**第 五 次 实 验 报 告**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 课程名称 | 内容安全实验 | | | | |
| 学生姓名 | 梁刘琪 | 学号 | 2020302181202 | 指导老师 | 张典 |
| 专业 | 网安 | 班级 | 7班 | 实验时间 | 2023.5.5-5.10 |

**实验任务**

1、使用Python3+OpenCV+dlib实现人脸识别与关键点（landmarks）实时检测 15

2、结合实验任务1使用Python3+OpenCV+Deepface实现人脸情感检测

3、使用Python3+dlib实现人脸伪造

4、使用Python3+Face-X-Ray实现人脸伪造图像检测

5、使用dlib实现人脸

**任务一 人脸识别与关键点（landmarks）实时检测**

**一．实验原理**

**1.人脸采集**

不同的人脸图像通过摄像镜头采集得到，比如静态图像、动态图像、不同位置、不同表情等，当采集对象在设备的拍摄范围内时，采集设备会自动搜索并拍摄人脸图像。

**主要影响因素**

·图像大小

人脸图像过小会影响识别效果，人脸图像过大会影响识别速度，图像大小反映在实际应用场景就是人脸离摄像头的距离。

·模糊程度

实际场景主要着力解决运动模糊，人脸相对于摄像头的移动经常会产生运动模糊。部分摄像头有抗模糊的功能，在成本有限的情况下，考虑通过算法模型优化此问题。

·采集角度

人脸相对于摄像头角度为正脸最佳，因此算法模型需训练包含左右侧人脸、上下侧人脸的数据。

·图像分辨率

越低的图像分辨率越难识别，图像大小综合图像分辨率，直接影响摄像头识别距离。

·遮挡程度

五官无遮挡、脸部边缘清晰的图像为最佳，在实际场景中，很多人脸都会被帽子、眼镜、口罩等遮挡物遮挡，这部分数据需要根据算法要求决定是否留用训练。

·光照环境

过曝或过暗的光照环境都会影响人脸识别效果，可以从摄像头自带的功能补光或滤光平衡光照影响，也可以利用算法模型优化图像光线。

**2.人脸检测**

是检测出图像中人脸所在位置的一项技术。

人脸检测算法的输入是一张图片，输出是人脸框坐标序列（0个人脸框或1个人脸框或多个人脸框）。一般情况下，输出的人脸坐标框为一个正朝上的正方形，但也有一些人脸检测技术输出的是正朝上的矩形，或者是带旋转方向的矩形。

**3.人脸配准**

是定位出人脸上五官关键点坐标的一项技术。

人脸配准算法的输入是“一张人脸图片”加“人脸坐标框”，输出五官关键点的坐标序列。五官关键点的数量是预先设定好的一个固定数值，可以根据不同的语义来定义（常见的有5点、68点、90点等等）。

**4.人脸属性识别**

是识别出人脸的性别、年龄、姿态、表情等属性值的一项技术。

一般的人脸属性识别算法的输入是“一张人脸图”和“人脸五官关键点坐标”，输出是人脸相应的属性值。人脸属性识别算法一般会根据人脸五官关键点坐标将人脸对齐（旋转、缩放、扣取等操作后，将人脸调整到预定的大小和形态），然后进行属性分析。

1. **人脸特征提取（Face Feature Extraction）**

是将一张人脸图像转化为一串固定长度的数值的过程。这个数值串被称为“人脸特征（Face Feature）”，具有表征这个人脸特点的能力。

人脸提特征过程的输入也是 “一张人脸图”和“人脸五官关键点坐标”，输出是人脸相应的一个数值串（特征）。人脸提特征算法都会根据人脸五官关键点坐标将人脸对齐预定模式，然后计算特征。

1. **人脸识别应用**

**人脸比对（Face Compare）**

是衡量两个人脸之间相似度的算法，人脸比对算法的输入是两个人脸特征（注：人脸特征由前面的人脸提特征算法获得），输出是两个特征之间的相似度。人脸验证、人脸识别、人脸检索都是在人脸比对的基础上加一些策略来实现。相对人脸提特征过程，单次的人脸比对耗时极短，几乎可以忽略。

**人脸验证（Face Verification**）

是判定两个人脸图是否为同一人的算法。

它的输入是两个人脸特征，通过人脸比对获得两个人脸特征的相似度，通过与预设的阈值比较来验证这两个人脸特征是否属于同一人（即相似度大于阈值，为同一人；小于阈值为不同）。

**人脸检索**

是查找和输入人脸相似的人脸序列的算法。

人脸检索通过将输入的人脸和一个集合中的说有人脸进行比对，根据比对后的相似度对集合中的人脸进行排序。根据相似度从高到低排序的人脸序列即使人脸检索的结果。

**人脸聚类（Face Cluster）**

是将一个集合内的人脸根据身份进行分组的算法。

人脸聚类也通过将集合内所有的人脸两两之间做人脸比对，再根据这些相似度值进行分析，将属于同一个身份的人划分到一个组里。

1. **Dlib**

它是一个 C++ 工具库，包含机器学习算法，图像处理，网络及一些工具类库。Dlib 里面有两个人脸检测模块：

·基于HOG+SVM分类

HOG 检测直接使用 dlib.get\_frontal\_face\_detector()就可以获得

HOG 检测方法速度可以实时

·基于 Maximum-Margin Object Detector（MMOD） 的深度学习人脸检测方案

MMOD 检测需要下载mmod模型， 然后使用dlib.cnn\_face\_detection\_model\_

v1(model\_path)加载模型，其他使用接口就是一样的了

MMOD速度非常慢，相对HOG方法是几十倍的差异

**Dlib人脸配准**

Dlib提供的这两个检测模型都只能检测脸部区域在70\*70以上的图片，对于脸部区域太小的图片无法检测。Dlib检测出的脸部区域对于下巴和额头区域会做过多的裁剪，这也是这两个模型的缺点。Dlib里面提供了两个关键点预测模型，分别是5 face landmarks，68 face landmarks。

根据关键点信息来判断人脸处于图像中的位置和倾斜的角度。比较简单的一种方法是根据两个眼睛的相对位置和大小来校正。一般而言校正之后左眼中心在图像0.2-0.4处，相对应右眼应该在0.6-0.8处，值越大则最后得到的人脸在图像中的占比越小。同时所有左右眼部关键点位置可以用来判断脸在平面内的倾斜程度。

1. **OpenCV**

是计算机视觉领域应用最广泛的开源工具包，基于C/C++， 支持Linux/Windows/MacOS/Android/iOS，并提供了Python，Matlab和Java等语言的接口，因为其丰富的接口，优秀的性能和商业友好的 使用许可，不管是学术界还是业界中都非常受欢迎。OpenCV旨在提供一个用于计算机视觉的科研和商业应用的高性能通用库。

根据功能和需求的不同，OpenCV中的函数接口大体可以分为如下部分：

**core：**核心模块，主要包含了OpenCV中最基本的结构（矩阵，点线和形状等），以及相关的基础运算/操作。

**imgproc：**图像处理模块，包含和图像相关的基础功能（滤波，梯度，改变大小等），以及一些衍生的高级功能

（图像分割，直方图，形态分析和边缘/直线提取等）。

**highgui**：提供了用户界面和文件读取的基本函数，比如图像显示窗口的生成和控制，图像/视频文件的IO等。

**·针对视频和一些特别的视觉应用，OpenCV也提供了强劲的支持：**

video：用于视频分析的常用功能，比如光流法（Optical Flow） 和目标跟踪等。

calib3d：三维重建，立体视觉和相机标定等的相关功能。

features2d：二维特征相关的功能，主要是一些不受专利保护的， 商业友好的特征点检测和匹配等功能，比如ORB特征。

object：目标检测模块，包含级联分类和Latent SVM

ml：机器学习算法模块，包含一些视觉中最常用的传统机器学习 算法。

flann：最近邻算法库，Fast Library for Approximate Nearest Nei ghbors，用于在多维空间进行聚类和检索，经常和关键点匹配搭 配使用。

gpu：包含了一些gpu加速的接口，底层的加速是CUDA实现。

photo：计算摄像学（Computational Photography）相关的接口， 当然这只是个名字，其实只有图像修复和降噪而已。

stitching：图像拼接模块，有了它可以自己生成全景照片。

nonfree：受到专利保护的一些算法，其实就是SIFT和SURF。

contrib：一些实验性质的算法，考虑在未来版本中加入的。

legacy：字面是遗产，意思就是废弃的一些接口，保留是考虑到 向下兼容。

ocl：利用OpenCL并行加速的一些接口。

superres：超分辨率模块，其实就是BTV-L1（Biliteral Total Varia tion – L1 regularization）算法

viz：基础的3D渲染模块，其实底层就是著名的3D工具包VTK（Vi sualization Toolkit）。

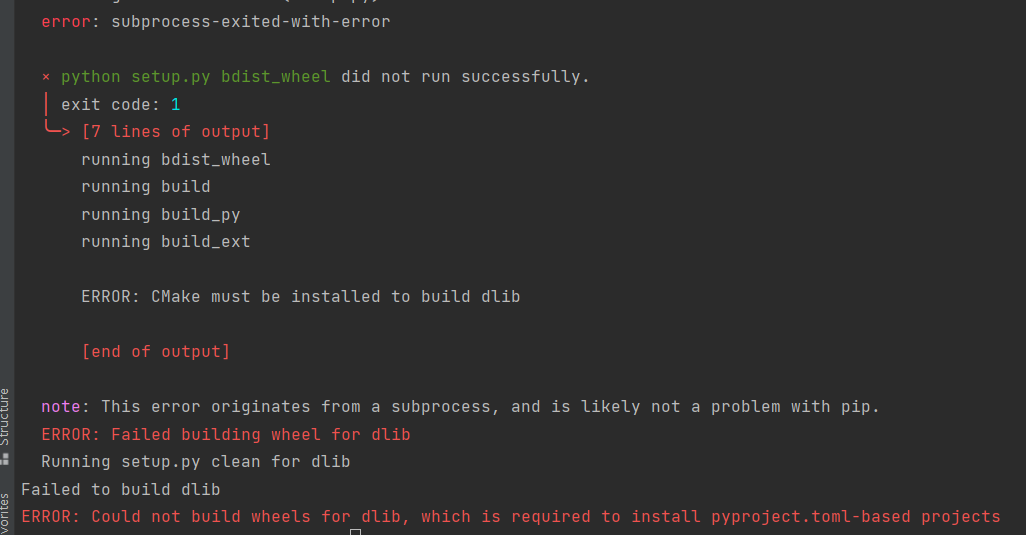
1. **环境准备**
2. **安装dlib**

**安装dlib前需要先安装cmake 和[boost](https://so.csdn.net/so/search?q=boost&spm=1001.2101.3001.7020" \t "https://blog.csdn.net/weixin_45875199/article/details/_blank)。然后才能正确安装dlib**

pip install dlib



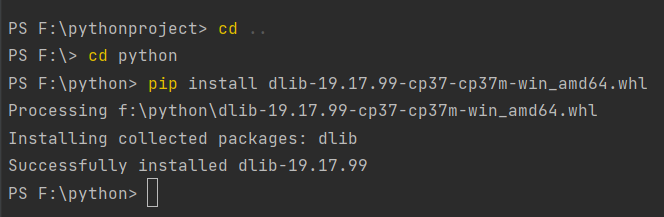
报错



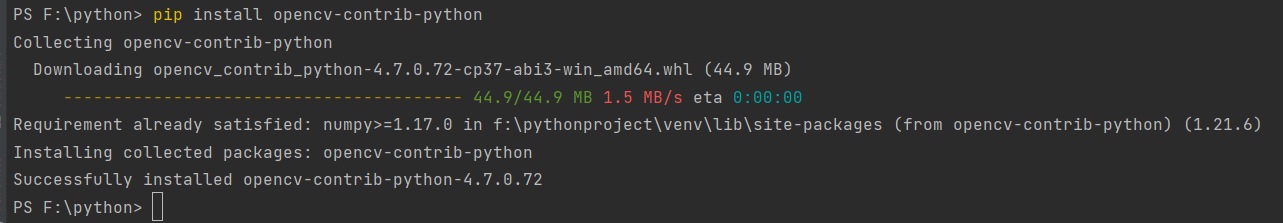
下载对应版本的dlib.whl文件https://blog.csdn.net/weixin\_42538848/article/details/128280584



