可视化作业 3

张言健 16300200020

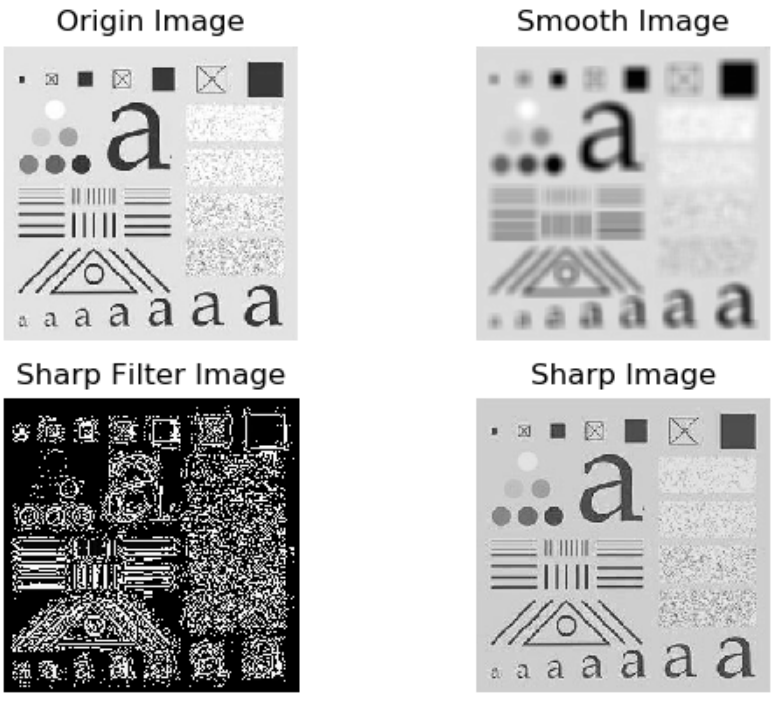
1. 编程实现基于空间滤波器的（1）平滑操作、（2）锐化算法算法；并把算法应用与图片上，显示与原图的对比差别。备注：实现的代码不能调用某个算法库里面的函数实现平滑或锐化。

   平滑操作

```
def smooth(image, N=9):
    operator = np.array([[1./(N*N)] * N] * N)
    new_image = copy.deepcopy(image)
    np_image = np.array(image)
    for i in range(int((N-1)/2), len(image)-int((N-1)/2)):
        for j in range(int((N-1)/2), len(image[0])-int((N-1)/2)):
            new_image[i][j] = np.sum(
                np_image[i-int((N-1)/2): i+int((N+1)/2), j-int((N-1)/2):
j+int((N+1)/2)] * operator)
            # new_image[i][j] = np.sum(np_image[i-1: i+2, j-1: j+2] *
operator)
    image_max = np.max(new_image)
    image_min = np.min(new_image)
    new_image = [[(p-image_min)/(image_max-image_min)*255
                  for p in ps] for ps in new_image]
    # print(new_image)
    return new_image


def sharp(image):
    operator = np.array(
        [[0.0, -1.0, 0.0], [-1.0, 4.0, -1.0], [0.0, -1.0, 0.0]])
    np_image = np.array(image)
    filter_image = np.zeros(np_image.shape)
    for i in range(1, len(image)-1):
        for j in range(1, len(image[0])-1):
            filter_image[i][j] = 0.1 * np.sum(np_image[i-1: i+2, j-1: j+2] *
operator)
    new_image = np_image + filter_image
    image_max = np.max(new_image)
    image_min = np.min(new_image)
    new_image = [[(p-image_min)/(image_max-image_min)*255
                  for p in ps] for ps in new_image]
    return filter_image, new_image

# image = data.moon()
image = io.imread("./image/pattern.jpg")
dtype = image.dtype.type
print(image)
image =  rgb2gray(image)
image = dtype(list([int(i*255) for i in j] for j in image))



print(image)
dtype = image.dtype.type
```

```
io.imsave("./image/moon_gray.jpg", image)
plt.subplot(2, 2, 1)
plt.imshow(image, plt.cm.gray)
plt.axis('off')
plt.title('Origin Image')

image_smooth = smooth(image)
image_smooth = dtype(image_smooth)
io.imsave("./image/img_smooth.jpg", image_smooth)
plt.subplot(2, 2, 2)
plt.imshow(image_smooth, plt.cm.gray)
plt.axis('off')
plt.title('Smooth Image')

image = io.imread("./image/moon_gray.jpg")
dtype = image.dtype.type
io.imsave("./image/img.jpg", image)
image_edge, image_sharp = sharp(image)
image_sharp = dtype(image_sharp)
image_edge = dtype(image_edge)
io.imsave("./image/img_sharp.jpg", image_sharp)

plt.subplot(2, 2, 3)
plt.imshow(image_edge, plt.cm.gray)
plt.axis('off')
plt.title('Sharp Filter Image')

plt.subplot(2, 2, 4)
plt.imshow(image_sharp, plt.cm.gray)
plt.axis('off')
plt.title('Sharp Image')
plt.show()
```

