人脸识别技术介绍

# 一、前言

人脸识别指判断两张人脸图片是否属于同一个人。

在实际应用中，公司需要提前准备一些已知姓名的人脸图片作为人脸库（白名单），当输入一张未知姓名的人脸图片时，通过计算这张图片和人脸库中每张图片的相似度，并挑选相似度大于阈值alpha的图片的姓名作为此人脸识别的结果。

通常，人脸识别拆分为如下步骤：

**活体检测：**判断图片中的人脸是真人图片还是拍摄真人图片得到的图片，若不为真人图片，则不进行人脸识别，直接禁止此图片，不进行接下来的操作。

**人脸检测**：检测图片中是否有人脸。

**人脸对齐裁剪：**对检测出来的人脸区域进行对齐然后裁剪。通常需要先对检测的人脸进行关键点检测，然后进行人脸对齐，最后对图片裁剪得到人脸区域。

**人脸特征提取和对比：**计算对齐裁剪的人脸的特征（一般是一维的float数组，长度512或1024等），随后计算两张对齐裁剪的人脸的特征之间的内积，即相似度，并基于此进行相关判断。

事实上，人脸识别的准确度依赖于人脸图片的质量，人脸的角度等。因此在构建人脸库时，我们需要对人脸库中的图片进行筛选，筛除掉部分不合格的图片。筛选条件如下：

姿态检测：检测图片中人脸的角度，抬头、低头、左/右转头角度太大都不合格，因此人脸正脸是最优解。

**分辨率判别：**对图片分辨率进行限制，图片中人脸区域分辨率过小不合格。

**亮度判别：**图片中的人脸区域不可过暗或过亮。

**清晰度判别：**图片中人脸区域不可过于模糊。

注意：web端和app端的人脸识别重度依赖于github中的一个开源的cpp项目：<https://github.com/seetafaceengine/SeetaFace6> ，以下简称seetaface6。web端由于需要支持并发性，所以我们将seetaface6中的模型转化为onnx模型，然后使用python进行调用。app端则是通过jni 包装seetaface6原生代码，随后在android studio中编译，并打包为APP应用。下文的讲解基于web端python代码(压缩包中目录FaceRegWeb)和seetaface6[原生代码](https://github.com/SeetaFace6Open/index)(见链接)，**相对路径都是相对于web端/cpp端项目的目录**。

# 二、活体检测

## 1.技术介绍：

采用的是seetaface6中的技术。其活体检测分为**全局活体检测**和**局部活体检测**。

**全局活体检测**就是对图片整体做检测，主要是判断是否出现了活体检测潜在的攻击介质，如手机、平板、照片等等。

**局部活体检测**是对具体人脸的成像细节通过算法分析，区别是一次成像和二次成像，如果是二次成像则认为是出现了攻击。

seetaface6中声明代码是：

#include <seeta/FaceAntiSpoofing.h>

seeta::FaceAntiSpoofing \*new\_fas\_v2() {

seeta::ModelSetting setting;

setting.append("fas\_first.csta");

setting.append("fas\_second.csta");

return new seeta::FaceAntiSpoofing(setting);

}

活体检测依赖于实时的视频，其会对视频中多帧图像判断活体，如果多帧图像中的人脸都判定为活体，才会被认定为真人并进行下一步。web端应用输入的是图片，因此不包含此功能。而APP端通常部署在闸机、门禁等地，需要考虑不法分子采用他人的图片通过门禁，因此需要包含此功能。

## 2.后续优化：

这部分较为成熟的技术都在大公司，且收费，暂时还是使用seetaface6提供的吧。但是另一种配合式活体检测，即按照要求完整指定动作来证明是真人，虽然较老，并且用户体验一般，但是相对更加安全（虽然无法排除不法分子提前录制包含指定动作的视频）。靠谱的开源项目是：<https://aitsuki.com/blog/liveness-detection-with-google-ml-kit/>。也可以自己实现，只是调试较为耗时，推荐参考代码：<https://github.com/DefTruth/lite.ai.toolkit> 。

## 3.cpp端代码介绍



（具体详见：[seetaface6官方文档](https://github.com/seetafaceengine/SeetaFaceTutorial)中对`活体检测`的介绍）