手动安装，成功



1. **安装opencv**



1. **实验步骤**

读取摄像头（或者视频文件）中的视频帧，实时检测人脸，并在屏幕上绘制出人脸关键点信息。

**导入包**

1. import threading
2. import cv2
3. import dlib

**加载人脸分类器和人脸检测器**

1. *# Load the detector*
2. detector = dlib.get\_frontal\_face\_detector()
3. *# Load the predictor*
4. predictor = dlib.shape\_predictor("shape\_predictor\_68\_face\_landmarks.dat")

get\_frontal\_face\_detector()函数返回一个对象，可以用于在图像中检测正面的人脸。从"shape\_predictor\_68\_face\_landmarks.dat"文件中加载了一个预训练的面部特征点检测器。该文件包含一个机器学习模型的参数，该模型可以将人脸图像作为输入，并输出人脸上68个特定点的位置，如眼睛和嘴巴的角落等。

**初始化类的实例对象**

1. def \_\_init\_\_(self, win\_name, cam\_name):
2. super().\_\_init\_\_()
3. self.cam\_name = cam\_name
4. self.win\_name = win\_name

super().\_\_init\_\_()：确保这个类的实例对象正确地初始化。

**读取测试视频test.mp4**

1. capture = cv2.VideoCapture('test.mp4')

**按帧读取视频帧**

1. ret, frame = capture.read()

ret,frame是获取cap.read()方法的两个返回值。

其中ret是布尔值，如果读取帧是正确的则返回True， 如果文件读取到结尾，它的返回值就为False。

frame就是每一帧的图像，是个三维矩阵。

**取灰度**

1. gray = cv2.cvtColor(src=frame, code=cv2.COLOR\_BGR2GRAY)

**识别人脸数**

1. faces = detector(frame)

**使用OpenCV和dlib库进行人脸检测和面部特征点检测**

1. if ret is True:
2. b, g, r = cv2.split(frame)  *# 分离色道 opencv读入的色道是B,G,R*
3. fame2 = cv2.merge([r, g, b])  *# 合成R,G,B*

检查一个名为"ret"的变量是否设置为True。如果是，代码将使用OpenCV的"cv2.split()"和"cv2.merge()"函数分离和重新组合视频帧中的颜色通道，然后将结果图像传递给dlib的"detector"对象进行人脸检测。传递给detector函数的参数"1"指定图像应该被上采样一次以提高检测准确性。

**人脸检测**

1. dets = detector(fame2, 1)  *# 使用detector进行人脸检测*
2. for i, face in enumerate(dets):  *# 遍历返回值 index是几个人脸*
3. shape = predictor(fame2, face)  *# 寻找人脸的68个脸*

如果检测器在图像中发现一个或多个人脸，代码将进入一个for循环，遍历检测到的人脸。对于每个人脸，代码使用dlib的"predictor"对象来查找对应于人脸的68个面部特征点。这些特征点可以用于各种用途，例如面部对齐、表情识别和面部识别。for循环中的索引变量"i"指示正在处理哪个人脸。

**将检测到的人脸的68个面部特征点用蓝色圆圈标记在视频帧上**

1. for index, pt in enumerate(shape.parts()):  *# 遍历所有的点，把点用蓝色的圈圈表示出来*
2. pt\_pox = (pt.x, pt.y)
3. cv2.circle(frame, pt\_pox, 2, (255, 0, 0), 2)

cv2.circle()"函数将蓝色圆圈绘制在视频帧上，以标示特征点的位置。函数的参数依次为：视频帧对象、特征点坐标、圆圈半径、颜色（蓝色）、线宽度。最终，视频帧上的所有特征点都会被用蓝色圆圈标记出来。

**展示每一帧图片**

1. cv2.imshow("me", frame)

**等待键盘输入，通常用q来判断是否退出**

1. k = cv2.waitKey(5)
2. if k & 0xff == ord('q'):
3. break

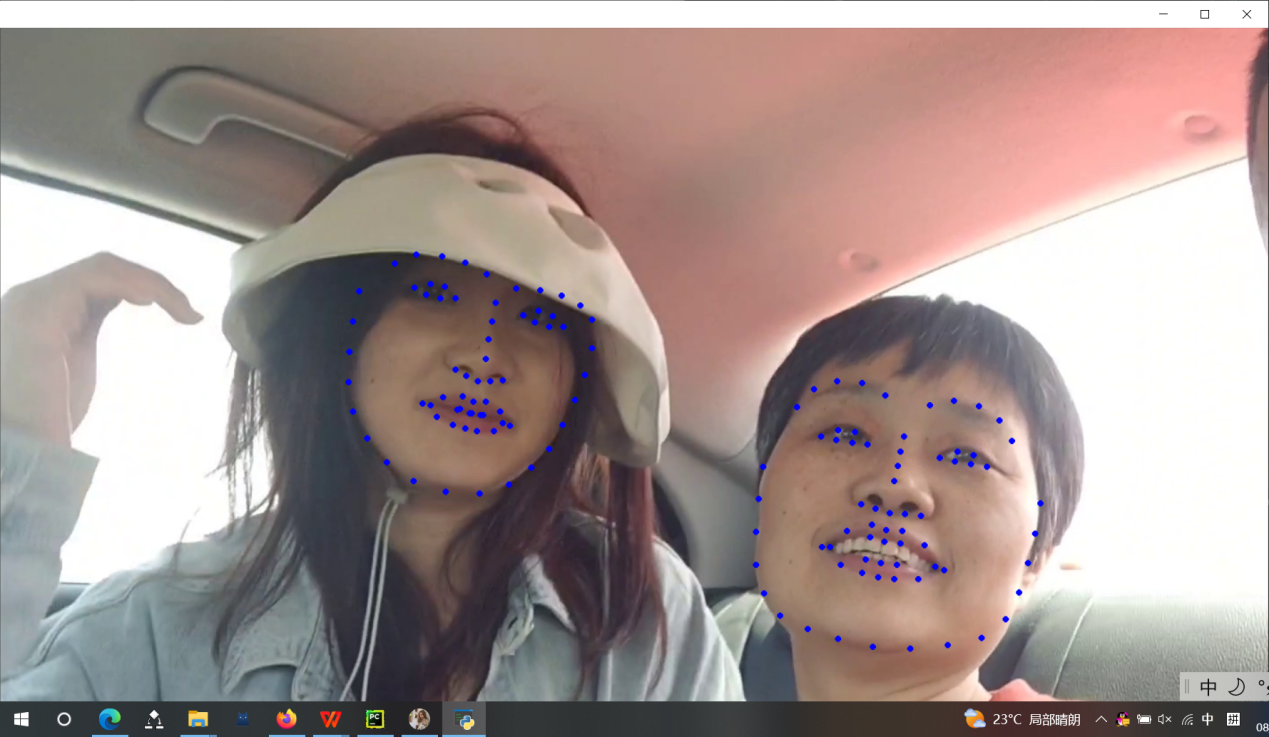
waitKey（）方法本身表示等待键盘输入，参数是1，表示延时1ms切换到下一帧图像，对于视频而言。

1. **实验结果**

**测试视频：test.mp4**



**识别结果：**



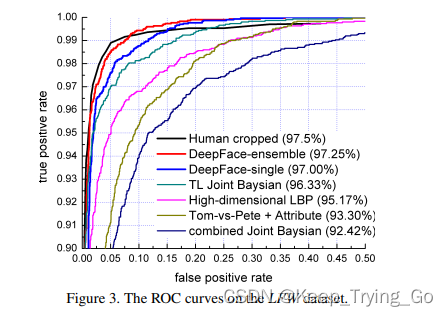
识别到视频中所有人脸，并正确标注

**任务二 使用Python3+OpenCV+Deepface实现人脸情感检测**

1. **实验原理**
2. **Deepface**

Deepface 是一个轻量级的python人脸识别和人脸属性分析（年龄、性别、情感和种族）框架。 它包含最先进的模型：VGG-Face、Google FaceNet、OpenFace、Facebook DeepFace、DeepID、ArcFace 和 Dlib。

**2.DeepFace中模型性能的比较**



一般情况下，根据LFW数据集上的实验结果，FaceNet, VGG-Face, ArcFace 和 Dlib 总体性能优于： OpenFace, DeepFace 和 DeepID。

FaceNet (/w 512d) ：99.65%

ArcFace ： 99.40%

Dlib ： 99.38%

VGG-Face ： 98.78%

OpenFace ： 93.80%

人类水平： 97.53%

1. **Deepface接口**

面部识别Demo

现代人脸识别流程由5个常见阶段组成：检测、对齐、归一化、表示和验证。虽然Deepface在后台处理所有这些常见阶段，但您不需要获得关于其背后所有过程的in-depth知识。只需一行代码即可调用其验证、查找或分析函数。

人脸验证Demo

此函数验证人脸对是否为同一个人或不同的人。它需要精确的图像路径作为输入。也欢迎传递numpy或based64编码图像。然后，它将返回一个字典，您应该只检查它的验证密钥。

人脸识别Demo

人脸识别需要多次应用人脸验证。这里，deepface有一个out-of-the-boxfind函数来处理这个动作。它将在数据库路径中查找输入图像的标识，并返回pandas数据帧作为输出。

**3.cv2.putText()函数用法**

cv2.putText(image, text, org, font, fontScale, color[, thickness[, lineType[, bottomLeftOrigin]]])

参数：

image:要在其上绘制文本的图像。

text:要绘制的文本字符串。

org:它是图像中文本字符串左下角的坐标。坐标表示为两个值的元组，即(X坐标值，Y坐标值)。

font:它表示字体类型。一些字体类型是FONT\_HERSHEY\_SIMPLEX,FONT\_HERSHEY\_PLAIN等。

fontScale:字体比例因子乘以font-specific基本大小。

color:它是要绘制的文本字符串的颜色。对于BGR，我们通过一个元组。例如：(255，0，0)为蓝色。

thickness:它是线的粗细像素。

lineType:这是一个可选参数，它给出了要使用的行的类型。

bottomLeftOrigin:这是一个可选参数。如果为true，则图像数据原点位于左下角。否则，它位于左上角。

**4.cv2.circle()**

用法： cv2.circle(image, center\_coordinates, [radius](https://so.csdn.net/so/search?q=radius&spm=1001.2101.3001.7020" \t "https://blog.csdn.net/weixin_39952074/article/details/_blank), color, thickness)

