第五次内容安全实验报告

课程名称	图像伪造				
学生姓名	陈曦	学号	2020302181081	指导老师	张典
专业	网络安全	班级	2020 级 3 班	实验时间	2023.5.10

一、实验内容

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

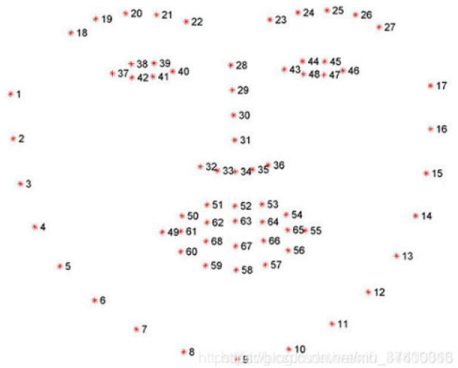
3. 使用 Python3+dlib 实现人脸伪造

4. 使用 Python3+Face-X-Ray 实现人脸伪造图像检测

二、实验原理

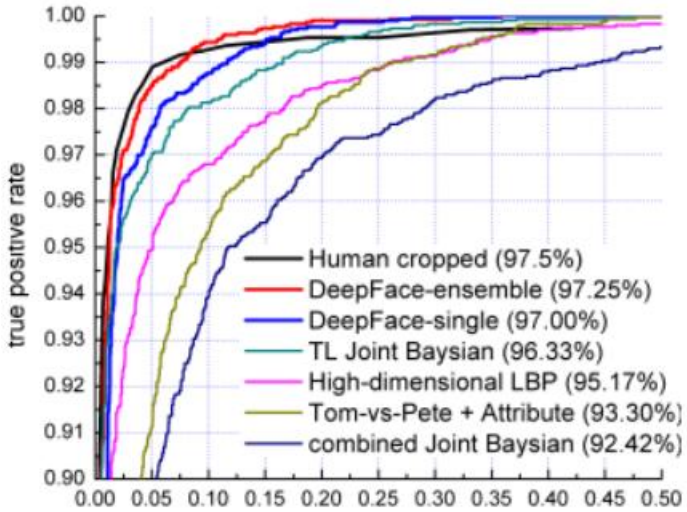
1. 人脸识别关键点

dlib 提取人脸特征点是用 68 个点包围每个部位，如下图，例如第 37 个点到第 42 个点就代表右眼，在图片上这几个点若显示出来就是把右眼那块区域包围着，可以通过这些点之间距离的变化来判断人脸的变化，比如是否眨眼等操作。



2. Deepface 人脸情感检测

deepFace 是轻量级人脸识别和面部属性分析（年龄、性别、情感、种族）框架。集成了人脸识别领域取得过历史最好成绩的几个有名的模型，包括：VGG-Face、Google FaceNet、OpenFace、Facebook DeepFace、DeepID、ArcFace、Dlib。DeepFace 中模型性能得比较如下。



3. 人脸伪造

（1） 使用 dlib 的 shape_predictor_68_face_landmarks.dat 模型，提取有正脸的源图片和目标图片 68 个人脸特征点。

（2） 根据人脸特征点分别获取人脸掩模。

（3） 对源图片仿射变换使其脸部对准目标图片中的脸部得到新的图片。

（4） 对人脸掩模执行相同的操作仿射。

（5） 将两个新得到的图取并集。

（6） 利用 opencv，对仿射变换后的源图片和目标图片进行泊松融合。

4. Face-X-Ray 实现人脸伪造图像检测

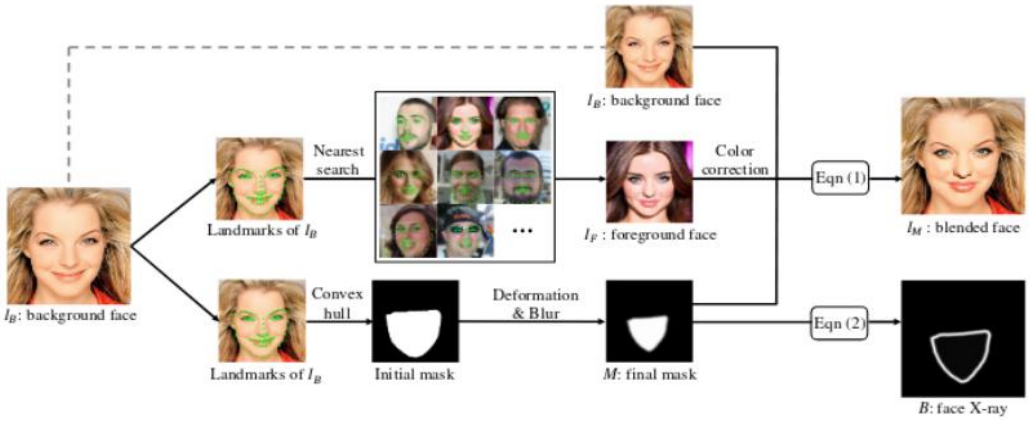
典型的人脸伪造方法主要包括以下三个阶段：

（1） 检测待伪造的面部区域

（2） 生成伪造的目标面部区域

（3） 将源图片和目标人脸进行融合

当前的人脸伪造方法主要集中于通过检测第二阶段中产生出伪造伪影来检测真伪。一个客观事实是对于每一张图片而言其都会存在一个独一无二特征标志，这样的一个特征标志往往是由硬件和软件两个部分共同造成的。作者认为第三阶段的融合过程会产生一个融合的边界，在边界的两侧会存在不一致图片特征，因此可以利用这种不一致来检测融合边界进而实现的对人脸伪造的检测。



需要注意的是，这里的掩码区域 M MM 与融合过程中的使用掩码区域在取值存在着一定的区别，在融合过程中的掩码区域的选值可以为二值的，如前文提及的泊松分布中的融合掩码，但是在 Face X-ray 的定义中掩码区域 M MM 的取值是不可以二值化的，因为二值化会导致生成的 Face X-ray 图像 B 为空白图像，进而导致无法实现检测伪造区域边界的目的，因此在上图生成最后的掩码区域 final mask 时需要进行一步模糊操作，其会使用一个 3*3 的高斯核来对图片进行模糊操作，将二者化的掩码区域转换成非二值化的掩码区域。

三、实验步骤

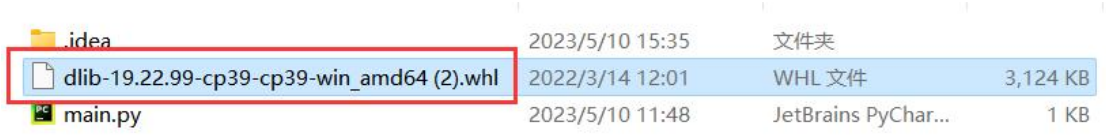
1. 使用 Python3+OpenCV+dlib 实现人脸识别与关键点（landmarks）实时检测

[安装 dlib 库]

下载该工具包。发现发生错误。

```
Failed to build dlib
ERROR: Could not build wheels for dlib, which is required to install pyproject.toml-based projects
```

下载轮子。注意不同 Python 版本的轮子不同。



在对应路径下载该文件。使用命令 pip install。

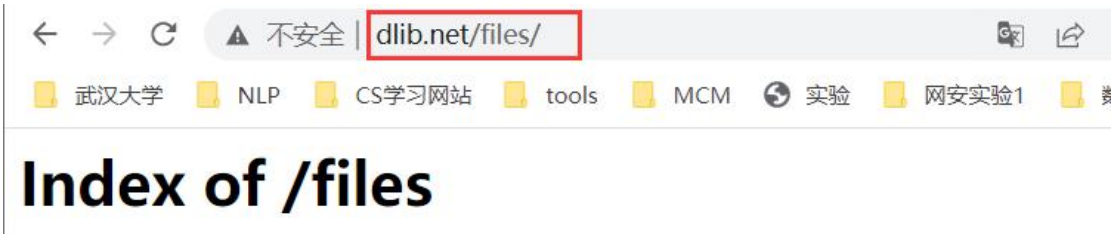
```
(base) D:\chenxi\code\neian5>pip install dlib-19.22.99-cp39-cp39-win_amd64.whl
WARNING: Ignoring invalid distribution -rotobuf (d:\chenxi\try\ide\anaconda\lib\site-packages)
Processing d:\chenxi\code\neian5\dlib-19.22.99-cp39-cp39-win_amd64.whl
WARNING: Ignoring invalid distribution -rotobuf (d:\chenxi\try\ide\anaconda\lib\site-packages)
Installing collected packages: dlib
Successfully installed dlib-19.22.99
```