# 三、人脸检测

## 1.技术介绍：

人脸检测，就是输入待检测的图片，输出检测到的每个人脸位置，用矩形表示。

人脸检测是后续算法的关键。流程如下



代码详见：web端的models/facedetector.py 中 class FaceBoxesV2(BasicOrtHandler)；或seetaface6的FaceBoxes/FaceDetector/src/seta/FaceDetector.cpp中 SeetaFaceInfoArray FaceDetector::Implement::detect(const SeetaImageData &image) const ；

## 2.后续优化

人脸检测是其他技术的基础，较为重要，幸运的是市面上有多种开源的优质人脸检测算法。我对部分优秀的进行了对比测试，结果如下：



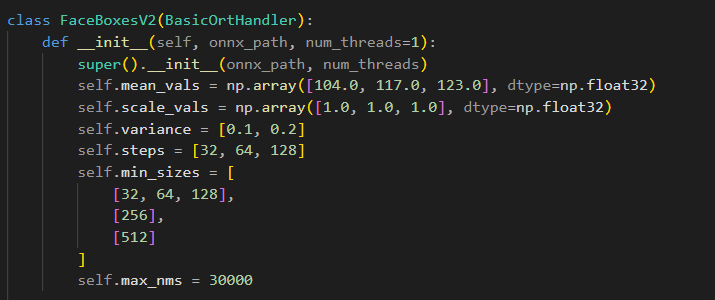
推荐使用`FaceBoxesv2`,效果好，时间短，优于seetaface6中的人脸检测。事实上web端使用的就是这个算法。但后续如果想使用seetaface6自带的原生人脸检测并保证在web端能并发推理，可以参考：<https://github.com/zjd1988/seetaface6_onnx_model> ，将这个项目中已经实现的seetaface6的人脸检测的onnx模型运用起来，并重写推理的预处理和后处理代码（这项目的作者对这部分代码写的不好，细节见此项目的issue）。

## 3.web端代码介绍

### 1）路径：

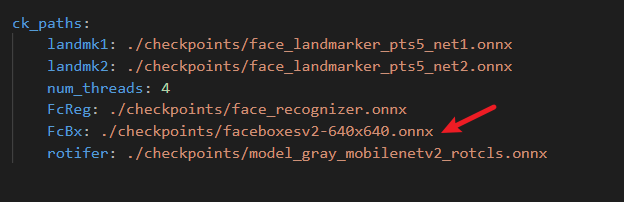
./models/facedetector.py

### 2）类定义：

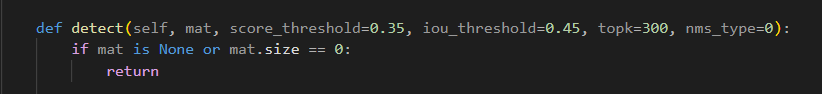


其中`num\_threads` 表示模型推理使用的线程数，线程数越大，推理的速度越快，但占用的资源越多，（同样在config.yaml 中`num\_threads`指定，推荐1或4）。