参数：

image:它是要在其上绘制圆的图像。

center\_coordinates：它是圆的中心坐标。坐标表示为两个值的元组，即(X坐标值，Y坐标值)。

radius:它是圆的半径。

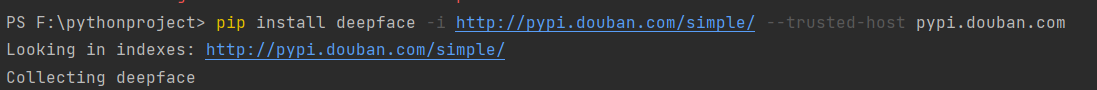
color:它是要绘制的圆的边界线的颜色。对于BGR，我们通过一个元组。例如：(255，0，0)为蓝色。

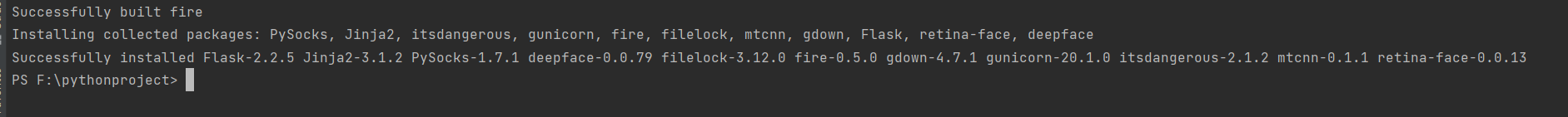
thickness:它是圆边界线的粗细像素。厚度-1像素将以指定的颜色填充矩形形状。

返回值：它返回一个图像。

1. **环境搭建**
2. **Deepface**

pip install deepface -i http://pypi.douban.com/simple/ --trusted-host pypi.douban.com





1. **实验步骤**

**导入包**

1. import threading
2. import cv2
3. import dlib

**加载人脸分类器和人脸检测器**

1. *# Load the detector*
2. detector = dlib.get\_frontal\_face\_detector()
3. *# Load the predictor*
4. predictor = dlib.shape\_predictor("shape\_predictor\_68\_face\_landmarks.dat")

get\_frontal\_face\_detector()函数返回一个对象，可以用于在图像中检测正面的人脸。从"shape\_predictor\_68\_face\_landmarks.dat"文件中加载了一个预训练的面部特征点检测器。该文件包含一个机器学习模型的参数，该模型可以将人脸图像作为输入，并输出人脸上68个特定点的位置，如眼睛和嘴巴的角落等。

**初始化类的实例对象**

1. def \_\_init\_\_(self, win\_name, cam\_name):
2. super().\_\_init\_\_()
3. self.cam\_name = cam\_name
4. self.win\_name = win\_name

super().\_\_init\_\_()：确保这个类的实例对象正确地初始化。

**读取测试视频test.mp4**

1. capture = cv2.VideoCapture('test.mp4')

**按帧读取视频帧**

1. ret, frame = capture.read()

ret,frame是获取cap.read()方法的两个返回值。

其中ret是布尔值，如果读取帧是正确的则返回True， 如果文件读取到结尾，它的返回值就为False。

frame就是每一帧的图像，是个三维矩阵。

**取灰度**

1. gray = cv2.cvtColor(src=frame, code=cv2.COLOR\_BGR2GRAY)

**识别人脸数**

1. faces = detector(frame)

**使用OpenCV和dlib库进行人脸检测和面部特征点检测**

1. if ret is True:
2. b, g, r = cv2.split(frame)  *# 分离色道 opencv读入的色道是B,G,R*
3. fame2 = cv2.merge([r, g, b])  *# 合成R,G,B*

检查一个名为"ret"的变量是否设置为True。如果是，代码将使用OpenCV的"cv2.split()"和"cv2.merge()"函数分离和重新组合视频帧中的颜色通道，然后将结果图像传递给dlib的"detector"对象进行人脸检测。传递给detector函数的参数"1"指定图像应该被上采样一次以提高检测准确性。

**人脸检测**

1. dets = detector(fame2, 1)  *# 使用detector进行人脸检测*
2. for i, face in enumerate(dets):  *# 遍历返回值 index是几个人脸*
3. shape = predictor(fame2, face)  *# 寻找人脸的68个脸*

如果检测器在图像中发现一个或多个人脸，代码将进入一个for循环，遍历检测到的人脸。对于每个人脸，代码使用dlib的"predictor"对象来查找对应于人脸的68个面部特征点。这些特征点可以用于各种用途，例如面部对齐、表情识别和面部识别。for循环中的索引变量"i"指示正在处理哪个人脸。

**将检测到的人脸的68个面部特征点用蓝色圆圈标记在视频帧上**

1. for index, pt in enumerate(shape.parts()):  *# 遍历所有的点，把点用蓝色的圈圈表示出来*
2. pt\_pox = (pt.x, pt.y)
3. cv2.circle(frame, pt\_pox, 2, (255, 0, 0), 2)

cv2.circle()"函数将蓝色圆圈绘制在视频帧上，以标示特征点的位置。函数的参数依次为：视频帧对象、特征点坐标、圆圈半径、颜色（蓝色）、线宽度。最终，视频帧上的所有特征点都会被用蓝色圆圈标记出来。

**Deepface检测年龄和情绪**

1. for k, d in enumerate(ffaces):
2. *# 用红色矩形框出人脸*
3. *#cv2.rectangle(frame, (d.left(), d.top()), (d.right(), d.bottom()), (0, 0, 255))*
4. *# 计算人脸热别框边长*
5. self.face\_width = d.right() - d.left()
6. obj = DeepFace.analyze(img\_path=frame, actions=['age', 'emotion'])
7. age = obj[0]['age']
8. emo = obj[0]['dominant\_emotion']
9. cv2.putText(frame, str(age), (d.left(), d.bottom() + 20), cv2.FONT\_HERSHEY\_SIMPLEX, 0.8, (0, 0, 255), 2,
10. 4)
11. cv2.putText(frame, str(emo), (d.left(), d.bottom() + 50), cv2.FONT\_HERSHEY\_SIMPLEX, 0.8, (0, 0, 255), 2,
12. 4)

一个for循环遍历检测到的人脸。对于每个人脸，代码使用OpenCV的"cv2.rectangle()"函数将红色矩形框绘制在视频帧上，以框出人脸的位置。

使用人脸的左右坐标计算人脸的宽度，并将其存储在"face\_width"变量中。

使用DeepFace的"analyze()"函数分析视频帧中的人脸。函数的参数包括视频帧、需要执行的动作（这里是年龄和情绪）。结果存储在"obj"变量中。

从"obj"变量中提取人脸的年龄和情绪，并使用OpenCV的"cv2.putText()"函数将它们添加到视频帧中。函数的参数包括视频帧对象、文本、文本的位置（在人脸下方）、字体、字体大小、颜色（红色）、线宽度和线类型。最终，视频帧上会显示检测到的人脸，以及人脸的年龄和情绪信息。

**展示每一帧图片**

1. cv2.imshow("me", frame)

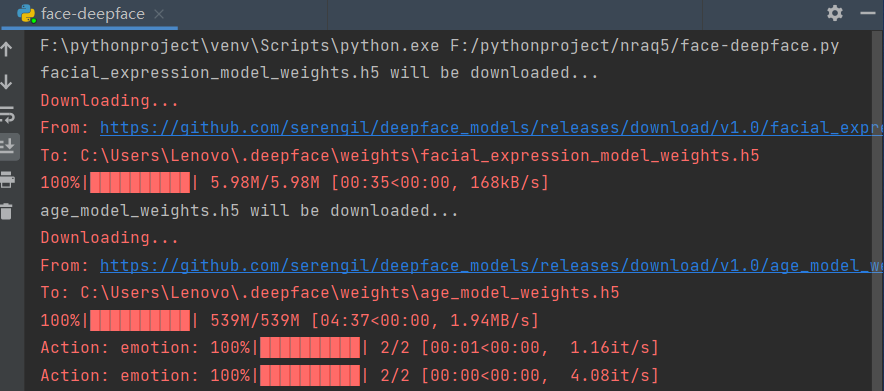
**等待键盘输入，通常用q来判断是否退出**

1. k = cv2.waitKey(5)
2. if k & 0xff == ord('q'):
3. break

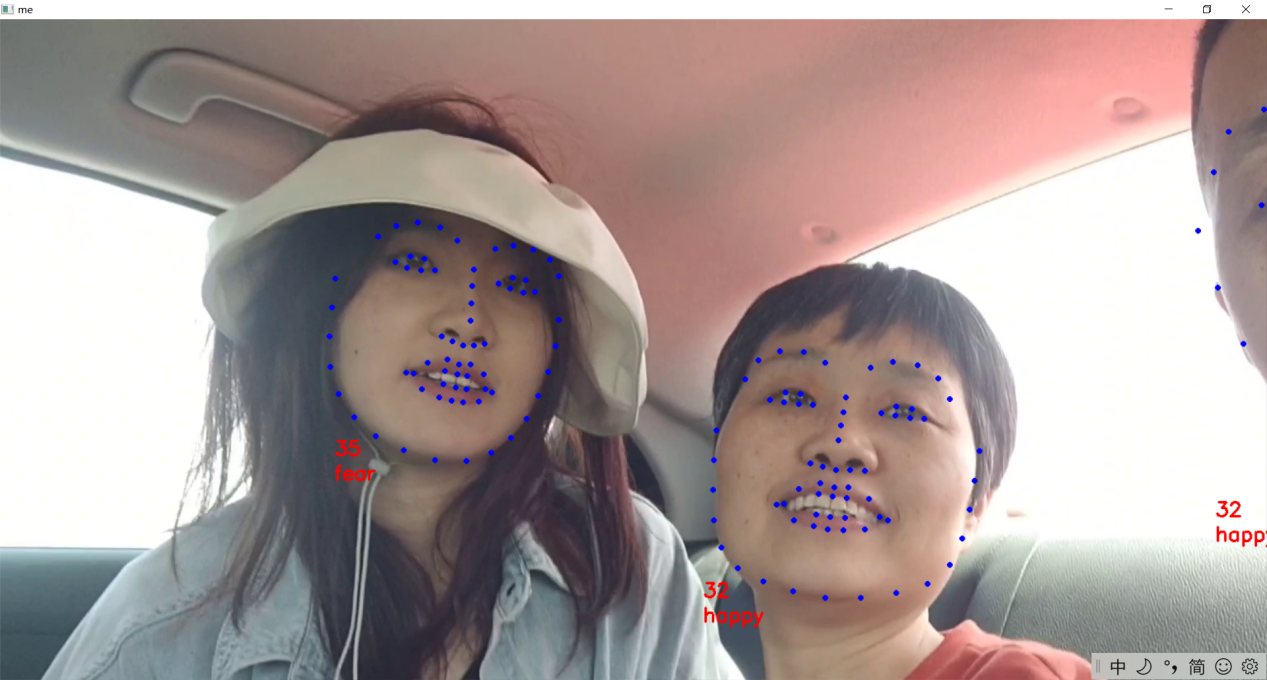
waitKey（）方法本身表示等待键盘输入，参数是1，表示延时1ms切换到下一帧图像，对于视频而言。

1. **实验结果**

**模型下载成功如图：**



**识别年龄和心情如图：**



左：实际年龄22，检测年龄：35

实际心情：natural 检测心情：fear（可能是因为左边检测对象嘴角弧度向下的原因）

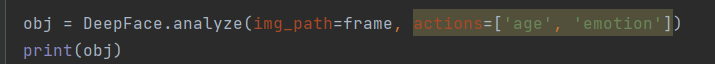
右：实际年龄48，检测年龄：32

实际心情：happy 检测心情：happy

心情检测较准，年龄检测非常不准确。

1. **实验心得**

输出obj，发现obj字符串中有”age” “domianant\_emotion”





所以cv2.puttext的第二个参数需要用str（age）

**任务三 使用Python3+dlib实现人脸伪造**

1. **实验原理**

**1.基础原理**

**1.1人脸互换**

人脸互换部分主要实现的功能是，给定任意两张人脸图片，通过一系列操作，使两个人的脸部交换，这部分需要的问题有：

1.不同的人的脸部结构千差万别，同一个人也会因为角度、面部表情的不同而导致差别，即如何实现不同图片的人脸对齐；

2.不同人脸的肤色、光照不同，即不同图片的面部亮度不同，在换脸后如何与整体亮度统一；

3.不同人脸的纹理不同，比如老人的皱纹等，如何实现换脸后纹理的统一。

**1.2人脸融合（face morph）**

在人脸融合部分，我们需要实现给定任意两张人脸图片和融合度α，通过一系列操作，实现两个人脸的融合。这一部分的困难在于：

1.人脸结构的检测与分割。对于给定的人脸图片，人脸的结构差异很大。

2.人脸融合度的构建。对于给定的融合度 α，如何对两张图片的人脸取样与映射。

**1.3本征脸（eigen face）**

在这一部分，需要对较大的数据集（几百张，几千张人脸图片）进行处理，通过主成分分析的方法，得到一定数量的人脸主成分。这一部分的主要困难在于数据集的预处理， 我们需要将不同图片中的人脸对齐，才能进行后续的处理。

**2.算法结构与处理过程**

**2.1人脸关键点检测**

实现人脸变换的第一步，便是人脸关键点的检测，得到图片中的人脸的结构。在这里，我们利用旷视face++的API 来定位人脸关键的集合{V1,V2,…,Vn}，其中关键点的数量可以选择 83 或者 106.

