哈爾濱工業大學

数字图像处理实验报告

实验 （三）

题　　目	Experiment 3
学　　院	计算机学院
专　　业	计算机科学与技术
学　　号	1160300909
学　　生	张志路
任课教师	姚鸿勋

哈尔滨工业大学计算机科学与技术学院

2018 年秋季

一、 实验内容

1. 综合运用图像处理的知识解决实际问题。
2. 了解颜色空间在图像处理中的用途。
3. 了解常见滤波在图像处理中的用途。
4. 了解图像修复（Image Inpainting）的常见方法。

注：仓库地址为 https://gitlab.com/1160300909/digital-image-processing-experiments

二、 实验目的

以下 2 个任务从中选做 1 个。

1. 去除图像中的背景，去除图像中的水印。
2. 去除人脸图像中的雀斑。（本实验）

三、 实验环境

1. 编程语言：python 3.6.4
2. 编程工具：PyCharm 5.0.3、Anaconda3-5.1.0
3. 操作系统：Windows 10 中文版 64-bit

四、 实验设计、算法和流程

4.1 整体流程

本实验的要求是去除人脸图像中的雀斑，根据所给图片的前后对照，我的理解是对图片进行“美颜处理”。为了增强算法的普适性，分五步对图片进行处理，如下所述。

1. 用具有保边效果的滤波算法对图像进行模糊处理。
2. 用肤色检测算法保护非皮肤区域。
3. 将第一步模糊后的图像和肤色外的背景进行图像融合。
4. 对融合后的图像进行美白处理。
5. 对美白后的图像进行锐化处理。

下面将详细叙述这五个步骤。

4.2 保边滤波器磨皮

1. 滤波器

磨皮算法是该美颜处理实验中最重要的一步，而在磨皮算法中，最重要的就是滤波器的选择，下面对滤波器的选择和导向滤波做简要介绍。

(1) 滤波器的选择

磨皮算法的实质是对图像进行平滑处理，在磨皮算法中，最重要的一步是对图像进行滤波处理，滤波算法可以选择高斯滤波、双边滤波、导向滤波等，只要能保边缘就行。但实际上，不同滤波方法对磨皮效果影响较大。

如果单单使用高斯模糊来磨皮，得到的效果是不尽人意的。原因在于，高斯模糊只考虑了像素间的距离关系，没有考虑到像素值本身之间的差异。举个例子来讲，头发与人脸分界处，如果采用高斯模糊则这个边缘也会模糊掉，这显然不是我们希望看到的。

与高斯滤波不同的是，双边滤波则考虑到了颜色的差异。它的像素点取值也是周边像素点的加权平均，而且权重也是高斯函数。不同的是，这个权重不仅与像素间距离有关，还与像素值本身的差异有关，具体讲是，像素值差异越小，权重越大，也是这个特性让它具有了保持边缘的特性，因此它是一个很好的磨皮工具。

而与双边滤波相比，导向滤波更具有优势。导向滤波不仅能实现双边滤波的边缘平滑，而且在检测到边缘附近有很好的表现，在一些细节上，引导滤波较好。此外，引导滤波最大的优势在于，可以写出与窗口大小无关的时间复杂度为的算法，因此在使用大窗口处理图片时，其效率更高。

而实际上，经过本实验也证明，利用导向滤波明显能得到比双边滤波更好的效果。下面简要介绍一下导向滤波。

(2) 导向滤波

a. 简介

导向滤波是何凯明在学生时代提出的一个保边滤波算法，最初来自于何明凯的图像去雾算法中。在 2015 年，何凯明又发表了一篇《Fast Guided Filter》的文章，阐述了一种更快速的导向滤波流程。这个算法速度比其他的保边缘磨皮算法都快，甚至快上好几倍。现在已经被应用封装 opencv 库中，可见算法堪称经典。

b. 基本原理

顾名思义，导向滤波就是有选择性的滤波，其与我们经常提及的高斯滤波、双边滤波相比，它具有导向性。

导向滤波通过一张引导图，对初始图像 p（输入图像）进行滤波处理，使得最后的输出图像大体上与初始图像 p 相似，但是纹理部分与引导图相似。具体来讲，它通过输入一副图像作为导向图，这样滤波器就知道什么地方是边缘，这样就可以更好的保护边缘，最终达到在滤波的同时，保持边缘细节。

