武汉大学国家网络安全学院实验报告

课程名称	内容安全实验	实验日期	2024-4-8
实验名称	实验 3：图像伪造与检测		
姓名	学号	专业	班级
邓鹏	2021302181152	网络空间安全	6 班

目录

一、实验目的及实验内容

1.1 实验目的

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

1.2 实验内容

1.2.1 OpenCV 相关原理

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。

本次实验中，OpenCV 主要用于图像获取、图像处理、人脸检测与特征点标记以及界面显示，以实现实时人脸识别。

1.2.2 Dlib 相关原理

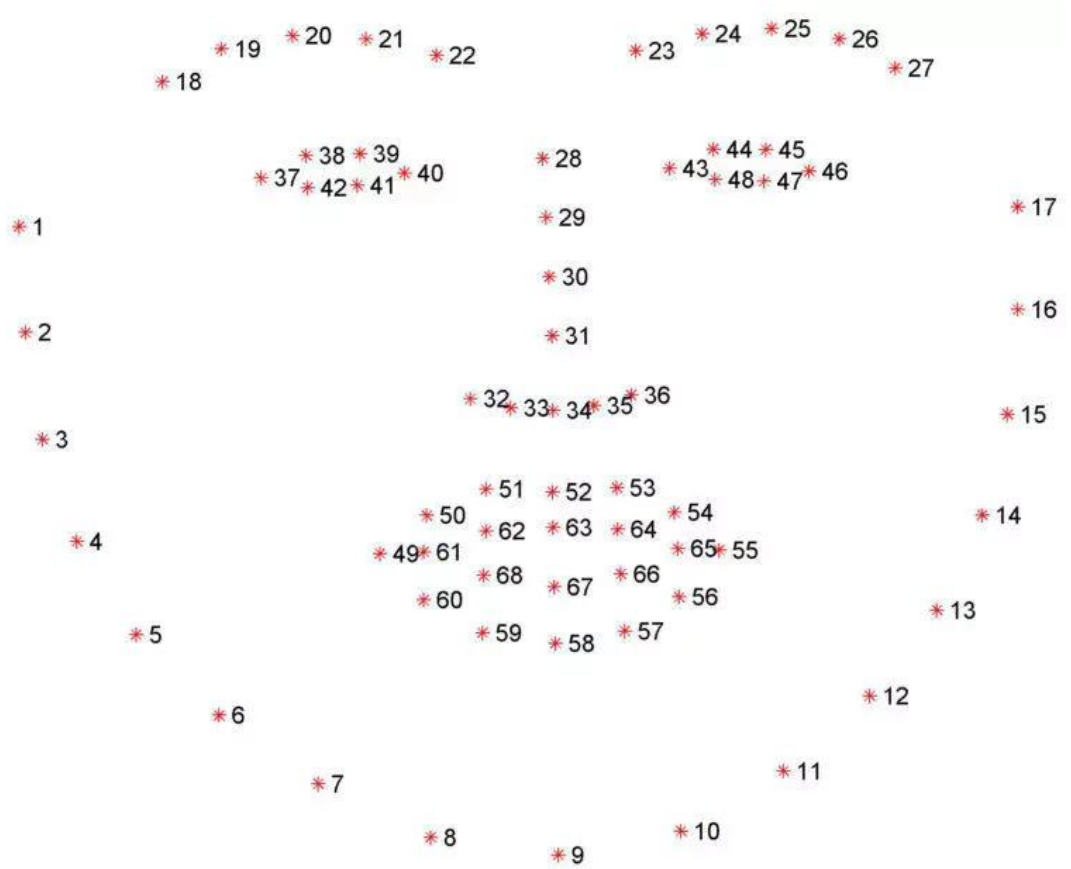
Dlib 是一个 C++ 工具库，包含机器学习算法，图像处理，网络及一些工具类库。Dlib 里面有两个人脸检测模块，基于 HOG+SVM 分类，以及基于 Maximum-Margin Object Detector（MMOD）的深度学习人脸检测方案。

- 基于 HOG+SVM
  - 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 MMOD
  - 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

另外，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。

为了实现实时检测，本次实验是基于 HOM+SVM 的。

Dlib 里面提供了两个关键点预测模型，分别是 5 face landmarks、68 face landmarks，其中 5 点位是左右眼睛分别两个，鼻子下一个。68 点位如下图。



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。

本次实验中使用 get_frontal_face_detector 方法。这个方法基于一种名为 HOG 加上线性分类器、图像金字塔和滑动窗口检测方案的组合技术。

• HOG 特征描述符：此方法首先将图像转换成灰度图，然后使用梯度的方向和强度计算出局部的 HOG 特征。这些特征能够捕获图像中的形状信息，并且对光照变化具有良好的鲁棒性。

• 滑动窗口：检测器会在图像中滑动一个小窗口，计算每个窗口的 HOG 特征，并使用预训练的线性分类器评估这些特征，判断窗口中是否包含人脸。

检测到人脸后，使用 shape_predictor 进行关键点定位，这是通过训练模型 shape_predictor_68_face_landmarks.dat 来实现的。这个模型使用了一种名为“Ensemble of Regression Trees”（集成回归树）的方法，对人脸的 68 个关键点进行精准定位。

• 回归树的集成：这种方法首先根据一组初始化的简单特征定位关键点的大致位置，然后逐步细化预测，每一步都利用前一步的结果。这种逐步学习的方法可以有效地提高预测的准确性。

• 特征点定位：模型通过评估面部区域内特定点的相对位置，预测关键点的精确坐标，如眼角、鼻尖、嘴角等。

1.2.3 Deepface 相关原理

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。 它包含的模型：VGG-Face、Google FaceNet、OpenFace、Facebook DeepFace、DeepID、ArcFace 和 Dlib。

DeepFace 框架主要涉及两大技术方向：深度神经网络和大规模数据集训练。

DeepFace 使用了一个深度卷积神经网络，该网络模仿了人类大脑处理视觉信息的方式。网络包含多个层级，包括卷积层、激活层、池化层和全连接层，这些层共同工作以从输入的面部图像中提取复杂的高维特征。

- 卷积层：用于提取图像中的局部特征，每个卷积层由多个卷积核组成，这些卷积核在图像上滑动以生成特征图。
- 激活层：用于引入非线性，帮助网络学习更复杂的模式。
- 池化层：用于降低特征图的空间尺寸，增强特征的鲁棒性，减少计算量。
- 全连接层：将前面卷积层提取的特征图整合，用于分类或其他任务。