**2.2计算凸包**

在获取人脸关键点集合后，我们需要计算这些关键点的凸包（convex hull）（凸包是一个计算几何（图形学）中的概念：在一个实数向量空间 V 中，对于给定集合 X，所有包含X 的凸集的交集 S 被称为 X 的凸包。X 的凸包可以用 X 内所有点(X1，…Xn)的凸组合来构造.）在这里，我们计算凸包是为了获取这些人脸关键点组成的一个人脸区域。

**2.3德劳内（Delaunay）三角划分**

在获得凸包以后，我们对凸包内的人脸关键点进行德劳内三角的划分。德劳内三角划分能将我们的凸包区域进行分割，本文转载自http://www.biyezuopin.vip/onews.asp?id=14906并且更易于保留人脸的细节部分，并且因为获取仿射变换需要原图片和目标图片的各三个点，正好对应于原图和目标图的对应的德劳内三角。

**2.4进行仿射变换**

在获得原图片和目标图图片的德劳内三角以后，我们需要寻找两张图对应的三角形对，对这样的每一对三角形，我们可以计算得到一个仿射函数。这个仿射函数将被用于三角形对之间的仿射变换。重复这个操作，直到所有区域都操作完毕，我们得到了人脸的位置变换。

**2.5无缝融合**

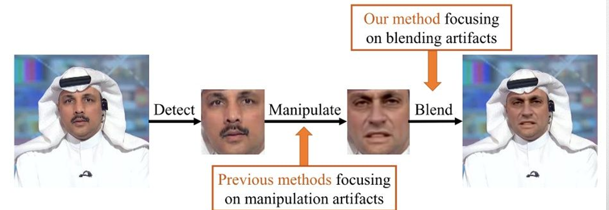
在上述人脸仿射变换后，我们得到人脸结构和位置的变换，但我们没有对人脸区域亮度进行调整，这样会造成人脸区域和其他区域的颜色协调的问题。所以最后我们用 opencv 的无缝融合函数seamlessClone()来实现无缝融合操作。

**3.传统的 AI 换脸一般分三步走：**

（1）检测目标图像的面部区域；

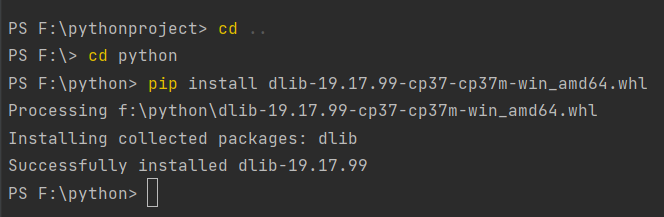
（2）利用 AI 换脸算法生成新的面部及一部分周围区域；

（3）将生成的新面部融合到原图像中，替换原图像中的面部。



1. **环境准备**

**安装dlib**



**三．实验步骤**

**导入库**

1. import cv2
2. import dlib
3. import numpy

cv2：OpenCV（开源计算机视觉库）是一个主要针对实时计算机视觉的编程函数库。它提供了执行图像和视频处理任务的工具，例如读取和写入图像，转换图像，过滤图像，对象检测等。

dlib：Dlib是一个现代的C++工具包，包含机器学习算法和用于在C++中创建复杂软件以解决实际问题的工具。

numpy：NumPy是Python编程语言的一个库，为其添加了支持大型多维数组和矩阵的功能，以及大量的高级数学函数来操作这些数组。

**定义常量并初始化**

1. PREDICTOR\_PATH = "shape\_predictor\_68\_face\_landmarks.dat"
2. SCALE\_FACTOR = 1
3. FEATHER\_AMOUNT = 11

PREDICTOR\_PATH = "shape\_predictor\_68\_face\_landmarks.dat"：这个常量定义了一个字符串，表示面部特征检测模型文件的路径和文件名。这个文件包含了一个训练好的机器学习模型，可以用来检测面部特征，比如眼睛、嘴巴、鼻子等。

SCALE\_FACTOR = 1：这个常量定义了一个数字，表示在进行面部对齐时应该缩放的因子。这个因子可以用来调整面部对齐的大小和比例。

FEATHER\_AMOUNT = 11：这个常量定义了一个数字，表示在进行面部融合时应该使用的羽化量。羽化是一种图像处理技术，可以用来减少两个图像之间的边缘锯齿状感觉，使它们看起来更加平滑和自然。

1. FACE\_POINTS = list(range(17, 68))
2. MOUTH\_POINTS = list(range(48, 61))
3. RIGHT\_BROW\_POINTS = list(range(17, 22))
4. LEFT\_BROW\_POINTS = list(range(22, 27))
5. RIGHT\_EYE\_POINTS = list(range(36, 42))
6. LEFT\_EYE\_POINTS = list(range(42, 48))
7. NOSE\_POINTS = list(range(27, 35))
8. JAW\_POINTS = list(range(0, 17))

FACE\_POINTS = list(range(17, 68))：这个列表包含了人脸所有关键点的索引，这些关键点可以用于进行面部对齐和面部融合等操作。

MOUTH\_POINTS = list(range(48, 61))：这个列表包含了人脸嘴巴部位的关键点索引。

RIGHT\_BROW\_POINTS = list(range(17, 22))：这个列表包含了人脸右眉毛部位的关键点索引。

LEFT\_BROW\_POINTS = list(range(22, 27))：这个列表包含了人脸左眉毛部位的关键点索引。

RIGHT\_EYE\_POINTS = list(range(36, 42))：这个列表包含了人脸右眼部位的关键点索引。

LEFT\_EYE\_POINTS = list(range(42, 48))：这个列表包含了人脸左眼部位的关键点索引。

NOSE\_POINTS = list(range(27, 35))：这个列表包含了人脸鼻子部位的关键点索引。

JAW\_POINTS = list(range(0, 17))：这个列表包含了人脸下巴部位的关键点索引。

1. ALIGN\_POINTS = (LEFT\_BROW\_POINTS + RIGHT\_EYE\_POINTS + LEFT\_EYE\_POINTS +
2. RIGHT\_BROW\_POINTS + NOSE\_POINTS + MOUTH\_POINTS)
3. OVERLAY\_POINTS = [
4. LEFT\_EYE\_POINTS + RIGHT\_EYE\_POINTS + LEFT\_BROW\_POINTS + RIGHT\_BROW\_POINTS,
5. NOSE\_POINTS + MOUTH\_POINTS,
6. ]
7. COLOUR\_CORRECT\_BLUR\_FRAC = 0.6
9. detector = dlib.get\_frontal\_face\_detector()
10. predictor = dlib.shape\_predictor(PREDICTOR\_PATH)

ALIGN\_POINTS = (LEFT\_BROW\_POINTS + RIGHT\_EYE\_POINTS + LEFT\_EYE\_POINTS + RIGHT\_BROW\_POINTS + NOSE\_POINTS + MOUTH\_POINTS)：这个常量定义了一个元组，包含了用于面部对齐的关键点索引。这些关键点位于人脸的眉毛、眼睛、鼻子和嘴巴部位。

OVERLAY\_POINTS = [LEFT\_EYE\_POINTS + RIGHT\_EYE\_POINTS + LEFT\_BROW\_POINTS + RIGHT\_BROW\_POINTS, NOSE\_POINTS + MOUTH\_POINTS]：这个常量定义了一个列表，包含了用于面部融合的关键点索引。这些关键点位于人脸的眉毛、眼睛、鼻子和嘴巴部位。

COLOUR\_CORRECT\_BLUR\_FRAC = 0.6：这个常量定义了一个数字，表示在进行颜色校正时应该使用的模糊量。颜色校正是一种图像处理技术，可以用来使不同图像之间的颜色更加一致。

detector = dlib.get\_frontal\_face\_detector()：这个语句初始化了一个人脸检测器对象，可以用来检测图像中的人脸。

predictor = dlib.shape\_predictor(PREDICTOR\_PATH)：这个语句初始化了一个面部特征检测器对象，可以用来检测人脸的关键点。这个对象需要一个面部特征检测模型文件的路径和文件名，这个路径和文件名被定义为之前的一个常量 PREDICTOR\_PATH。

**定义异常类**

1. class TooManyFaces(Exception):
2. pass

定义异常类 TooManyFaces。这个异常类表示检测到了超过一个人脸，而期望只有一个人脸的情况。当检测到多张人脸时，可以抛出这个异常来通知调用者发生了错误。

**检测人脸函数**

1. def get\_landmarks(im):
2. rects = detector(im, 1)
4. if len(rects) > 1:
5. raise TooManyFaces
6. if len(rects) == 0:
7. raise NoFaces
9. return numpy.matrix([[p.x, p.y] for p in predictor(im, rects[0]).parts()])

get\_landmarks，它接受一个图像 im 作为参数，并返回一个矩阵，其中包含了检测到的人脸的所有关键点坐标。

**标记人脸函数**

1. def annotate\_landmarks(im, landmarks):
2. im = im.copy()
3. for idx, point in enumerate(landmarks):
4. pos = (point[0, 0], point[0, 1])
5. cv2.putText(im, str(idx), pos,
6. fontFace=cv2.FONT\_HERSHEY\_SCRIPT\_SIMPLEX,
7. fontScale=0.4,
8. color=(0, 0, 255))
9. cv2.circle(im, pos, 3, color=(0, 255, 255))
10. return im

接受一个图像 im 和一个关键点矩阵 landmarks 作为参数，并返回一个带有关键点标记的图像。绘制了当前关键点的索引号，绘制了当前关键点的位置标记，使用黄色圆圈表示。

**面部融合函数**

1. def draw\_convex\_hull(im, points, color):
2. points = cv2.convexHull(points)
3. cv2.fillConvexPoly(im, points, color=color)

cv2.convexHull() 函数来计算指定点集的凸包，并将结果保存在变量 points 中，cv2.fillConvexPoly() 函数，在图像 im 上绘制一个填充的凸多边形，其顶点为 points，颜色为 color。

