数字图像处理

第五次作业

董浩

自动化少 61

2140506111

2019 年 4 月 2 日

摘要：

本次作业内容包括分别通过 Butterworth 和 Gaussian 频域低通和高通滤波器，选取不同的截止频率，对图像进行边缘增强，并计算图像的功率谱比。此外还通过拉普拉斯和 Unmask 高通滤波器的方法对图像处理等等。本次作业报告的内容首先对基本原理进行了介绍，然后通过编程的方式实现，并打印出图像处理结果。编程环境为 Python3.6.3 和 Opencv-python3.4.2。

一、频域低通滤波器

1.频域滤波的步骤和功率谱

1)先将图像做频域内的水平移动，然后求原图像 f(x,y)的 DFT，得到其图像的傅里叶谱 F(u,v)

$$F(u,v)=\Im\left[(-1)^{x+y}f_p(x,y)\right]$$

2)与频域滤波器做乘积

$$G(u,v)=H(u,v)F(u,v)$$

3)求取 G(u,v)的 IDFT，然后再将图像做频域内的水平移动（移动回去），其结果可能存在寄生的虚数，此时忽略即可

$$g_p(x,y)=\{\mathrm{Re}(\mathrm{IDFT}(\mathrm{G(u,\ v)}))\}(-1)^{x+y}$$

4) 这里使用 ifft2 函数进行 IDFT 变换，得到的图像的尺寸为 PxQ，切取左上角的 MxN 的图像，就能得到结果了。

功率谱定义：

$$P_f(u,v)=|F(u,v)|^2P_g(u,v)=|G(u,v)|^2$$

$P_f(u,v)$为滤波前图像的功率谱，$P_g(u,v)$为滤波后图像的功率谱。

滤波器的功率谱理解为：

$$\mathrm{L}=\frac{P_g(u,v)}{P_f(u,v)}$$

2.理想的低通滤波器

$$H(u,v)=\begin{cases}1, D(u,v)\le D_0\\ 0, D(u,v)>D_0\end{cases}$$

其中，D0 表示通带的半径。D(u,v)的计算方式也就是两点间的距离，很简单就能得到。

$$D(u,v)=\sqrt{\left(u-\frac{P}{2}\right)^2+\left(v-\frac{Q}{2}\right)^2}$$

低通滤波器滤除了高频成分，所以使得图像模糊。由于理想低通滤波器的过度特性过于急峻，所以会产生了振铃现象。

3.巴特沃斯低通滤波器

$$H(u,v)=\frac{1}{1+\left(D(u,v)/D_0\right)^{2n}}$$

同样的，D0 表示通带的半径，n 表示的是巴特沃斯滤波器的次数。随着次数的增加，振铃现象会越来越明显。

4.高斯低通滤波器

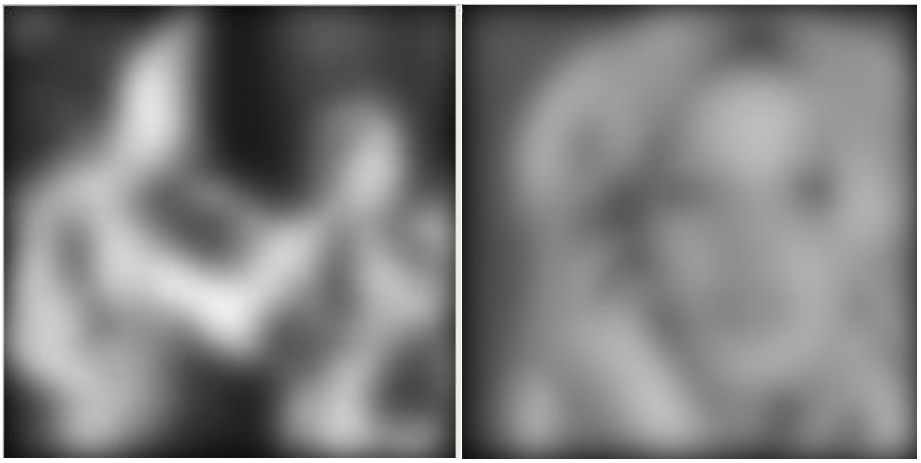
$$H(u,v) = e^{\frac{-D^2(u,v)}{2D_0^2}}$$