DeepFace 系统通过在大规模数据集上进行训练来优化其网络参数。这些数据集通常包含数百万张带有标注的人脸图像。

- 前向传播：在训练过程中，图像数据通过网络前向传递，每一层的输出都是下一层的输入。
- 反向传播和优化：使用反向传播算法来计算损失函数（如交叉熵损失）对网络参数的梯度，并利用优化算法（如 SGD、Adam）调整参数以最小化损失。

为了提高识别的准确性，DeepFace 在处理任何人脸图像之前，会先进行面部校准和对齐。这通常涉及以下步骤：

- 检测面部关键点（如眼睛、鼻子和嘴巴的位置）。
- 基于这些关键点进行仿射变换，将所有面部图像转换到一个标准的姿态和尺度，确保输入网络的面部图像在几何上是一致的。

在提取了面部的深度特征之后，DeepFace 可以执行多种任务，如验证、识别和属性分析：

- 验证：比较两张面部图像的特征，判断它们是否属于同一人。
- 识别：将提取的特征与数据库中已知的特征进行匹配，以识别个人的身份。
- 属性分析：使用专门训练的分类网络来预测年龄、性别和情绪。

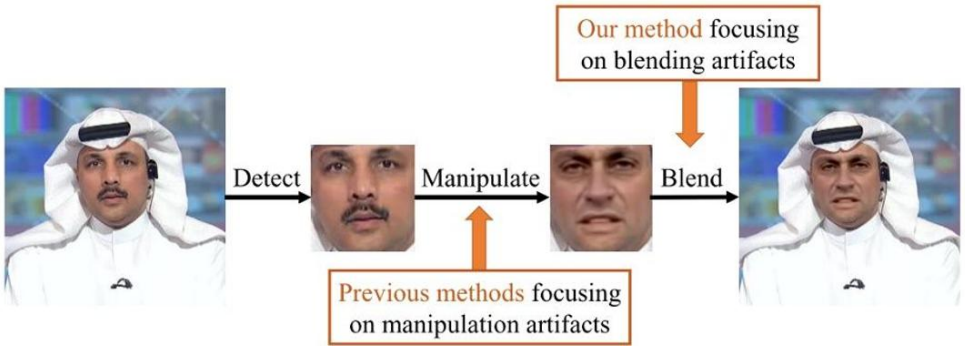
1.2.4 人脸伪造相关原理

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。

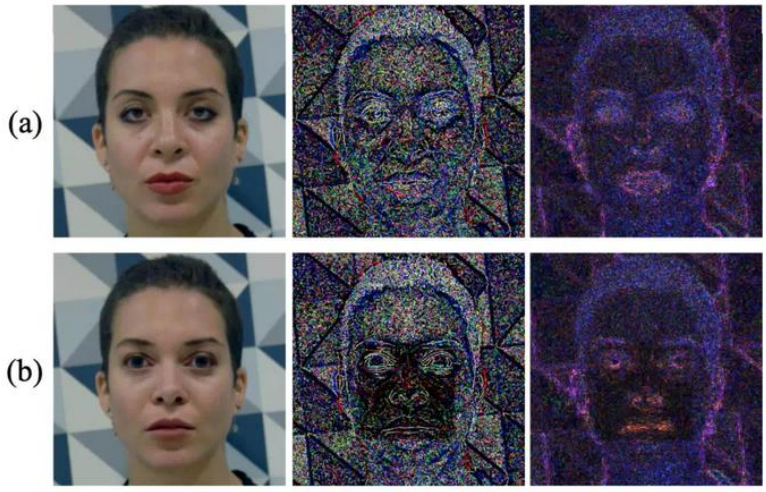


1.2.5 Face X-Ray 人脸伪造检测相关原理

常见换脸鉴别方法主要从利用 AI 换脸算法生成新的面部及一部分周围区域入手，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。

真实图像展现出一致地噪声模式，而换脸明显会有所不同。



当然，Face X-Ray 并不是完美的。如果图像是整体合成，那么 Face X-Ray 是难以检测出来的；如果针对 Face X-Ray 训练一个新换脸模型，该算法也有可能被攻击到。

二、实验环境

Python3.8+opencv+dlib+Deepface+scikit_image+color_transfer

三、实验步骤及结果分析

3.1 人脸识别与关键点实时检测

首先，创建一个人脸检测类，初始化时加载 dlib 的人脸检测器和面部特征点模型，并设置 OpenCV 的视频捕捉对象。

```
1个用法
class FaceDetector():

    def __init__(self):
        # 使用特征提取器 get_frontal_face_detector
        self.detector = dlib.get_frontal_face_detector()
        # dlib 的68点模型，使用官方训练好的特征预测器
        self.predictor = dlib.shape_predictor("shape_predictor_68_face_landmarks.dat")

        # 建cv2摄像头对象
        self.cap = cv2.VideoCapture(0)
        # 设置视频参数
        self.cap.set( propId: 3,  value: 480)
```

循环读取摄像头视频流，转换为灰度图像以提高处理速度，使用 dlib 检测图像中的人脸，并用 OpenCV 标记出来。如果有人脸，就使用 dlib 的预测器获取并显示 68 个面部特征点。最后，通过按下 q 键来退出循环，释放摄像头资源并关闭所有窗口。

```
def detect_and_display(self):
    while self.cap.isOpened():
        flag, im_rd = self.cap.read()
        k = cv2.waitKey(1)
        # 取灰度
        img_gray = cv2.cvtColor(im_rd, cv2.COLOR_RGB2GRAY)

        # 使用人脸检测器检测每一帧图像中的人脸
        faces = self.detector(img_gray, 0)

        # 如果检测到人脸
        if (len(faces) != 0):
            # 对每个人脸都标出68个特征点
            for i in range(len(faces)):
                for k, d in enumerate(faces):
                    cv2.rectangle(im_rd, (d.left(), d.top()), (d.right(), d.bottom()), (0, 0, 255))

                    # 使用预测器得到68点数据的坐标
                    shape = self.predictor(im_rd, d)
                    # 圆圈显示每个特征点
                    for i in range(68):
                        cv2.circle(im_rd,  center: (shape.part(i).x, shape.part(i).y),  radius: 1,  color: (0, 255, 0), -1,  lineType: 8)
        else:
            cv2.putText(im_rd,  text: "No Face",  org: (20, 50), cv2.FONT_HERSHEY_SIMPLEX,  fontScale: 0.6,  color: (0, 0, 255),  thickness: 1, cv2.LINE_AA)

        # 窗口显示
        cv2.imshow( winname: "Face Detector", im_rd)

        # 按下 q 键退出
        if (cv2.waitKey(1) & 0xFF) == ord('q'):
            break

    # 释放摄像头
    self.cap.release()
    # 删除建立的窗口
    cv2.destroyAllWindows()
```

