第 四 次 实 验 报 告

课程名称	内容安全实验				
学生姓名		学号		指导老师	张典
专业	网络空间安全	班级		实验时间	2022.05.12

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

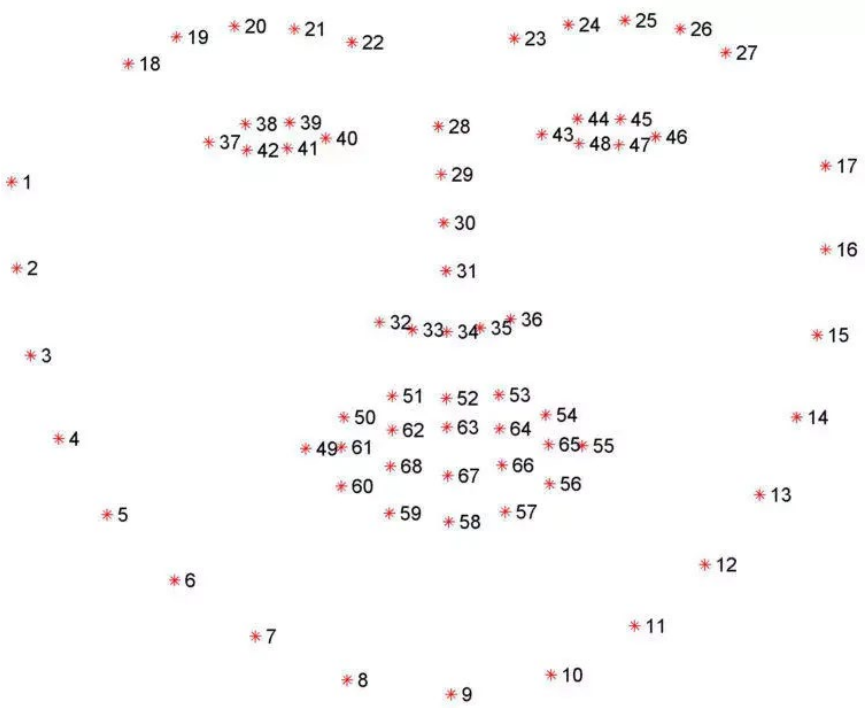
有关人脸的任务可以分类为好多种，比如，人脸检测指的是在图像中检测出人脸的位置；人脸配准指根据输入图像和人脸的位置，输出人脸上五官关键点的坐标；人脸属性识别一般是在人脸配准的基础上，根据人脸判断人的年龄、性别、种族、情感等属性；人脸特征提取指将图像中的人脸转换为用数字表示的向量，可以用于深度学习模型中；人脸比对表示衡量两张人脸图像之间的相似度，这个任务可以通过计算两张人脸图像对应的特征向量之间的距离实现；人脸验证指判断两张人脸图像是否来源于同一个人，也可以通过比较相应的特征向量来实现；人脸识别指在库中找到一个与输入图像中的人脸相似度超过一定阈值，且相似度最高的人脸图像；人脸检索与人脸识别类似，将库中的图像按照与输入图像中的人脸相似度从高到低进行排列；人脸聚类指将库中彼此相似度较高的图像划分为同一类。

本实验中主要使用到的人脸检测工具是 python 中的 dlib 库，dlib 库主要使用基于图像的 HOG 特征，搭配支持向量机(SVM)的人脸检测算法。dlib 库使用图像的 HOG 特征表示人脸，这是因为相较于其他特征而言，HOG 特征对图像的

几何和光学形变拥有较好地不变性。除了人脸检测，实际上在大多数物体检测任务中，HOG 特征都常常与 SVM 相搭配使用。

dlib 库的人脸检测算法主要思路如下，从包含人脸的正样本和不包含人脸的负样本中分别提取 HOG 特征，得到特征描述子，然后使用 SVM 进行训练，视为二分类问题。训练后得到一个初步的模型，然后不断调整或缩放负样本，用模型进行分类，如果模型在负样本中检测出了人脸，则将检测出的区域裁剪下来并加入负样本集中。如此反复训练得到最终的分类模型。在应用模型检测图像中的人脸时，因为会在不同尺度上对图像进行检测，所以结果可能会出现多张人脸检测边框有所重合的情况，再使用极大抑制(NMS)保留唯一边框作为最终检测结果即可。