再次下载 dlib 包发现已经下载完成。

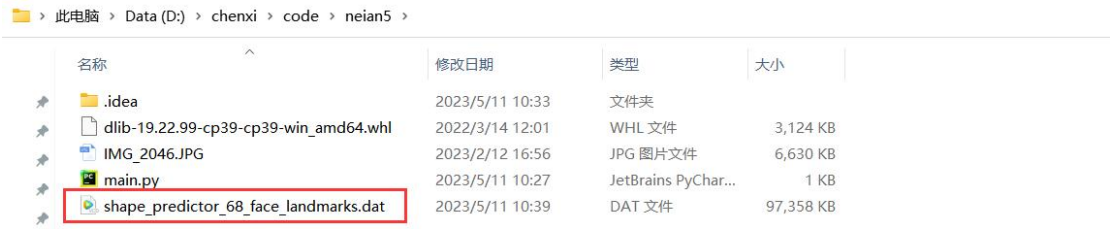
```
(base) D:\chenxi\code\neian5>pip install dlib
WARNING: Ignoring invalid distribution -rotobuf (d:\chenxi\try\ide\anaconda\lib\site-packages)
Requirement already satisfied: dlib in d:\chenxi\try\ide\anaconda\lib\site-packages (19.22.99)
WARNING: Ignoring invalid distribution -rotobuf (d:\chenxi\try\ide\anaconda\lib\site-packages)
```

下载人脸检验关键点模型文件 shape_predictor_68_face_landmarks.dat.bz2。

使用网址 dlib.net/files/，找到该文件并下载。



下载完成后，解压放在目标文件夹中，即可被程序使用。



[编程实现]

[图像检测]

选择一张带人脸的照片 test.png 作为检测目标。放在 pics 文件夹下，作为待监测的图片。

导入工具包 cv2 和 dlib，进行图像处理。

```
import cv2
import dlib
```

读入 test.png 为 img。

```
path = "pics/test.png"
img = cv2.imread(path)
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

使用人脸分类器，获取人脸检测器 shape_predictor_68_face_landmarks.dat。

```
detector = dlib.get_frontal_face_detector()
predictor = dlib.shape_predictor(
    "shape_predictor_68_face_landmarks.dat"
)
```

寻找人连的 68 个标定点，遍历所有点，打印出坐标，并圈出来。

```
dets = detector(gray, 1)
```

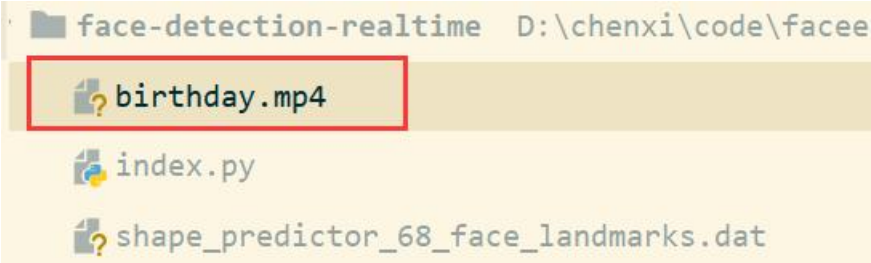
```
    for face in dets:
        shape = predictor(img, face)
        for pt in shape.parts():
            pt_pos = (pt.x, pt.y)
            cv2.circle(img, pt_pos, 2, (0, 255, 0), 1)
        cv2.imshow("image", img)
```

显示该图像。

```
cv2.waitKey(0)
cv2.destroyAllWindows()
```

[实时检测]

将需要进行人脸检测的视频 mp4 文件放在代码文件夹下。



导入需要的工具包。

```
import threading
import cv2
import os
import json
import dlib
```

使用人脸分类器，获取人脸检测器 shape_predictor_68_face_landmarks.dat。

```
detector = dlib.get_frontal_face_detector()
predictor = dlib.shape_predictor(
    "shape_predictor_68_face_landmarks.dat"
)
```

初始化对象的属性。在这个方法中，super().__init__()调用父类的构造函数，即 threading.Thread 类的构造函数，用于初始化线程对象。self.cam_name 和 self.win_name 分别用于存储传入的相机名称和窗口名称，以便在后续操作中使用。

```
class OpcvCapture(threading.Thread):
    def __init__(self, win_name, cam_name):
```

```
            super().__init__()
            self.cam_name = cam_name
            self.win_name = win_name
```

使用之前初始化的人脸检测器 detector 检测当前帧中的所有人脸，并将检测结果存储在 faces 中。cv2.waitKey(1)用于等待用户按下按键，而 1 表示等待 1 毫秒。这样，就可以不断读取视频文件中的每一帧图像，并使用人脸检测器检测人脸。

```
    def run(self):
        #capture = cv2.VideoCapture(self.cam_name)
        capture = cv2.VideoCapture("birthday.mp4")
        while (True):
            ret, frame = capture.read()
            gray = cv2.cvtColor(src=frame, code=cv2.COLOR_BGR2GRAY)
            faces = detector(frame)
        cv2.waitKey(1)
```

运行读取 birthday.mp4 的程序，实现人脸标注。

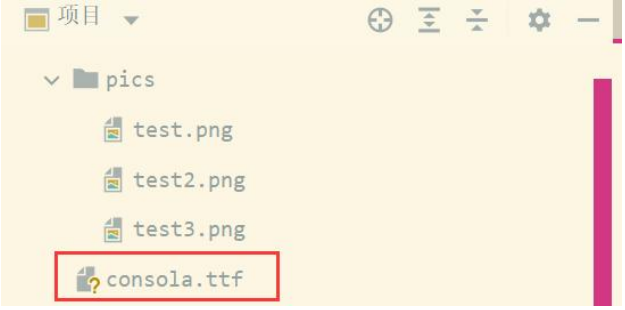
```
if __name__ == "__main__":
    camera1 = OpcvCapture("Face", 0)
    camera1.start()
```

2. 结合任务 1 使用 Python3+OpenCV+Deepface 实现人脸情感检测

使用命令 pip install deepface -i https://pypi.tuna.tsinghua.edu.cn/simple，下载 deepface 工具包。

```
(base) D:\chenxi\code\neian5>pip install deepface -i https://pypi.tuna.tsinghua.edu.cn/simple
WARNING: Ignoring invalid distribution -rotobuf (d:\chenxi\try\ide\anaconda\lib\site-packages)
Looking in indexes: https://pypi.tuna.tsinghua.edu.cn/simple
Collecting deepface
  Downloading https://pypi.tuna.tsinghua.edu.cn/packages/48/ae/b3e7704245ffa505e0733aadcb5794220e4a5b72941c5
b/deepface-0.0.79-py3-none-any.whl (49 kB)
```

将字体包放在项目中，用来图片信息显示。



[图像同人检测]

对比两张人脸是否为同一个人。

导入工具包。

```
import os
import cv2
import pickle
import numpy as np
import numpy.ma
from deepface import DeepFace
from PIL import Image, ImageDraw, ImageFont
```

编写 Print_DeepFace 函数查看 deepface 包含的特征点。可以被调用。