运行效果：

略

3.2 人脸情感检测

首先，在前一个实验的基础上，在构造函数中，添加帧计数器以及定义存储人脸属性。

```
1个用法
class FaceDetector:

    def __init__(self):
        # 使用特征提取器 get_frontal_face_detector
        self.detector = dlib.get_frontal_face_detector()
        # dlib 的68点模型，使用官方训练好的特征预测器
        self.predictor = dlib.shape_predictor("shape_predictor_68_face_landmarks.dat")

        # 建cv2摄像头对象
        self.cap = cv2.VideoCapture(0)
        # 设置视频参数
        self.cap.set( propId: 3,  value: 480)
        # 初始化帧计数器
        self.frame_count = 0

        # 存储分析后的人脸属性
        self.last_age = None
        self.last_emotion = None
        self.last_gender = None
```

循环读取摄像头画面，将图像转换为灰度以提高处理速度，使用 dlib 进行人脸检测并标记。每隔 10 帧，使用 DeepFace 对检测到的人脸进行年龄、情绪和性别分析。

```
1个用法
def detect_and_display(self):
    while self.cap.isOpened():
        flag, im_rd = self.cap.read()
        if not flag:
            print("Failed to grab frame")
            break

        # 取灰度
        img_gray = cv2.cvtColor(im_rd, cv2.COLOR_RGB2GRAY)
        # 使用人脸检测器检测每一帧图像中的人脸
        faces = self.detector(img_gray, 0)

        if len(faces) != 0:
            # 对每个人脸都标出68个特征点
            for k, d in enumerate(faces):
                cv2.rectangle(im_rd, (d.left(), d.top()), (d.right(), d.bottom()), (0, 0, 255))
                shape = self.predictor(im_rd, d)
                for i in range(68):
                    cv2.circle(im_rd, center: (shape.part(i).x, shape.part(i).y), radius: 1, color: (0, 255, 0), -1)

                # 每10帧进行一次深度分析
                if self.frame_count % 10 == 0:
                    # 保存当前帧图像
                    cv2.imwrite( filename: "temp.jpg", im_rd)
                    try:
                        face_attributes = DeepFace.analyze(img_path="temp.jpg", actions=['age', 'emotion', 'gender'], enforce_detection=False)
                        # 处理分析结果
                        self.process_face_attributes(face_attributes)
                    except Exception as e:
                        print("Error in DeepFace analysis:", e)

                # 显示分析的属性信息
                self.display_attributes(im_rd, d)

        # 帧数累加
        self.frame_count += 1
        # 显示窗口
        cv2.imshow( winname: "Face Detector", im_rd)
        # 按'q'退出
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break

    # 释放摄像头
    self.cap.release()
    # 关闭所有窗口
    cv2.destroyAllWindows()
```

下面是存储和显示人物属性的函数

```
# 存储分析后的人脸属性
1个用法
def process_face_attributes(self, face_attributes):
    if isinstance(face_attributes, list) and face_attributes:
        face_attributes = face_attributes[0]
    self.last_age = face_attributes.get('age')
    self.last_emotion = face_attributes.get('dominant_emotion')
    self.last_gender = face_attributes.get('gender')

# 显示人脸属性
1个用法
def display_attributes(self, im_rd, d):
    if self.last_age and self.last_emotion and self.last_gender:
        cv2.putText(im_rd, text: f"Age: {self.last_age}", org: (d.left(), d.top() - 60), cv2.FONT_HERSHEY_SIMPLEX, fontScale: 0.5, color: (255, 255, 0), thickness: 2)
        cv2.putText(im_rd, text: f"Emotion: {self.last_emotion}", org: (d.left(), d.top() - 40), cv2.FONT_HERSHEY_SIMPLEX, fontScale: 0.5, color: (255, 255, 0), thickness: 2)
        cv2.putText(im_rd, text: f"Gender: {self.last_gender}", org: (d.left(), d.top() - 20), cv2.FONT_HERSHEY_SIMPLEX, fontScale: 0.5, color: (255, 255, 0), thickness: 2)
```

效果展示：

略

3.3 人脸伪造

首先，进行初始化设置，加载必要的库和人脸特征点预测器；定义面部关键区域的点集，

用于对齐和覆盖操作；设定图像处理的基本参数，如缩放因子和模糊参数。

```
import cv2
import dlib
import numpy

import sys

PREDICTOR_PATH = "./shape_predictor_68_face_landmarks.dat"
# 图像缩放比例
SCALE_FACTOR = 1
# 用于模糊掩码边界的参数
FEATHER_AMOUNT = 11

# 定义各个部分的特征点索引
FACE_POINTS = list(range(17, 68))
MOUTH_POINTS = list(range(48, 61))
RIGHT_BROW_POINTS = list(range(17, 22))
LEFT_BROW_POINTS = list(range(22, 27))
RIGHT_EYE_POINTS = list(range(36, 42))
LEFT_EYE_POINTS = list(range(42, 48))
NOSE_POINTS = list(range(27, 35))
JAW_POINTS = list(range(0, 17))

# 用于对齐图像的特征点
ALIGN_POINTS = (LEFT_BROW_POINTS + RIGHT_EYE_POINTS + LEFT_EYE_POINTS +
                RIGHT_BROW_POINTS + NOSE_POINTS + MOUTH_POINTS)

# 用于覆盖的特征点组，使用凸包进行覆盖
OVERLAY_POINTS = [
    LEFT_EYE_POINTS + RIGHT_EYE_POINTS + LEFT_BROW_POINTS + RIGHT_BROW_POINTS,
    NOSE_POINTS + MOUTH_POINTS,
]

# 颜色校正的模糊度
COLOUR_CORRECT_BLUR_FRAC = 0.6

# 初始化dlib的人脸检测器和特征点预测器
detector = dlib.get_frontal_face_detector()
predictor = dlib.shape_predictor(PREDICTOR_PATH)
```