人脸关键点模型有许多种，其中 dlib 库主要使用的人脸关键点模型是 68 点位的模型。Dlib 库在检测人脸关键点时，先检测出输入图像中的人脸，由于图像中的人脸尺寸多种多样，所以需要使用仿射变换调整人脸关键点的尺寸。将调整后的人脸关键点做平均，再进行残差计算从而拟合成为最终的人脸关键点。



人脸伪造算法一般分为以下三个步骤：检测目标图像中的人脸；再利用换脸算法尝试将目标人脸的特征融合在待伪造图像的人脸中，从而生成新的待替换人

脸以及一部分人脸周围区域，从而使得伪造出的人脸更为真实；最后将生成的图像替换到待伪造图像中，从而在待伪造图像中生成新的人脸，完成人脸伪造任务。

根据上述人脸伪造算法步骤，人脸伪造的检测也有相应的方法。针对第二步，可以在训练集中加入大量伪造图像，让模型学习伪造人脸的特征，从而识别出伪造的人脸图像。这种方法的优点在于算法思想较为简单，较为容易实现；缺点在于不同的人脸伪造算法生成的人脸特征不尽相同，因此在训练集中存在该人脸伪造算法生成的样本时，算法表现较好，对于没有在训练集中出现过的人脸伪造算法，该模型的分类准确率不高。

本实验中使用到的 Face-X-Ray 工具弥补了上述方法的不足，该方法直接从人脸伪造算法的第三步入手，而不只是检测被替换的人脸。该方法通过生成灰度图像，判断该图像是否可以分解为两张不同来源的图像的混合，从而检测出换脸的边界，这个思想类似于图像伪造的检测。该方法不需要了解伪造者使用了何种人脸伪造算法，因为哪种人脸伪造算法都需要进行不同来源的图像的融合，所以该方法更具有普适性。

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

首先需要获取用于定位人脸关键点的模型，既可以使用自定义数据集训练，也可以在 dlib 官网上下载训练好的模型。本实验中我使用的是 dlib 官网提供的模型文件 shape_predictor_68_face_landmarks.dat。

然后开始编写代码，先定义人脸检测器和特征提取器。

```
# 人脸检测器
detector = dlib.get_frontal_face_detector()
# 特征提取器
predictor = dlib.shape_predictor('../models/shape_predictor_68_face_landmarks.dat')
```

接下来使用 cv2 库的 VideoCapture 函数读取 video.mp4 视频文件，再使用 read 方法按帧读取视频帧，从而实现视频抽帧的功能。Read 方法返回 ret 和 img 两个值，其中返回值 ret 是一个布尔值，当正确读取视频帧时返回 True，当读取到视频结尾或错误读取视频帧时返回 False，因此可以通过这个返回值判断当

前是否已读取到视频结尾。返回值 img 是读取到的当前帧的图像，将 img 作为参数可以传入至 landmarks 函数或 recognition 函数中，从而实现不同功能。这两个函数的功能以及定义会在下文中讲述。在代码的最后分别释放两个库所占用的资源。

```
if __name__ == '__main__':
    cap = cv2.VideoCapture('../data/video.mp4')
    while True:
        ret, img = cap.read()
        if ret == False:
            break
        landmarks(img)
        # recognition(img)

    cap.release()
    cv2.destroyAllWindows()
```

landmarks 函数用于人脸关键点检测，即在视频的每帧图像中绘制人脸关键点。首先将该帧图像转换为灰度图，从而能够传入人脸检测器中进行检测，然后将检测到的人脸数据传入特征提取器中，以得到相应人脸的 68 个关键点。再将这些关键点绘制在每帧的图像中。

```
# 人脸关键点检测
def landmarks(img):
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    dets = detector(gray, 1)
    for face in dets:
        # 寻找68个人脸关键点
        shape = predictor(img, face)
        # 绘制所有点
        for pt in shape.parts():
            cv2.circle(img, (pt.x, pt.y), 2, (0, 255, 0), 1)
        cv2.namedWindow('image', cv2.WINDOW_FREERATIO)
        cv2.imshow('image', img)
        cv2.waitKey(1)
```