D0 表示通带的半径。高斯滤波器的过度特性非常平坦，因此是不会产生振铃现象的。

5.实验结果

5.1 巴特沃斯低通滤波器

5.1.1 D0=10　L1=0.830371308187　L2=0.891247053846



5.1.2 D0=30　L1=0.930928609369　L2=0.949503255826

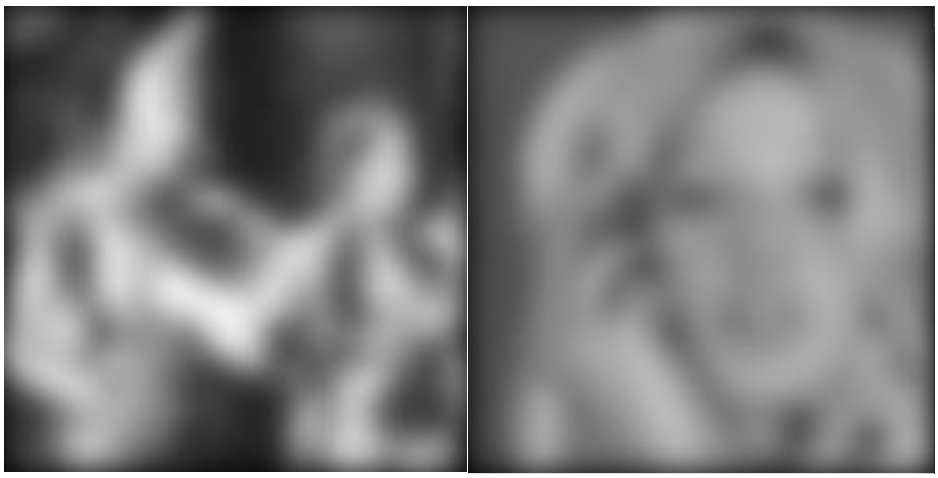
5.1.3 D0=60　L1=0.964113104977　L2=0.96660876847

5.1.4 D0=160 L1=0.987770394211 L2=0.988383259221

5.1.5 D0=460 L1=0.993600457158 L2=0.995267720661

5.2 高斯低通滤波器

5.2.1 D0=10 L1=0.820398723127 L2=0.880092189054



5.2.2 D0=30 L1=0.92616456987 L2=0.945000063875

5.2.3 D0=60 L1=0.961899245134 L2=0.96660876847

5.2.4 D0=160 L1=0.985508347745 L2=0.982364061411

5.2.5 D0=460　L1=0.992387649652　L2=0.990854406642


6.结果分析

通过观察选取不同 D0 得到的结果可以发现，当 D0 为 10 时，图像严重模糊，表面图像多数尖锐的细节信息包含在被滤波器滤除的 13%功率内。随着滤波器半径的增大，滤除的功率也越来越少，导致的模糊也越来越弱。随着被滤除的高频内容的数量的减少，图像的纹理变得越来越好。

此外二阶巴特沃斯低通滤波器和高斯低通滤波器都没有明显的振铃现象，这要归功于这两种滤波器在低频和高频之间的平滑过渡。而且对于相同的截止频率，高斯低通滤波器和二阶巴特沃斯低通滤波器相比导致的平滑效果要稍差一些，主要因为高斯低通滤波器的剖面图不像二阶巴特沃斯低通滤波器的那样紧凑。

二、频域高通滤波器

1.理想高通滤波器

$$H(u,v)=\begin{cases}0, D(u,v)\le D_0\\ 1, D(u,v)>D_0\end{cases}$$

这里的 D0 是滤波器的阻带半径，而 D(u,v)是点到滤波器中央的距离。用这个滤波器对图像进行处理，与理想的低通滤波器一样，所得到的图像有很明显的振铃现象。结果图像从视觉上来看，有些偏暗，这是因为图像的直流分量被滤掉的原因。