```
def Print_DeepFace():
    print(dir(DeepFace))
```

编写 VerifyPerson 函数，验证两张人脸是否为同一个人。使用 VGG-Face 模型。

打印出两张人脸的相似性，并判断是否相似。并展示对比这两张人脸图。

```
if __name__ == '__main__':
    # Print_DeepFace()
    VerifyPerson()
    # FaceEmotion()
```

[图像情感检测]

使用下面的 FaceEmotion 函数可以检测图像中人脸的情绪，种族，性别和年龄。

主要功能是对人的上述特征进行检测。使用 DeepFace.analyze 分析人脸特征，并打印出检测结果。

其中性别判断需要多加一层判断逻辑。

```
  def FaceEmotion(img_path='pics/test.png'):
      img = cv2.imread(img_path)

      emotion = DeepFace.analyze(img_path=img_path)

      if emotion[0]['gender']['Woman'] < emotion[0]['gender']['Man']:
          str_gender = 'Woman'
      else:
          str_gender = 'Man'

      print("gender:", str_gender)
      print("age:", emotion[0]["age"])
      print("dominant_race:", emotion[0]["dominant_race"])
      print("dominant_emotion:", emotion[0]["dominant_emotion"])
```

将需要打印在图片上的文字存为字符串变量 words。

```
  words = 'gender: ' + str_gender + '\n' \
                                                "age: " + str(emotion[0]["age"]) + '\n' \
                                                  "dominant_race: " + str(emotion[0]["dominant_race"]) + '\n' \
                                                "dominant_emotion: " + str(emotion[0]["dominant_emotion"])
```

使用方框将人脸框出。

```
  img = cv2.resize(src=img, dsize=(450, 450))
      x0=emotion[0]['region']['x']
      y0=emotion[0]['region']['y']
      w=emotion[0]['region']['w']
      h=emotion[0]['region']['h']
      cv2.rectangle(img=img,pt1=(x0,y0),pt2=(x0+w,y0+h),color=(0,255,0),thickness=2)
```

设置字体颜色，字体类型，字体大小，文本位置等参数，使用 draw.text 将文本显示。

```
  img_ptl = Image.fromarray(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
      fillColor = (255, 0, 0)
      position = (100, 50)
```

```
        font = ImageFont.truetype('consola.ttf', 18)
        draw = ImageDraw.Draw(img_ptl)
        draw.text(position, words, font=font, fill=fillColor)
        img = cv2.cvtColor(numpy.ma.asarray(img_ptl), v2.COLO
    R_BGR2RGB)
```

显示该图像。

```
        cv2.imshow('img', img)
        cv2.waitKey(0)
        cv2.destroyAllWindows()
```

运行 FaceEmotion 函数。

```
    if __name__ == '__main__':
        # Print_DeepFace()
        # VerifyPerson()
        FaceEmotion()
```

3. 使用 Python3+dlib 实现人脸伪造

导入工具包。

```
    import cv2
    import dlib
    import numpy as np
    import matplotlib.pyplot as plt
```

定义可视化图像函数。opencv 读入图像格式为 BGR，matplotlib 可视化格式为 RGB，

因此需将 BGR 转 RGB。

```
    def look_img(img):
        img_RGB = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        plt.imshow(img_RGB)
        plt.show()
```

获取图片大小（高度和宽度）。

```
    def get_image_size(image):
        image_size = (image.shape[0], image.shape[1])
        return image_size
```

获取人脸标志，68 个特征点。

```
def get_face_landmarks(image, face_detector, shape_predictor):
    dets = face_detector(image, 1)
    shape = shape_predictor(image, dets[0])
    face_landmarks = np.array([[p.x, p.y] for p in shape.parts()])
    return face_landmarks
```

获取人脸掩模。

```
def get_face_mask(image_size, face_landmarks):
    mask = np.zeros(image_size, dtype=np.uint8)
    points = np.concatenate([face_landmarks[0:16], face_landmarks[26:17:-1]])
    cv2.fillPoly(img=mask, pts=[points], color=255)
    return mask
```

获取待伪造人脸仿射变换后的图片。

```
def get_affine_image(image1, image2, face_landmarks1, face_landmarks2):
    three_points_index = [18, 8, 25]
    M = cv2.getAffineTransform(face_landmarks1[three_points_index].astype(np.float32),
                               face_landmarks2[three_points_index].astype(np.float32))
    dsize = (image2.shape[1], image2.shape[0])
    affine_image = cv2.warpAffine(image1, M, dsize)
    return affine_image.astype(np.uint8)
```

获取掩模的中心点坐标。

```
def get_mask_center_point(image_mask):
    image_mask_index = np.argwhere(image_mask > 0)
    miny, minx = np.min(image_mask_index, axis=0)
    maxy, maxx = np.max(image_mask_index, axis=0)
    center_point = ((maxx + minx) // 2, (maxy + miny) // 2)
    return center_point
```

获取两个掩模掩盖部分的并集。

```
def get_mask_union(mask1, mask2):
    mask = np.min([mask1, mask2], axis=0)  # 掩盖部分并集
    mask = ((cv2.blur(mask, (5, 5)) == 255) * 255).astype(np.uint8)  # 缩小掩模大小
```

```
        mask = cv2.blur(mask, (3, 3)).astype(np.uint8)  # 模糊掩模
        return mask
```

肤色调整。

```
def skin_color_adjustment(im1, im2, mask=None):
    if mask is None:
        im1_ksize = 55
        im2_ksize = 55
        im1_factor = cv2.GaussianBlur(im1, (im1_ksize, im1_ksize), 0).astype(np.float)
        im2_factor = cv2.GaussianBlur(im2, (im2_ksize, im2_ksize), 0).astype(np.float)
    else:
        im1_face_image = cv2.bitwise_and(im1, im1, mask=mask)
        im2_face_image = cv2.bitwise_and(im2, im2, mask=mask)
        im1_factor = np.mean(im1_face_image, axis=(0, 1))
        im2_factor = np.mean(im2_face_image, axis=(0, 1))

    im1 = np.clip((im1.astype(np.float64) * im2_factor / np.clip(im1_factor, 1e-6, None)), 0, 255).astype(np.uint8)
    return im1
```

创建人脸检测器，加载标志点检测器。

取两个待仿造的源图像文件，并调整两张图片的大小。

展示两张图片和图片 1 仿射变换后的图片，最后实现人脸仿造。

```
if __name__ == '__main__':
    det_face = dlib.get_frontal_face_detector()
    det_landmarks = dlib.shape_predictor("shape_predictor_68_face_landmarks.dat")  # 68 点
    im1 = cv2.imread('pics/img1.jpg')  # 源图片
    im1 = cv2.resize(im1, (600, im1.shape[0] * 600 // im1.shape[1]))
    landmarks1 = get_face_landmarks(im1, det_face, det_landmarks)  # 68_face_landmarks
    im1_size = get_image_size(im1)  # 脸图大小
    im1_mask = get_face_mask(im1_size, landmarks1)  # 脸图人脸掩模
    im2 = cv2.imread('pics/img2.jpg')  # 目标图片
```

