的人体数组，具体内容见下文。  注：如果没有检测出人体则为空
image_id	String	被检测的图片在系统中的标识
time_used	Int	整个请求所花费的时间，单位为毫秒
error_message	String	当请求失败时才会返回此字符串，具体返回内容见后续错误信息章节。否则此字段不存在

skeletons 中单个元素的结构



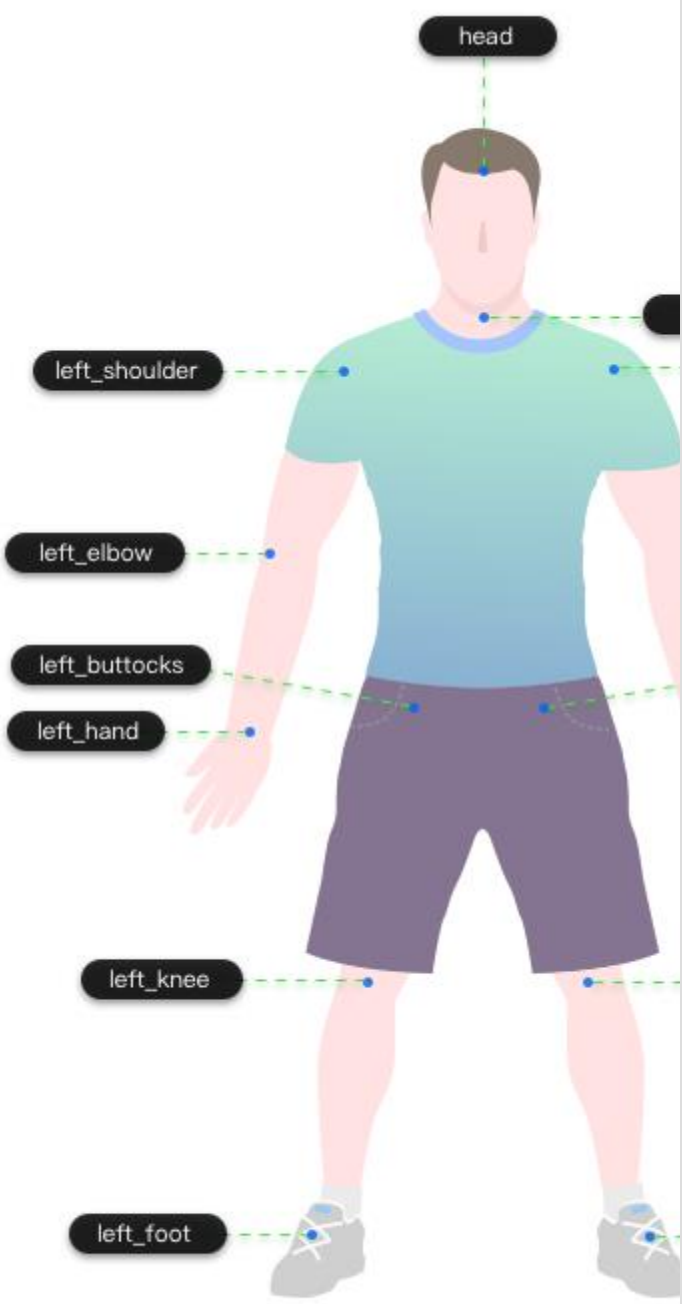
字段	类型	说明
body_rectangle	Object	人体矩形框的位置，包括以下属性。每个属性的值都是整数： <b>top</b> : 矩形框左上角像素点的纵坐标 <b>left</b> : 矩形框左上角像素点的横坐标 <b>width</b> : 矩形框的宽度 <b>height</b> : 矩形框的高度
landmark	Object	包含 14 个骨骼关键点的对象类型，具体包含信息见下表及说明图

landmark 对象结构

下文字段均代表一个骨骼关键点，包含字段 x: 横坐标位置；y: 纵坐标位置，单位均为像素

（注：以下字段中坐标均为相对于人体框的坐标）

字段	类型	说明	图示
head	Object	头部	
neck	Object	脖子	
left_shoulder	Object	左肩	
left_elbow	Object	左肘	
left_hand	Object	左手	
right_shoulder	Object	右肩	
right_elbow	Object	右肘	
right_hand	Object	右手	
left_buttocks	Object	左臀	

left_knee	Object	左膝	
left_foot	Object	左脚	
right_buttocks	Object	右臀	
right_knee	Object	右膝	
right_foot	Object	右脚	

请求成功返回值示例

```
{
  "time_used": 2299,
  "image_id": "lHjvzhnGndA/zZja3ho2fA==",
  "skeletons": [
    {
```