结果如下：



2. 证明二维变量的离散傅里叶变换的卷积定理即：

f(x,y)*h(x,y) <==> F(u,v)H(u,v)

f(x,y)h(x,y) <==> F(u,v)*H(u,v)

其中, * 表示卷积运算。

Proof: f(x,y)*h(x,y) <==> F(u,v)H(u,v)

let $t(x,y) = f(x,y) * h(x,y)$

$$
\begin{aligned}
T(u,v) &= \sum_{x=0}^{M-1}(\sum_{y=0}^{N-1} t(x,y)e^{-j2\pi\left(\frac{ux}{M}+\frac{vy}{N}\right)}) \\
&= \sum_{x=0}^{M-1}(\sum_{y=0}^{N-1}\sum_{k=0}^{N-1} f(x,k)h(x,y-k)e^{-j2\pi\left(\frac{ux}{M}+\frac{vy}{N}\right)}) \\
&= \sum_{x=0}^{M-1}\sum_{y=0}^{N-1}\sum_{k=0}^{N-1} f(x,k)(h(x,y-k)e^{-j2\pi\left(\frac{ux}{M}+\frac{vy}{N}\right)}) \\
&= \sum_{x=0}^{M-1}\sum_{y=0}^{N-1}\sum_{l=0}^{M-1}\sum_{k=0}^{N-1} f(l,k)(h(x-l,y-k)e^{-j2\pi\left(\frac{ux}{M}+\frac{vy}{N}\right)}) \\
&= \sum_{l=0}^{M-1}\sum_{k=0}^{N-1} f(l,k)(\sum_{x=0}^{M-1}\sum_{y=0}^{N-1} h(x-l,y-k)e^{-j2\pi\left(\frac{u(x-l)+ul}{M}+\frac{v(y-k)+vk}{N}\right)}) \\
&= \sum_{l=0}^{M-1}\sum_{k=0}^{N-1} f(l,k)(\sum_{x-l=0}^{M-1}\sum_{y-k=0}^{N-1} h(x-l,y-k)e^{-j2\pi\left(\frac{u(x-l)+ul}{M}+\frac{v(y-k)+vk}{N}\right)}) \\
&= H(x,y)(\sum_{l=0}^{M-1}\sum_{k=0}^{N-1} f(l,k)e^{-j2\pi\left(\frac{ul}{M}+\frac{vk}{N}\right)}) \\
&= F(x,y)H(x,y)
\end{aligned}
$$

Proof: f(x,y)h(x,y) <==> F(u,v)*H(u,v)

let $T(u,v) = F(x,y) * H(x,y)$

Which means

$\sum_{x=0}^{M-1}(\sum_{y=0}^{N-1} t(x,y)e^{-j2\pi\left(\frac{ux}{M}+\frac{vy}{N}\right)}) = \sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(x,y)\sum_{l=0}^{M-1}\sum_{k=0}^{N-1} h(l-x,k-y)e^{-j2\pi\left(\frac{um}{M}+\frac{vn}{N}\right)})$

Then we have

$left = \sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(x,y)\sum_{l-x=0}^{M-1}\sum_{k-y=0}^{N-1} h(l-x,k-y)e^{-j2\pi\left(\frac{um}{M}+\frac{vn}{N}\right)})$

Thus we have

$left = \sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(x,y)h(x,y)e^{-j2\pi\left(\frac{um}{M}+\frac{vn}{N}\right)})$

