数字图像处理 大作业报告

Introduction

本次大作业选择的论文是*Face Swapping: Automatically Replacing Faces in Photographs*，来自SIGGRAPH 08'。这篇论文的目标是实现人脸的“无缝”替换，但并不是替换成指定的目标人脸，而是在一个很大的图片库中进行筛选，选择“最适合替换”的人脸来进行替换。因此，这篇文章可以分为两部分，第一部分就是对图片库的建立和人脸筛选，第二部分是把人脸替换以后的后处理方法。因为这篇文章使用到很多比较复杂的图像处理技巧，尤其是需要用到复杂的公式，这些部分因为自己的能力不足，并没有用原始的方法实现，而是采用了实现简单，但是效果差一些的方法来进行替换。

下面我按照图片库的建立，人脸替换过程，图片库中人脸排序，以及后处理的过程来对自己的实现进行介绍。

图片库的建立

因为我们的算法并没有对人脸采用3D模型的重建，因此控制**被替换人脸**和**替换人脸**之间的**角度关系**很重要。一张脸的角度可以用三个参数来表示：yaw，pitch和roll，其中yaw度量了脸的左右偏转的程度，pitch衡量的是抬头或者低头的程度，roll则是在人脸平面上的转动的衡量。如果我们用一张yaw和pitch相同的人脸来替换我们的目标人脸，那么在替换过程中，只需要在平面上进行偏转(in-plane)，操作起来很简单。

根据论文中的技术，我们这里把所有的人脸按照yaw和pitch两个维度来进行划分。这里pitch取-15到15之间，每个区间取10；yaw取-25到25之间，每个区间取10。因此我们现在在yaw上有5个区间，在pitch上有3个区间，因此我们把所有的图片库中的人脸划分成15个不同的**角度分类**(以下用posebin来指称)。

我的代码的大部分工作主要是集中在这个部分，下面我对建立流程进行一个概述：

1. 图片库建立的入口在 `facetrain.py` 中。该程序接受两个参数，一个是图片库的根目录，一个是把图片库导入的目标posebin的位置。我这里用到的图片库来自于LFW(Labeled Face in the Wild) Face Database，选用这个图片库的原因主要是该图片库提供很详细的人名标记数据，以及对齐过的人脸照片。对齐这个特性在之后替换的过程中很管用。
2. 程序首先遍历图片库的根目录，得到所有图片的路径。之后遍历这个列表，依次

处理每张人脸照片。处理的过程大致如下：

1. 用Face++人脸识别服务获得yaw，pitch，roll这几个角度(pose)参数，根据yaw和pitch来决定应该放在哪个posebin中
2. 同时获得人脸的中心位置和人脸的范围，这里直接把人脸截取保存下来
3. 最后也会获得一些人脸上的landmark的位置，比如人脸的轮廓，眼睛的轮廓等

到此位置图片库建立完成了，在指定的posebin路径下出现了15个posebin，以及一个编号为0的不满足条件的posebin。在posebin的目录下也保存有一个所有人脸的相关数据的数据文件。

这一部分的代码主要有(在 `facelib/` 下)：

1. `facetrain.py`：程序的入口
2. `faceposebin.py`：与posebin有关的处理
3. `faceinfo.py`：对脸部信息的转换
4. `faceproc.py`：对人脸的处理，主要是截取，保存到指定的posebin等等
5. `faceset.py`：对图片库的抽象，主要函数就是提取这个图片库中所有的图片路径

人脸的替换

这里我们称**替换人脸**为从图片库中选出来的，换到我们的目标图片上的人脸；称**目标人脸**为要被替换的人脸。在不考虑替换人脸的选择、排序和后处理等机制时，我们先讨论一下如何把一张图片库中的脸换到指定的目标人脸的方法。第一步首先要把替换人脸提取出来，这里采用的方法如下：