c. 算法

下面给出导向滤波的一般算法。

其中，p 为输入图，I 为引导图，f_{mean} 为一个均值滤波器，参数ε为平滑项参数，滤波半径为 r。

另外，从算法中公式也可看出，*var* 为自相关方差，*cov* 为互相关协方差。

1: $\text{mean}_I = f_{\text{mean}}(I, r)$
$\text{mean}_p = f_{\text{mean}}(p, r)$
$\text{corr}_I = f_{\text{mean}}(I. * I, r)$
$\text{corr}_{Ip} = f_{\text{mean}}(I. * p, r)$

2: $\text{var}_I = \text{corr}_I - \text{mean}_I. * \text{mean}_I$
$\text{cov}_{Ip} = \text{corr}_{Ip} - \text{mean}_I. * \text{mean}_p$

3: $a = \text{cov}_{Ip} ./ (\text{var}_I + \epsilon)$
$b = \text{mean}_p - a. * \text{mean}_I$

4: $\text{mean}_a = f_{\text{mean}}(a, r)$
$\text{mean}_b = f_{\text{mean}}(b, r)$

5: $q = \text{mean}_a. * I + \text{mean}_b$

算法的时间复杂度为 O(N)。

d. 应用

导向滤波作为一种保边滤波，不仅能实现双边滤波的边缘平滑，而且在检测到边缘附近有很好的表现，可应用在图像增强、HDR 压缩、图像抠图、美颜以及图像去雾等场景。

2. 磨皮算法描述

假设原始图像层为 src，磨皮算法描述如下。

(1) 复制图层，复制 src 到 img。

(2) 对 img 进行滤波处理。滤波器可以选择表面模糊、导向滤波、双边滤波、各向异性扩散、BEEP、局部均方差等任何具有保边效果的滤波器，在这里我们选择效果较好的导向滤波。滤波后得到图像 temp1。

(3) 应用图像，令 temp2 = temp1- Src + 128。这个和高反差保留的算法是一模一样的，只是 Photoshop 内嵌的高反差保留用的是高斯模糊，这里用的是导向滤波而已。

(4) 对 temp2 进行高斯滤波得到 temp3。高斯模糊本质上是低通滤波器，这里使用高斯模糊以减少图像的高频分量。

(5) 进行图层混合。首先进行线性光混合，计算公式为 temp4 = img + 2*temp3 – 256。再进行透明度的处理，计算公式为 dst = (img*(100 - p) + temp4*p)/100。

确定各个步骤的参数很重要，下面介绍一下参数问题。

对于第二步的导向滤波，其原理我们前面已经介绍，不再赘言。在这里经过不断的实验，滤波半径取 3 时较好，滤波的正则化项取 125 时较好。

第三步应用图像中常数 128，实际上也不一定是个定值，如果把他调大，则处理后的图像整体偏亮，调小则图像偏暗。

第四步的高斯模糊用以恢复皮肤质感，这个模糊的半径一般越大，质感越强，但是太大，磨皮效果就没有了。高斯模糊半径在 0.5-2 之间比较合适，我们这里取半径为 1。

第五步的图层的透明度参数也是一个道理，如果透明度值偏小，则图片整体的斑点可能会偏多；透明度值偏大，那么图像又会过于模糊。在实际实验中，我们省略了这一步，即透明度参数为 100。