**覆盖面部**

get\_face\_mask，它接受一个图像 im 和一个关键点矩阵 landmarks 作为参数，并返回一个掩膜图像，用于指定在进行面部融合时哪些区域需要被覆盖。

1. def get\_face\_mask(im, landmarks):
2. im = numpy.zeros(im.shape[:2], dtype=numpy.float64)
4. for group in OVERLAY\_POINTS:
5. draw\_convex\_hull(im,
6. landmarks[group],
7. color=1)
9. im = numpy.array([im, im, im]).transpose((1, 2, 0))
11. im = (cv2.GaussianBlur(im, (FEATHER\_AMOUNT, FEATHER\_AMOUNT), 0) > 0) \* 1.0
12. im = cv2.GaussianBlur(im, (FEATHER\_AMOUNT, FEATHER\_AMOUNT), 0)
14. return im

im = numpy.array([im, im, im]).transpose((1, 2, 0))：这个语句将 im 中的数据转换为一个三通道图像，其中每个通道都包含了相同的数据。

im = (cv2.GaussianBlur(im, (FEATHER\_AMOUNT, FEATHER\_AMOUNT), 0) > 0) \* 1.0：这个语句对图像进行高斯模糊，并将模糊后的像素值大于 0 的部分视为掩膜的有效区域。这个有效区域被赋值为 1.0，其余部分为 0。

im = cv2.GaussianBlur(im, (FEATHER\_AMOUNT, FEATHER\_AMOUNT), 0)：这个语句对掩膜图像进行高斯模糊，以便实现一定程度的羽化效果。

**将一个人脸的关键点对齐到另一个人脸的关键点。**

1. def transformation\_from\_points(points1, points2):
2. points1 = points1.astype(numpy.float64)
3. points2 = points2.astype(numpy.float64)
5. c1 = numpy.mean(points1, axis=0)
6. c2 = numpy.mean(points2, axis=0)
7. points1 -= c1
8. points2 -= c2
10. s1 = numpy.std(points1)
11. s2 = numpy.std(points2)
12. points1 /= s1
13. points2 /= s2
15. U, S, Vt = numpy.linalg.svd(points1.T \* points2)
16. R = (U \* Vt).T
18. return numpy.vstack([numpy.hstack(((s2 / s1) \* R,
19. c2.T - (s2 / s1) \* R \* c1.T)),
20. numpy.matrix([0., 0., 1.])])

points1 = points1.astype(numpy.float64) 和 points2 points2.astype(numpy.float64)：这两个语句将输入的两个关键点矩阵转换为浮点数类型。

c1 = numpy.mean(points1, axis=0) 和 c2 = numpy.mean(points2, axis=0)：这两个语句分别计算两个关键点矩阵的均值，并将结果保存在变量 c1 和 c2 中。

points1 -= c1 和 points2 -= c2：这两个语句将每个关键点的坐标减去对应矩阵的均值，以便进行后续的标准化操作。

s1 = numpy.std(points1) 和 s2 = numpy.std(points2)：这两个语句分别计算两个关键点矩阵的标准差，并将结果保存在变量 s1 和 s2 中。

points1 /= s1 和 points2 /= s2：这两个语句将每个关键点的坐标除以对应矩阵的标准差，以标准化两个关键点矩阵。

U, S, Vt = numpy.linalg.svd(points1.T \* points2)：这个语句使用奇异值分解（SVD）来计算两个关键点矩阵之间的旋转矩阵。

R = (U \* Vt).T：这个语句将旋转矩阵从奇异值分解的结果中提取出来，并将其转置。

return numpy.vstack([numpy.hstack(((s2 / s1) \* R, c2.T - (s2 / s1) \* R \* c1.T)), numpy.matrix([0., 0., 1.])])：这个语句将旋转矩阵和平移向量合并为一个 3x3 的变换矩阵，并返回该矩阵。其中，平移向量是通过将一个人脸的关键点矩阵的均值变换到另一个人脸的关键点矩阵的均值得到的。

**生成一个经过变换后的图像**

1. def warp\_im(im, M, dshape):
2. output\_im = numpy.zeros(dshape, dtype=im.dtype)
3. cv2.warpAffine(im,
4. M[:2],
5. (dshape[1], dshape[0]),
6. dst=output\_im,
7. borderMode=cv2.BORDER\_TRANSPARENT,
8. flags=cv2.WARP\_INVERSE\_MAP)
9. return output\_im

cv2.warpAffine(im, M[:2], (dshape[1], dshape[0]), dst=output\_im, borderMode=cv2.BORDER\_TRANSPARENT, flags=cv2.WARP\_INVERSE\_MAP)：这个语句使用 OpenCV 库的 cv2.warpAffine() 函数来对输入图像 im 进行仿射变换。变换矩阵 M 的前两行用于表示旋转、缩放和平移，因此这个语句只取变换矩阵的前两行。输出图像的大小由参数 dshape 指定。变换后的图像被保存在 output\_im 中。borderMode=cv2.BORDER\_TRANSPARENT 表示输出图像的边界使用透明像素填充。flags=cv2.WARP\_INVERSE\_MAP 表示变换使用的是反向映射。

**矫正颜色**

1. def correct\_colours(im1, im2, landmarks1):
2. blur\_amount = COLOUR\_CORRECT\_BLUR\_FRAC \* numpy.linalg.norm(
3. numpy.mean(landmarks1[LEFT\_EYE\_POINTS], axis=0) -
4. numpy.mean(landmarks1[RIGHT\_EYE\_POINTS], axis=0))
5. blur\_amount = int(blur\_amount)
6. if blur\_amount % 2 == 0:
7. blur\_amount += 1
8. im1\_blur = cv2.GaussianBlur(im1, (blur\_amount, blur\_amount), 0)
9. im2\_blur = cv2.GaussianBlur(im2, (blur\_amount, blur\_amount), 0)
10. im2\_blur = im2\_blur + 128 \* (im2\_blur <= 1.0)
12. return (im2.astype(numpy.float64) \* im1\_blur.astype(numpy.float64) /
13. im2\_blur.astype(numpy.float64))

第一个语句计算了需要对第二张图像进行高斯模糊的程度，以便自动校正颜色。具体来说，它计算了左眼和右眼之间的距离，并将其乘以一个常数 COLOUR\_CORRECT\_BLUR\_FRAC，得到需要进行模糊的程度。im2\_blur = im2\_blur + 128 \* (im2\_blur <= 1.0)：这个语句将第二张图像进行缩放，以便调整颜色平衡。具体来说，它将第二张图像中的每个像素值乘以 128，如果最终值小于等于 1.0，则将该像素值加上 128。

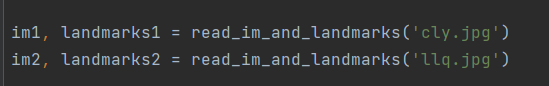
**主函数**

1. im1, landmarks1 = read\_im\_and\_landmarks('cly.jpg')
2. im2, landmarks2 = read\_im\_and\_landmarks('llq.jpg')
4. M = transformation\_from\_points(landmarks1[ALIGN\_POINTS],
5. landmarks2[ALIGN\_POINTS])
7. mask = get\_face\_mask(im2, landmarks2)
8. warped\_mask = warp\_im(mask, M, im1.shape)
9. combined\_mask = numpy.max([get\_face\_mask(im1, landmarks1), warped\_mask],
10. axis=0)
12. warped\_im2 = warp\_im(im2, M, im1.shape)
13. warped\_corrected\_im2 = correct\_colours(im1, warped\_im2, landmarks1)
15. output\_im = im1 \* (1.0 - combined\_mask) + warped\_corrected\_im2 \* combined\_mask
17. cv2.imwrite('cly\_llqversion.jpg', output\_im)

输入需要伪造的人脸图，将伪造后的图片输出。

1. **实验结果**

**换脸前：**



llq.jpg cly.jpg

**换脸后**

将cly的面部特征融合到llq.jpg原图像



将llq面部特征融合到cly.jpg原图像



结果分析：llq更适合没有刘海的发型，可以使用换脸技术来实现发型尝试。

**任务四 使用Python3+Face-X-Ray实现人脸伪造图像检测**

1. **实验原理**

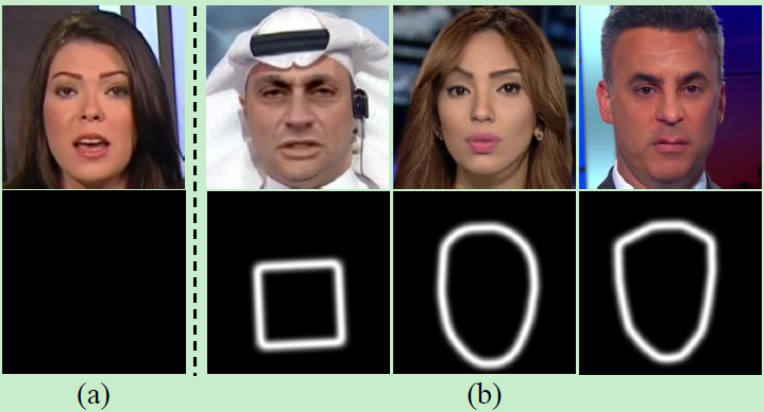
**1．基本原理**

常见换脸鉴别方法主要从人脸伪造第二步入手，通过基于数据集的有监督训练学习大量换脸图像，检测换脸过程中产生的瑕疵，判断真伪。但是，不同的换脸算法合成时的瑕疵各不相同，因此针对一种换脸算法进行训练后，应用于另外一种算法上时准确率明显下降，这就是已有换脸鉴别算法不具通用性的原因。

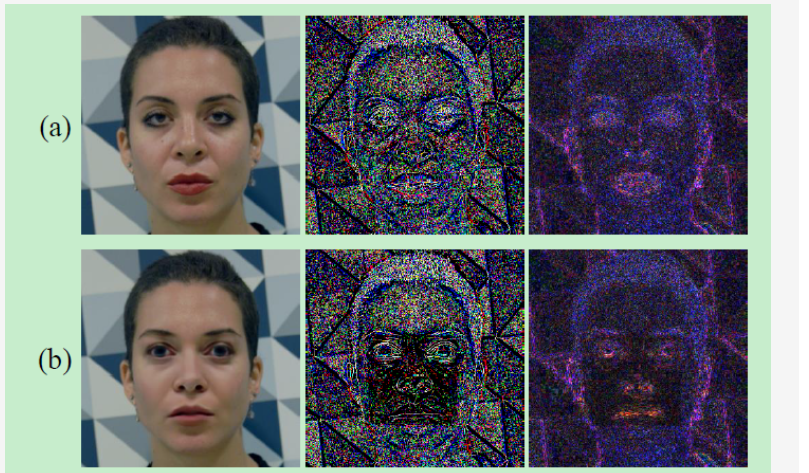
Face X-Ray不需要事先知道操作方法或人工监督，而是从第三步入手，通过生成灰度图像，显示该图像是否可以分解为来自不同来源的两个图像的混合，从而检测出换脸的边界，就像照 X 光一样，让这个边界清晰可见