`onnx\_path`表示模型路径，其在`./config.yaml`中指定：



### 3）方法定义：



推荐使用默认值即可，调用方式如下：

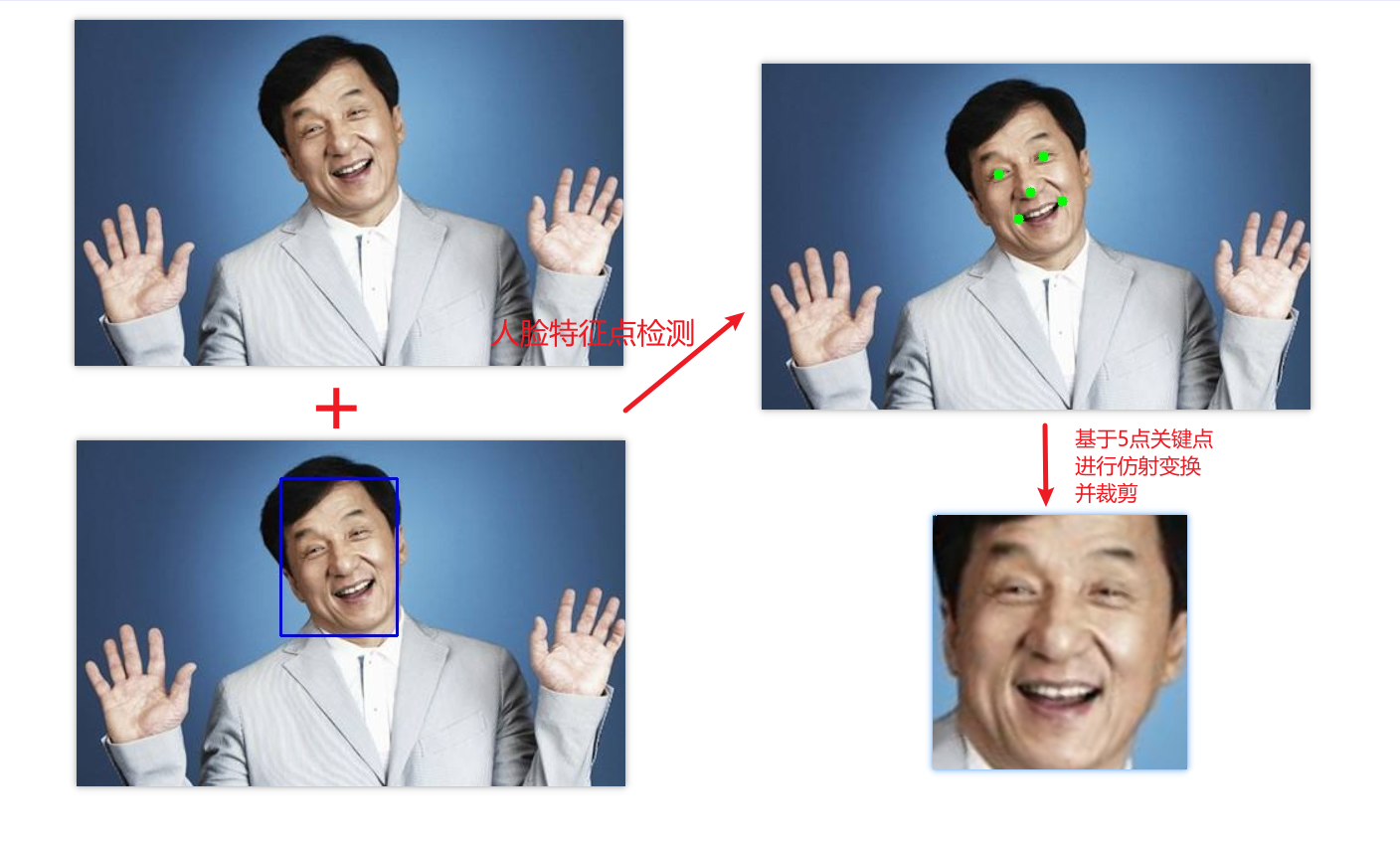
detect(mat=image) # image 是通过cv2.imread读取得到

返回的是image 中人脸矩形区域的信息。

# 四、人脸对齐裁剪

## 1. 算法介绍

在人脸检测的基础上，人脸对齐的目的是将人脸图像进行标准化，以便随后进行裁剪人脸区域。



我们使用的人脸对齐裁剪算法基于人脸特征点。首先寻找人脸的5点特征点（左眼中，右眼中，鼻尖，左嘴角，右嘴角），然后使用仿射变换将它们对齐并裁剪人脸区域并将区域调整为指定尺寸。

具体代码详见：web端的models/facelandmarks5er.py中class Landmark5er()和models/facealign.py中class FaceAlign； 或seetaface6 代码的Landmarker\Landmarker\src\seeta\FaceLandmarker.cpp 中FaceLandmarker::Implement::mark(const SeetaImageData &image, const SeetaRect &face) const及其FaceRecognizer6\FaceRecognizer\src\seeta\FaceRecognizer.cpp中bool FaceRecognizer::CropFace(const SeetaImageData &image, const SeetaPointF \*points, SeetaImageData &face)。

## 2. 后续优化

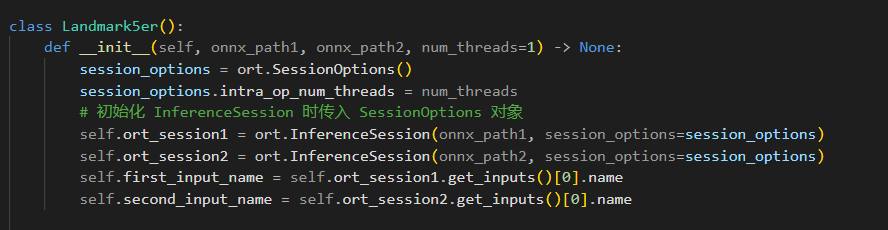
我们使用的seetaface6中的人脸对齐裁剪算法，从而适配后续的seetaface6的人脸特征提取算法。web端我已经将seetaface6的cpp代码更改为python代码，并将其csta模型转化为onnx模型从而支持并发请求。Web端中人脸对齐裁剪部分的python代码等价于对应的seetaface6 cpp代码，推荐阅读python代码，更易于理解。同时推荐观看学习当下5点特征点检测的部分算法：<https://blog.csdn.net/qq_14845119/article/details/107105841> 。 我测试了[mtcnn](https://github.com/kpzhang93/MTCNN_face_detection_alignment)，效果一般，主要是mtcnn人脸检测劣于`faceboxV2`算法，而mtcnn强制依赖于它自己的人脸检测，不好分开。