recognition 函数用于人脸识别，即在视频的每帧图像中绘制人脸识别结果边框。该函数与 landmarks 函数类似，先将图像转换为灰度图，再传入到人脸检

测器中检测人脸，最后提取识别到的人脸边框在图像中的坐标，并根据这些坐标将人脸识别结果边框在图像中绘制出来。

```
# 人脸识别
def recognition(img):
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    dets = detector(gray, 1)
    for face in dets:
        left = face.left()
        right = face.right()
        top = face.top()
        bottom = face.bottom()
        # 绘制人脸识别结果边框
        cv2.rectangle(img, (left, top), (right, bottom), (0, 255, 0), 2)
        cv2.namedWindow('image', cv2.WINDOW_FREERATIO)
        cv2.imshow('image', img)
        cv2.waitKey(1)
```

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

与实验任务 1 类似，因为需要使用到 dlib 库的人脸关键点检测接口，所以需要先定义人脸检测器和特征提取器。

```
# 人脸检测器
detector = dlib.get_frontal_face_detector()
# 特征提取器
predictor = dlib.shape_predictor('../models/shape_predictor_68_face_landmarks.dat')
```

然后按帧读取视频帧，每 10 帧进行一次人脸属性检测。

```
if __name__ == '__main__':
    cap = cv2.VideoCapture('../data/video.mp4')
    i = 0
    while True:
        ret, img = cap.read()
        i += 1
        if ret == False:
            break
        # 每隔10帧检测一次人脸属性
        if i % 10 != 0:
            continue
        analyze(img, i)
```

人脸属性检测的代码封装在 analyze 函数中，函数先将待检测的视频帧作为图像保存在本地，从而方便 DeepFace 库的 analyze 函数的调用。然后调用 DeepFace 库的 analyze 函数根据人脸检测年龄、性别、种族和情感这四个属性。

```
img_path = '../data/video%d.jpg' % (i)
# 将抽取的视频帧保存在本地
cv2.imwrite(img_path, img)
# 人脸属性检测
ana = DeepFace.analyze(img_path=img_path, actions=['age', 'gender', 'race', 'emotion'], enforce_detection=False)
age = ana['age']
gender = ana['gender']
race = ana['dominant_race']
emotion = ana['dominant_emotion']
```

函数的最后使用人脸检测器和特征提取器检测人脸关键点，并将其绘制在图像上，同时在图像中输出人脸年龄、性别、种族和情感检测结果。

```
# 人脸关键点检测
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
dets = detector(gray, 1)
for face in dets:
    # 寻找68个人脸关键点
    shape = predictor(img, face)
    # 绘制所有点
    for pt in shape.parts():
        cv2.circle(img, (pt.x, pt.y), 2, (0, 255, 0), 1)
    # 绘制性别检测结果
    cv2.putText(img, 'gender: %s' % (gender), (90, 400), cv2.FONT_HERSHEY_SIMPLEX, 1.2, (35, 170, 242), 4)
    # 绘制种族检测结果
    cv2.putText(img, 'race: %s' % (race), (90, 450), cv2.FONT_HERSHEY_SIMPLEX, 1.2, (35, 170, 242), 4)
    # 绘制年龄检测结果
    cv2.putText(img, 'age: %s' % (age), (90, 500), cv2.FONT_HERSHEY_SIMPLEX, 1.2, (35, 170, 242), 4)
    # 绘制情感检测结果
    cv2.putText(img, 'emotion: %s' % (emotion), (90, 550), cv2.FONT_HERSHEY_SIMPLEX, 1.2, (35, 170, 242), 4)
    cv2.namedWindow('image', cv2.WINDOW_FREERATIO)
    # 展示图像
    cv2.imshow('image', img)
    cv2.waitKey(1)
```