2.巴特沃斯高通滤波器

$$H(u,v)=\frac{1}{1+\left(D_0/D(u,v)\right)^{2n}}$$

同样的，巴特沃斯高通滤波器也可以通过改变次数 n，对过度特性进行调整。过大的 n 会造成振铃现象。

3.高斯高通滤波器

$$H(u,v)=1-e^{\frac{-D^2(u,v)}{2D_0^2}}$$

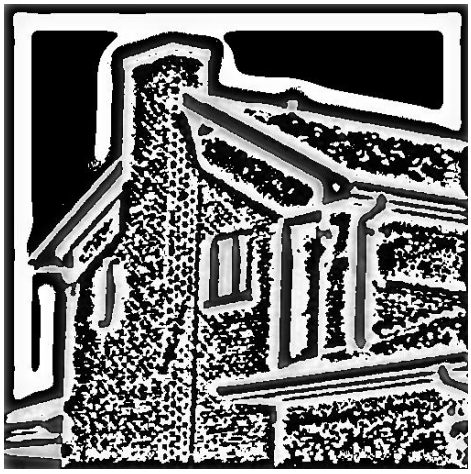
高斯滤波器的过度特性很好，所以不会发生振铃现象。

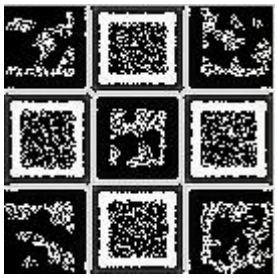
4.实验结果

4.1 巴特沃斯高通滤波器

4.1.1 D0=30

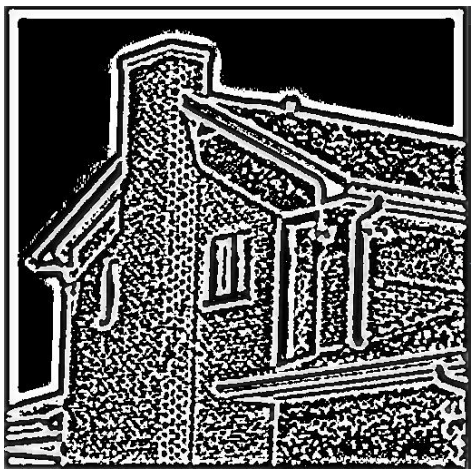
L1=1.18542575049　　L2=0.786451223374

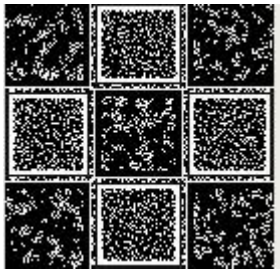




4.1.2 D0=60

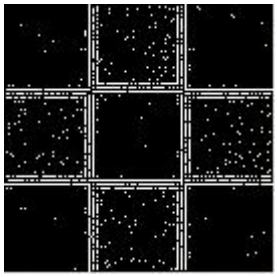
L1=1.01439133281　　L2=0.638253964473





4.1.3 D0=160

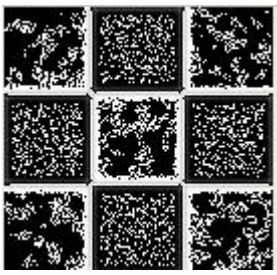
L1=0.706401009005　　L2=0.213132555844



4.2 高斯高通滤波器

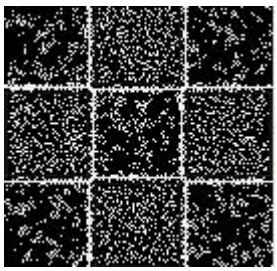
4.2.1 D0=30

L1=0.913584916765　　L2=0.658227025639



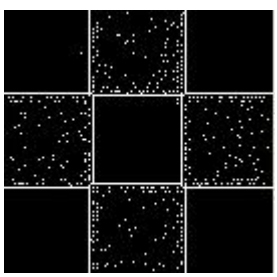
4.2.2 D0=60

L1=0.851828297001　　L2=0.479685636526



4.2.3 D0=160

L1=0.596703471419　　L2=0.128320753276



5.结果分析

通过观察选取不同 D0 得到的结果可以发现，截止频率越大，使用两种高通滤波器得到的结果就越平滑。高通滤波后的图像失去了它的灰度色调，因为直流项已被减小为 0，最终结果是高通滤波后的图像中典型的暗色调已成为主流，因此需要额外的处理来增强感兴趣的细节。

