数字图像处理第四次实验对比度增强报告（文末附代码）

1 问题

应用图像对比度增强算法提高 DR 图像的对比度

2 问题分析

本次实验要求设计算法提高DR图像的对比度，根据要求建议我们使用 re tinex 算法。我采用retinex算法的SSR算法。该算法的核心原理如下：

该算法构建的成像模型为： R1 = R \* L + F 其中R1为观测到的图像，R为原始图像,L为光照，F是噪声图像，在不考虑噪声的基础上，成像模型变为 R1 =R\* L，因此该算法的核心为如何通过R1估计出R。

在对数域下进行计算， 两边同时取对数， 有

L(R1) = L(R)+L(L) （1）

(L表示取对数)。但是L我们是不知道的。因此采用以下的方法来估计L

L = GuassFilter( R1) （2）

通过对观测图像 R1 进行高斯滤波，高斯滤波的结果就为L ，将估计的L代入（1）式，就可以得到 L(R)。接下来的步骤很关键。由于该算法一开始用在图像去雾上的，在得到L(R)，之后 不是进行指数反变换，而是直接对 L(R)进行线性拉伸。拉伸的结果就是恢复后的R。

对于高斯滤波来说， 高斯滤波核满足：

F(x,y) = lambda \* e ^ ( -(x^2 + y^2)/c^2)

且F(x,y)的 求和值为一。

因此本算法的参数主要就是c 和 高斯滤波的滤波核大小

一开始在没有得到待处理图像的时候，我主要用这个算法进行了去雾，滤波核参数选区了将整张图片大小作为滤波核的大小。C取 80 – 300。去雾得到了效果。

但是当我得到待处理的胸片之后发现上述算法完全不起作用。我尝试用之前学过的garma校正来进行对比度增强，效果差强人意。经过思考原有的retinex算法，我发现问题出在线性拉伸的步骤。对于去雾任务来说，其原图是被“雾笼罩”的图片，通过对原图进行高斯滤波，我们得到的事实上是“雾”（原始图像的低频部分），在对数域将雾去掉得到部分原始图像的细节（少量细节），也就是高频部分。这时候为了从少量的细节还原出原图，我们需要直接用线性拉伸，从细节还原出原图。而当将retinex算法用在图像对比度增强的时候这个算法就失灵了。因为现在原图像经过高斯滤波之后得到的是细节减弱的图像，它是图像的低频部分包含了图像大部分的信息。做减法得到图像的高频部分，但这些高频部分只是图像的细节（其包含的图像信息较少），直接用这些细节，用线性拉伸重建原图自然得到的效果很不好。因此我对原本的retinex算法做了修正：

在（2）之后

L(R2) = L(R1) – L(L)

L(R) = L(R1) + alpha \* L(R2)

R = e ^(L(R))

修正后的算事实上是在对数域上用 观测减去滤波后的图得到细节，再在对数域上将细节叠加到对数域上的观测图后增强细节，最后用e反变换得到原图。做的工作实质上是在对数域进行细节增强！通过这种方式，最后得到了很好的效果。

本次实验步骤如下： 1、用上述“原始” retinex ssr算法 做图像去雾

2、用上述“原始” retinex ssr算法 做图像增强

3、用garmma矫正算法做图像增强

4 用改进后的 retinex算法 做图像增强

3 实验结果：

图像去雾：



Retinex算法 ： 滤波核大小为整张图片 c =300



用retinex 算法进行图像对比度增强， 滤波核大小同上，c =80



用原始retinex 算法进行图像对比度增强， 滤波核大小同上，c =300



用原始retinex 算法进行图像对比度增强， 滤波核大小同上，c =10



用 garmma校正 c =1 gamma =2



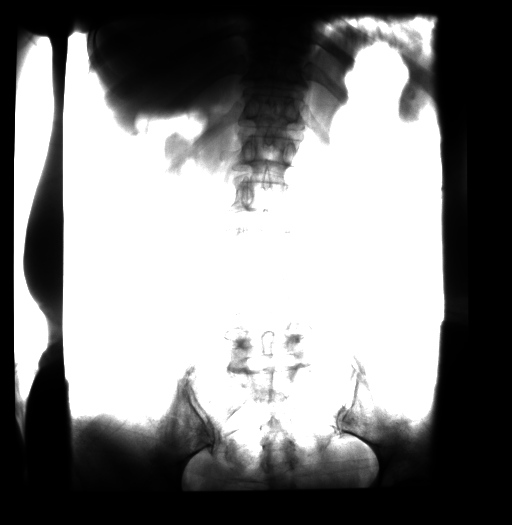
用改进后的retinex算法，滤波核大学为整张图片，c =10 ， alpha = 5



用改进后的retinex算法，滤波核大学为整张图片，c =80 alpha = 5



用改进后的retinex算法，滤波核大学为整张图片，c =300 alpha = 5



4 实验结果

分析实验结果，如同我在2部分所说的直接用原始的retinex算法效果极差，原始的retinex算法最后的线性拉伸处理部分比较适用于图像去雾，在图像去雾上使用原始的retinex算法确实能够取得一定的效果，但如果要用在对一幅信息丰富的、相对清晰的图形（如本实验的DR图片）进行相同的处理就会丢失大量低频信息，从而导致结果极差。而改进后的在对数域进行细节加强，取得了很好的结果，其本质的原理就是加强原图在细节部分的光照（因为在对数域做加法就是在空间域做乘法）从而达到了增强对比度的效果。而使用gamma校正的效果差强人意是因为gamma校正是对在全局上进行像素的压缩，有针对性的增强图像细节部分的对比度。所以效果没有改进后的retinex效果好。 同时通过这个实验我也发现retinex算法的参数（c）会很大地影响算法效果，在实际使用场景中必须针对具体的图像进行调参。 （结果图片在result文件夹下）