1. **获得轮廓**：调用Face++的API，获得人脸的轮廓。注意这里并不是要获得整张脸的轮廓，而是包含关键的面部器官的区域。因此我们选择的轮廓线是从眉毛上方出发，向下到眼角的外侧，再到下嘴唇的外侧。
2. **修改轮廓**：考虑到Face++返回的landmark都是跟面部器官紧紧贴合的，这样在之后匹配时，如果目标人脸的器官过大(-_-)，比如眼睛很大等等，就可能覆盖不上。所以这里我们会对landmark的位置进行一下修改，方法是：从鼻子的位置出发，往这些landmark连线，然后对沿着这根线的角度扩展一下距离。这样得到的新的轮廓线可以保证是包裹在原先的轮廓线之上的。
3. **提取替换人脸**：提取的过程依然是基于轮廓线的。我们这里用的是Python OpenCV提供的库：`cv2.pointPolygonTest` 可以获得一个点跟轮廓线的距离，如果为负的话则表示在轮廓线之外。根据这个判断条件，我们把判定为负的像素点都改为0
4. **仿射变换**：尽管我们已经得到了替换人脸，但是目标人脸跟替换人脸之间可能有如下两种不匹配的情况：**大小不相等**和**旋转角度不相等**，而这两种不匹配都可以用仿射变换(affine transform)来解决。要想进行仿射变换，首先是要得到

变换矩阵。这个变换矩阵可以用三个从原图到目标图的点对，来解一个线性方程组得到。我们选用的点对是 `左眼` 、 `右眼` 和 `鼻子` 的位置。求转换矩阵的过程可以用OpenCV的 `warpAffine` 函数，执行affine transform可以用OpenCV的 `getAffineTransform` 函数。(之前我自己实现过一个affine transform的版本，但是因为效果不好放弃了……)

5. **边界处理**：边界的feathering比我想象中更加重要，加上这个效果之后原先许多看起来不和谐的结构都变和谐了。具体的做法就是通过在脸部的轮廓周围d0的范围内，对人脸的边界进行融合，具体的公式是：

$$I_{\text{结果}} = \alpha I_{\text{目标人脸}} + (1-\alpha) I_{\text{替换人脸}}$$

> 注意，这个边界处理的办法其实在后面放弃掉了，转而使用OpenCV3.0的新特性——Seamless Clone，具体细节请见最后一节

这部分的代码放在 `facerepl/` 目录下：

1. `affine.py` ：是自己实现的仿射变换函数
2. `contour.py` ：提取轮廓线，以及脸部区域图片
3. `facealign.py` ：得到替换的结果，中间使用了 `contour.py` 的函数，以及仿射变换
4. `faceblending.py` ：这是使用feathering的方法得到替换结果的函数，效果比 `facealign.py` 好很多

候选人脸的排序

由于之后的后处理用到的Seamless Clone实在是太强大，这里并不需要对除了**位置**、**角度**之外的其他因素进行排序了。这两个指标直接用yaw，pitch和roll匹配即可。至于论文中提到的resolution的问题，我这里处理的办法跟文章中一模一样，就是计算两只眼睛之间的像素距离，距离越大那么分辨率越高。

代码是 `facerank/facerank.py`

后处理

由于后处理需要用到各种奇怪的技巧，而且需要理解大量的难懂数学公式，更关键的是没有一些方便的API可供调用，这样自己实现的成本太高而且很可能实现不出来(比如affine transform就是一次失败的尝试，里面有些细节不是自己实现就能发现的)。

不过相对于使用论文中提供的球谐函数，我这里有一个非常简便而有效的办法，就是使用Poisson Image Editing里面提到的方法：Seamless cloning。这个方法的本质是，使用梯度而不是RGB值来对图像进行融合。融合的结果几乎看不出来二者

的边界。这个方法对我们这个应用简直完美：

1. 首先，在边界处理那一步中，不用简单的带alpha的mask了，而是直接生成一个0-1mask
2. 把这个0-1mask与目标人脸和替换人脸一起作为参数传给 `cv2.seamlessClone`

得到的结果可以看出来，不仅是没有了边界的问题，连色差都一起解决了。

> 其实我觉得这个甚至可以作为对论文方法的改进……

在 `facerepl/facecloning.py` 中实现了这个方法。

> 在 `facerepl/facehistogram.py` 中实现了用直方图匹配方法调整颜色的函数。尽管效果并不好……

代码的使用

这次大作业的实现分成两个包，一个是附件中的代码包，一个是保存posebin的数据包(运行时请把数据包解压到代码包下)。

使用的时候直接可以用 `faceswap.py` 脚本进行测试：

```
python faceswap.py <image>
```

会在本地目录下生成六张替换后的图片

测试结果

我们的方法毕竟是原始论文的简化版，所以说并不能适用于所有的情况，但是还是有些不错的结果。结果在代码目录下的一些tmp文件中。