```
        landmarks2 = get_face_landmarks(im2, det_face, det_landmarks)  # 68_face_landmarks
        im2_size = get_image_size(im2)  # 目标图片大小
        im2_mask = get_face_mask(im2_size, landmarks2)  # 目标图片人脸掩模
        affine_im1 = get_affine_image(im1, im2, landmarks1, landmarks2)  # im1（脸图）仿射变换后的图片
        affine_im1_mask = get_affine_image(im1_mask, im2, landmarks1, landmarks2)  # im1（脸图）仿射变换后的图片的人脸掩模
        union_mask = get_mask_union(im2_mask, affine_im1_mask)  # 掩模合并
        affine_im1 = skin_color_adjustment(affine_im1, im2, mask=union_mask)  # 肤色调整
        point = get_mask_center_point(affine_im1_mask)  # im1（脸图）仿射变换后的图片的人脸掩模的中心点
        seamless_im = cv2.seamlessClone(affine_im1, im2, mask=union_mask, p=point, flags=cv2.NORMAL_CLONE)  # 进行泊松融合
        look_img(im1)
        look_img(im2)
        look_img(affine_im1)
        look_img(seamless_im)
```

4. 使用 Python3+Face-X-Ray 实现人脸伪造图像检测

[项目结构]

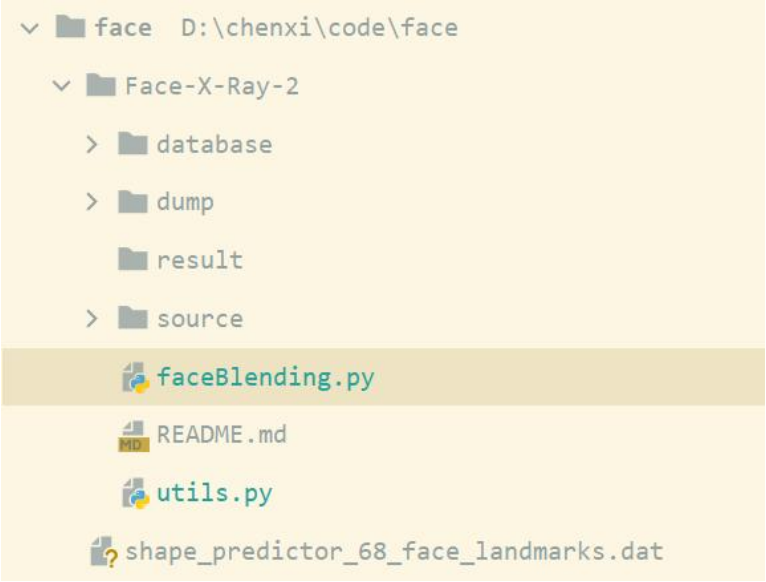
Source 文件夹中存放待检测的伪造图像。

Database 中存放待被识别的正常图像。

Dump 中存放识别的过程图像和结果图像。

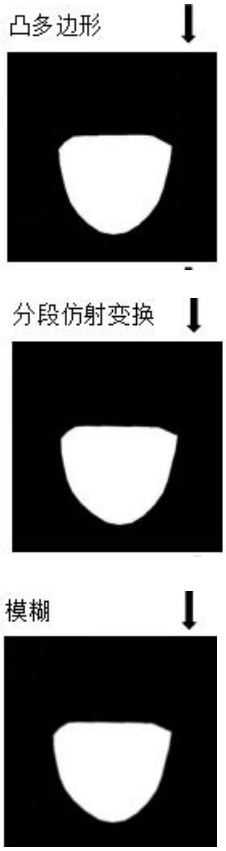
faceBlending.py 和 utils.py 为运行代码。

Shape_predictor_68_Face_landmarks.dat 为 68 个检测点模型。



[编程实现]

将人脸转换成凸多边形，分段仿射变换，模糊来判断。



faceBlending.py：

导入工具包。

```
import argparse, sys, os
from os.path import basename, splitext
from PIL import Image
from functools import partial
from skimage.transform import PiecewiseAffineTransform, warp
import numpy as np
import cv2
import dlib
from tqdm import tqdm
from color_transfer import color_transfer
from utils import files, FACIAL_LANDMARKS_IDXS, shape_to_np
```

设置源文件脸。设置人脸检测模型。

```
args = get_parser()
    # source faces
    srcFaces = tqdm(files(args.srcFacePath, ['.jpg']))
    # real faces database
    #ds = image2pilBatch(files(args.faceDatabase, ['.jpg']))
    # face detector
    detector = dlib.get_frontal_face_detector()
    predictor = dlib.shape_predictor(args.shapePredictor)
```

在源文件脸中的每一张脸都读入。

```
    for i, srcFace in enumerate(srcFaces):
        # load bgr
        try:
            srcFaceBgr = cv2.imread(srcFace)
        except:
            tqdm.write(f'Fail loading: {srcFace}')
            continue
```

从人脸中获得标志点。

```
srcLms = get_landmarks(detector, predictor, cv2.cvtColor(srcFaceBgr, cv2.COLOR_BGR2RGB))
        if srcLms is None:
            tqdm.write(f'No face: {srcFace}')
            continue
```

查找正常图片中哪一个和源文件脸最相似。

```
1.  targetRgb = find_one_neighbor(detector, predictor, srcFac
    e, srcLms, files(args.faceDatabase, ['.jpg']), args.thresh
    old)
2.          if targetRgb is None: # if not found
3.              tqdm.write(f'No Match: {srcFace}')
4.              continue
```

如果找到了最相似的人脸，执行：

```
1.      targetBgr = cv2.cvtColor(targetRgb, cv2.COLOR_RGB2BG
    R)
2.          hullMask = convex_hull(srcFaceBgr.shape, srcLms)
    # size (h, w, c) mask of face convex hull
```

生成仿射变换，并模糊。

```
1.          anchors, deformedAnchors = random_deform(hullMask.
    shape[:2], 4, 4)
2.          # piecewise affine transform and blur
3.          warped = piecewise_affine_transform(hullMask, anc
    hors, deformedAnchors) # size (h, w) warped mask
4.          blured = cv2.GaussianBlur(warped, (5,5), 3)
5.          # swap
6.          left, up, right, bot = min(srcLms[:,0]), min(srcL
    ms[:,1]), max(srcLms[:,0]), max(srcLms[:,1])
7.          targetBgrT = color_transfer(srcFaceBgr[up:bot,lef
    t:right,:], targetBgr)
8.          resultantFace = forge(srcFaceBgr, targetBgrT, blu
    red) # forged face
```

保存人脸图片到 dump 文件夹。

```
1.      cv2.imwrite(f'./dump/mask_{i}.jpg', hullMask)
2.          cv2.imwrite(f'./dump/deformed_{i}.jpg', warped*25
    5)
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', targetBgr
    T)
7.          cv2.imwrite(f'./dump/forge_{i}.jpg', resultantFac
    e)
```

Get_landmarks 函数可以得到脸部的特征。

```
1.  def get_landmarks(detector, predictor, rgb):
```

```
        # first get bounding box (dlib.rectangle class) of face.
        boxes = detector(rgb, 1)
        for box in boxes:
            landmarks = shape_to_np(predictor(rgb, box=box))
            break
        else:
            return None
        return landmarks.astype(np.int32)
```

返回 mask。

```
    def forge(srcRgb, targetRgb, mask):
        #mask = np.dstack([mask]*3)
        return (mask * targetRgb + (1 - mask) * srcRgb).astype(np.uint8)
    def convex_hull(size, points, fillColor=(255,)*3):
        mask = np.zeros(size, dtype=np.uint8) # mask has the same depth as input image
        points = cv2.convexHull(np.array(points))
        corners = np.expand_dims(points, axis=0).astype(np.int32)
        cv2.fillPoly(mask, corners, fillColor)
        return mask
```

设置规格尺寸。

```
    def random_deform(imageSize, nrows, ncols, mean=0, std=5):
        h, w = imageSize
        rows = np.linspace(0, h-1, nrows).astype(np.int32)
        cols = np.linspace(0, w-1, ncols).astype(np.int32)
        rows, cols = np.meshgrid(rows, cols)
        anchors = np.vstack([rows.flat, cols.flat]).T
        assert anchors.shape[1] == 2 and anchors.shape[0] == ncols * nrows
        deformed = anchors + np.random.normal(mean, std, size=anchors.shape)
        np.clip(deformed[:,0], 0, h-1, deformed[:,0])
        np.clip(deformed[:,1], 0, w-1, deformed[:,1])
        return anchors, deformed.astype(np.int32)
```

一些简单的组织函数。