再定义两个异常处理类：TooManyFaces 和 NoFaces，分别用于处理检测到多个或零个人脸的情况。

```
# 如果检测到多于一个人脸，抛出异常。
1个用法
class TooManyFaces(Exception):
    pass

# 如果没有检测到人脸，抛出异常。
1个用法
class NoFaces(Exception):
    pass
```

定义面部特征点检测函数 get_landmarks(im)，在给定图像中检测人脸并提取 68 个特征点

```python
# 检测给定图像中的人脸，并提取68个面部特征点
1个用法
def get_landmarks(im):
    rects = detector(im, 1)

    if len(rects) > 1:
        raise TooManyFaces
    if len(rects) == 0:
        raise NoFaces

    return numpy.matrix([[p.x, p.y] for p in predictor(im, rects[0]).parts()])
```

定义图像标注函数 annotate_landmarks(im, landmarks)，在图像上标记检测到的面部特征点

```python
#  在图像上标记面部特征点，包括在每个特征点位置绘制圆点和标号，用于调试或可视化特征点位置。
def annotate_landmarks(im, landmarks):
    im = im.copy()
    for idx, point in enumerate(landmarks):
        pos = (point[0, 0], point[0, 1])
        cv2.putText(im, str(idx), pos,
                    fontFace=cv2.FONT_HERSHEY_SCRIPT_SIMPLEX,
                    fontScale=0.4,
                    color=(0, 0, 255))
        cv2.circle(im, pos, radius: 3, color=(0, 255, 255))
    return im
```

定义凸包绘制和掩码生成函数，其中 draw_convex_hull(im, points, color)根据指定的特征点绘制凸包，并填充指定颜色；get_face_mask(im, landmarks)生成面部掩码，这个掩码将用于图像融合中，确保只有面部区域被修改。

```python
# 在图像上根据给定的点绘制凸包并填充指定颜色。这在生成面部区域掩码时用于定义要进行融合的面部区域。
1个用法
def draw_convex_hull(im, points, color):
    points = cv2.convexHull(points)
    cv2.fillConvexPoly(im, points, color=color)


# 生成面部掩码。根据特征点，为给定的面部区域（例如眼睛、鼻子、嘴巴等区域）生成一个掩码，用于之后的图像融合过程。
2个用法
def get_face_mask(im, landmarks):
    im = numpy.zeros(im.shape[:2], dtype=numpy.float64)

    for group in OVERLAY_POINTS:
        draw_convex_hull(im,
                         landmarks[group],
                         color=1)

    im = numpy.array([im, im, im]).transpose((1, 2, 0))

    im = (cv2.GaussianBlur(im, ksize: (FEATHER_AMOUNT, FEATHER_AMOUNT), sigmaX: 0) > 0) * 1.0
    im = cv2.GaussianBlur(im, ksize: (FEATHER_AMOUNT, FEATHER_AMOUNT), sigmaX: 0)

    return im
```

定义图像对齐和变形函数，其中 transformation_from_points(points1, points2)计算两组特征点之间的仿射变换矩阵，用于将一张图像的面部特征对齐到另一张图像；warp_im(im, M, dshape)使用计算得到的仿射变换矩阵变形图像，使其与目标图像对齐。

```
# 根据两组特征点计算仿射变换矩阵，用于将一个人脸对齐到另一个人脸。
1个用法
def transformation_from_points(points1, points2):

    points1 = points1.astype(numpy.float64)
    points2 = points2.astype(numpy.float64)

    c1 = numpy.mean(points1, axis=0)
    c2 = numpy.mean(points2, axis=0)
    points1 -= c1
    points2 -= c2

    s1 = numpy.std(points1)
    s2 = numpy.std(points2)
    points1 /= s1
    points2 /= s2

    U, S, Vt = numpy.linalg.svd(points1.T * points2)

    R = (U * Vt).T

    return numpy.vstack([numpy.hstack(((s2 / s1) * R,
                                       c2.T - (s2 / s1) * R * c1.T)),
                         numpy.matrix([0., 0., 1.])])


# 使用计算得到的仿射变换矩阵（M）来变换图像，使之与目标图像对齐。
2个用法
def warp_im(im, M, dshape):
    output_im = numpy.zeros(dshape, dtype=im.dtype)
    cv2.warpAffine(im,
                   M[:2],
                   dsize: (dshape[1], dshape[0]),
                   dst=output_im,
                   borderMode=cv2.BORDER_TRANSPARENT,
                   flags=cv2.WARP_INVERSE_MAP)
    return output_im
```

定义图像对齐和变形函数，其中 transformation_from_points(points1, points2)计算两组特征点之间的仿射变换矩阵，用于将一张图像的面部特征对齐到另一张图像；warp_im(im, M, dshape)使用计算得到的仿射变换矩阵变形图像，使其与目标图像对齐。

```
# 颜色校正，以匹配两张融合图像的颜色。通过应用高斯模糊来平滑颜色差异，并调整颜色强度，使融合后的图像看起来更自然。
1个用法
def correct_colours(im1, im2, landmarks1):
    blur_amount = COLOUR_CORRECT_BLUR_FRAC * numpy.linalg.norm(
        numpy.mean(landmarks1[LEFT_EYE_POINTS], axis=0) -
        numpy.mean(landmarks1[RIGHT_EYE_POINTS], axis=0))
    blur_amount = int(blur_amount)
    if blur_amount % 2 == 0:
        blur_amount += 1
    im1_blur = cv2.GaussianBlur(im1, ksize: (blur_amount, blur_amount), sigmaX: 0)
    im2_blur = cv2.GaussianBlur(im2, ksize: (blur_amount, blur_amount), sigmaX: 0)

    # Avoid divide-by-zero errors.
    im2_blur += (128 * (im2_blur <= 1.0)).astype(im2_blur.dtype)

    return (im2.astype(numpy.float64) * im1_blur.astype(numpy.float64) /
            im2_blur.astype(numpy.float64))
```

定义读取图像文件函数 read_im_and_landmarks(fname)，调整大小，提取面部特征点

```
# 读取图像文件，调整其大小，并提取面部特征点。这是预处理步骤，为面部融合准备图像和数据。
2个用法
def read_im_and_landmarks(fname):
    im = cv2.imread(fname, cv2.IMREAD_COLOR)
    im = cv2.resize(im, dsize: (im.shape[1] * SCALE_FACTOR,
                        im.shape[0] * SCALE_FACTOR))
    s = get_landmarks(im)

    return im, s
```