## 3.web端代码介绍

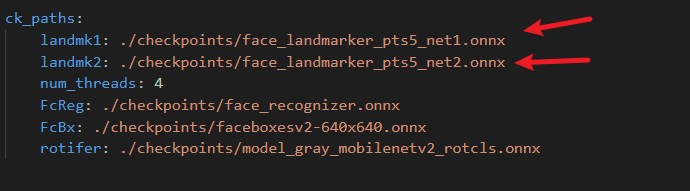
### 1）路径：

./models/facelandmarks5er.py 获取人脸面部的5点关键点坐标

### 2）类定义：

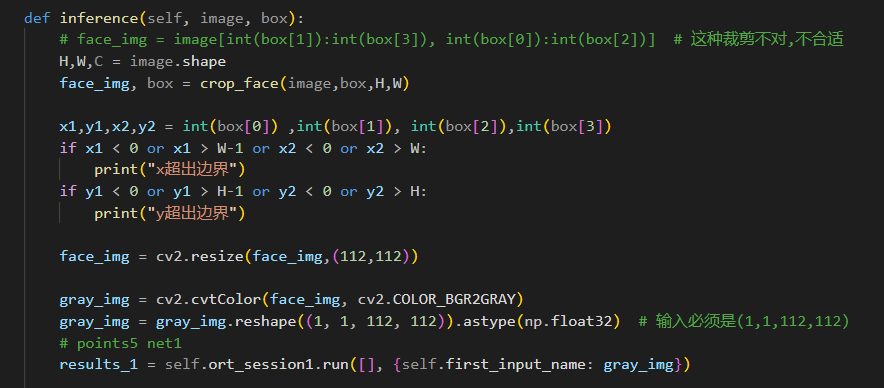


这个类需要两个模型，其路径分别为`onnx\_path1`和`onnx\_path2`，都在`./config.yaml`中指定：



`num\_thread`介绍同上，不再赘述。

### 3）方法定义：



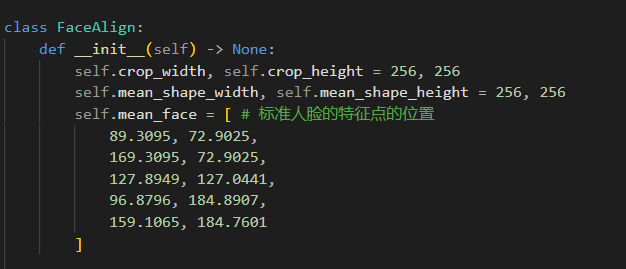
`image` 表示输入的人脸图像，`box`表示这个人脸图像中人脸矩形框的信息。

此方法返回这个box指定的人脸的5点关键点(左眼、右眼、笔尖、左嘴角、右嘴角)。

### 1）路径：

./models/facealign.py 根据人脸5点关键点，裁剪人脸区域并返回

### 2）类定义：



为了适配后续的人脸识别操作，在调用类创建对象时，不允许任何个性化操作。

### 3）方法调用：

`image`表示原始的人脸图像，`landmarks5`表示其中人脸的5点特征点。

此方法输出的是裁剪的人脸区域图片。(如上图)