```
    def piecewise_affine_transform(image, srcAnchor, tgtAnchor):
        trans = PiecewiseAffineTransform()
```

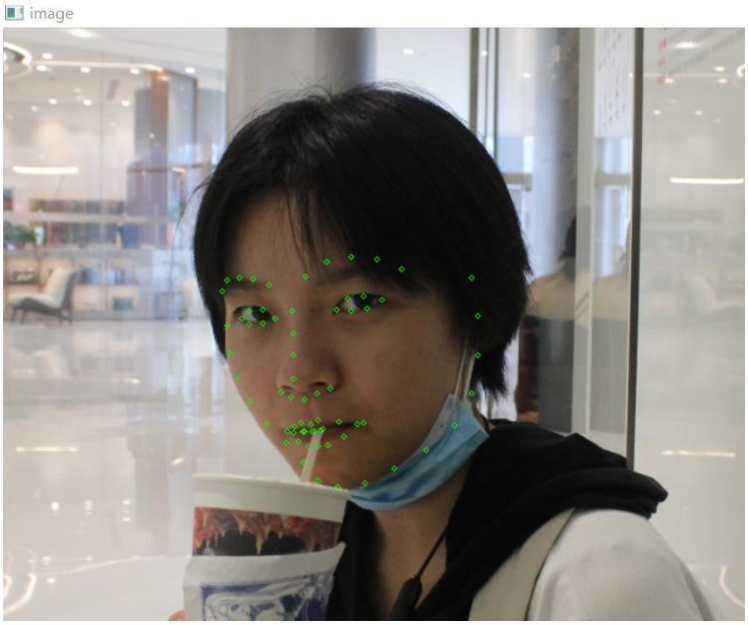
```
        trans.estimate(srcAnchor, tgtAnchor)
        warped = warp(image, trans)
        return warped
    def distance(lms1, lms2):
        return np.linalg.norm(lms1 - lms2)
    def get_parser():
        parser = argparse.ArgumentParser(description='Demo for face x-ray fake sample generation')
        parser.add_argument('--srcFacePath', '-sfp', type=str)
        parser.add_argument('--faceDatabase', '-fd', type=str)
        parser.add_argument('--threshold', '-t', type=float, default=25, help='threshold for facial landmarks distance')
        parser.add_argument('--shapePredictor', '-sp', type=str, default='D:/chenxi/code/face/shape_predictor_68_face_landmarks.dat', help='Path to dlib facial landmark predictor model')
        return parser.parse_args()


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

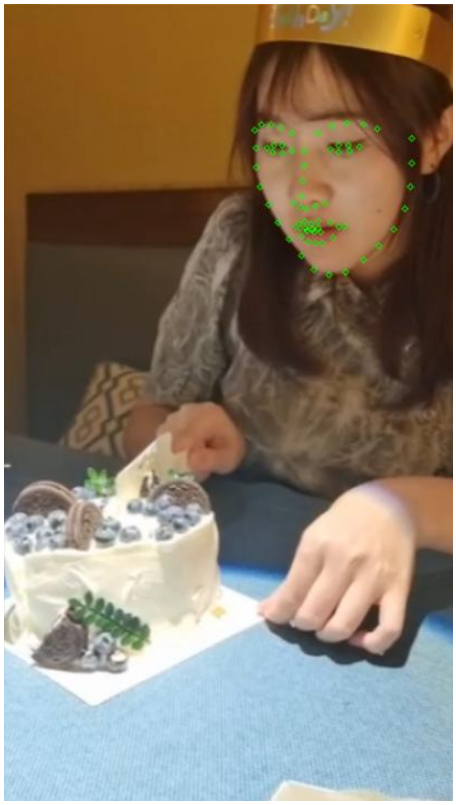
四、实验结果

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

图片人脸关键点检测。



视频关键点检测。



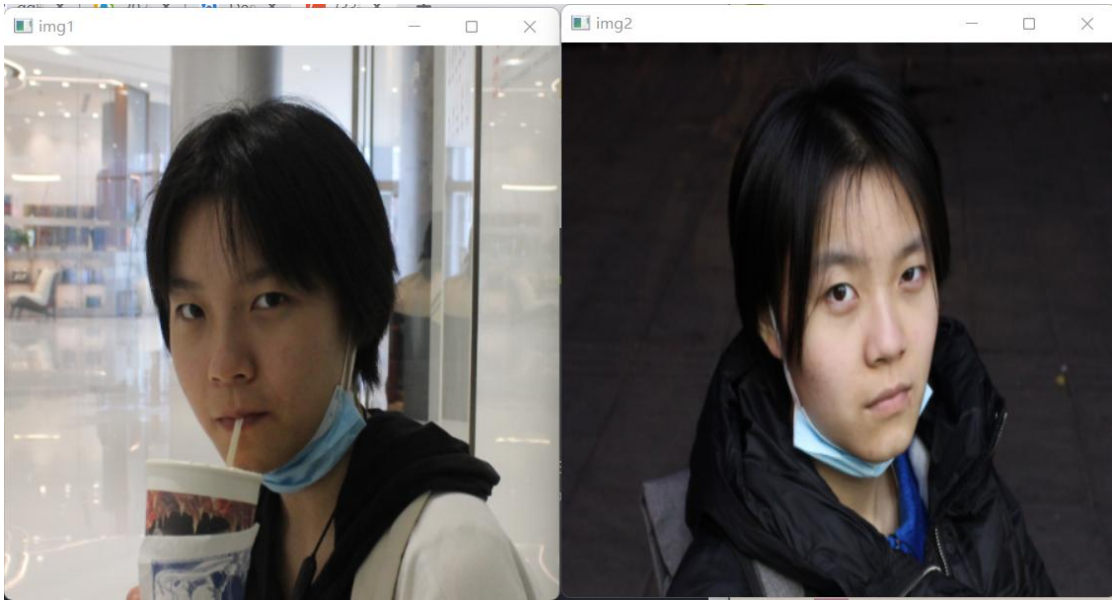
2.结合实验任务 1 使用 Python3+OpenCV+Deepface 实现人脸情感检测

打印属性，可以看到 DeepFace 的各种属性。

```
main2
D:\chenxi\try\ide\anaconda\python.exe D:/chenxi/code/neian5/main2.py
Directory  C:\Users\raven /.deepface created
Directory  C:\Users\raven /.deepface/weights created
['Age', 'ArcFace', 'DeepID', 'DlibWrapper', 'Emotion', 'Facenet', 'Facenet512', 'FbDeepFace', 'Gender', 'OpenFace', 'Race', 'SFace', 'VGGFace',

进程已结束,退出代码0
```