---

```
"body_rectangle": {
  "width": 781,
  "top": 315,
  "height": 910,
  "left": 134
},
"landmark": {
  "left_buttocks": {
    "y": 901,
    "x": 452,
    "score": 0.7299447
  },
  "head": {
    "y": 9,
    "x": 305,
    "score": 0.5368539
  },
  "neck": {
    "y": 673,
    "x": 378,
    "score": 0.6446256
  },
  "left_shoulder": {
    "y": 692,
    "x": 769,
    "score": 0.7006054
  },
  "left_hand": {
    "y": 597,
    "x": 769,
    "score": 0.41823307
  },
  "left_knee": {
    "y": 901,
    "x": 329,
    "score": 0.6185796
  },
  "right_elbow": {
    "y": 901,
    "x": 12,
    "score": 0.4332367
  },
  "right_shoulder": {
    "y": 635,
```

```
        "x": 12,  
        "score": 0.724305  
    },  
    "right_hand": {  
        "y": 901,  
        "x": 232,  
        "score": 0.56872773  
    },  
    "left_foot": {  
        "y": 901,  
        "x": 452,  
        "score": 0.3753785  
    },  
    "left_elbow": {  
        "y": 901,  
        "x": 769,  
        "score": 0.51084995  
    },  
    "right_buttocks": {  
        "y": 901,  
        "x": 134,  
        "score": 0.5565492  
    },  
    "right_knee": {  
        "y": 901,  
        "x": 85,  
        "score": 0.47690523  
    },  
    "right_foot": {  
        "y": 901,  
        "x": 452,  
        "score": 0.52765757  
    }  
}  
},  
{  
    "body_rectangle": {  
        "width": 559,  
        "top": 533,  
        "height": 740,  
        "left": 394  
    },  
    "landmark": {  
        "left_buttocks": {
```

---

```
        "y": 732,  
        "x": 114,  
        "score": 0.8086717  
    },  
    "head": {  
        "y": 270,  
        "x": 464,  
        "score": 0.4876535  
    },  
    "neck": {  
        "y": 270,  
        "x": 79,  
        "score": 0.5550944  
    },  
    "left_shoulder": {  
        "y": 378,  
        "x": 359,  
        "score": 0.64766926  
    },  
    "left_hand": {  
        "y": 362,  
        "x": 481,  
        "score": 0.6320832  
    },  
    "left_knee": {  
        "y": 563,  
        "x": 271,  
        "score": 0.49012265  
    },  
    "right_elbow": {  
        "y": 378,  
        "x": 551,  
        "score": 0.6178683  
    },  
    "right_shoulder": {  
        "y": 347,  
        "x": 324,  
        "score": 0.5143001  
    },  
    "right_hand": {  
        "y": 378,  
        "x": 551,  
        "score": 0.6799432  
    },  
    },
```

```
        "left_foot": {
            "y": 732,
            "x": 9,
            "score": 0.4409459
        },
        "left_elbow": {
            "y": 362,
            "x": 499,
            "score": 0.56940734
        },
        "right_buttocks": {
            "y": 732,
            "x": 201,
            "score": 0.6996023
        },
        "right_knee": {
            "y": 424,
            "x": 551,
            "score": 0.48145446
        },
        "right_foot": {
            "y": 732,
            "x": 9,
            "score": 0.43120167
        }
    }
}

],
"request_id": "1533886107,224911b6-e136-42bd-aebb-e1aec327e018"
}
```

## 6.2 python 的 socket 库

本部分内容使用统一的模版，具体内容见格式范例，提交时作者和导师须亲笔签名。

### 1. 中文摘要

#### 什么是 Socket?

Socket 又称“套接字”，应用程序通常通过“套接字”向网络发出请求或者应答网络请求，使主机间或者一台计算机上的进程间可以通讯。

---

#### socket() 函数

Python 中，我们用 `socket()` 函数来创建套接字，语法格式如下：

```
socket.socket([family[, type[, proto]]])
```

参数

- family: 套接字家族可以使 AF\_UNIX 或者 AF\_INET
- type: 套接字类型可以根据是面向连接的还是非连接分为 SOCK\_STREAM 或 SOCK\_DGRAM
- protocol: 一般不填默认为 0.