3. 代码

磨皮算法代码如下。

```
# 磨皮算法
def dermabrasion(ori_img):
    Gaussia_r = 1  # 高斯滤波半径
    guided_r = 3  # 导向滤波半径
    guided_eps = 125  # 导向滤波的正则化项，类似于 bilateralFilter 中的 sigma
    # p = 80  # 透明度
    img = ori_img.copy()

    temp1 = cv2.ximgproc.guidedFilter(img, img, guided_r, guided_eps)  # 导向滤波
    # dst = cv.ximgproc.guidedFilter(guide, src, radius, eps[, dst[, dDepth]])
    temp2 = (np.uint32(temp1) - np.uint32(img) + 128)  # 高反差保留，应用图像
    temp2 = np.uint8(np.clip((temp2), 0, 255))  # 类型转换，截断
    temp3 = cv2.GaussianBlur(temp2, (Gaussia_r, Gaussia_r), 0)  #高斯模糊

    temp4 = np.uint32(img) + 2 * np.uint32(temp3) - 256  # 线性光混合
    # dst = (np.uint32(img)*(100 - p) + temp4*p) / 100  # 透明度
    return np.uint8(np.clip(temp4, 0, 255))
```

在实际调用时，该算法迭代了三次。

4.3 肤色检测

1. 肤色检测方法介绍

为了使图片中肤色外的景象在处理时尽量不受影响，我们有必要进行肤色检测，以便仅对肤色部分进行处理。

下面介绍几种常用的肤色检测方法。

(1) 基于 RGB 色彩空间

肤色在 RGB 模型下的范围基本满足以下约束.

在均匀光照下应满足以下判别式：

r>95 and g>40 and b>20 and max(r,g,b)-min(r,g,b)>15 and r-g>15 and r>b

在侧光拍摄环境下：

r>220 and g>210 and b>170 and abs(r-g)<=15 and r>b and g>b

上述两个公式取并集，得到的集合即认为是皮肤区域，其范围外的图像设为黑色。

(2) 基于椭圆皮肤模型

如果将皮肤信息映射到 YCrCb 空间，则在 CrCb 二维空间中这些皮肤像素点近似成一个椭圆分布。因此如果我们得到了一个 CrCb 的椭圆，下次来一个坐标(Cr,Cb)我们只需判断它是否在椭圆内（包括边界），如果是，则可以判断其为皮肤，否则就是非皮肤像素点。

(3) 基于 YCrCb 颜色空间

YCrCb，其中“Y”表示明亮度（Luminance），也就是灰阶值；而“U”和“V” 表示的则是色度（Chroma），作用是描述影像色彩及饱和度，用于指定像素的颜色。“亮度”是透过 RGB 输入信号来建立的，方法是将 RGB 信号的特定部分叠加到一起。“色度”则定义了颜色的两个方面—色调与饱和度，分别用 Cr 和 Cb 来表示。其中，Cr 反映了 RGB 输入信号红色部分与 RGB 信号亮度值之间的差异。而 Cb 反映的是 RGB 输入信号蓝色部分与 RGB 信号亮度值之间的差异。

这个方法与方法（1）其实大同小异，只是颜色空间不同而已。资料显示，正常黄种人的 Cr 分量大约在 133 至 173 之间，Cb 分量大约在 77 至 127 之间。可以根据此范围确定肤色区域。

(4) 基于 HSV 颜色空间

同样地，也是在不同的颜色空间下采取相应的颜色范围将皮肤分割出来。其中 H 分量大约在 7 至 20 之间，S 分量大约在 28 至 255 之间，V 分量大约在 50 至 255 之间。

(5) 基于 YCrCb 颜色空间和 Otsu 法阈值分割

该方法主要分两步，一是将 RGB 图像转换到 YCrCb 颜色空间，提取 Cr 分量图像；二是对 Cr 做自二值化阈值分割处理（Otsu 法）。

2. 肤色检测算法描述

为了尽可能准确的检测出皮肤区域，本实验中的算法采用基于 YCrCb 颜色空间和基于 HSV 颜色空间两种方法的联合，即两种方法中只要有一种检测出某区域为皮肤，则就认定该区域为皮肤。