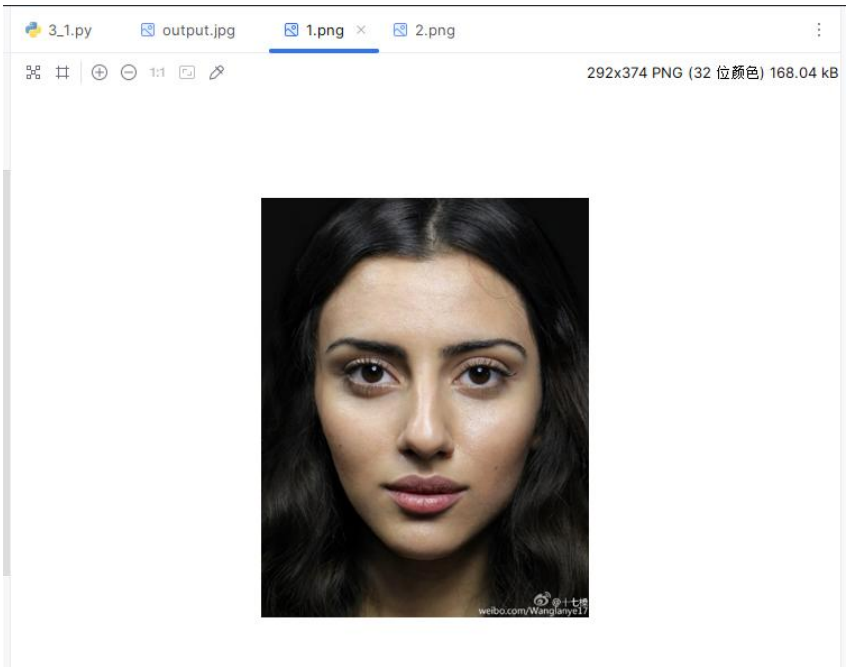
最后，将功能整合起来，实现从读取图像到输出融合后的图像的完整流程。程序首先读取两张图像，提取特征点，计算变换矩阵，生成面部掩码，应用变换，进行颜色校正，最后生成并保存最终的融合图像。

```
# 读取图像及其特征
im1, landmarks1 = read_im_and_landmarks(sys.argv[1])
im2, landmarks2 = read_im_and_landmarks(sys.argv[2])

# 计算从第一张图的特征点到第二张图特征点的仿射变换矩阵
M = transformation_from_points(landmarks1[ALIGN_POINTS], landmarks2[ALIGN_POINTS])

# 为第二张图生成面部掩码
mask = get_face_mask(im2, landmarks2)
# 将面部掩码应用仿射变换，使之与第一张图对齐
warped_mask = warp_im(mask, M, im1.shape)

# 合并第一张图和变形后的第二张图的面部掩码
combined_mask = numpy.max( a: [get_face_mask(im1, landmarks1), warped_mask], axis=0)

# 应用变换矩阵变形第二张图像，使之与第一张图像对齐
warped_im2 = warp_im(im2, M, im1.shape)
# 对变形后的第二张图像进行颜色校正，使其颜色与第一张图像更加吻合
warped_corrected_im2 = correct_colours(im1, warped_im2, landmarks1)

# 生成最终融合图像，通过掩码来混合两张图像的相应部分
output_im = im1 * (1.0 - combined_mask) + warped_corrected_im2 * combined_mask

# 将最终融合后的图像保存到文件
cv2.imwrite( filename: 'output.jpg', output_im)
```