3. 使用 Python3+dlib 实现人脸伪造

首先定义用于保存模型路径和区分人脸关键点等作用的常量，然后定义 dlib 库的人脸检测器和根据预训练模型导入的特征提取器。

```
PREDICTOR_PATH = "../models/shape_predictor_68_face_landmarks.dat"
SCALE_FACTOR = 1
FEATHER_AMOUNT = 11

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))

# 用于描绘图像的点
ALIGN_POINTS = (LEFT_BROW_POINTS + RIGHT_EYE_POINTS + LEFT_EYE_POINTS +
                               RIGHT_BROW_POINTS + NOSE_POINTS + MOUTH_POINTS)

# 在第二张图像中需要覆盖第一张图像的点，每个元素的凸包将会被覆盖
OVERLAY_POINTS = [
    LEFT_EYE_POINTS + RIGHT_EYE_POINTS + LEFT_BROW_POINTS + RIGHT_BROW_POINTS,
    NOSE_POINTS + MOUTH_POINTS,
]

# 颜色校正过程使用的模糊量，作为瞳孔距离的一部分
COLOUR_CORRECT_BLUR_FRAC = 0.6

# 人脸检测器
detector = dlib.get_frontal_face_detector()
# 特征提取器
predictor = dlib.shape_predictor(PREDICTOR_PATH)
```

然后读取两张图像，一张是待伪造图像，另一张是待伪造人脸的图像，后者图像中的人脸会覆盖前者图像中的人脸，最终输出图像中除了人脸部分外，其余内容都与待伪造图像相同。代码中将前者设为图像 1，后者设为图像 2。读入两张图像后，返回图像的数据以及人脸关键点信息。再计算两张图像中人脸的变换矩阵，提取两张图像中人脸的 mask，并得到伪造后人脸，最后将伪造后人脸覆盖到待伪造图像中，从而得到伪造后的图像。

```
input_path1 = '../data/face1.jpg'
input_path2 = '../data/face2.png'
output_path = '../data/output.jpg'

# 读入两张图像
im1, landmarks1 = read_im_and_landmarks(input_path1)
im2, landmarks2 = read_im_and_landmarks(input_path2)

# 计算两张图像的变换矩阵
M = transformation_from_points(landmarks1[ALIGN_POINTS],
                               landmarks2[ALIGN_POINTS])

# 提取两张图像中人脸的mask
mask = get_face_mask(im2, landmarks2)
warped_mask = warp_im(mask, M, im1.shape)
combined_mask = numpy.max([get_face_mask(im1, landmarks1), warped_mask],
                          axis=0)

# 将待伪造人脸变换到能与待伪造图像相符合
warped_im2 = warp_im(im2, M, im1.shape)
warped_corrected_im2 = correct_colours(im1, warped_im2, landmarks1)

# 将待伪造人脸覆盖到待伪造图像中
output_im = im1 * (1.0 - combined_mask) + warped_corrected_im2 * combined_mask

# 将修改后的图像保存在本地
cv2.imwrite(output_path, output_im)
```

4. 使用 Python3+Face-X-Ray 实现人脸伪造图像检测

代码支持四个参数，-sfp 表示待检测的伪造人脸图像目录，-fd 表示正常人脸图像目录，-t 表示人脸关键点距离的阈值，-sp 表示特征提取器使用的预训练模型文件路径。检测结果存放于 dump 目录中。

```
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='../models/shape_predictor_68_face_landmarks.dat', \
        help='Path to dlib facial landmark predictor model')
    return parser.parse_args()
```