**2.Face-X-Ray**

提出一种新的图像表示称为face X-ray，用于检测伪造的人脸。一个输入人脸的face X-ray是灰度图，可表明输入图是否可分解为两张不同源图像的合成。它通过显示伪造图像的混合边界和真实图像的absence来实现。绝大多数人脸变换算法有一个共同步骤：将更改后的脸部混合到现有背景图像中。为此，face X-ray提供有效方法检测伪造人脸。Face X-ray 仅仅假设blending步骤存在。事实上训练face X-ray算法不需要牛逼人脸造假算法（DeepFakes、Face2Face、FaceSwap、NeuralTextures）生成的假脸 。当将face X-ray用于未知算法生成的假脸时仍有效，其他算法则不太顶。如下图（a）真脸返回的face X-ray是黑图，因为没有合成边界。而（b）中的合成人脸是有边界的。



假脸检测的难点之一是在实际应用中无法得知生成假脸的算法。而多数假脸检测算法是在已知算法条件下监督训练的。这就导致这些算法能达到98%的准确率。当然缺点也很明显：过拟合和受限于特定算法。本文观察到大多数人脸合成算法都有同一步骤：将转换后的人脸混合到已有的背景图像中。这就在混合边界上存在固有的图像差异。这些差异可以探测到边界。实际上，采集过程每个图像都有自己的区别标记，这些标记是从硬件（例如，传感器、镜头）或软件组件（例如，压缩、合成算法）引入的，并且这些标记倾向于呈现在整个图像中。本文描述了噪声分析和错误级别分析作为两种代表性的区别标记：



 face X-ray算法通过自监督算法训练大量由真实图像合成的混合图像，无需利用其他算法生成的假脸。

**3.方法：**

方法包括三个阶段：检测面部区域、合成期望的目标脸、 将目标人脸融合到原始图像中。之前的假脸检测集中于第二阶段：即通过训练一个真假脸分类网络来解决识别假脸。但是这种方法在识别其他方法生成的假脸时就不太好使。本文主要针对这个泛化性能来讲。

每幅图像都有自己独特的标记或基本统计数据，主要来源于硬件和软件。因此，可以利用边界上底层图像统计的不一致性来发现混合边界来检测伪造的人脸图像。

**Face XRay Definition**

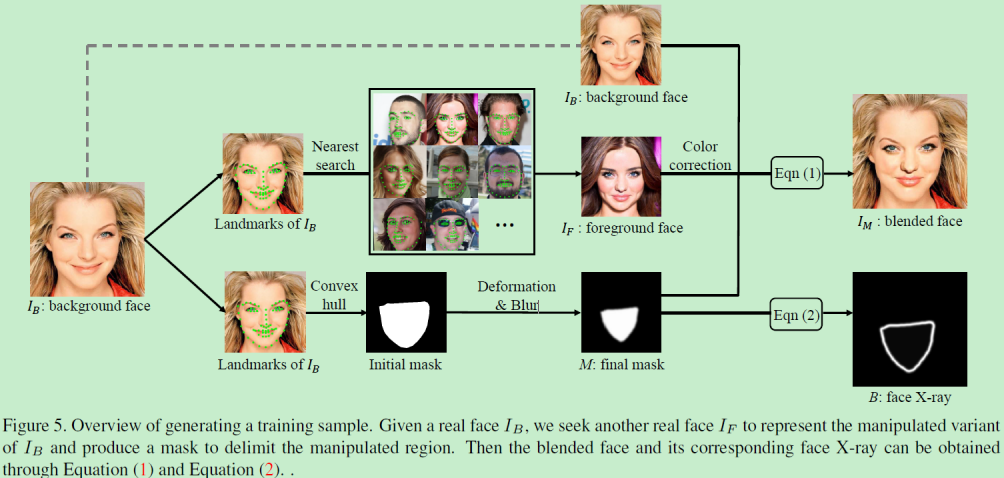
给定输入图，想去判定这张图是否是两张图的乘积：

IMG_256

前者是含期望面部属性的前景，后者是背景。M代表mask，划定操纵区域。M灰度值在0-1之间。为此定义X-ray为图像B，若输入图为合成图则B将会透露出混合边界信息，若为真图则B的像素为全零。定义输入图像的face X-ray为B：

IMG_257

整个流程图：



通过混合两个真实的图像来产生非平凡的面部X射线。

1. **分段仿射变换和模糊处理**

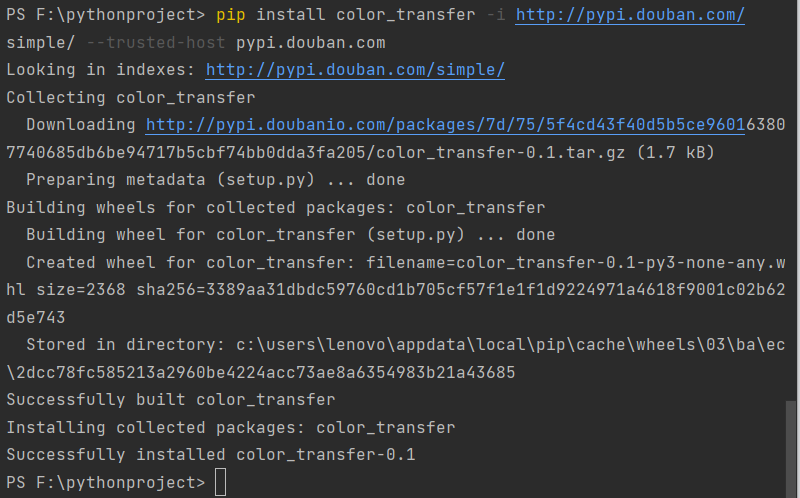
在图像处理和计算机视觉中，分段仿射变换是一种常见的图像变换技术，它可以将图像分成多个小块，对每个小块进行仿射变换，从而实现对整幅图像的变换。分段仿射变换通常用于图像配准、图像纠正、图像变形等应用中。而模糊处理是一种常见的图像滤波技术，用于减少图像噪声、平滑图像、模糊图像等。模糊处理通常通过对图像进行卷积操作来实现，其中卷积核的选择和卷积操作的参数会对处理效果产生重要影响。常用的模糊处理方法包括高斯模糊、均值模糊、中值模糊等。

1. **环境搭建**

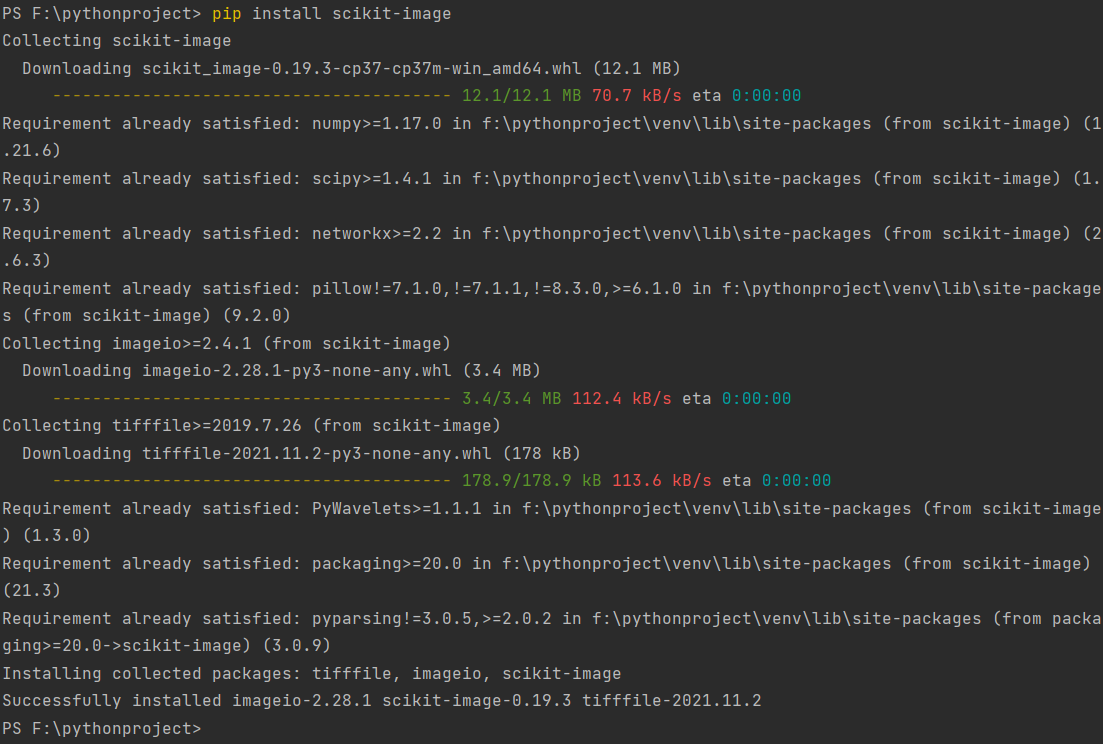
**1.安装依赖包：**

pip install numpy, opencv-python, scikit-image, dlib, tqdm

**color\_transfer**



**scikit-image**

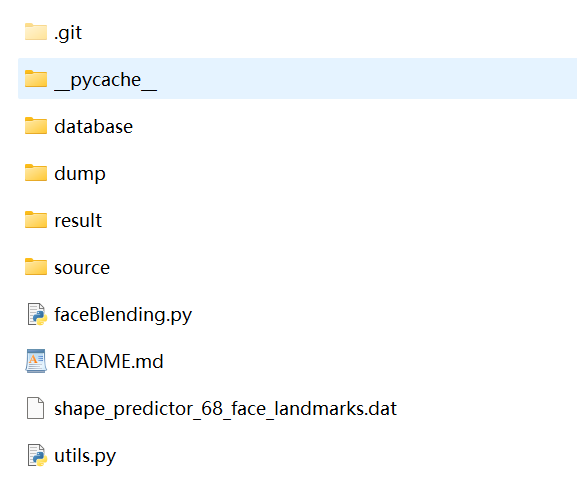


**下载“shape\_predictor\_68\_face\_landmarks.dat”文件**



**下载源代码：**

git clone https://github.com/neverUseThisName/Face-X-Ray



database：正常人脸图片全集文件夹

source：待检测的人脸伪造图片文件夹

dump：检测结果

faceBlending.py：main文件

1. **实验步骤**

**导入库**

1. import argparse, sys, os
2. from os.path import basename, splitext
3. from skimage.transform import PiecewiseAffineTransform, warp
4. import numpy as np
5. import cv2
6. import dlib
7. from tqdm import tqdm
8. from color\_transfer import color\_transfer
9. from utils import files, FACIAL\_LANDMARKS\_IDXS, shape\_to\_np

**检测人脸函数**

1. def get\_landmarks(detector, predictor, rgb):
2. # first get bounding box (dlib.rectangle class) of face.
3. boxes = detector(rgb, 1)
4. for box in boxes:
5. landmarks = shape\_to\_np(predictor(rgb, box=box))
6. break
7. else:
8. return None
9. return landmarks.astype(np.int32)

for box in boxes: 和 landmarks = shape\_to\_np(predictor(rgb, box=box))：这两个语句分别遍历 boxes 中的所有人脸，并使用 predictor 对象检测每张人脸的关键点。shape\_to\_np 函数将检测到的关键点转换为 NumPy 数组，并将结果保存在变量 landmarks 中。

**找相似图像**

1. def find\_one\_neighbor(detector, predictor, srcPath, srcLms, faceDatabase, threshold):
2. for face in faceDatabase:
3. rgb = dlib.load\_rgb\_image(face)
4. landmarks = get\_landmarks(detector, predictor, rgb)
5. if landmarks is None:
6. continue
7. dist = distance(srcLms, landmarks)
8. if dist < threshold and basename(face).split('\_')[0] != basename(srcPath).split('\_')[0]:
9. print("yes")
10. return rgb
11. print("no")
12. return None