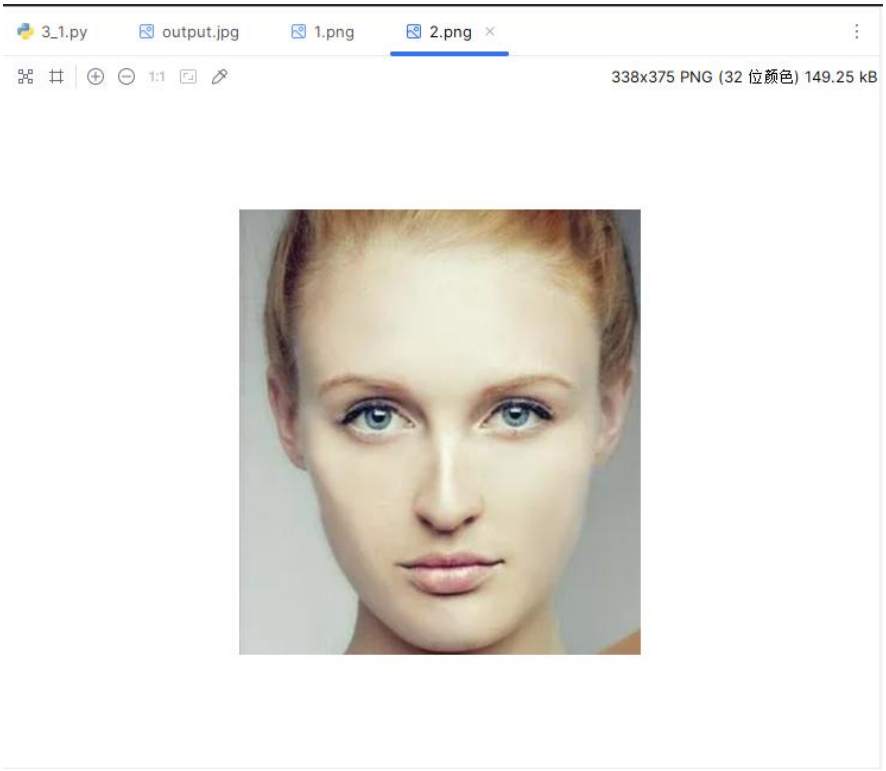
效果展示：

原图

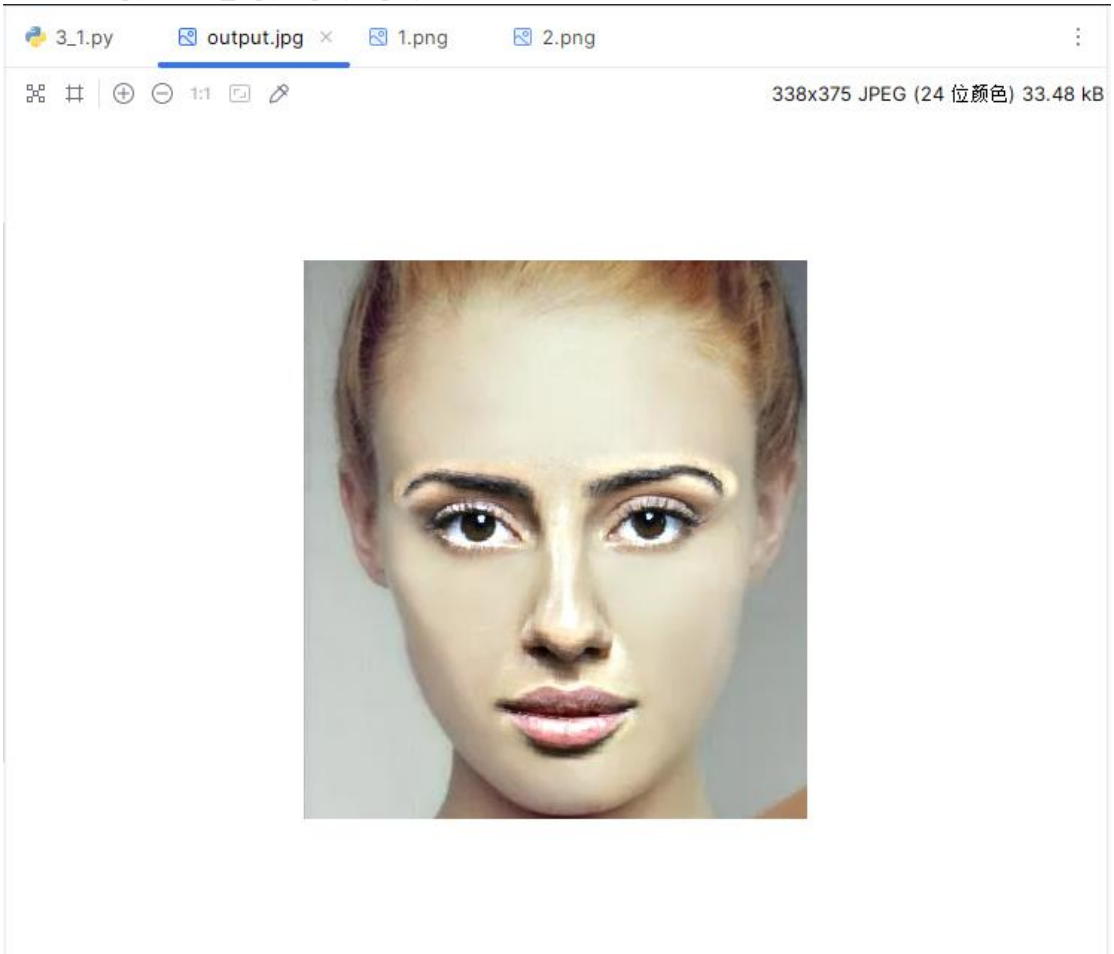
1.png



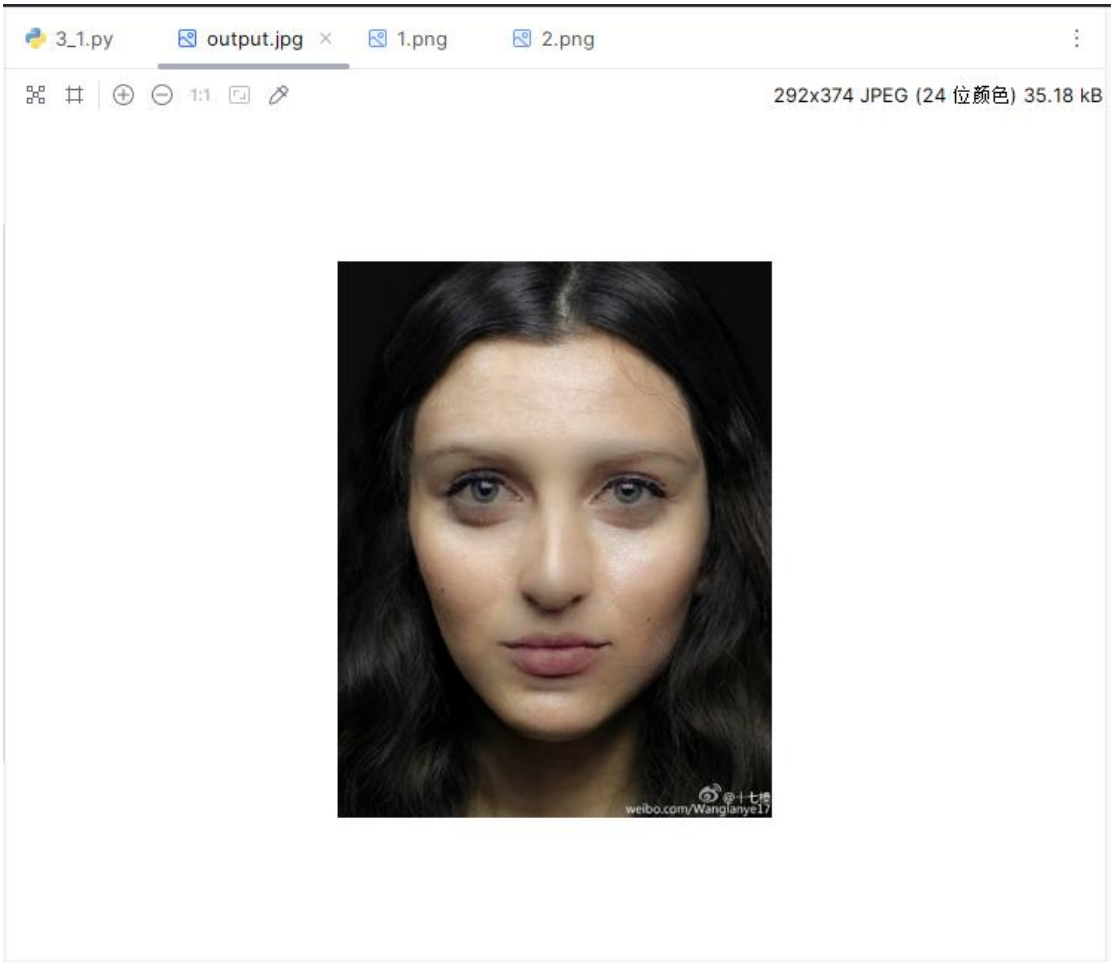
2.png



运行代码 python 3_1.py 2.png 1.png



运行代码 python 3_1.py 1.png 2.png



3.4 人脸伪造图像检测

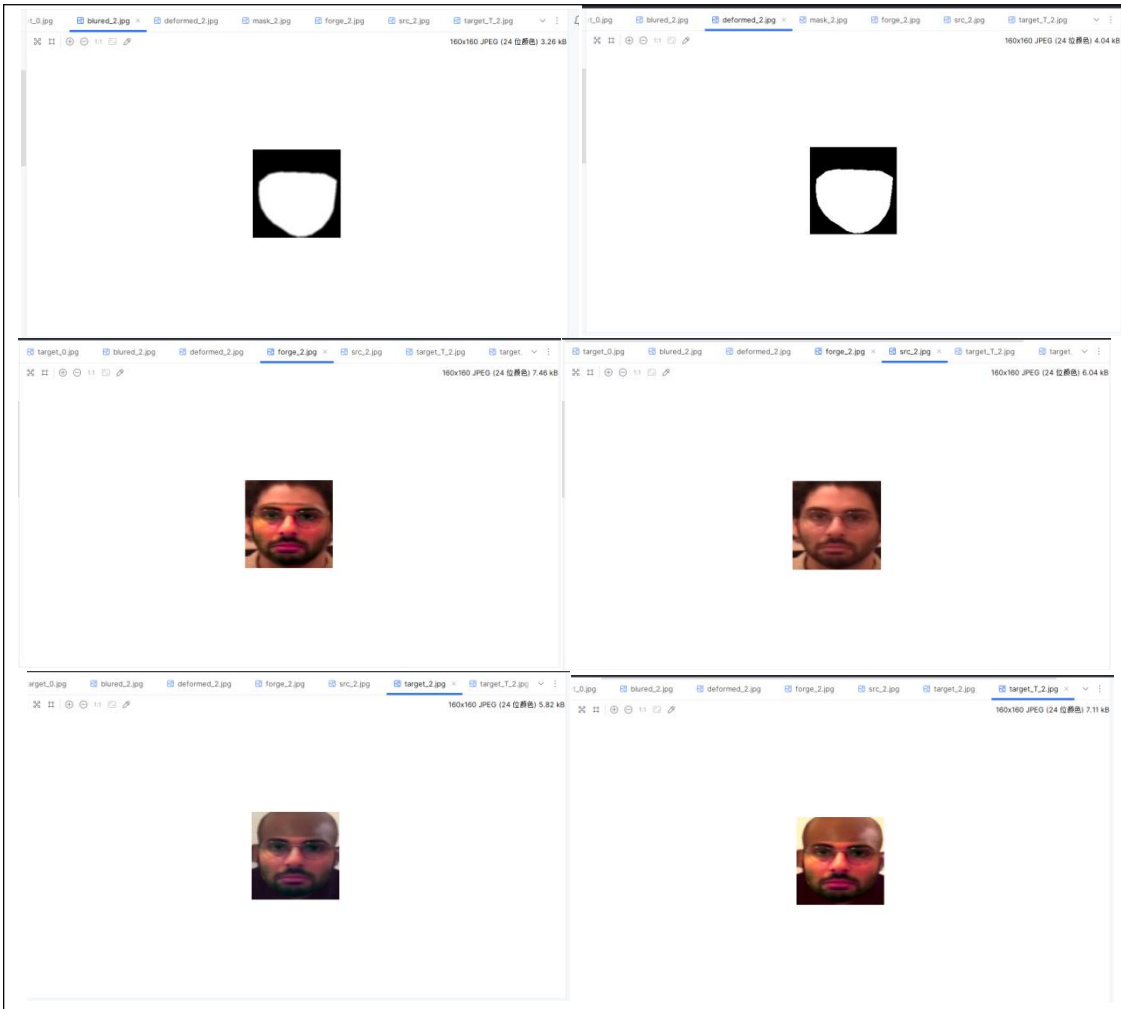
尝试运行 Face-X-Ray-2 的代码，会发现报错

```
(deepface) PS D:\Content_Secu\task3\Face-X-Ray-2> python faceBlending.py --srcFacePath D:\Content_Secu\task3\Face-X-Ray-2\source\ --faceDatabase D:\Content_Secu\task3\Face-X-Ray-2\database
No Match: D:\Content_Secu\task3\Face-X-Ray-2\source\output.jpg
No Match: D:\Content_Secu\task3\Face-X-Ray-2\source\output1.jpg
2it [00:03,  1.85s/it]
```

但是将 database 目录下的文件重命名以 target 开头复制到 source 目录下即可运行成功。
如复制原有的 target_0.jpg 到 source 目录下运行

```
(deepface) PS D:\Content_Secu\task3\Face-X-Ray-2> python faceBlending.py --srcFacePath D:\Content_Secu\task3\Face-X-Ray-2\source\ --faceDatabase D:\Content_Secu\task3\Face-X-Ray-2\database
No Match: D:\Content_Secu\task3\Face-X-Ray-2\source\output.jpg
No Match: D:\Content_Secu\task3\Face-X-Ray-2\source\output1.jpg
3it [00:04,  1.44s/it]
```