可以使用实验目录中的 src/project4.py 进行人脸伪造检测，使用命令如下所示。

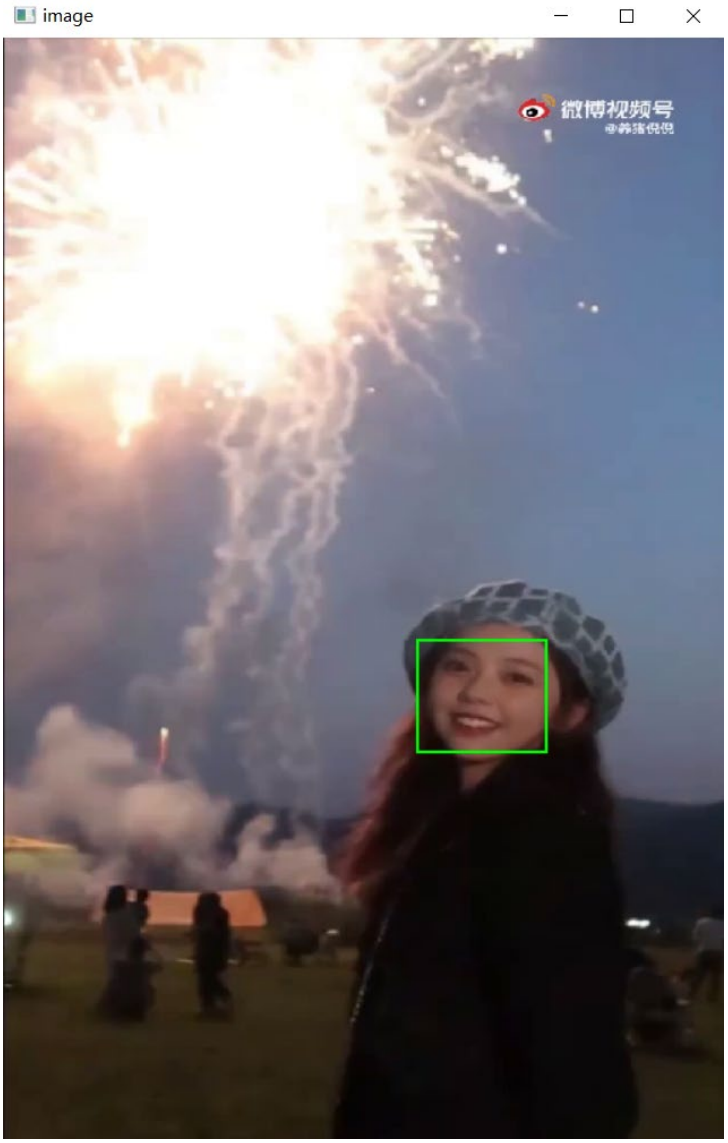
python project4.py -sfp=..\Face-X-Ray-2\source -fd=..\Face-X-Ray-2\database

```
(cs) C:\Users\22848\Desktop\Prominent\内容安全\内容安全第四次实验\src>python project4.py -sfp=..\Face-X-Ray-2\source -fd=..\Face-X-Ray-2\database
No Match: ..\Face-X-Ray-2\source\222222.jpg
No Match: ..\Face-X-Ray-2\source\output.jpg
No Match: ..\Face-X-Ray-2\source\output1.jpg
5it [00:06,  1.33s/it]
```