三、其他高通滤波器

1.频率域的拉普拉斯算子

拉普拉斯算子可使用如下滤波器在频率域中实现：

$$H(u,v) = -4\pi^2(u^2 + v^2)$$

在关于频率矩形的中心，使用如下滤波器：

$$H(u,v) = -4\pi^2[(u - p/2)^2 + (v - Q/s)^2] = -4\pi^2 D^2(u,v)$$

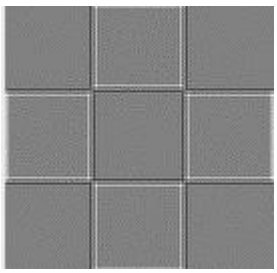
其中，D(u,v)是距离函数。

2.Unmask 滤波器

$$g(x,y) = \Im^{-1}\{[k_1 + k_2 * H_{HP}(u,v)]]F(u,v)\}$$

3.实验结果

3.1 拉普拉斯



3.2 Unmask

4.结果分析

对比每组图像处理结果中的原始图像和滤波后的图像,可以隐约看到滤波器的边缘增强效果。由于拉普拉斯高通滤波器将原始图像完全加回到滤波后的结果中，因此解决了巴特沃斯滤波器和高斯滤波器除去了傅里叶变换零频率成分的问题,从而使得滤波后的图像背景的平均 强度增加变亮，但同时引入了噪声干扰，使得滤波后的图像有一定程度的失真。可见 Unmask 的图像边缘信息更加清晰，但同时带来了过度锐化的问题，出现了多重轮廓。

四、比较并讨论空域低通高通滤波（Project3）与频域低通和高通的关系

空间域和频域滤波间的纽带是卷积定理。空间域中的滤波定义为滤波函数 h（x，y）与输入图像 f（x，y）进行卷积；频率域中的滤波定义为滤波函数 H（u，v）与输入图像的傅里叶变换 F（u，v）进行相乘。空间域的滤波器和频率域的滤波器互为傅里叶变换。

频域增强技术与空域增强技术有密切的联系。一方面，许多空域增强技术可借助频域概念来分析和帮助设计；另一方面，许多空域增强技术可转化到频域实现，而许多频域增强技术可转化到空域实现。空域滤波主要包括平滑滤波和锐化滤波。平滑滤波是要滤除不规则的噪声或干扰的影响，从频域的角度看，不规则的噪声具有较高的频率，所以可用具有低通能力的频域滤波器来滤除。由此可见空域的平滑滤波对应频域的低通滤波。锐化滤波是要增强边缘和轮廓处的强度，从频域的角度看，边缘和轮廓处都具有较高的频率，所以可用具有高通能力的频域滤波器来增强。由此可见，空域的锐化滤波对应频域的高通滤波。频域里低通滤波器的转移函数应该对应空域里平滑滤波器的模板函数的傅里叶变换。频域里高通滤波器的转移函数应该对应空域里锐化滤波器的模板函数的傅里叶变换。即空域和频域的滤波器组

成傅里叶变换对。给定一个域内的滤波器，通过傅里叶变换或反变换得到在另一个域内对应的滤波器。空域的锐化滤波或频域的高通滤波可用两个空域的平滑滤波器或两个频域的低通滤波器实现。在频域中分析图像的频率成分与图像的视觉效果间的对应关系比较直观。空域滤波在具体实现上和硬件设计上有一定优点。区别：例如，空域技术中无论使用点操作还是模板操作，每次都只是基于部分像素的性质，而频域技术每次都利用图像中所有像素的数据，具有全局性，有可能更好地体现图像的整体特性，如整体对比度和平均灰度值等。