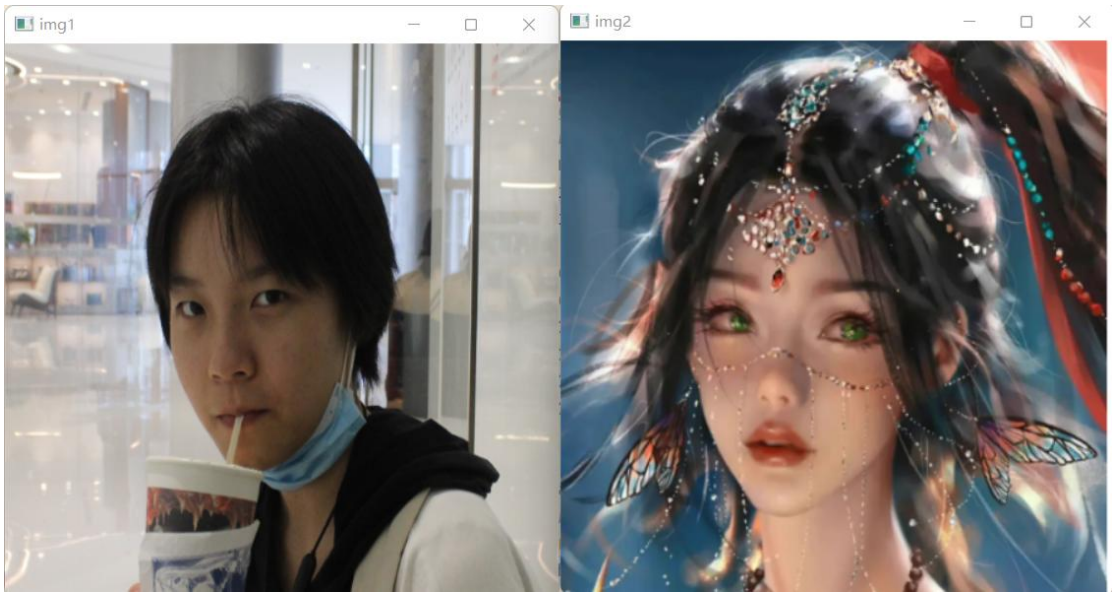
判断两个人是否为同一个人。



输出结果为是同一个人。

```
Downloading...
From: https://github.com/serengil/deepface_models/releases/download/v1.0/vgg_face_weights.h5
To: C:\Users\raven\.deepface\weights\vgg_face_weights.h5
100%|██████████| 580M/580M [00:50<00:00, 11.5MB/s]
{'verified': True, 'distance': 0.3410092190180912, 'threshold': 0.4, 'model': 'VGG-Face', 'detector_backend'
是否相似: True
```

换掉照片，检测是否为同一个人。

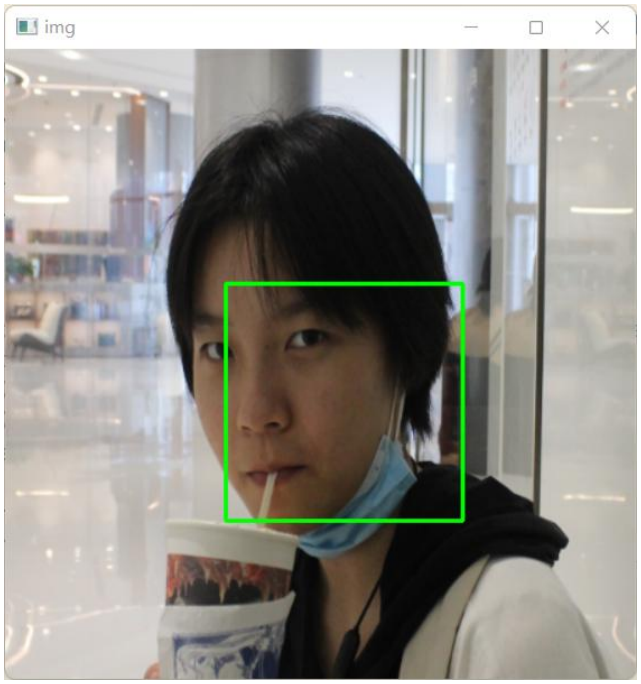


检测结果为不是同一个人。

```
D:\chenxi\try\ide\anaconda\python.exe D:/chenxi/code/neian5/main2.py
{'verified': False, 'distance': 0.449984769281052, 'threshold': 0.4, 'model': 'VGG-Face', 'detector_backend': 'opencv', 'similarity_metric': 'cosine', 'facia
是否相似: False

进程已结束,退出代码0
```