hence we have

$t(x,y) = f(x,y)h(x,y)$

3. 编程实现基于课件中频率域滤波5步骤的：

（1）低通平滑操作，并把算法应用与图片上，显示原图的频谱图、频域操作结果的频谱图，以及操作结果；

（2）频率操作，去除大脑CT体膜Shepp-Logan图像中的条纹。

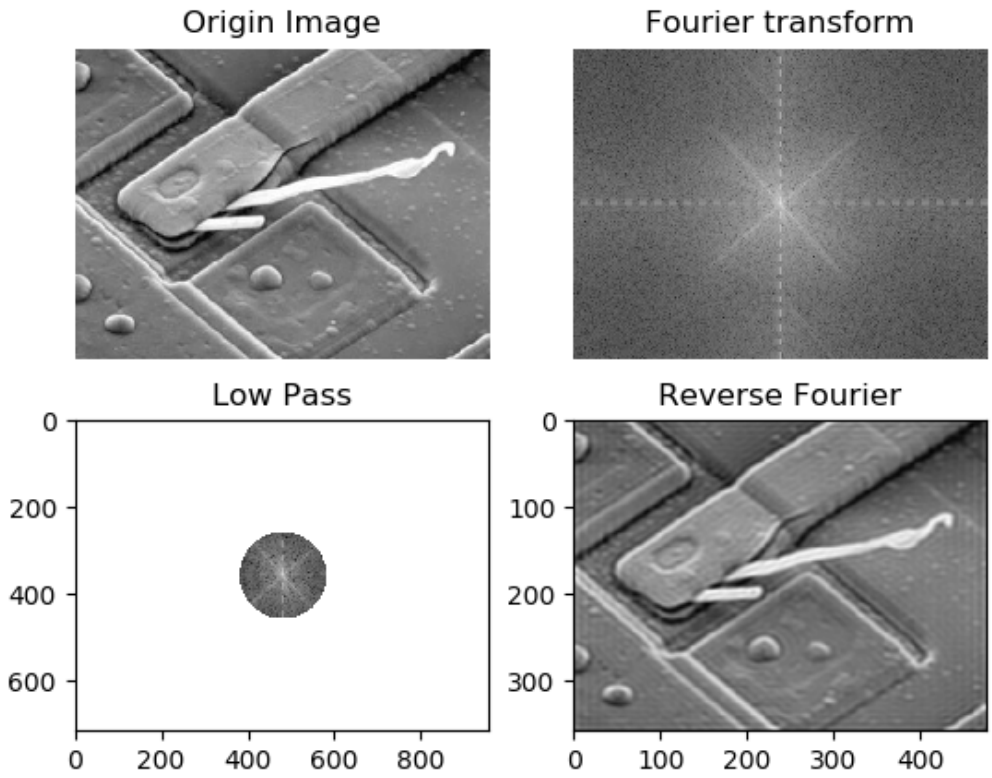
备注：图像的时空-频域变换过程（即离散傅里叶变换和逆变换）可以调用库函数。

低通平滑

```
![smooth_and_sharp](D:\SP2(资料存储室)\复旦学习资料\大四下\数据可视化\可视化作业3\image\smooth_and_sharp.png)def low_pass(image):

    M, N = image.shape
    P,Q = 2*M, 2*N
    pad_image = np.ones((P, Q))
    for i in range(M):
        for j in range(N):
            pad_image[i][j] = image[i][j]

    fc = np.zeros((P, Q))
    for i in range(P):
        for j in range(Q):
            fc[i][j] = pad_image[i][j] * ((-1) ** (i + j))

    f_image = fft2(fc)  #  2-D discrete Fourier transform.

    plt.subplot(2, 2, 2)
    plt.imshow(np.log(np.abs(f_image)), "gray")
    plt.axis('off')
    plt.title('Fourier transform')

    low_pass_filter = np.zeros((P, Q))
    size = 100
    for i in range(P):
        for j in range(Q):
            if (i-M)**2+(j-N)**2 < size**2:
                low_pass_filter[i][j] = 1
    new_f_image = f_image*low_pass_filter

    plt.subplot(2, 2, 3)
    plt.imshow(np.log(abs(new_f_image)), "gray")
    plt.title('Low Pass')

    re_image = ifft2(new_f_image)

    fc1 = np.zeros((P, Q))
    for i in range(P):
        for j in range(Q):
            fc1[i][j] = re_image[i][j] * ((-1) ** (i + j))
    re_image = fc1[:M,:N]

    plt.subplot(2, 2, 4)
    plt.imshow(re_image, "gray")
    plt.title('Reverse Fourier')

# low-freq filter
image = io.imread("./image/slice.jpg")
dtype = image.dtype.type
print(image)
image =  rgb2gray(image)
image = dtype(list([int(i*255) for i in j] for j in image))
io.imsave("./image/pattern_grey.jpg", image)
```

```
plt.subplot(2, 2, 1)
plt.imshow(image, plt.cm.gray, label='First Line')
plt.axis('off')
plt.title('Origin Image')
image = np.array(image)
low_pass(image)
plt.show()
```

