HW3

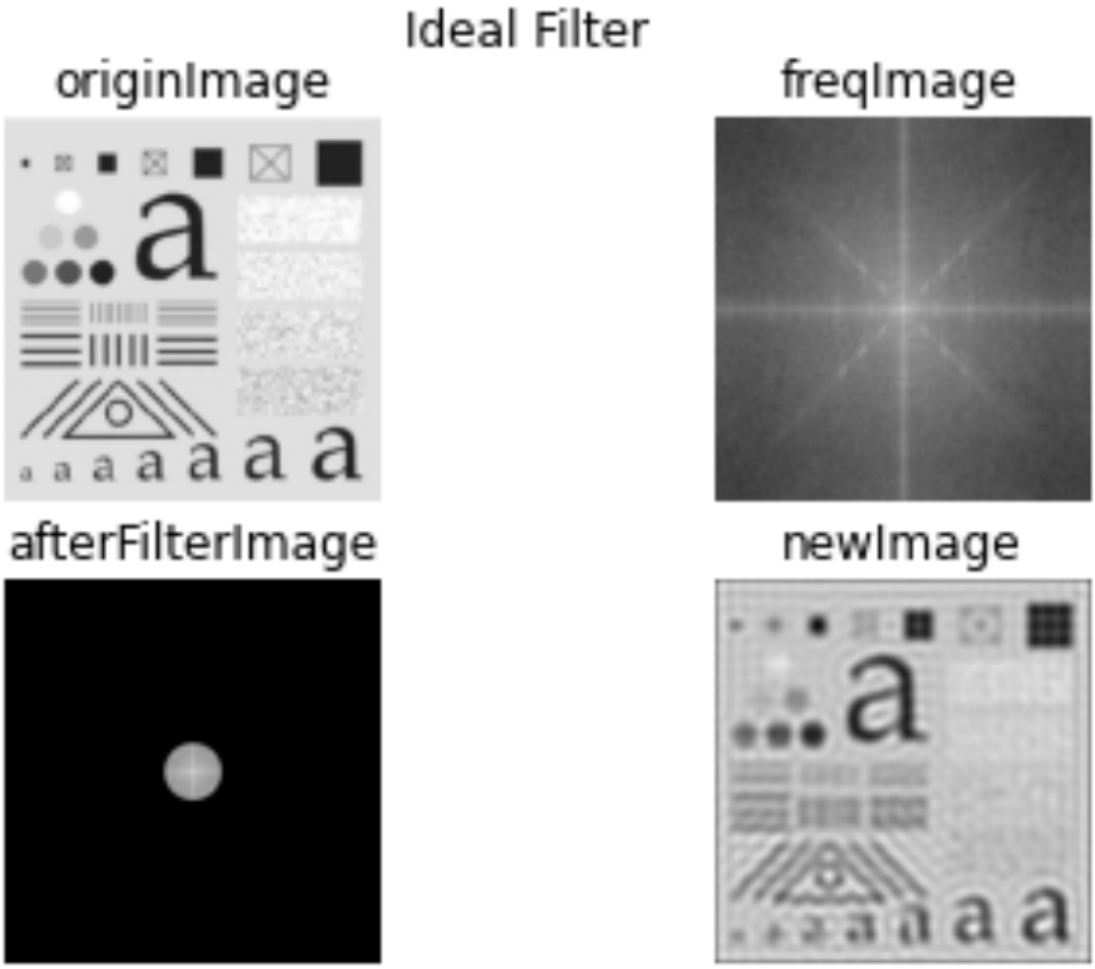
赵心怡 19307110452

1.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；

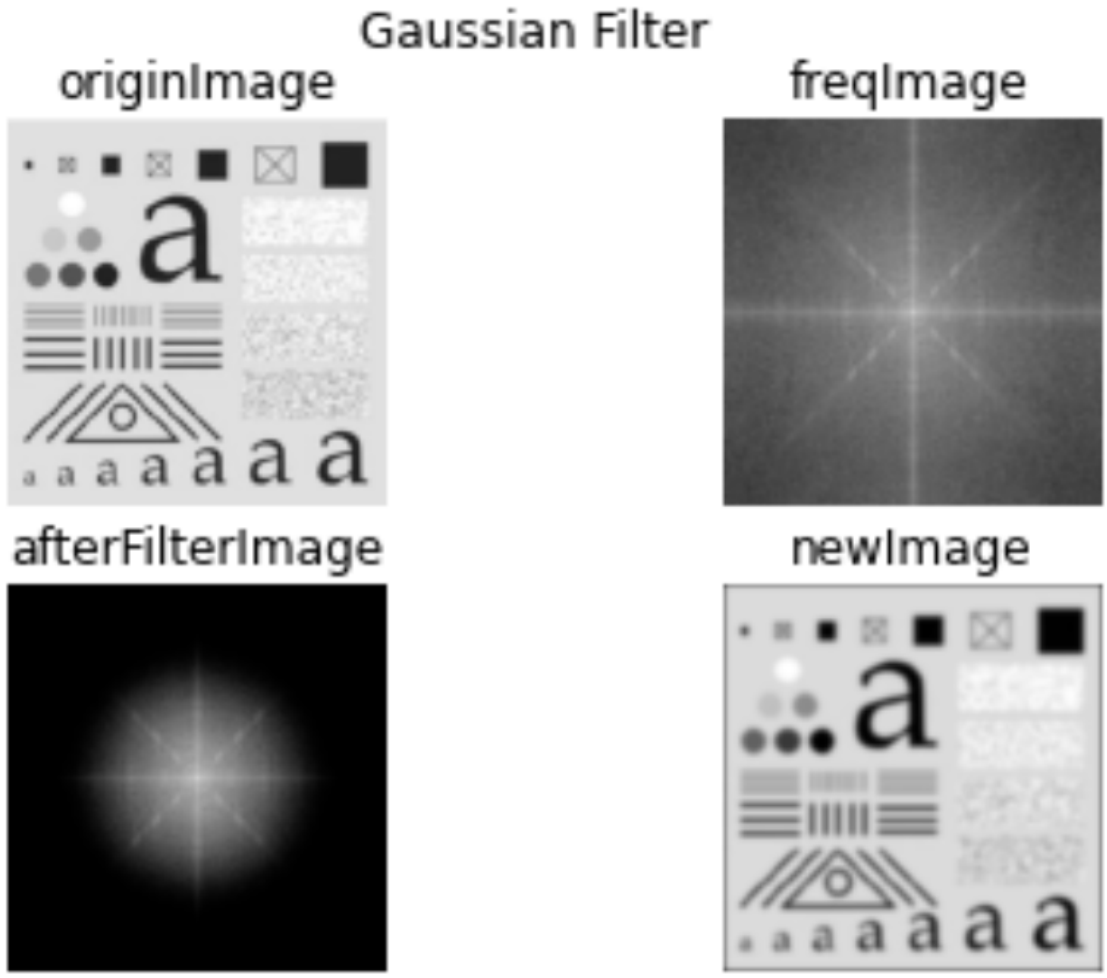
```
def smoothFilter(img, a, method):
    # 光滑滤波函数，len_patch是指核的大小
    height, width = img.shape
    paddingImg = np.pad(img, ((0, height), (0, width)), constant_values=0)
    new_height, new_width = height * 2, width * 2
    y, x = np.meshgrid(np.arange(new_width),np.arange(new_height) )  # 产生横坐标与纵坐标的网络
    centralized_op = (-1) ** (x + y)      # 中心化的算子
    freqImg = np.fft.fft2(paddingImg * centralized_op)  # 离散傅里叶变换
    filter = np.sqrt((y - width) ** 2 + (x - height) ** 2)    # 各位置距离中心点距离
    if method == 'Gauss':
        sigma = a
        filter = np.exp(-filter**2/(2*sigma**2)) # 高斯低通滤波器
    if method == 'Ideal':
        filter[np.where(filter <= a)] = 1
        filter[np.where(filter > a)] = 0      # 理想低通滤波器
    new_freqImg = freqImg * filter # 对频率域滤波
    filteredImg = np.real(np.fft.ifft2(new_freqImg)) * centralized_op    # 滤波后的图片
    newImg = filteredImg[0:height, 0:width] #截取前面m,n长度的图片
    newImg = (newImg - newImg.min()) / (newImg.max() - newImg.min()) * 255 # 归一化
    return freqImg, new_freqImg, newImg
```

理想滤波器下，a=50的结果



观察发现理想低通滤波器的特点是会让图像有明显的振铃现象。然后再选择method为高斯：

高斯滤波器,sigma=40:

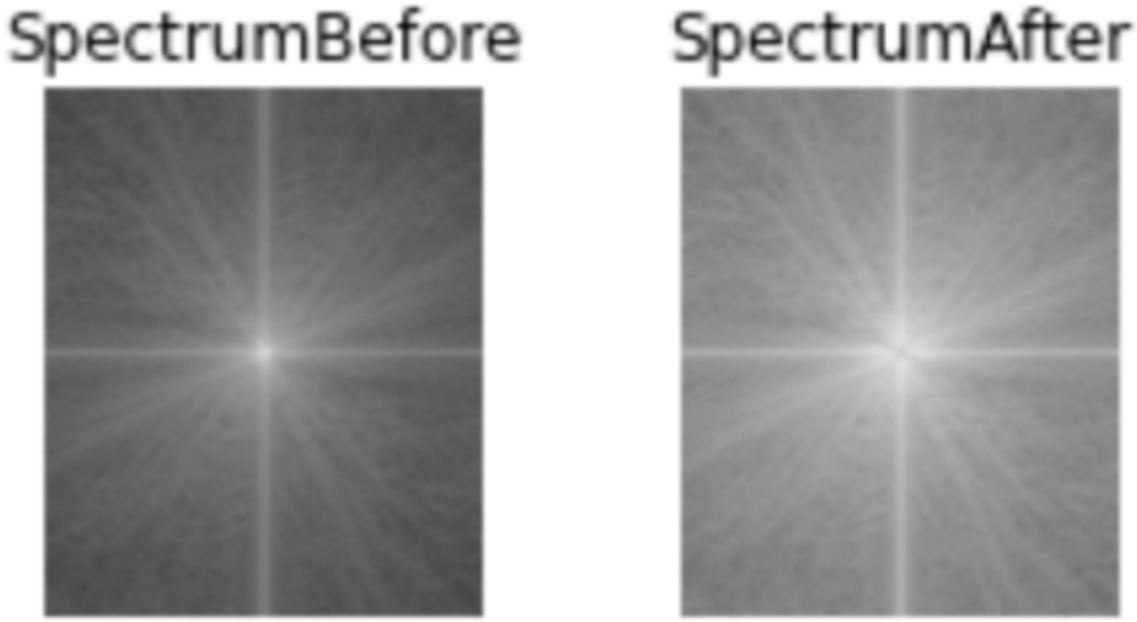


高斯模糊没有产生振铃现象，模糊效果较好。

（2）实现至少一种图像的锐化操作，该操作是基于频域操作的。

```
def sharpenFilter(img, a, method):
    # 光滑滤波函数，len_patch是指核的大小
    height, width = img.shape
    paddingImg = np.pad(img, ((0, height), (0, width)), constant_values=0)
    new_height, new_width = height * 2, width * 2
    y, x = np.meshgrid(np.arange(new_width),np.arange(new_height) )  # 产生横坐标与纵坐标的网络
    centralized_op = (-1) ** (x + y)      # 中心化的算子
    freqImg = np.fft.fft2(paddingImg * centralized_op)  # 离散傅里叶变换
    filter = np.sqrt((y - width) ** 2 + (x - height) ** 2)    # 各位置距离中心点距离
    if method == 'Gauss':
        sigma = a
        filter = 1 - np.exp(-filter**2/(2*sigma**2)) # 高斯高通滤波器
    if method == 'Ideal':
        filter[np.where(filter <= a)] = 0
        filter[np.where(filter > a)] = 1      # 理想高通滤波器
    new_freqImg = freqImg * filter # 对频率域滤波
    filteredImg = np.real(np.fft.ifft2(new_freqImg)) * centralized_op    # 滤波后的图片
    newImg = filteredImg[0:height, 0:width] #截取前面m,n长度的图片
    newImg1 = img + 1 * newImg # 这里令k=1，不同的k值比较
    return freqImg, new_freqImg, newImg, newImg1
```