### 4）总结：

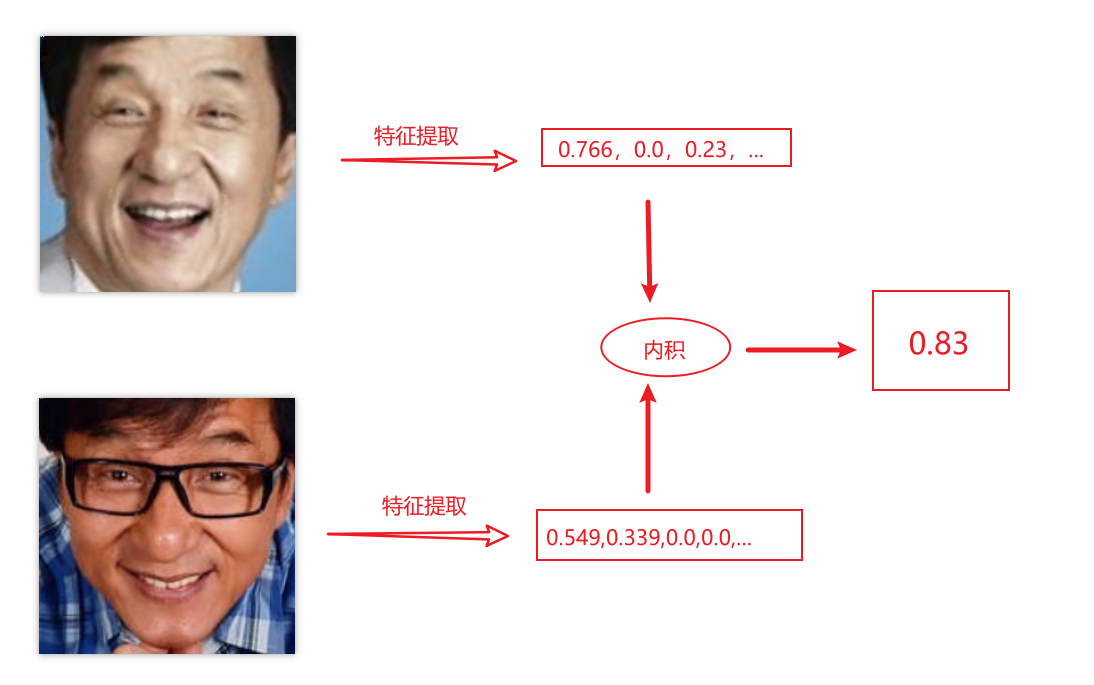
为便于理解，我们将这两部分的调用展示出来(路径：./process.py)：



# 五、人脸特征提取和对比

## 1. 算法介绍

人脸特征提取依赖于经过人脸对齐裁剪后得到的人脸。其是整个项目的核心关键，但是也是最难以调试修改的，事实上我们只能通常是直接使用当前优秀的开源项目，在此使用的是seetaface6中的人脸特征提取算法。对于人脸特征提取算法，其输入是一张经过人脸对齐裁剪后的人脸图片，输出是一个一维的float数组（称为特征向量）。



所谓特征对比就是计算两人脸的对应特征向量之间的内积，其也被称为这两张人脸的相似度。同一个人的两张人脸图片，如果它们都是正脸且面部完整，则相似度更高。

相似度越高，表明其是同一个人的可能性越大，我们设定一个**相似度阈值**，大于这个阈值则表明为同一人。人脸识别是1: N，即输入的人脸会和人脸库每张脸比对，考虑到人脸库中人的复杂性多样性，阈值设定过低会导致返回多个识别结果，更加严重的是假如人脸库中无待检测人的人脸图片（陌生人），但是此人被识别为别人从而通过门禁（陌生人通过门禁的人脸识别）。因此我们倾向于提高相似度阈值，固然这样会无法成功识别部分已经录入人脸库的人，但是未识别成功可以通过调整人脸角度和人脸位置来识别成功，远远好过陌生人通过人脸识别。

事实上人脸的完整性和角度对特征提取的影响极大，人脸库中的人脸对这些要求高一点有利于后续识别，证件照通常是较为优秀的人脸库图片。待检测人脸则没有这些要求，如果识别失败，返回调整人脸的提示进行后续识别即可。

因此我们需要对人脸图片的质量等进行判定，来返回调整人脸的提示信息。

代码详见：web端models/facerecoger.py中class FaceRecoger() ；或seetaface6的FaceRecognizer6\FaceRecognizer\src\seeta\FaceRecognizer.cpp

