说明

环境：

Anaconda 集成 python 3.6.5

主要使用了 dlib numpy opencv (调用为cv2) os这四个库

录制+截屏：FSCapture 8.0

目的：

将视频中的人脸更换为指定照片中的人脸，并且输出视频。

思路:

首先使用opencv将一个视频分割为帧，将每一帧保存至origin文件夹内，然后利用transfer.py将每一帧图片转换并且保存至transfer文件夹内，然后继续使用opencv将每一帧的图片在转换为视频，最终保存新视频并且删除origin和 transfer内的缓存帧图。

算法：（以transfer为主）

在transfer.py里面主要使用了dlib库去提取人脸的68个特征点，通过特征点的重叠和转换以及色彩校正实现了换脸。

这个过程主要分以下四步：

1、检测脸部标记。

2、旋转、缩放、平移和第二张图片，以配合第一步。

3、调整第二张图片的色彩平衡，以适配第一张图片。

4、把第二张图像的特性混合在第一张图像中。

**1、检测脸部标记:（准备工作）**

读取图片：

def read\_im\_and\_landmarks(fname):

im = cv2.imread(fname, cv2.IMREAD\_COLOR)

im = cv2.resize(im, (im.shape[1] \* SCALE\_FACTOR,

im.shape[0] \* SCALE\_FACTOR))

s = get\_landmarks(im)

return im, s

已经训练好的模型路径：（[下载](http://sourceforge.net/projects/dclib/files/dlib/v18.10/shape_predictor_68_face_landmarks.dat.bz2)）

PREDICTOR\_PATH = "shape\_predictor\_68\_face\_landmarks.dat"

人脸检测器：

detector = dlib.get\_frontal\_face\_detector()

预测器：

predictor = dlib.shape\_predictor(PREDICTOR\_PATH)

预测大致人脸：

预测器需要粗略的边界框作为算法的输入，这是由检测器提供的，该检测器返回矩形列表，每个矩形对应图像中的面部，代码如下：

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()])

**2.用 Procrus**[**te**](http://www.elecfans.com/tags/te/)**s 分析调整脸部：**

现在我们已经有了两个标记矩阵，每行有一组坐标对应一个特定的面部特征（如第30行的坐标对应于鼻头）。我们现在要解决如何旋转、翻译和缩放第一个向量，使它们尽可能适配第二个向量的点。一个想法是可以用相同的变换在第一个图像上覆盖第二个图像**，**其实最终是一个正交矩阵的解决办法，代码如下：（[参考](https://en.wikipedia.org/wiki/Orthogonal_Procrustes_problem)文档，维基百科）

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.])])

代码实现了这几步：

1.将输入矩阵转换为浮点数。这是后续操作的基础。

2.每一个点集减去它的矩心。一旦为点集找到了一个最佳的缩放和旋转方法，这两个矩心 c1 和 c2 就可以用来找到完整的解决方案。

3.同样，每一个点集除以它的标准偏差。这会消除组件缩放偏差的问题。

4.使用奇异值分解计算旋转部分。可以在维基百科上看到关于解决正交 Procrustes 问题的细节。

5.利用仿射变换矩阵返回完整的转化。

**3、色彩校正**

两幅图像之间不同的肤色和光线造成了覆盖区域的边缘不连续，若无此步，则制作的图片色彩不均匀。

此函数试图改变 im2（第二张图） 的颜色来适配 im1。它通过用 im2 除以 im2 的高斯模糊值，然后乘以im1的高斯模糊值。代码如下：

def correct\_colors(im1, im2, landmarks1,landmarks2): #修改

blur\_amount = COLOUR\_CORRECT\_BLUR\_FRAC \* numpy.linalg.norm(

numpy.mean(landmarks1[LEFT\_EYE\_POINTS], axis=0) -

numpy.mean(landmarks2[RIGHT\_EYE\_POINTS], axis=0))

blur\_amount = int(blur\_amount)

if blur\_amount % 2 == 0:

blur\_amount += 1

im1\_blur = cv2.GaussianBlur(im1, (blur\_amount, blur\_amount), 0)

im2\_blur = cv2.GaussianBlur(im2, (blur\_amount, blur\_amount), 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))

**4、第二张图特征混合在第一张图**

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, (FEATURE\_AMOUNT, FEATURE\_AMOUNT), 0) > 0) \* 1.0

im = cv2.GaussianBlur(im, (FEATURE\_AMOUNT, FEATURE\_AMOUNT), 0)

return im

get\_face\_mask()的定义是为一张图像和一个标记矩阵生成一个遮罩，它画出了两个白色的凸多边形：一个是眼睛周围的区域，一个是鼻子和嘴部周围的区域。之后它由11个像素向遮罩的边缘外部羽化扩展，可以帮助隐藏任何不连续的区域。最终返回优化过后的图像。

其余的py算法较为简单，主要为opencv和os的使用，在此不再赘述。

最终我是使用的main.py去循环读取图片并且将其更改，最终合并视频删除缓存，大致流程和思想就是这样了

**总结：**

通过此次实验，我知道了dlib库的详细使用，但其中还有许多许多的不足之处和不理解之处，在以后的学习中还需要逐步完善。而且合并的视频也并不理想，还有很多可拓展的空间，联想到最近实训的机器学习，我就在想如果通过机器学习去识别更多的图片，训练更好的模型，会不会更加准确呢？ 但是作业就只能到此为止了，再次谢谢老师的阅读和教导！