此外在上述处理结束后，再进行一次开运算，即先腐蚀后膨胀，以消除皮肤外的无关小区域。

算法描述如下。

(1) RGB 颜色空间分别转换为 YCrCb 颜色空间和 HSV 颜色空间。

(2) 对每个像素点，如果其对应的 Cr、Cb、H、S、V 五个分量都满足肤色认定条件，则认定该像素点为肤色。

(3) 肤色外的区域置为黑色。

(4) 进行开运算。

对于上述算法的参数选择问题，也是经过不断的实验选择出较优的参数，以更准确地检测出所有皮肤区域。具体参数可见如下代码。

3. 代码

肤色检测算法代码如下。

```
# 肤色检测
def skin_detect(img):
    len1, len2, len3 = img.shape
    result = np.zeros(img.shape, np.uint8)
    hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
    ycrcb = cv2.cvtColor(img, cv2.COLOR_BGR2YCrCb)

    if len1 == 0 or len3 != 3:
        return result

    for i in range(len1):
        for j in range(len2):
            h = hsv[i, j, 0]
            s = hsv[i, j, 1]
            v = hsv[i, j, 2]
            cr = ycrcb[i, j, 1]
            cb = ycrcb[i, j, 2]
            if 128 <= cr <= 173 and 77 <= cb <= 122 and 0 <= h <= 35 and s >= 23
and v >= 90:
                result[i, j, 0] = img[i, j, 0]
                result[i, j, 1] = img[i, j, 1]
                result[i, j, 2] = img[i, j, 2]

    kernel = np.ones((15, 15), np.uint8)
    result = cv2.morphologyEx(result, cv2.MORPH_OPEN, kernel)
    return result
```

4.4 图像融合

此部分的目的主要是合成肤色检测后的和经滤波处理后的图像，得到一张具有较强背景真实感和肤色磨皮效果的结果图。此部分比较简单。

1. 图像融合算法描述

(1) 经滤波器处理后的图像为 pro_img，肤色检测后的图像为 det_img，原始图像为 ori_img。

(2) 对于每一个像素点，如果对应 det_img 像素点处的 RGB 值都为 0，则将对应 ori_img 像素点值赋值给对应 pro_img 的像素点。

(3) 最后返回 pro_img。

2. 代码

图像融合代码如下。

```
# 合成肤色检测后的和经滤波处理后的图像
# pro_img：经滤波器处理后的图像
# det_img：肤色检测后后的图像
# ori_img：原始图像
def synthesis(pro_img, det_img, ori_img):
    for i in range(pro_img.shape[0]):
        for j in range(pro_img.shape[1]):
            if det_img[i, j, 0] == det_img[i, j, 1] == det_img[i, j, 2] == 0:
                pro_img[i, j] = ori_img[i, j]
    return pro_img
```

4.5 美白

1. 美白方法介绍

美白其实主要使皮肤变白变亮，因此，如果能有个映射，使得原图在色阶上有所增强，并且在亮度两端增强的稍弱，中间稍强，则可作为一个简单的美白方法。

在图像增强论文《A Two-Stage Contrast Enhancement Algorithm for Digital Images》中提出一个公式，非常适合上述情况。公式如下。

$$v(x,y) = \frac{log(w(x,y)*(\beta-1)/255+1)}{log\,\beta} * 255$$

其中 w(x,y)表示输入图像数据，v(x,y)表示输出结果，β 为调节参数。β 分别为 2、3、4、5 时的曲线，如下图 4.5.1 所示。

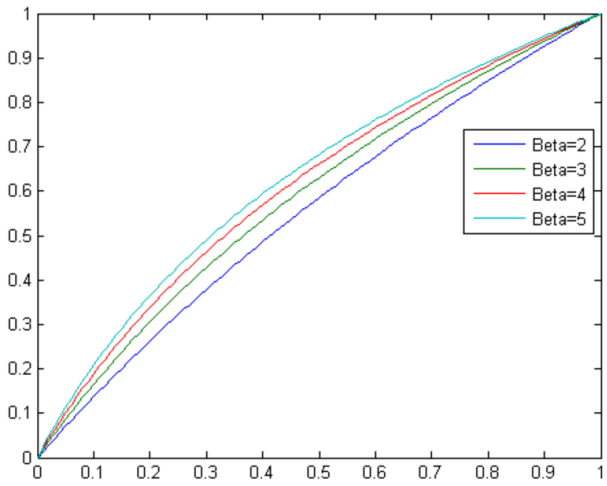

图 4.5.1

可见，β 越大，美白的程度越强。

在本实验中，经过反复验证，β 取 1.8 时效果较好。

2. 代码

由于此美白算法比较简单，下面直接给出代码。

```
# 美白
def whiten(img):
    beta = 1.8
    white = np.log(img / 255 * (beta - 1) + 1) / np.log(beta)
    white = np.uint8(np.clip(white * 255, 0, 255))
    return white
```