5代码

**import** cv2 **as** cv  
**import** numpy **as** np  
**import** os  
  
**def** linear\_stretch(input):  
 *#对灰度进行线性拉伸* **if** len(input.shape) <3:  
 shape = [input.shape[0],input.shape[1],1]  
 input = np.resize(input,shape)  
 **for** i **in** range(input.shape[-1]):  
 max\_value = np.max(input[:,:,i])  
 min\_value = np.min(input[:,:,i])  
 scale = 255/(max\_value-min\_value)  
 input[:,:,i] = (input[:,:,i]-min\_value)\*scale  
 **return** input  
  
**def** cal\_gauss\_kernal(gauss\_c, kernal\_size=[3,3]):  
 *'''  
 计算高斯核函数* **:param** *gauss\_c: 高斯环绕尺度* **:param** *kernal\_size: 核大小, 为方便起见长和宽必须为奇数，如果输入为偶数的话会将其+1变为奇数 [h,w]* **:return***: 高斯核 一个 rank=2的 numpy数组  
 '''* h,w = kernal\_size  
 gauss\_c\_2 = np.square(gauss\_c)  
 **if** h%2 == 0:  
 h=h+1  
 **if** w%2 ==0:  
 w=w+1  
 center = [h//2,w//2]  
 kernal = np.zeros(kernal\_size,dtype= np.float32)  
 **for** i **in** range(kernal\_size[0]):  
 **for** j **in** range(kernal\_size[1]):  
 *#遍历计算kernal* y = np.abs(i-center[0])  
 x = np.abs(j-center[1])  
 kernal[i][j] = np.exp(-(np.square(x)+ np.square(y))/gauss\_c\_2 ) *# e ^ ( -(x^2 + y^2)/c^2)* lam = np.sum(kernal)  
 **return** kernal/lam  
  
*#利用 retinex算法实现图像对比度增强***def** retinex\_ssr(input,gauss\_c=80,kernal\_size=[3,3]):  
 *'''  
 采用 单尺度比较简单的retinex ssr算法 ， 算法思想： r(x,y) = s(x,y) - l(x,y) (r,s,l 是对数化后的值）  
 l(x,y)是光照函数，我们认为光照函数变化比较平滑，在ssr中 采用 高斯滤波器进行近似，即：  
 r(x,y) = s(x,y) - log( F(x,y) \*\* S(x,y)) (\*\*表示卷积）  
 F(x,y) = lambda \* e ^ ( -(x^2 + y^2)/c^2) （c是高斯环绕尺度，lambda是一个自适应尺度，再确定c的情况下 使得F积分为1）* **:param** *input: 输入，可以是灰度图像也可以是彩色图像，尝试写自适应算法* **:param** *gauss\_c: 高斯标准差 c* **:return***: 处理之后的图像  
 '''* input = np.array(input,dtype=np.float32)  
 shape = input.shape  
 kernal = cal\_gauss\_kernal(gauss\_c,kernal\_size)  
 input\_L = np.copy(input)  
 input\_L = cv.filter2D(input\_L,-1,kernal)  
 input = np.clip(input,0.000001,255)  
 input\_L = np.clip( input\_L , 0.000001, 255)  
 input = np.log(input)  
 input\_L= np.log(input\_L)  
 output = input - input\_L  
 *#output = np.exp(output)  
 #output = linear\_stretch(output)  
 #output = np.array(output,dtype= np.uint8)* output = input + 6\*output  
 output = np.exp(output)  
 **return** np.resize(output,shape)  
  
  
**if** \_\_name\_\_ ==**'\_\_main\_\_'**:  
 BASE\_PATH = os.path.dirname(os.path.abspath(\_\_file\_\_))  
 RESULT\_PATH = os.path.join(BASE\_PATH, **"result"**)  
 **if not** os.path.exists(RESULT\_PATH):  
 os.mkdir(RESULT\_PATH)  
 **import** argparse  
  
 parser = argparse.ArgumentParser()  
 parser.add\_argument(**'--img'**, help=**"the path of the input img"**)  
 parser.add\_argument(**'--c'**, help=**'the param of gauss\_c'**)  
 *#parser.add\_argument('--kernal\_size', help='the size of the kernal')* args = parser.parse\_args()  
 *################# 图片读取模块* img\_name = os.path.split(args.img)[-1]  
 img\_name, ext = os.path.splitext(img\_name)  
 input = cv.imread(args.img, flags=cv.IMREAD\_GRAYSCALE) *#读取灰度图片  
 #input = cv.imread(args.img, flags=cv.IMREAD\_COLOR) #读取rgb图片  
 ########################## 处理模块* output = retinex\_ssr(input,float(args.c),[input.shape[0],input.shape[1]])  
 *#########图片输出模块* img\_name = img\_name + **"\_retinex\_gauss\_"** + str(args.c)  
 img\_name = img\_name + ext  
 img\_name = os.path.join(RESULT\_PATH, img\_name)  
 cv.imwrite(img\_name,output)