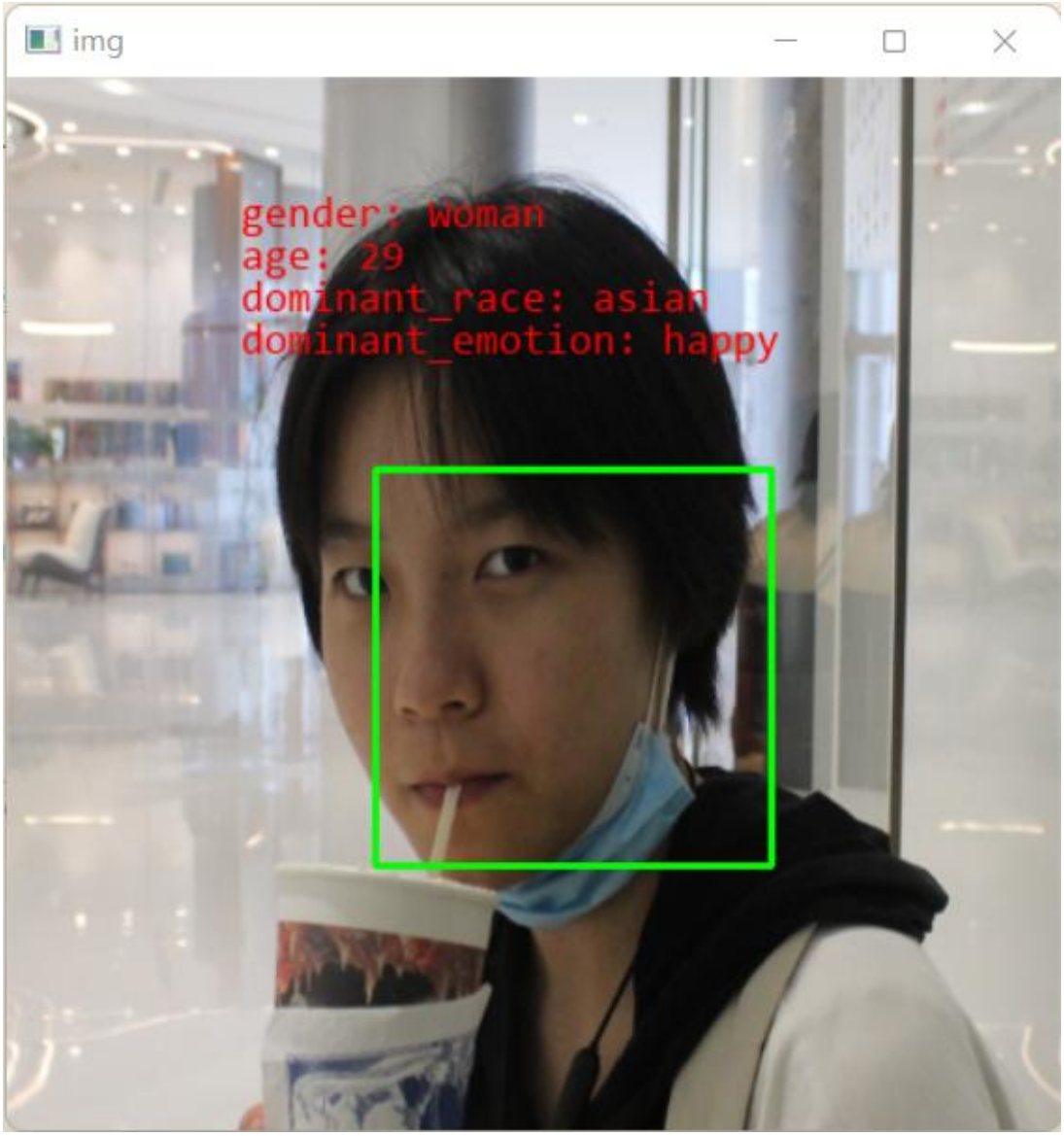
人脸情感检测，可以在终端输出信息。



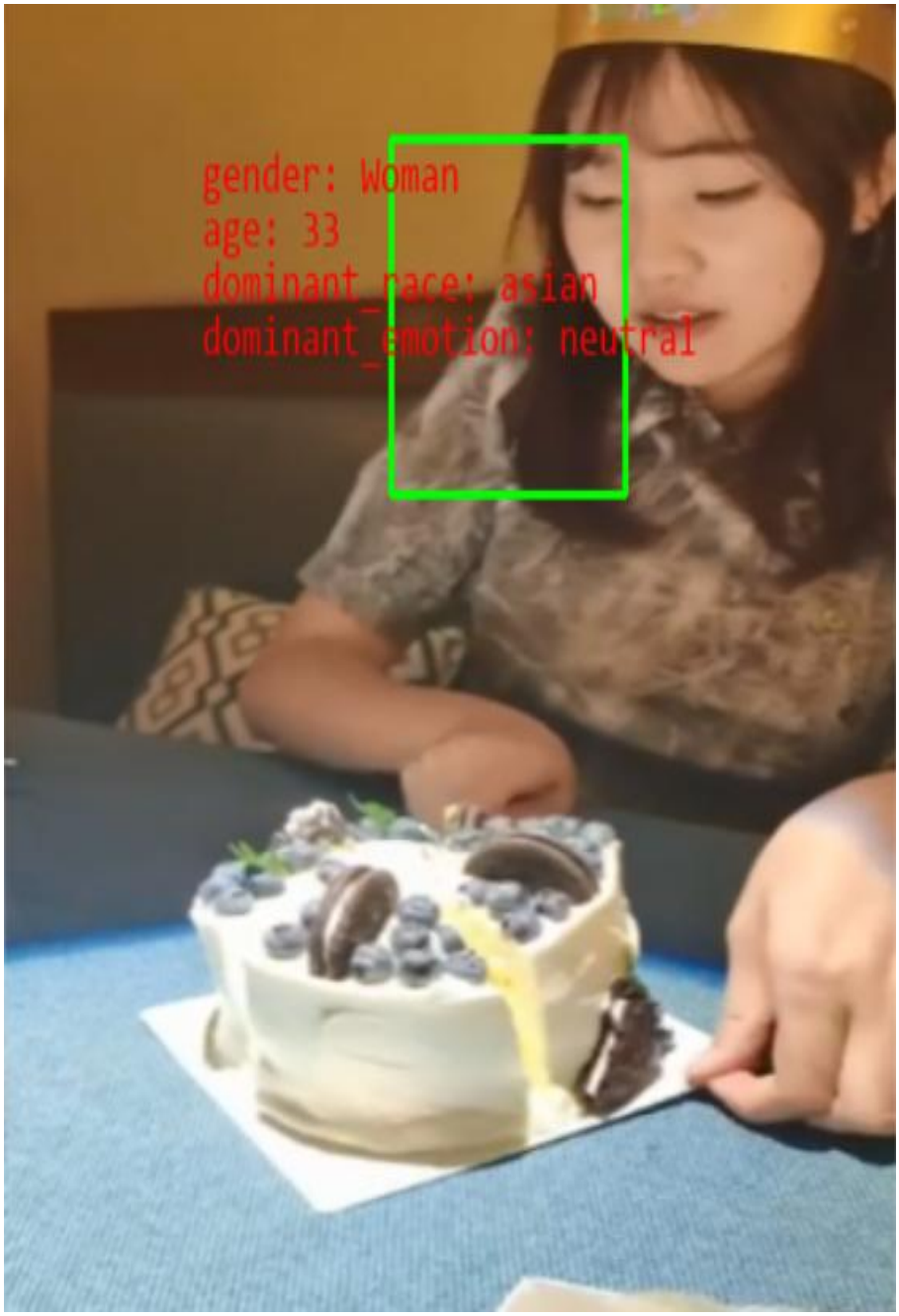
在终端查看信息，可得该人脸是 29 岁开心的亚洲女人。

```
D:\chenxi\try\ide\anaconda\python.exe D:/chenxi/code/neian5/main2.py
Action: race: 100%|██████████| 4/4 [00:01<00:00,  3.94it/s]
gender: Woman
age: 29
dominant_race: asian
dominant_emotion: happy
```