高斯滤波器,a=20的结果，同时输出了原来的频谱图和锐化后的频谱图



不知道什么原因plot函数画的结果看起来变灰了，把图片保存下来打开看锐化的结果更明显。

通过变换后边界变得更清晰了

不同的sigma值会让锐化的效果不同，当sigma增大，锐化效果变得不明显。

2. 实现噪声的生成（不可以调用别的库实现的函数）

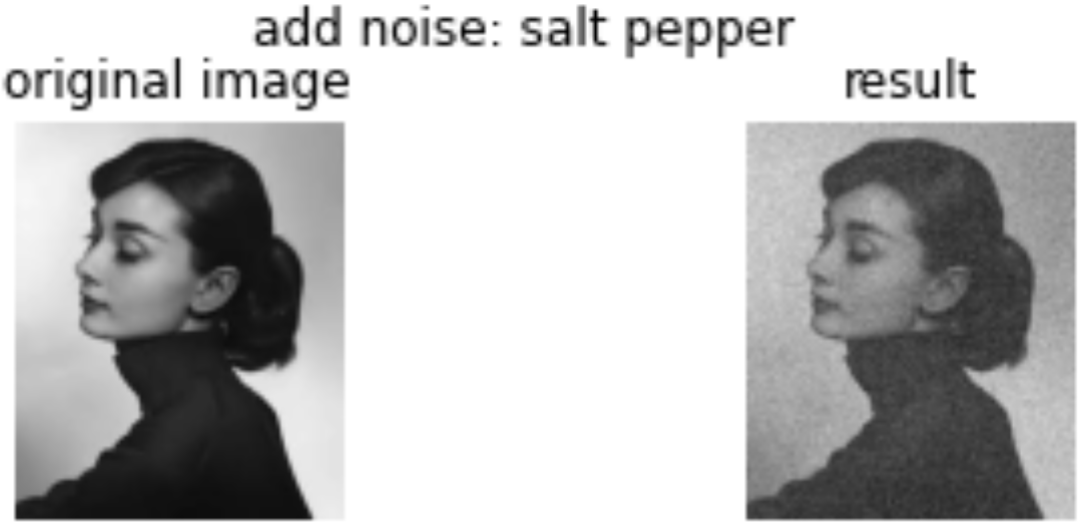
针对对大脑、心脏图像（或其他多类图像），生成两种不同类型、不同强度的噪声，并使用生成的噪声污染图像，对比展示噪声污染前后的图像：

白噪声

代码：

```
def white_noise(img, k):
    noise = np.random.normal(0,0.1,img.shape)
    newImg = noise*255+img #加白噪声，选取sigma为0.1
    newImg = (newImg-newImg.min())/(newImg.max()-newImg.min())*255#归一化
    return newImg
```

处理结果如图所示，可以看到噪点增加了



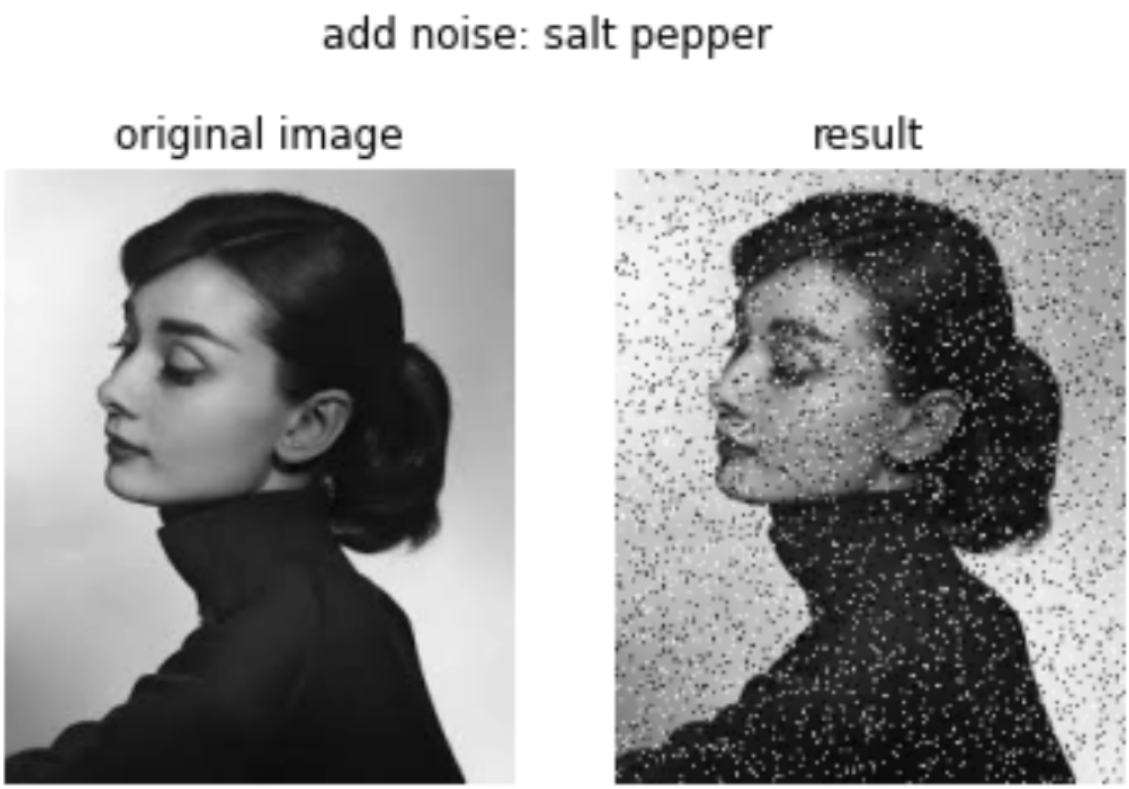
plt函数不清晰，保存图片结果：

椒盐噪声

代码：

```python
def salt_pepper(img,p1=0.05,p2=0.95):
    height, width = img.shape
    polluted_img = np.array(img)
    for i in range(height):
        for j in range(width): # 遍历每个像素点
            rand_num = random.random() # 生成0-1随机数
            if rand_num < p1:
                polluted_img[i,j] = 0 # 椒噪声
            elif rand_num > p2:
                polluted_img[i,j] = 2555 # 盐噪声
    return polluted_img
```