遍历人脸数据库中的所有图像，load\_rgb\_image() 函数将当前图像加载到内存中，并将它保存在变量 rgb 中，

dist = distance(srcLms, landmarks)：这个语句计算源图像的关键点数组 srcLms 与当前图像的关键点数组 landmarks 之间的欧几里得距离，并将结果保存在变量 dist 中。

如果源图像与当前图像之间的距离小于阈值 threshold，且当前图像与源图像属于同一个人，则执行yes

**面部填充**

1. def convex\_hull(size, points, fillColor=(255,)\*3):
2. mask = np.zeros(size, dtype=np.uint8) *# mask has the same depth as input image*
3. points = cv2.convexHull(np.array(points))
4. corners = np.expand\_dims(points, axis=0).astype(np.int32)
5. cv2.fillPoly(mask, corners, fillColor)
6. return mask

points = cv2.convexHull(np.array(points))：这个语句使用 OpenCV 库的 convexHull() 函数计算输入点集 points 的凸包，并将结果保存在变量 points 中。

corners = np.expand\_dims(points, axis=0).astype(np.int32)：这个语句将凸包点集 points 转换为 NumPy 数组，并将其添加一个维度，以便与 fillPoly() 函数的输入格式相同。同时，它将数组元素的数据类型设置为有符号 32 位整数。

cv2.fillPoly(mask, corners, fillColor)：这个语句使用 OpenCV 库的 fillPoly() 函数在掩膜 mask 上填充凸包内部，颜色为 fillColor。

**图像的行列排布**

1. def random\_deform(imageSize, nrows, ncols, mean=0, std=5):
2. h, w = imageSize
3. rows = np.linspace(0, h-1, nrows).astype(np.int32)
4. cols = np.linspace(0, w-1, ncols).astype(np.int32)
5. rows, cols = np.meshgrid(rows, cols)
6. anchors = np.vstack([rows.flat, cols.flat]).T
7. assert anchors.shape[1] == 2 and anchors.shape[0] == ncols \* nrows
8. deformed = anchors + np.random.normal(mean, std, size=anchors.shape)
9. np.clip(deformed[:,0], 0, h-1, deformed[:,0])
10. np.clip(deformed[:,1], 0, w-1, deformed[:,1])
11. return anchors, deformed.astype(np.int32)

h, w = imageSize：这个语句将图像大小的元组 imageSize 拆分为两个变量 h 和 w，分别表示图像的高度和宽度。

rows = np.linspace(0, h-1, nrows).astype(np.int32) 和 cols = np.linspace(0, w-1, ncols).astype(np.int32)：这两个语句分别在图像的行和列方向上创建均匀分布的采样点，并将它们保存在变量 rows 和 cols 中。

rows, cols = np.meshgrid(rows, cols)：这个语句使用 NumPy 库的 meshgrid() 函数将行和列方向上的采样点组合成网格，并将结果保存在变量 rows 和 cols 中。

anchors = np.vstack([rows.flat, cols.flat]).T：这个语句将网格中的每个点展平成一维数组，并将它们沿着列方向堆叠起来，得到一个二维数组 anchors。注意，这里的 flat 属性将二维数组展平成一维数组。

assert anchors.shape[1] == 2 and anchors.shape[0] == ncols \* nrows：这个语句检查 anchors 的形状是否满足要求。

deformed = anchors + np.random.normal(mean, std, size=anchors.shape)：这个语句将 anchors 中的每个点进行随机位移，位移量服从均值为 mean，标准差为 std 的正态分布。具体来说，它使用 NumPy 库的 random.normal() 函数生成一个与 anchors 形状相同的随机数组，并将其与 anchors 相加得到 deformed。

np.clip(deformed[:,0], 0, h-1, deformed[:,0]) 和 np.clip(deformed[:,1], 0, w-1, deformed[:,1])：这两个语句将 deformed 中的行和列坐标限制在图像范围内，避免产生越界的点。

**仿射变化**

1. def piecewise\_affine\_transform(image, srcAnchor, tgtAnchor):
2. trans = PiecewiseAffineTransform()
3. trans.estimate(srcAnchor, tgtAnchor)
4. warped = warp(image, trans)
5. return warped

trans = PiecewiseAffineTransform()：这个语句创建一个 PiecewiseAffineTransform() 对象，并将它保存在变量 trans 中。PiecewiseAffineTransform 是 Scikit-image 库中的一个类，用于执行分段仿射变换。

trans.estimate(srcAnchor, tgtAnchor)：这个语句使用 srcAnchor 和 tgtAnchor 中的坐标点来估计仿射变换矩阵，并将结果保存在 trans 对象中。具体来说，它使用 estimate() 方法计算输入和输出锚点之间的仿射变换关系。

warped = warp(image, trans)：这个语句使用 Scikit-image 库的 warp() 函数将输入图像 image 进行分段仿射变换，并将结果保存在变量 warped 中。变换矩阵 trans 描述了输入图像中每个锚点与输出图像中对应锚点之间的仿射变换关系。在变换过程中，非锚点像素的新位置是通过插值计算得到的。

**解析命令行参数**

1. def get\_parser():
2. parser = argparse.ArgumentParser(description='Demo for face x-ray fake sample generation')
3. parser.add\_argument('--srcFacePath', '-sfp', type=str)
4. parser.add\_argument('--faceDatabase', '-fd', type=str)
5. parser.add\_argument('--threshold', '-t', type=float, default=25, help='threshold for facial landmarks distance')
6. parser.add\_argument('--shapePredictor', '-sp', type=str, default='shape\_predictor\_68\_face\_landmarks.dat', help='Path to dlib facial landmark predictor model')
7. return parser.parse\_args()

参数 '--srcFacePath'：正常图片集

参数 '--faceDatabase'：待检测的伪造图片集

参数 '--threshold'：设为100

**主函数**

加载解释器

1. args = get\_parser()

加载人脸检测器

1. detector = dlib.get\_frontal\_face\_detector()
2. predictor = dlib.shape\_predictor(args.shapePredictor)

读取正常图片集

1. srcFaces = tqdm(files(args.srcFacePath, ['.jpg']))

在图片集中对每一张图片：

加载图像的bgr

1. try:
2. srcFaceBgr = cv2.imread(srcFace)
3. except:
4. tqdm.write(f'Fail loading: {srcFace}')
5. continue

检测关键点

1. srcLms = get\_landmarks(detector, predictor, cv2.cvtColor(srcFaceBgr, cv2.COLOR\_BGR2RGB))
2. if srcLms is None:
3. tqdm.write(f'No face: {srcFace}')
4. continue

找到第一个与正常图片相似的照片

1. targetRgb = find\_one\_neighbor(detector, predictor, srcFace, srcLms, files(args.faceDatabase, ['.jpg']), args.threshold)
2. if targetRgb is None: *# if not found*
3. tqdm.write(f'No Match: {srcFace}')
4. continue

生成随机形变

1. anchors, deformedAnchors = random\_deform(hullMask.shape[:2], 4, 4)

分段仿射变换和模糊处理

1. warped = piecewise\_affine\_transform(hullMask, anchors, deformedAnchors) *# size (h, w) warped mask*
2. blured = cv2.GaussianBlur(warped, (5,5), 3)

交换

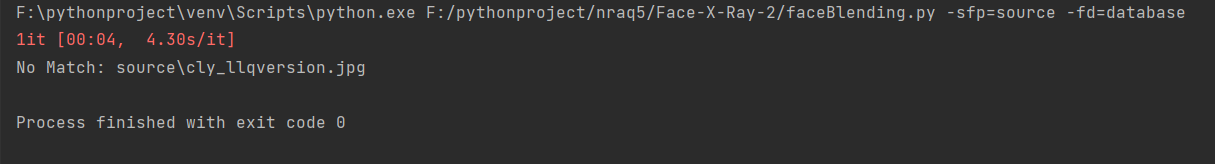
1. left, up, right, bot = min(srcLms[:,0]), min(srcLms[:,1]), max(srcLms[:,0]), max(srcLms[:,1])
2. targetBgrT = color\_transfer(srcFaceBgr[up:bot,left:right,:], targetBgr)
3. resultantFace = forge(srcFaceBgr, targetBgrT, blured) # forged face

保存图片到dump文件夹下面

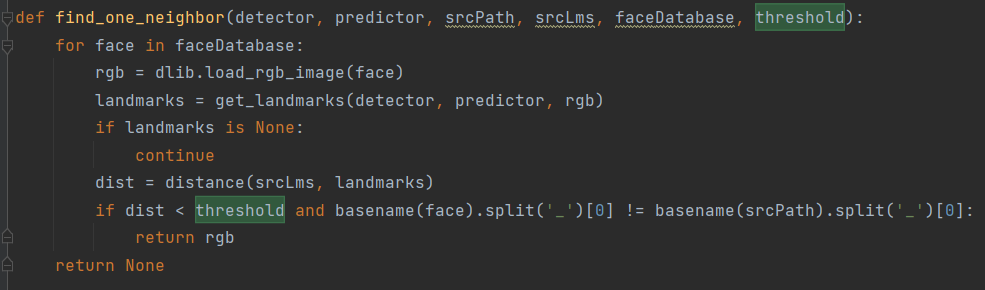
1. cv2.imwrite(f'./dump/mask\_{i}.jpg', hullMask)
2. cv2.imwrite(f'./dump/deformed\_{i}.jpg', warped\*255)
3. cv2.imwrite(f'./dump/blured\_{i}.jpg', blured\*255)
4. cv2.imwrite(f'./dump/src\_{i}.jpg', srcFaceBgr)
5. cv2.imwrite(f'./dump/target\_{i}.jpg', targetBgr)
6. cv2.imwrite(f'./dump/target\_T\_{i}.jpg', targetBgrT)
7. cv2.imwrite(f'./dump/forge\_{i}.jpg', resultantFace)
8. **实验结果**