将文字添加在图片上，如图所示。

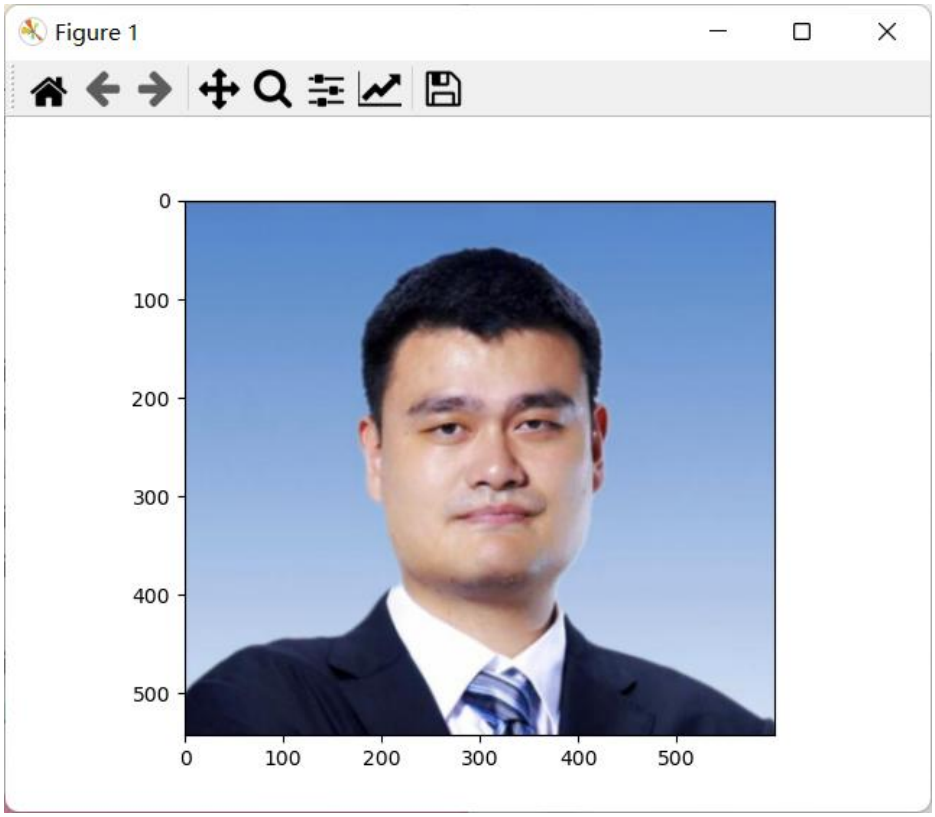


将该程度和任务一的实时监测结合起来，可以检测到视频中人脸情感特征。

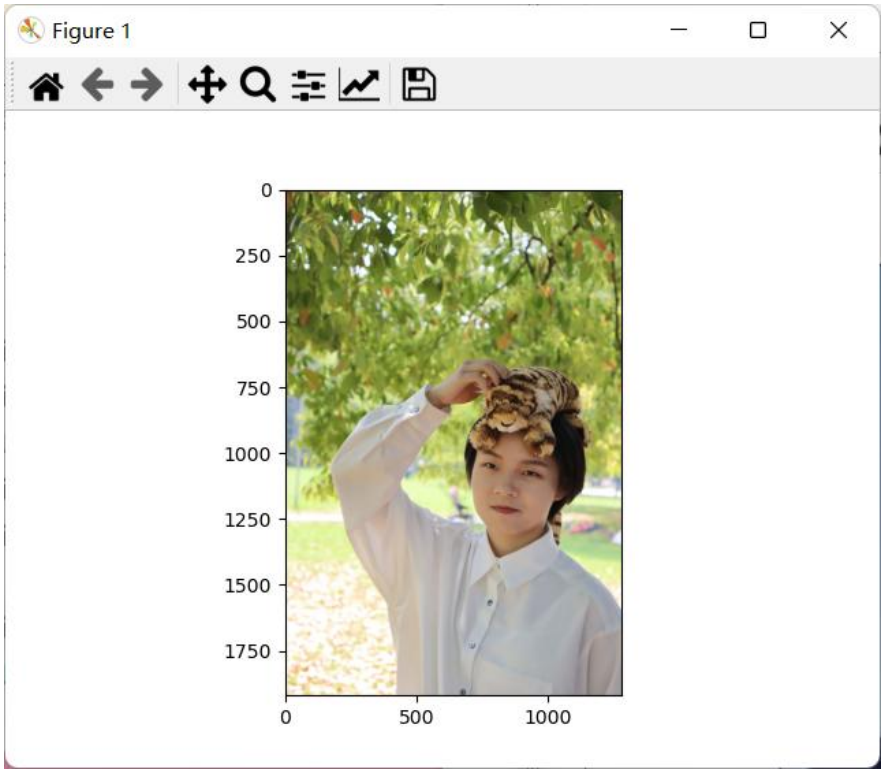


3.使用 Python3+dlib 实现人脸伪造

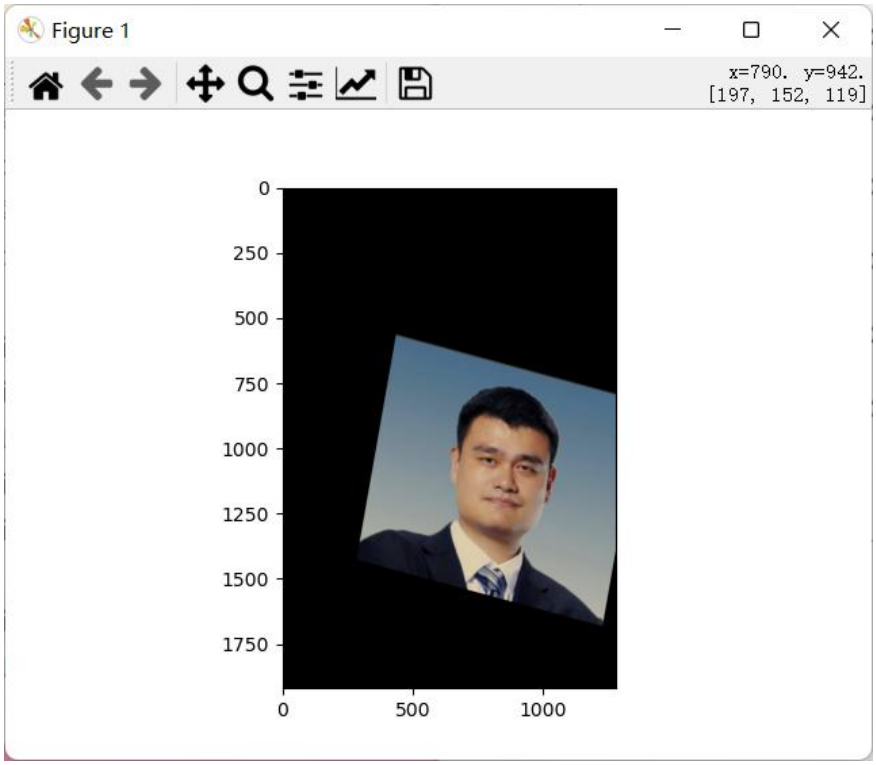
仿造人脸为姚明。



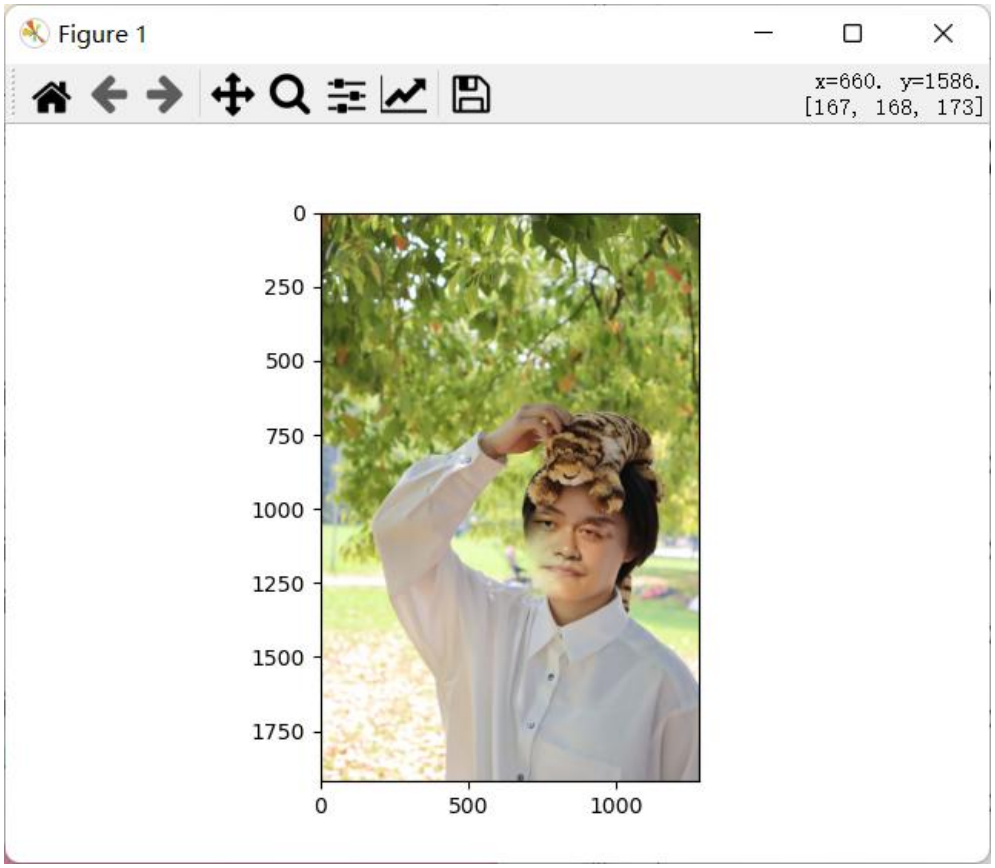
展示被仿造图片。



仿造人脸的仿射变换。

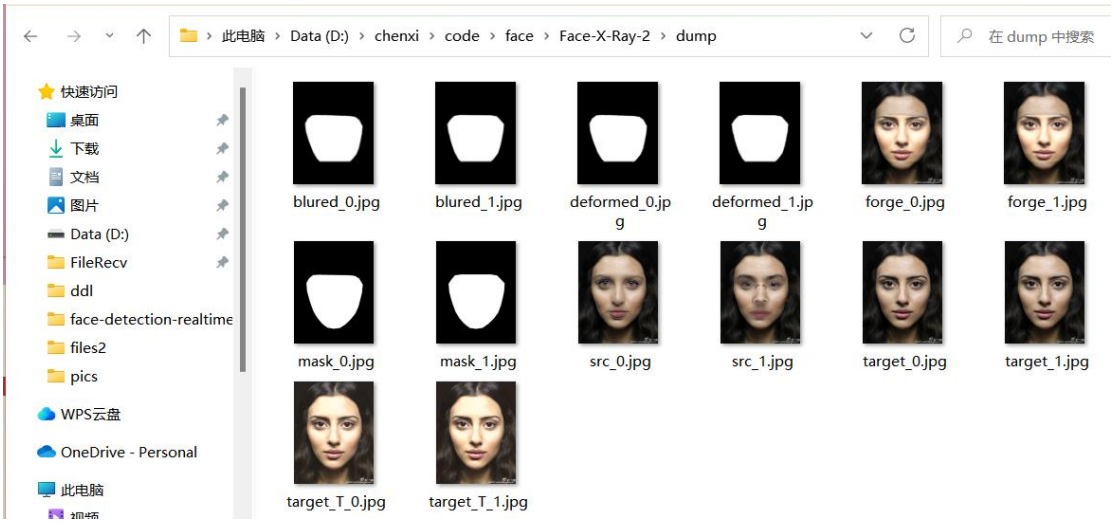


人脸伪造结果如图所示。



4. 使用 Python3+Face-X-Ray 实现人脸伪造图像检测

在 dump 文件夹中可以查看到检测结果。



五．实验心得

本次实验我学习了图像和视频的人脸识别，人脸情感分析，人脸伪造和人脸伪造检测，受益匪浅。通过学习了人脸识别工具 dlib 和 deepface 的基本用法，实现了对人脸的各种操作。