4.6 锐化

1. 锐化方法介绍

在得到磨皮融合的效果图后，我们还需要进行一定的锐化算法，来进一步增强细节感，强化边缘，让图像看起来更清晰。经典的锐化方法有 Robert 算子、Sobel 算子、拉普拉斯算子等等，这里我们采用 Sobel 算子。

Sobel 算子是一种一阶微分算子，它能根据像素点周围的梯度值来计算该像素点的梯度，之后会根据事先设好的阈值比较，来进行取舍。Sobel 算子是 3×3 算子模板，该算子包含两组 3×3 的矩阵，分别为横向及纵向，将之与图像作平面卷积，即可分别得出横向及纵向的亮度差分近似值。

如果以 A 代表原始图像，G_x 及 G_y 分别代表经横向及纵向边缘检测的图像，则结果 G 可表示如下。

$$\text{可令 } G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} * A \ ， \ G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} * A$$

$$G = \sqrt{{G_x}^2 + {G_y}^2}$$

2. 锐化算法描述

(1) 设原图像为 img，将原图像复制到 Sobel。

(2) 对第 i 行 j 列像素点的第 k 个通道 img [i][j][k]，有 Sobel [i][j][k] = G（如上所述 G）。

(3) Sobel[Sobel<=threshold] = 0。

(4) 计算 out = weight * Sobel + img，其中 weight 为锐化权重。

(5) 返回 out。

本实验中取 threshold = 50，weight = 0.13。

3. 代码

锐化算法代码如下。

```
# 利用Sobel 算子锐化
def sobel_sharpen(img):
    weight = 0.13  # 锐化参数
    len1, len2, len3 = img.shape
    image_out = img.copy()

    for i in range(1, len1 - 1):
        for j in range(1, len2 - 1):
            for k in range(len3):
                a = int(img[i - 1, j - 1, k] + 2 * img[i, j - 1, k] +
                        img[i + 1, j - 1, k])
                b = int(img[i - 1, j + 1, k] + 2 * img[i, j + 1, k] +
                        img[i + 1, j + 1, k])
                c = int(img[i + 1, j - 1, k] + 2 * img[i + 1, j, k] +
                        img[i + 1, j + 1, k])
                d = int(img[i - 1, j - 1, k] + 2 * img[i - 1, j, k] +
                        img[i - 1, j + 1, k])
                image_out[i][j][k] = np.sqrt(
                    np.power(a - b, 2) + np.power(c - d, 2))

    threshold = 50  # 阈值
    image_out[image_out <= threshold] = 0
    image_out = np.uint32(weight * image_out) + np.uint32(img)
    image_out = np.uint8(np.clip(image_out, 0, 255))
    return image_out
```