附录 1：参考文献

[1] 冈萨雷斯.数字图像处理(第三版)北京:电子工业出版社,2011

附录 2：源代码

1.low_pass.py

```
'''频域低通滤波器：设计低通滤波器包括 butterworth and Gaussian (选择合适的半径，计算功率谱比),
平滑测试图像 test1 和 2;分析各自优缺点；'''

import cv2
import math
import numpy as np
import matplotlib.pyplot as plt

test = cv2.imread("./test1.pgm", 0)
# test = cv2.imread("./test2.tif", 0)
test_c1 = test.copy()
test_c2 = test.copy()
width = test.shape[0]
height = test.shape[1]
test_l = np.zeros((2*width, 2*height), np.int)
test_l_g = np.zeros((2*width, 2*height), np.int)
test_l_b = np.zeros((2*width, 2*height), np.int)
test_l_G = np.zeros((2*width, 2*height), dtype=np.complex128)
test_l_B = np.zeros((2*width, 2*height), dtype=np.complex128)
test_extend = cv2.copyMakeBorder(test, 0, width, 0, height, cv2.BORDER_CONSTANT, value=[0, 0,
0])

D0 = 60

for i in range(0, width):
    for j in range(0, height):
        if (i + j) % 2 == 1:
            test_l[i, j] = -1 * test_extend[i, j]
        else:
            test_l[i, j] = test_extend[i, j]
```

```
test_l_F = np.fft.fft2(test_l)


def cal_energy(img):
    energy = 0
    for i in range(0, img.shape[0]):
        for j in range(0, img.shape[1]):
            energy = energy + abs(img[i, j])**2
    return energy

def gause_lp_H(u, v):
    return math.exp(((u - width)*(u - width)+(v - height)*(v - height))*(-1)/2/D0/D0)


def butterworth_lp_H(u, v):
    D_uv_2 = (u - width)*(u - width)+(v - height)*(v - height)
    return 1/(1+D_uv_2*D_uv_2/D0/D0/D0/D0)


for i in range(0, 2*width):
    for j in range(0, 2*height):
        test_l_G[i, j] = test_l_F[i, j] * gause_lp_H(i, j)
        test_l_B[i, j] = test_l_F[i, j] * butterworth_lp_H(i, j)

test_l_G_IF = np.fft.ifft2(test_l_G)
test_l_G_IF_real = test_l_G_IF.real
test_l_B_IF = np.fft.ifft2(test_l_B)
test_l_B_IF_real = test_l_B_IF.real

for i in range(0, 2*width):
    for j in range(0, 2*height):
        if (i + j) % 2 == 1:
            test_l_g[i, j] = -1 * test_l_G_IF_real[i, j]
            test_l_b[i, j] = -1 * test_l_B_IF_real[i, j]
        else:
            test_l_g[i, j] = test_l_G_IF_real[i, j]
            test_l_b[i, j] = test_l_B_IF_real[i, j]

for i in range(0, width):
    for j in range(0, height):
        test_c1[i, j] = test_l_g[i, j]
        test_c2[i, j] = test_l_b[i, j]
```

```
'''plt.subplot(221)
plt.imshow(abs(test_l_F))
plt.title('fft')
plt.subplot(222)
plt.imshow(abs(test_l_G))
plt.title('G')
plt.subplot(223)
plt.imshow(abs(test_l_B))
plt.title('B')
plt.show()'''

g_l = cal_energy(test_c1)/cal_energy(test)
b_l = cal_energy(test_c2)/cal_energy(test)

print('gl:', g_l)
print('bl', b_l)
cv2.imshow('test_g_l', test_c1)
cv2.imshow('test_b_l', test_c2)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

2.high_pass.py

```
'''频域高通滤波器：设计高通滤波器包括 butterworth and Gaussian，在频域增强边缘。
选择半径和计算功率谱比，测试图像 test3,4：分析各自优缺点；'''

import cv2
import math
import numpy as np

test = cv2.imread("./test4.tif", 0)
# test = cv2.imread("./test3_corrupt.pgm", 0)
test_c3 = test.copy()
test_c4 = test.copy()
width = test.shape[0]
height = test.shape[1]
test_l = np.zeros((2*width, 2*height), np.int)
test_h_g = np.zeros((2*width, 2*height), np.int)
test_h_b = np.zeros((2*width, 2*height), np.int)
test_h_G = np.zeros((2*width, 2*height), dtype=np.complex128)
test_h_B = np.zeros((2*width, 2*height), dtype=np.complex128)
test_extend = cv2.copyMakeBorder(test, 0, width, 0, height, cv2.BORDER_CONSTANT, value=[0, 0,
0])
```

```
D0 = 160

for i in range(0, width):
    for j in range(0, height):
        if (i + j) % 2 == 1:
            test_l[i, j] = -1 * test_extend[i, j]
        else:
            test_l[i, j] = test_extend[i, j]

test_l_F = np.fft.fft2(test_l)


def cal_energy(img):
    energy = 0
    for i in range(0, img.shape[0]):
        for j in range(0, img.shape[1]):
            energy = energy + abs(img[i, j])**2
    return energy


def gause_lp_H(u, v):
    return math.exp(((u - width)*(u - width)+(v - height)*(v - height))*(-1)/2/D0/D0)


def gause_hp_H(u, v):
    return 1-gause_lp_H(u, v)


def butterworth_hp_H(u, v):
    D_uv_2 = (u - width)*(u - width)+(v - height)*(v - height)
    if D_uv_2 != 0:
        return 1/(1+D0*D0*D0*D0/D_uv_2/D_uv_2)
    else:
        return 0

for i in range(0, 2*width):
    for j in range(0, 2*height):
        test_h_G[i, j] = test_l_F[i, j] * gause_hp_H(i, j)
        test_h_B[i, j] = test_l_F[i, j] * butterworth_hp_H(i, j)

test_h_G_IF = np.fft.ifft2(test_h_G)
test_h_G_IF_real = test_h_G_IF.real
test_h_B_IF = np.fft.ifft2(test_h_B)
test_h_B_IF_real = test_h_B_IF.real
```