## 2. 后续改进

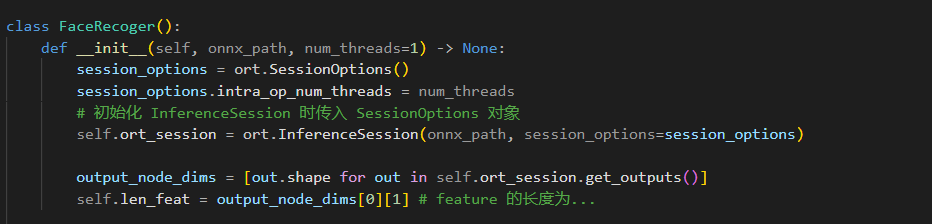
我测试过部分开源算法：dlib, [lite.ai.toolkit](https://github.com/DefTruth/lite.ai.toolkit) 中的类型为`faceid`的人脸识别算法，[deepface](https://github.com/serengil/deepface)中介绍的算法等。其在欧洲人脸上表现优秀，但是在亚洲人脸上表现较差。因此在web端和cpp端我们都采用seetaface6中的人脸识别算法，并在web端中，将此算法的csta模型转化为onnx模型并进行使用。我暂时没有改进的思路。

## 3. web端代码介绍

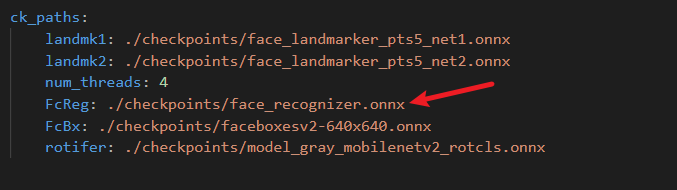
### 1）路径：

./models/facerecoger.py

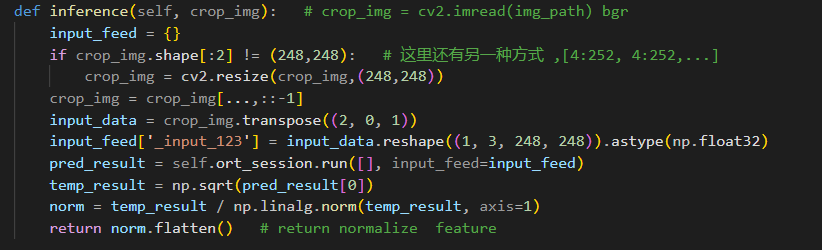
### 2）类定义：



`onnx\_path`表示调用的模型路径，通过`./config.yaml`指定：



### 3）方法定义：



`crop\_img`表示的是裁剪的人脸图片，return的是长度为1024特征向量（一维float数组，长度为1024）。对于两张人脸的特征向量，我们可以计算其相似度(内积),并将相似度和阈值比较，大于阈值则同一人。阈值设定在config.yaml中（人脸库中的人越多，这个阈值要设的越大，但请保证在[0.6 ,0.8]之间）：



# 六、图片质量评估

## 1.算法介绍

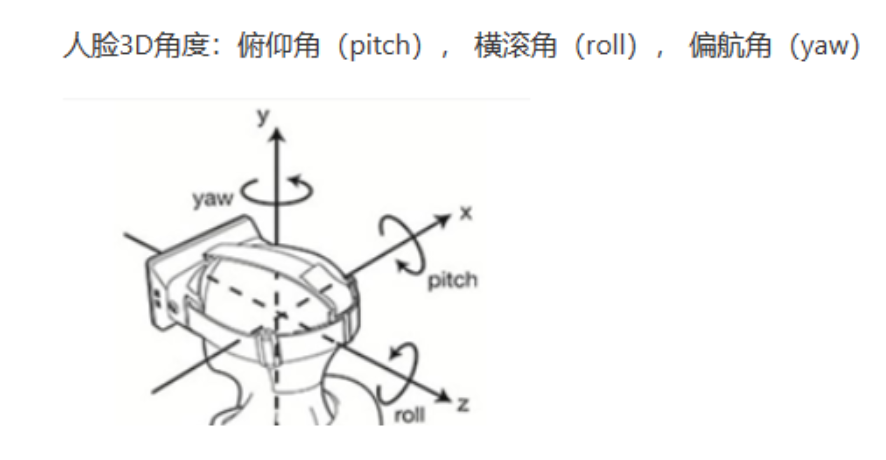
正文上文所说，我们需要进行图片质量评估，评估图片的：亮度、清晰度、完整度、姿态、分辨率，对其中不合格的属性返回并提示需要调整。

**亮度：**过亮或过暗的人脸都不好。

**清晰度：**清晰度这里是传统方式通过二次模糊后图像信息损失程度统计的清晰度（清晰度对于识别精度的影响相对不大）。

**完整度：**人脸检测得到的人脸框如果超出图片的边界，则认定其人脸不完整。

**姿态：**检测人脸的三个角度pitch，roll和yaw。如果其都在合理范围（人脸正对着摄像头或略微有旋转），则合格，否则不合格。(Yaw: 表示头部向左(正值)或向右旋转的角度。 Pitch: 表示头部向上(正值)或向下旋转的角度。Roll: 表示头部向左或向右侧倾的角度。)