五、 实验结果

如下图 5.1.1，为实验说明中所给定的原图和目标图；下图 5.1.2，为经程序处理后的结果。

图 5.1.1

图 5.1.2

如下图 5.2.1，为从网上找到带有雀斑的图；下图 5.2.2，为原图经该程序直接处理后的结果。

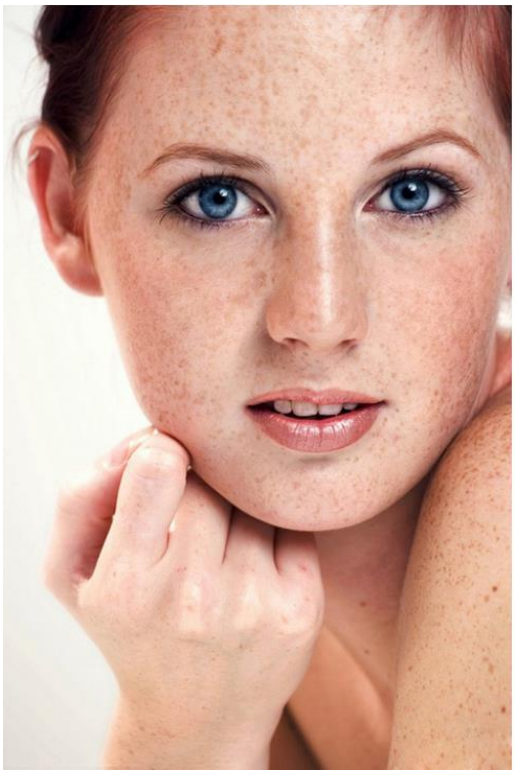

图 5.2.1

图 5.2.2

如下图 5.3.1，为从网上找到的原图；下图 5.3.2，为原图经该程序直接处理后的结果。

图 5.3.1

图 5.3.2

此外，程序还做了许多测试，测试结果大部分较好，不再一一展示。

六、 结论

1. 实验结果与目标图像吻合度较高，该“美颜”算法具有一定的普适度，但也存在一些问题。
2. 本实验中的关键步骤是滤波器的选择和皮肤检测，滤波器的选择问题在前面已经详细介绍，下面简要说一下肤色检测结果对结果的影响。肤色检测的准确度直接影响着最后图像的真实感和质感。对于眉毛、头发以及皮肤边缘部分，这些部分往往在肤色检测时不能很好地确定，这样就会导致结果的真实感不强，某些皮肤边缘存在瑕疵。
3. 关于整体流程问题。理论上处理时应该是先对皮肤部分美白，再与背景进行融合。然而这个流程会导致融合后的图像皮肤与非皮肤部分的交界处有明显的合成迹象，真实感不强。因此本实验中采用的是先融合后美白，这个流程的效果明显会比先美白后融合要好，但却会导致背景也偏白。
4. 关于实验改进。从最后结果也可以看出，本实验中已经对磨皮和美白算法取得了很好的效果，但皮肤检测和图像融合算法仍有改进之处，这两部分的改进也直接影响着结果的质感。关于皮肤检测算法，经实验，一般性的算法很难完全检测出皮肤区域，容易产生误识，且普适度不高。下一步的想法是采用深度学习方法，以更好的检测皮肤区域。

七、 体会

1. 本次实验可以说是较好的达到了实验目的，完成了实验内容。
2. 实现磨皮和皮肤检测算法时，我查阅了许多资料，让我对整个流程以及具体算法有了基本的思路，但是网上直接给出算法大都“美颜”效果不太好。于是我又查阅了几篇论文，对一些算法进行了优化（例如从双边滤波到导向滤波的转变），才使得效果有了较大提升。这让我感到实验真正的意义，就是不断探索，不断尝试，不断追求完美的结果。
3. 图像处理是一件有趣的事，把有趣的事做好是一件美好的事。

八、 参考文献

[1] Kaiming He, Jian Sun, Xiaoou Tang. Guided image filtering[J]. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2013.

[2] Shenchuan Tai, Naiching Wang, Yiying Chang. A Two-Stage Contrast Enhancement Algorithm for Digital Images[J]. Image and Signal Processing, 2008.

[3] 陈锻生,刘政凯.肤色检测技术综述[J].计算机学报,2006(02):194-207.

[4] 王微. 基于三通道协同调节和细节平滑的自动人脸美化[D].天津大学,2013.

[5] 黄明杨. 图像去模糊技术及相关图像增强系统[D].北京邮电大学,2017.

[6] 徐舒畅. 基于色素分离的皮肤图像处理与分析[D].浙江大学,2007.

[7] Kaiming He, Jian Sun. Fast Guided Filter. 2015.

[8] 冈萨雷斯, 伍兹. 数字图像信号处理[M]. 北京：电子工业出版社，2011：249-267.

[9] 简单探讨可牛影像软件中具有肤质保留功能的磨皮算法及其实现细节[EB/OL]. [2018-12-18]. http://www.cnblogs.com/Imageshop/p/4709710.html.