发现 dump 文件夹下也生成了文件

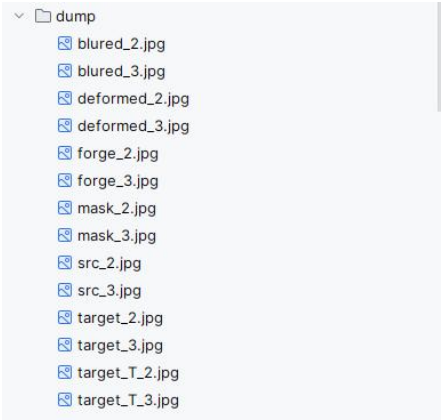


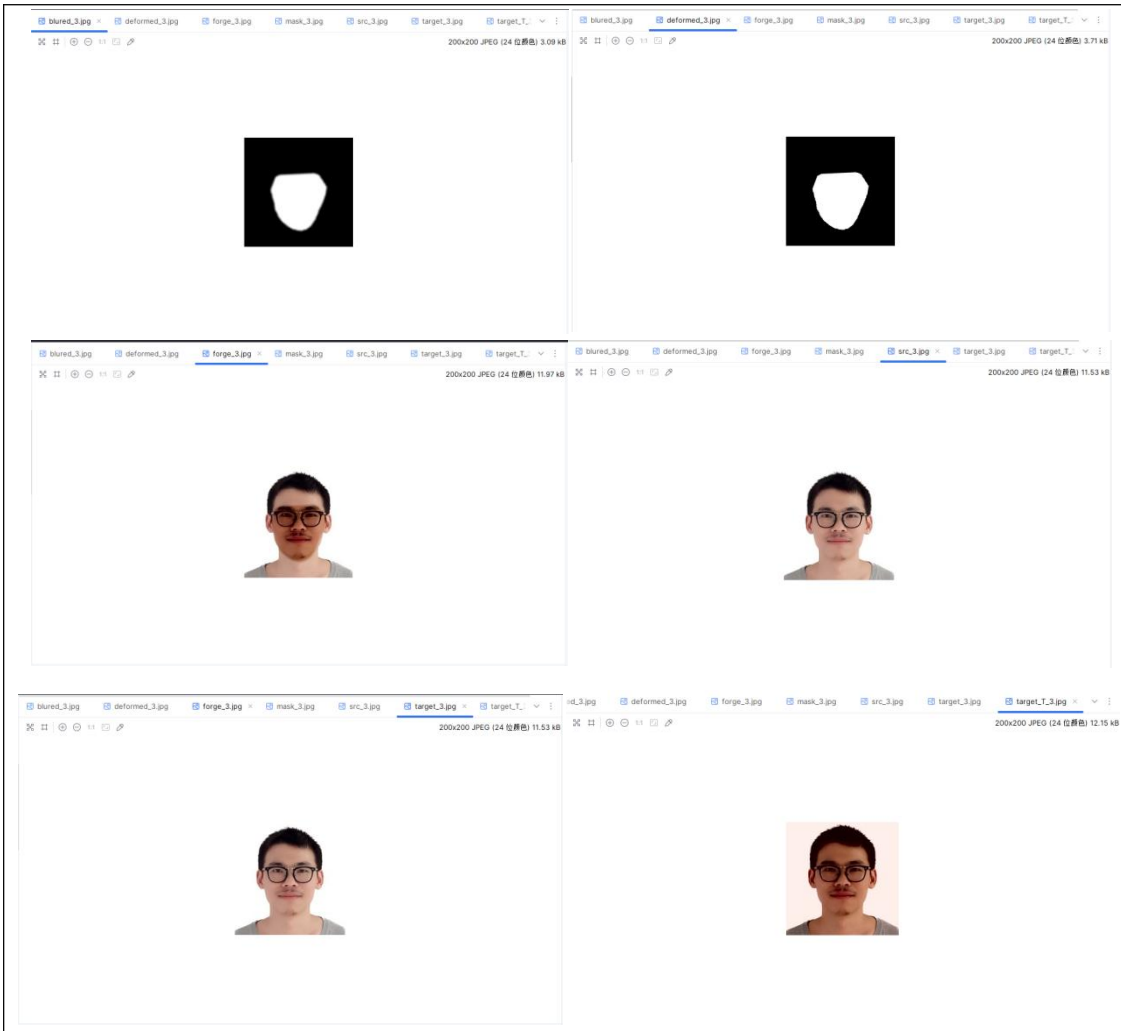
可见，src_2.jpg 与 target_2.jpg 不一致，所以鉴定为换脸

再来一次尝试，将 database 目录下的 1.jpg 重命名为 target_1.jpg 复制到 source 目录下,运行

```
(deepface) PS D:\Content_Secu\task3\Face-X-Ray-2> python faceBlending.py --srcFacePath D:\Content_Secu\task3\Face-X-Ray-2\source\ --faceDatabase D:\Content_Secu\task3\Face-X-Ray-
2\database
No Match: D:\Content_Secu\task3\Face-X-Ray-2\source\output.jpg
No Match: D:\Content_Secu\task3\Face-X-Ray-2\source\output1.jpg
4it [00:04,  1.11s/it]
```

也在 dump 目录下生成了文件

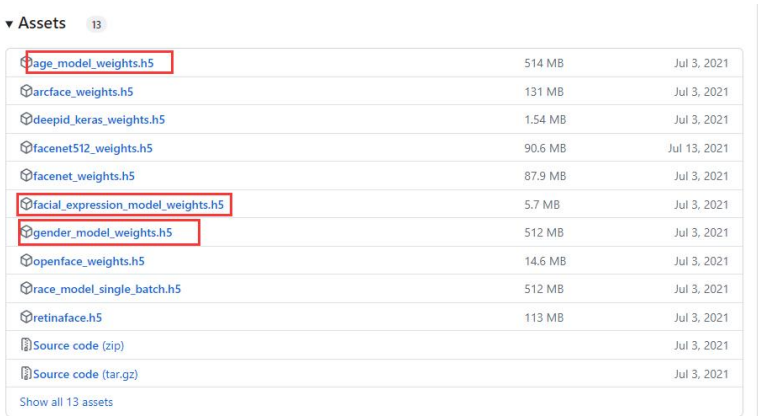




可以发现，src_3.jpg 与 target_3.jpg 一致，故未换脸。

四、出现的问题及解决方法

问题：任务二代码运行时会报错模型下载失败
解决方法：在官网 https://github.com/serengil/deepface_models/releases 中下载相关模型代码



模型下载下来后存储到 C:\Users\用户名\.deepface\weights 中，这样代码运行时才能够调用。

五、个人小结

本次实验中，我学会了如何利用 OpenCV 和 Dlib 进行图像处理和特征点检测。通过学习 DeepFace 框架，我对深度学习在人脸属性分析（如情绪识别）中的应用有了实际的操作经验。在使用 Dlib 进行人脸伪造实验中，我不仅学到了如何生成和应用伪造面部，还理解了这一技术背后的原理，以及如何通过改变和优化算法参数来获得更自然的伪造效果。

在使用 DeepFace 进行情绪检测时，初次配置环境遇到了模型下载失败的问题。通过查询相关资料和社区讨论，我学会了如何手动下载模型并配置到本地库。

通过这次实验，我不仅提升了自己的技术能力，更加深了对人脸识别技术潜在影响的理解。

六、教师评语及评分

教师签名：

年　　月　　日