处理结果如图所示：当我们设置椒盐的比例为0.05的时候图像上已经有很多噪声点了，随着p值的增大，噪声点的数量也会变多。



3. 编程实现基于频域的选择滤波器方法，去除大脑CT体膜图像（Shepp-Logan）中的条纹；或自己设计一个有周期噪声的图片，并用频域选择滤波器去除噪声。

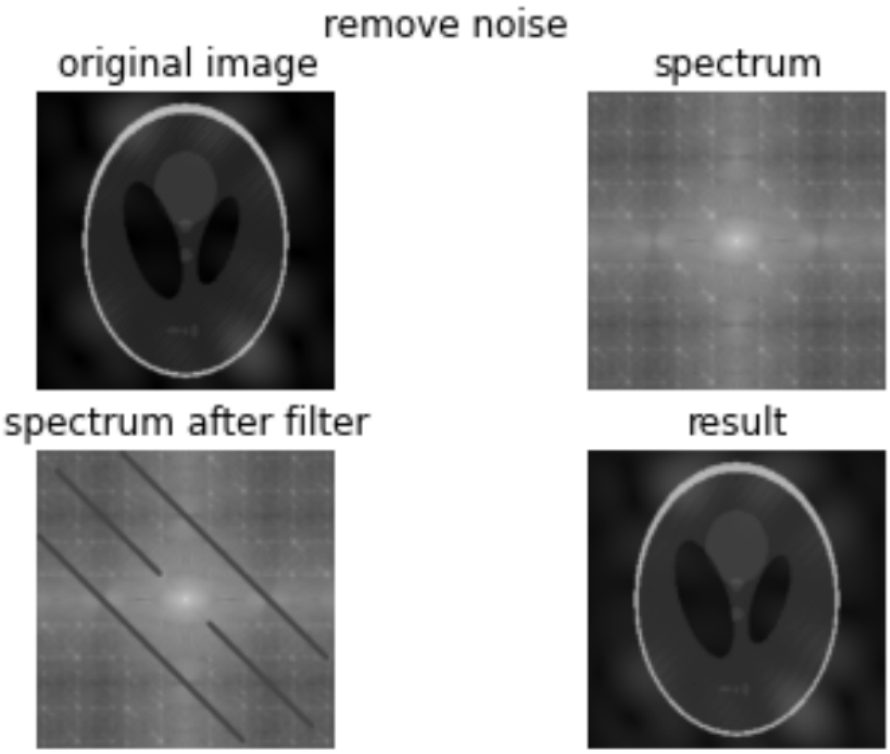
尝试了很多不同的消除噪声的方法，试着采用斜向的噪声消除方法。

```
def noiseRemove(img, a):
    height, width = img.shape
    paddingImg = np.pad(img, ((0, height), (0, width)), constant_values=0)
    new_height, new_width = height * 2, width * 2
    y, x = np.meshgrid(np.arange(new_width),np.arange(new_height) )  # 产生横坐标与纵坐标的网络
    centralized_op = (-1) ** (x + y)      # 中心化的算子
    freqImg = np.fft.fft2(paddingImg * centralized_op)  # 离散傅里叶变换
    f=freqImg
    for i in range(100,600,10):
        f=cover(f,i,i,a);
        f=cover(f,730+i,730+i,a); #中心两边的斜向噪声消除
    for i in range(12,1000,10):
        f=cover(f,i+400,i,a);
        f=cover(f,i,i+400,a); # 左下和右上的斜向噪声消除
    new_freqImg=f
    filteredImg = np.real(np.fft.ifft2(new_freqImg)) * centralized_op    # 滤波后的图片
    newImg = filteredImg[0:height, 0:width] #截取前面m,n长度的图片
    return freqImg, new_freqImg, newImg

def cover(img,x,y,k): #覆盖原有像素
```

```
    B=deepcopy(img);
    for i in range(x-k,x+k+1):
        for j in range(y-k,y+k+1):
            B[i,j]=250
    return B
```

图像处理结果，输出了频谱图和处理后的频谱图：



同样因为plt的画质太差，单独保存result为图片比较：

可以看到大部分噪声消除了， 但是还有一些边缘部分的噪声还存在。猜测单纯遮挡噪音还是有很大的局限性，尤其是当频谱图中的噪音较多时。