结果如下



频率操作（notch filter）

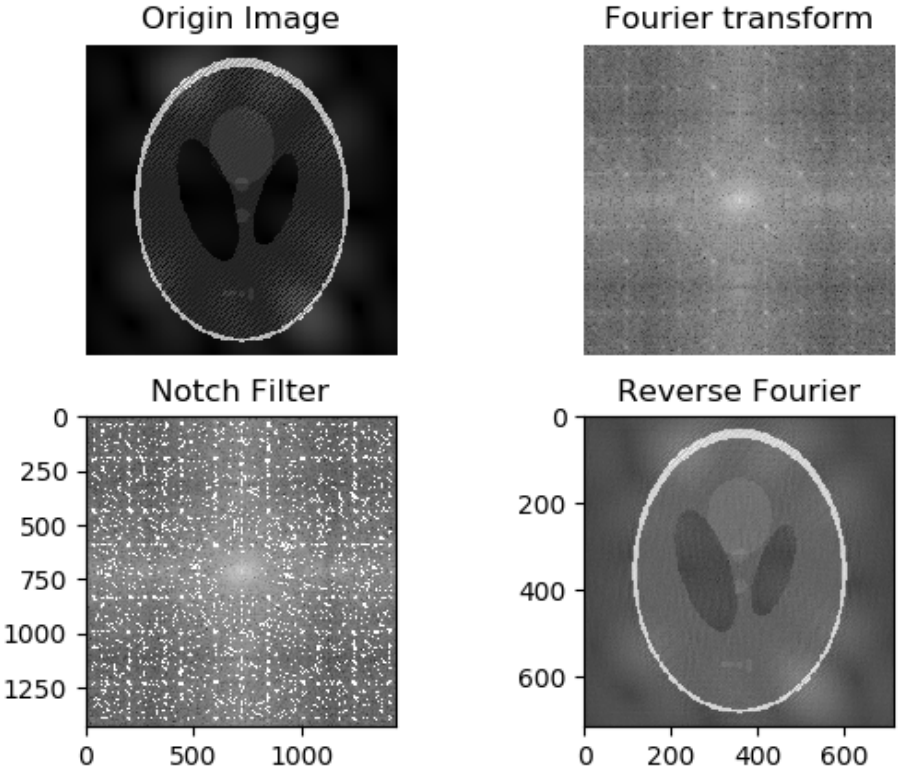
思路，将亮斑用notch filter相乘去掉

筛选亮斑的方法：其亮度高于周围平均值的1.8倍

```
def notch_filter(image):

    M, N = image.shape
    P,Q = 2*M, 2*N
    pad_image = np.ones((P, Q))
    for i in range(M):
        for j in range(N):
            pad_image[i][j] = image[i][j]

    fc = np.zeros((P, Q))
    for i in range(P):
        for j in range(Q):
            fc[i][j] = pad_image[i][j] * ((-1) ** (i + j))

    f_image = fft2(fc)                          #  2-D discrete Fourier
transform.
    plt.subplot(2, 2, 2)
```

```
    plt.imshow(np.log(np.abs(f_image)), "gray")
    plt.axis('off')
    plt.title('Fourier transform')
    notch_filt = np.ones((P, Q))

    abs_f_image = abs(f_image)
    size = 30
    center = 60                                  # keep the center part
    for i in range(size,P-size):
        for j in range(size,Q-size):
            if not (i-M)**2+(j-N)**2 < center**2 \
          and f_image[i][j] > 1.8 * np.mean(abs_f_image[i-size:i+size, j-size:j
+size]):
                notch_filt[i-1:i+2,j-1:j+2] = np.zeros((3,3))  # set empty

    new_f_image = f_image * notch_filt

    plt.subplot(2, 2, 3)
    plt.imshow(np.log(abs(new_f_image)), "gray")
    plt.title('Notch Filter')

    re_image = ifft2(new_f_image)

    fc1 = np.zeros((P, Q))
    for i in range(P):
        for j in range(Q):
            fc1[i][j] = re_image[i][j] * ((-1) ** (i + j))
    re_image = fc1[:M,:N]

    plt.subplot(2, 2, 4)
    plt.imshow(re_image, "gray")
    plt.title('Reverse Fourier')

# notch filter
image = io.imread("./image/freq_testimage_shepplogan.PNG")
dtype = image.dtype.type
image =  rgb2gray(image)
image = dtype(list([int(i*255) for i in j] for j in image))
io.imsave("./image/pattern_grey.jpg", image)
plt.subplot(2, 2, 1)
plt.imshow(image, "gray")
plt.axis('off')
plt.title('Origin Image')
notch_filter(image)
plt.show()
```

结果如下


Origin Image
Fourier transform
Notch Filter
Reverse Fourier
0
250
500
750
1000
1250
0
500
1000
0
200
400
600
0
200
400
600