```
for i in range(0, 2*width):
    for j in range(0, 2*height):
        if (i + j) % 2 == 1:
            test_h_g[i, j] = -1 * test_h_G_IF_real[i, j]
            test_h_b[i, j] = -1 * test_h_B_IF_real[i, j]
        else:
            test_h_g[i, j] = test_h_G_IF_real[i, j]
            test_h_b[i, j] = test_h_B_IF_real[i, j]

for i in range(0, width):
    for j in range(0, height):
        test_c3[i, j] = test_h_g[i, j]
        test_c4[i, j] = test_h_b[i, j]

g_l = cal_energy(test_c3)/cal_energy(test)
b_l = cal_energy(test_c4)/cal_energy(test)

print('gl:', g_l)
print('bl', b_l)

cv2.imshow('test_g_h', test_c3)
cv2.imshow('test_b_h', test_c4)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

3.high_pass2.py

```
"其他高通滤波器：拉普拉斯和 Unmask，对测试图像 test3,4 滤波；分析各自优缺点；"

import cv2
import math
import numpy as np

# test = cv2.imread("./test4.tif", 0)
test = cv2.imread("./test3_corrupt.pgm", 0)
test_c3 = test.copy()
test_c4 = test.copy()
width = test.shape[0]
height = test.shape[1]
test_l = np.zeros((2*width, 2*height), np.int)
test_h_b = np.zeros((2*width, 2*height), np.float32)
test_h_B = np.zeros((2*width, 2*height), dtype=np.complex128)
test_h_u = np.zeros((2*width, 2*height), np.float32)
```

```
test_h_U = np.zeros((2*width, 2*height), dtype=np.complex128)
test_extend = cv2.copyMakeBorder(test, 0, width, 0, height, cv2.BORDER_CONSTANT, value=[0, 0,
0])

for i in range(0, 2*width):
    for j in range(0, 2*height):
        if (i + j) % 2 == 1:
            test_l[i, j] = -1 * test_extend[i, j]
        else:
            test_l[i, j] = test_extend[i, j]

test_l_F = np.fft.fft2(test_l)


def laplace_hp_H(u, v):
    D_uv_2 = (u - width) * (u - width) + (v - height) * (v - height)
    return 1 + 4 * math.pi * math.pi * D_uv_2


def unmask_hp_H(u, v):
    return 0.5+0.75*laplace_hp_H(u, v)

for i in range(0, 2*width):
    for j in range(0, 2*height):
        test_h_B[i, j] = test_l_F[i, j] * laplace_hp_H(i, j)
        test_h_U[i, j] = test_l_F[i, j] * unmask_hp_H(i, j)

test_h_B_IF = np.fft.ifft2(test_h_B)
test_h_B_IF_real = test_h_B_IF.real
test_h_U_IF = np.fft.ifft2(test_h_U)
test_h_U_IF_real = test_h_U_IF.real

for i in range(0, 2*width):
    for j in range(0, 2*height):
        if (i + j) % 2 == 1:
            test_h_b[i, j] = -1 * test_h_B_IF_real[i, j]
            test_h_u[i, j] = -1 * test_h_U_IF_real[i, j]
        else:
            test_h_b[i, j] = test_h_B_IF_real[i, j]
            test_h_u[i, j] = test_h_U_IF_real[i, j]

test_h_min = test_h_b.min()
test_h_m = test_h_b.max()-test_h_min
test_h_min_u = test_h_u.min()
```

```
test_h_m_u = test_h_u.max()-test_h_min_u

for i in range(0, width):
    for j in range(0, height):
        test_h_b[i, j] = test_h_b[i, j] - test_h_min
        test_h_b[i, j] = int(test_h_b[i, j] * 255 / test_h_m)
        test_c3[i, j] = test_h_b[i, j]
        test_h_u[i, j] = test_h_u[i, j] - test_h_min_u
        test_h_u[i, j] = int(test_h_u[i, j] * 255 / test_h_m_u)
        test_c4[i, j] = test_h_u[i, j]

cv2.imshow('test_l_h', test_c3)
cv2.imshow('test_u_h', test_c4)
cv2.waitKey(0)
cv2.destroyAllWindows()
```