**分辨率：**人脸区域部分的分辨率不应过小，我们设置的是112x112，当然可以设置的更小点。

代码详见：seetaface6的QualityAssessor3\QualityAssessor\src\seeta\QualityAssessor.cpp 或web端的./models/imgchecker.py。

## 2.web端代码介绍

### 1）config.yaml

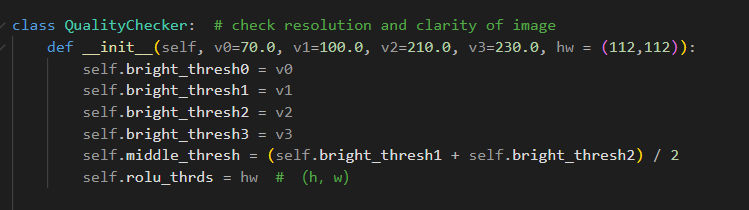
图片质量评估的相关配置在此一并给出。



图片质量评估都定义在：./models/imgchecker.py。

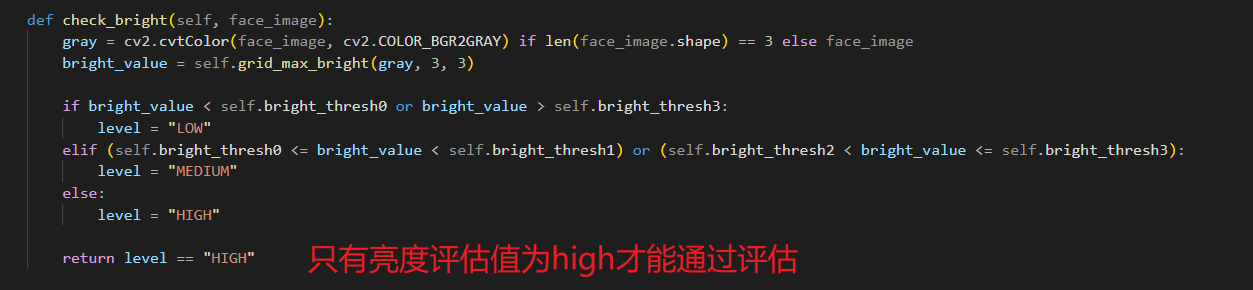
### 2）亮度/完整度/分辨率评估：

2.1. 类定义：

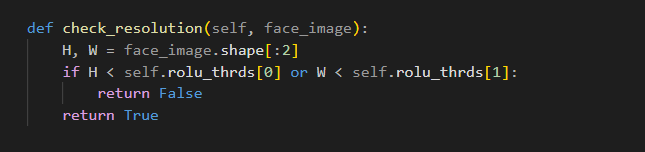


这是评估`亮度/完整度/分辨率`的类。`v0`, `v1`, `v2`, `v3` 表示图片亮度的四个阈值。如果图片的亮度值在[v1,v2] 之间，则表明图片亮度评估值为HIGH；落在区间[v0,v1] 或 [v2,v3]，则亮度评估值为MEDIUM，其余情况则为LOW。hw表示最低人脸最低分辨率，人脸高度H必须大于hw[0] ,宽度W必须大于hw[1]。