**1.报错**

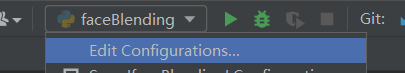
出现不匹配情况

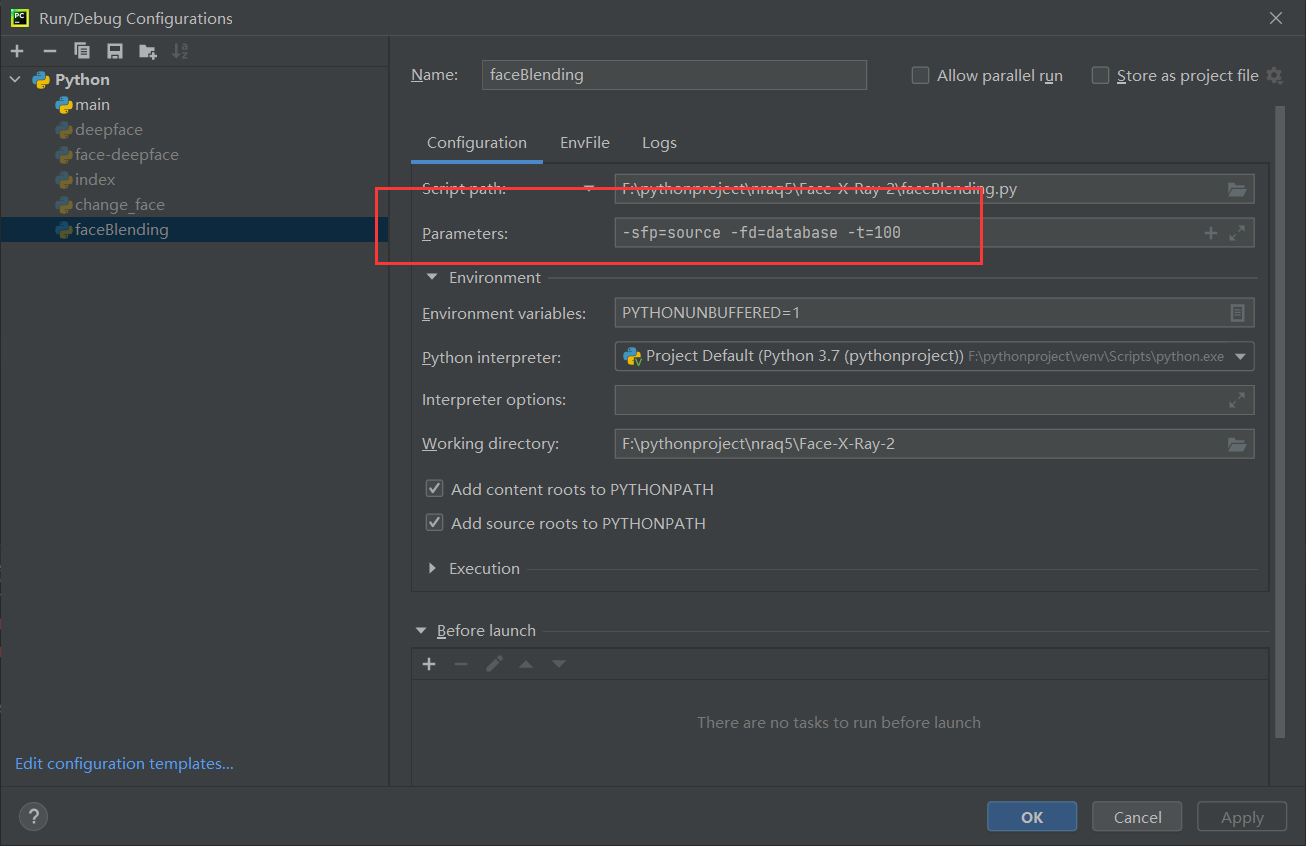


返回 No Match 是因为find\_one\_neibor()返回了none

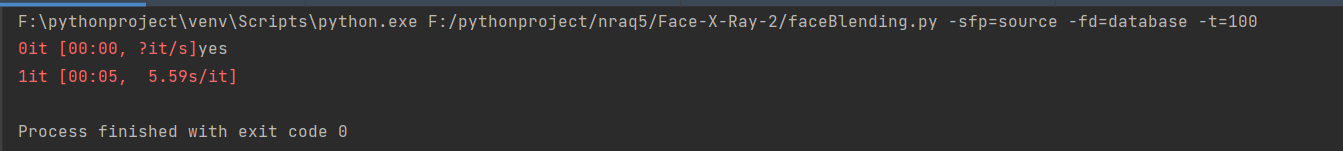


Threshold设置的太小，导致dist总大于threshold,跳转到none，所以将threshold设为100





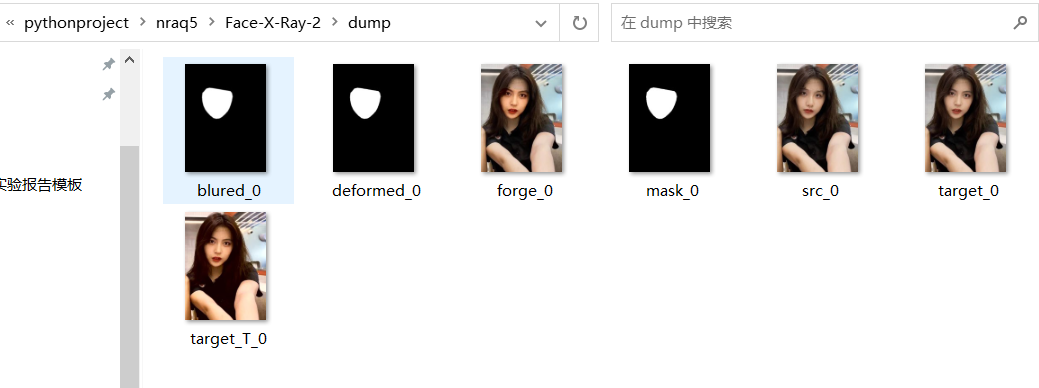
**2.成功运行**



**待检测图片**



**运行结果**



**任务二的实验拓展------用dlib实现人脸情绪检测**

1. **实验步骤**

**导入包**

1. import threading
2. import cv2
3. import dlib
4. import numpy as np

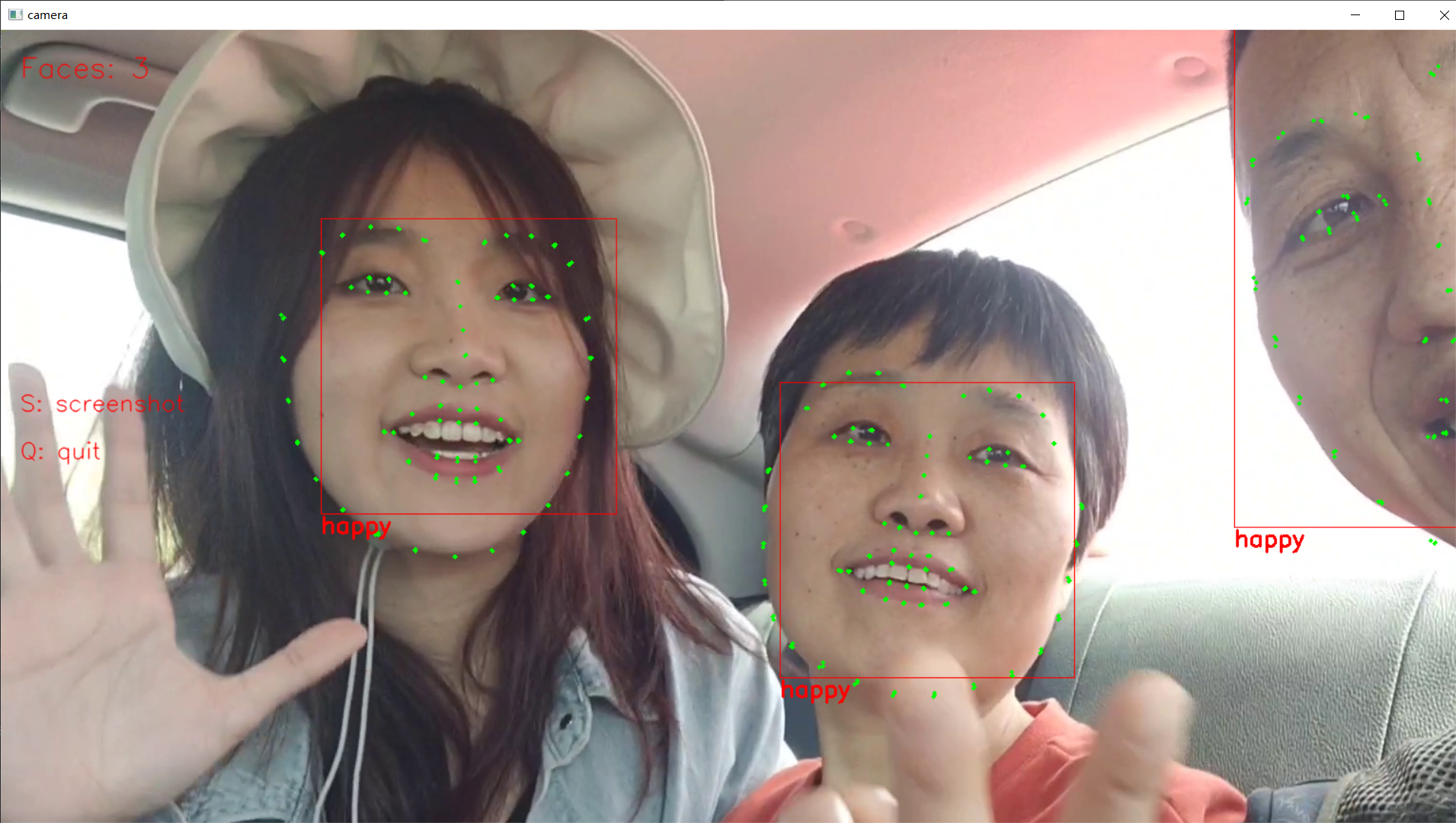
代码其余部分与任务二相同，此处不再赘述，下面是关键代码

**分析任意n点的位置关系作为表情识别的依据**

1. 嘴巴张开程度
2. mouth\_width = (shape.part(54).x - shape.part(48).x) / self.face\_width  *# 嘴巴咧开程度*
3. mouth\_higth = (shape.part(66).y - shape.part(62).y) / self.face\_width  *# 嘴巴张开程度*
5. 眉毛的弧度
6. *# 通过两个眉毛上的10个特征点，分析挑眉程度和皱眉程度*
7. brow\_sum = 0  *# 高度之和*
8. frown\_sum = 0  *# 两边眉毛距离之和*
9. for j in range(17, 21):
10. brow\_sum += (shape.part(j).y - d.top()) + (shape.part(j + 5).y - d.top())
11. frown\_sum += shape.part(j + 5).x - shape.part(j).x
12. line\_brow\_x.append(shape.part(j).x)
13. line\_brow\_y.append(shape.part(j).y)
14. *# self.brow\_k, self.brow\_d = self.fit\_slr(line\_brow\_x, line\_brow\_y)  # 计算眉毛的倾斜程度*
15. tempx = np.array(line\_brow\_x)
16. tempy = np.array(line\_brow\_y)
17. z1 = np.polyfit(tempx, tempy, 1)  *# 拟合成一次直线*
18. self.brow\_k = -round(z1[0], 3)  *# 拟合出曲线的斜率和实际眉毛的倾斜方向是相反的*
19. brow\_hight = (brow\_sum / 10) / self.face\_width  *# 眉毛高度占比*
20. brow\_width = (frown\_sum / 5) / self.face\_width  *# 眉毛距离占比*
22. 眼睛睁开程度
23. eye\_sum = (shape.part(41).y - shape.part(37).y + shape.part(40).y - shape.part(38).y +
24. shape.part(47).y - shape.part(43).y + shape.part(46).y - shape.part(44).y)
25. eye\_hight = (eye\_sum / 4) / self.face\_width

4.分情况讨论

1. if round(mouth\_higth >= 0.03):
2. if eye\_hight >= 0.056:
3. cv2.putText(im\_rd, "amazing", (d.left(), d.bottom() + 20), cv2.FONT\_HERSHEY\_SIMPLEX,
4. 0.8,
5. (0, 0, 255), 2, 4)
6. else:
7. cv2.putText(im\_rd, "happy", (d.left(), d.bottom() + 20), cv2.FONT\_HERSHEY\_SIMPLEX, 0.8,
8. (0, 0, 255), 2, 4)
9. *# 没有张嘴，可能是正常和生气*
10. else:
11. if self.brow\_k <= -0.3:
12. cv2.putText(im\_rd, "angry", (d.left(), d.bottom() + 20), cv2.FONT\_HERSHEY\_SIMPLEX, 0.8,
13. (0, 0, 255), 2, 4)
14. else:
15. cv2.putText(im\_rd, "nature", (d.left(), d.bottom() + 20), cv2.FONT\_HERSHEY\_SIMPLEX, 0.8,
16. (0, 0, 255), 2, 4)
17. **实验结果**



检测到3人均为happy情绪，与事实相符合

优点：比deepface更准确