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

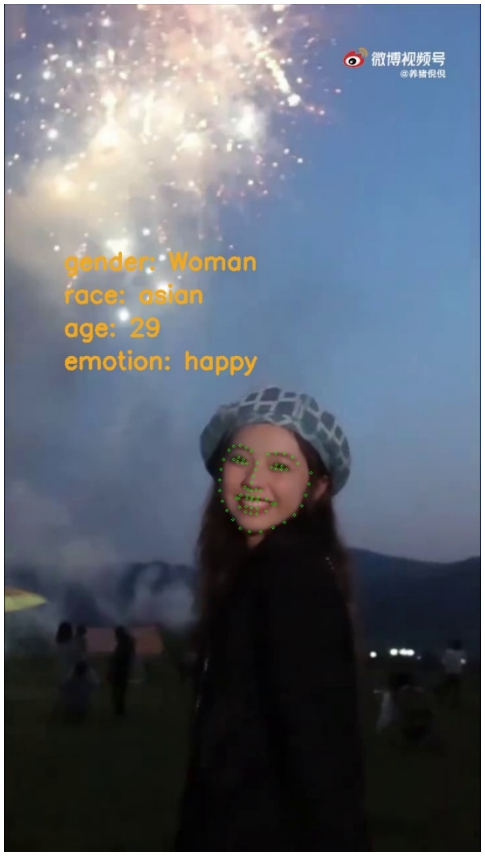
视频人脸识别检测结果



视频人脸关键点检测结果

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

每隔 10 帧抽取的视频帧的人脸属性检测结果


gender: Woman
race: asian
age: 29
emotion: happy

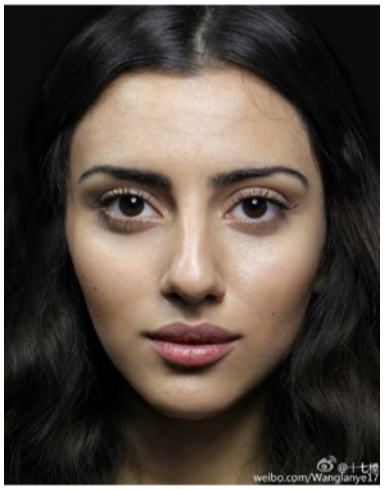

gender: Woman
race: asian
age: 28
emotion: happy


gender: Woman
race: white
age: 29
emotion: fear

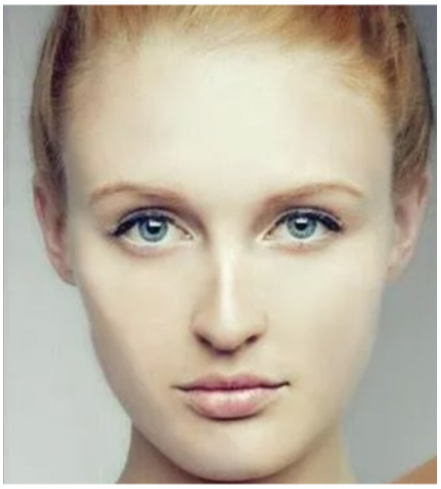

gender: Man
race: asian
age: 31
emotion: happy

3. 使用 Python3+dlib 实现人脸伪造

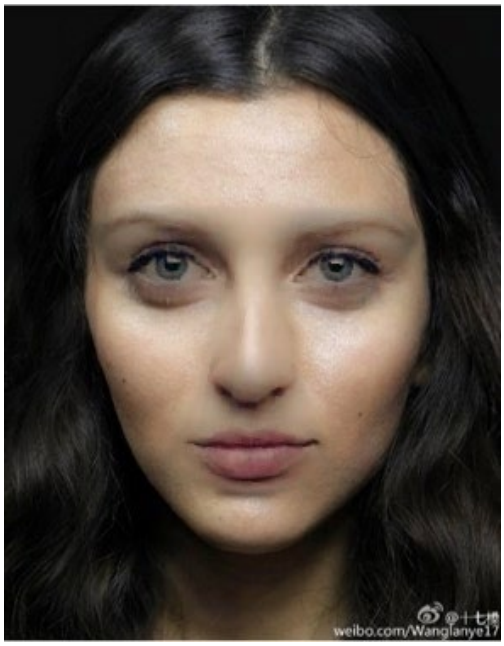
待伪造图像



待伪造人脸的图像



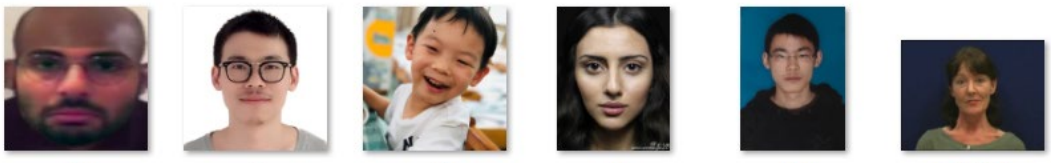
伪造后的图像



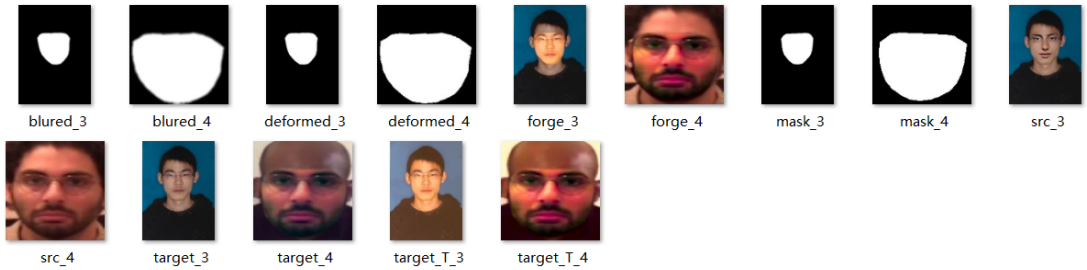
4. 使用 Python3+Face-X-Ray 实现人脸伪造图像检测

有五张待检测的人脸伪造图像

正常人脸图像



检测结果如下所示，五张待检测的图像中有两张图像被检测出了是伪造的人脸图像



只有一张待检测图像的情况

人脸伪造图像检测结果