Socket 对象(内建)方法

函数	描述
服务器端套接字	
s.bind()	绑定地址 (host,port) 到套接字， 在 AF_INET 下,以元组 (host,port) 的形式
s.listen()	开始 TCP 监听。backlog 指定在拒绝连接之前，操作系统可以挂起的最大连接数。大部分应用程序设为 5 就可以了。
s.accept()	被动接受 TCP 客户端连接,(阻塞式)等待连接的到来
客户端套接字	
s.connect()	主动初始化 TCP 服务器连接，。一般 address 的格式为元组 (hostname,port)。失败会抛出 socket.error 错误。
s.connect_ex()	connect()函数的扩展版本,出错时返回出错码,而不是抛出异常
公共用途的套接字函数	
s.recv()	接收 TCP 数据，数据以字符串形式返回，bufsize 指定要接收的最大数据量。返回的数据可能小于 bufsize，通常可以忽略。
s.send()	发送 TCP 数据，将 string 中的数据发送到连接的套接字。返回值是要发送的字节大小，可能小于 string 的字节大小。



函数	描述
s.sendall()	完整发送 TCP 数据，完整发送 TCP 数据。将 string 中的数据发送到连接的套接字。尝试发送所有数据。成功返回 None，失败则抛出异常。
s.recvfrom()	接收 UDP 数据，与 recv()类似，但返回值是（data,address）。其中 data 是包含数据的字节字符串，address 是发送数据的套接字地址。
s.sendto()	发送 UDP 数据，将数据发送到套接字，address 是形式为（ipaddr, port）的元组。返回值是发送的字节数。
s.close()	关闭套接字
s.getpeername()	返回连接套接字的远程地址。返回值通常是元组（ipaddr,port）。
s.getsockname()	返回套接字自己的地址。通常是一个元组(ipaddr,port)
s.setsockopt(level,optname,value)	设置给定套接字选项的值。
s.getsockopt(level,optname[.buflen])	返回套接字选项的值。
s.settimeout(timeout)	设置套接字操作的超时期，timeout 是一个浮点数，单位是秒。值为 None 表示永不超时。这个时期应该在刚创建套接字时设置，因为它们可能用于连接的操作（如 connect）。
s.gettimeout()	返回当前超时期的值，单位是秒，如果没有设置超时期，则返回 None。
s.fileno()	返回套接字的文件描述符。
s.setblocking(flag)	如果 flag 为 0，则将套接字设为非阻塞模式，否则将套接字设为阻塞模式（默认）。如果调用 recv()没有发现任何数据，或 send()调用无法立即发送数据，那么将引发异常。
s.makefile()	创建一个与该套接字相关连的文件

## 6.3 python 的 requests 库

requests 是 python 实现的最简单易用的 HTTP 库

```
import requests
```

```
url = "https://api.github.com/events"
```

---

## 获取某个网页

```
import requests
r = requests.get("https://api.github.com/events")
print(r)                # <Response [200]>
print(type(r))           # <class 'requests.models.Response'>
print(r.status_code)     # 200
```

## 各种请求

# 发送一个 HTTP POST 请求:

```
r = requests.post("http://httpbin.org/post", data = {'key': 'value'})
```

r = requests.delete('http://httpbin.org/delete') # 发送一个  
HTTP delete 请求:

r = requests.head('http://httpbin.org/get') # 发送一个  
HTTP head 请求:

r = requests.options('http://httpbin.org/get') # 发送一个  
HTTP options 请求:

## get 传递 URL 参数

?+键值对

```
response1 = requests.get("http://httpbin.org/get?key1=value1")
print(response1.url)
#http://httpbin.org/get?key1=value1
```

## requests 提供了 params 关键字参数来传递参数

```
parameter = {
    "key1": "value1",
    "key2": "value2"
}

response2 = requests.get("http://httpbin.org/get", params =
parameter)
print(response2.url)
# http://httpbin.org/get?key1=value1&key2=value2
```

---

还可以将一个列表作为值传入

```
parameter = {
    "key1": "value1",
    "key2": ["value21", "value22"]
}

response3 = requests.get("http://httpbin.org/get", params =
parameter)
print(response3.url)
# http://httpbin.org/get?key1=value1&key2=value21&key2=value22
```

注意字典里值为 None 的键都不会被添加到 URL 的查询字符串里。

```
parameter = {
    "key1": "value",
    "key2": None
}

response4 = requests.get("http://httpbin.org/get", params =
parameter)
print(response4.url) #http://httpbin.org/get?key1=value
```

---

## 参 考 资 料

- [1] <https://console.faceplusplus.com.cn/documents/37664576>.
- [2] <https://console.faceplusplus.com.cn/documents/4888373>
- [3] <https://www.runoob.com/>
- [4] <https://morvanzhou.github.io/>
- [5] <https://wygng.github.io/>
- [6] <https://www.jianshu.com/p/ecb4d54ad8cf>