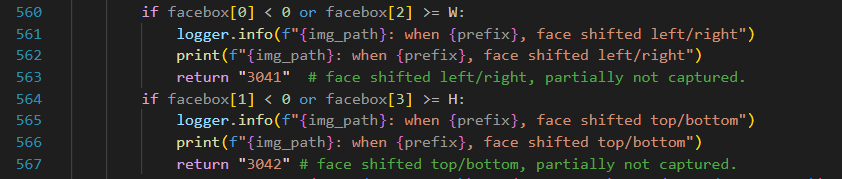
2.2 亮度评估方法：



2.3 分辨率：



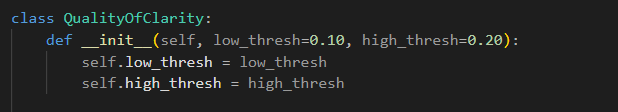
2.4 完整性：（./process.py中line560-567）



`facebox`表示人脸框（x1,y1,x2,y2）。点坐标：左上角点(x1,y1)，右下角点(x2,y2)。

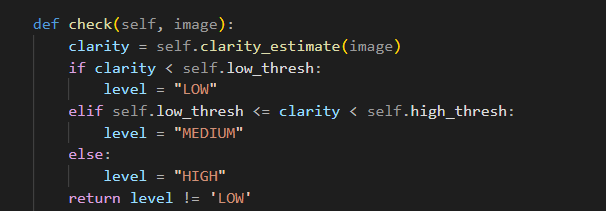
### 3）清晰度评估：

3.1类定义：



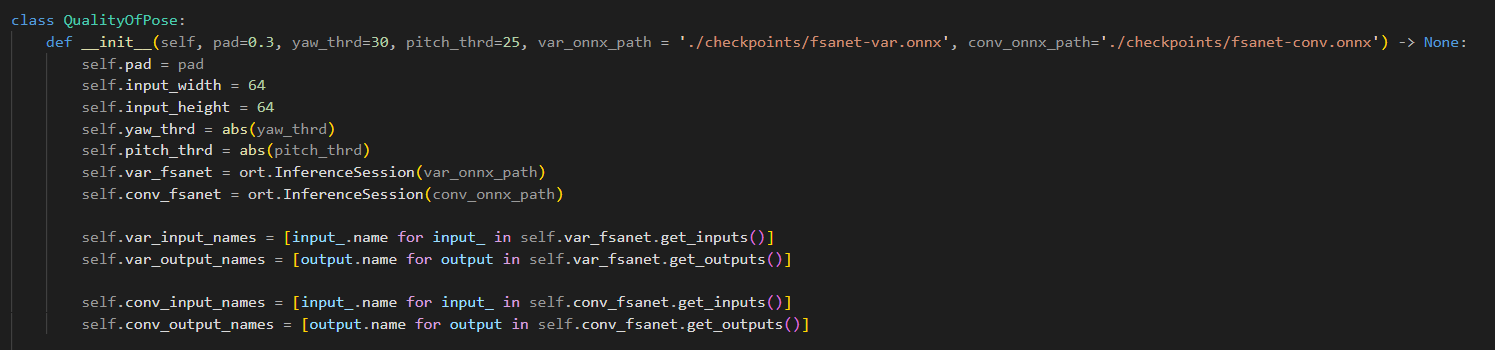
`low\_thresh`和`high\_thresh`表示低质量阈值和高质量阈值。

3.2 方法定义：



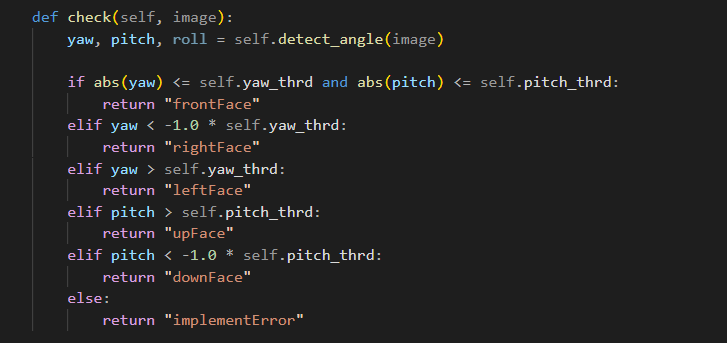
### 4）姿态评估：

4.1 类定义：



`pad`表示图片拓展的比例，`yaw\_thrd`表示yaw阈值，`pitch\_thrd`表示pitch阈值, `var\_onnx\_path`和 `conv\_onnx\_path`模型路径。其配置在config.yaml中：

4.2 方法定义：



输入的是通过人脸检测后裁剪的人脸区域图片（注意：没有经过人脸对齐）。输出的是这个人脸的姿态状态。

### 5）总结：

介绍下这个图片质量评估在实际代码中如何运用。（路径：./process.py）



# 七、总结

我们的项目使用的是开源项目[seetaface6](https://github.com/seetafaceengine/SeetaFace6)，其有更为优秀的教程：<https://github.com/seetafaceengine/SeetaFaceTutorial>。 推荐阅读。

Web端为了支持响应并发请求并进行并发运行，使用python的fastapi库，并将seetaface6中的csta模型转化为onnx模型（借鉴的是项目：<https://github.com/zjd1988/seetaface6_onnx_model> ，其issue有别人的提问，推荐阅读利于理解此项目）。