图像处理与可视化 Homework4 报告

林子开 21307110161

2023 年 11 月 10 日

目录

1 频率域滤波

1.1 平滑

1.1.1 频域滤波器：高斯低通滤波器（GLPF）

使用的高斯低通滤波器如下

$$H(u,v) = \exp[-D^2(u,v)/2\sigma^2]$$

其中

$$D^2(u,v) = (u-u_0)^2 + (v-v_0)^2$$

(u_0, v_0) 表示频域的中心点，σ 表示高斯滤波器的标准差，σ 越大，滤波后的频域图像会保留越多高频信号。

1.1.2 Python 代码实现

Python 代码实现频域低通滤波的代码如下，其中，**频域滤波的五个标准步骤：**

1. 填充

2. 中心化，DFT

3. 频域滤波

4. IDFT，去中心化

5. 提取左上象限图像

均已写在 smooth 函数中的注释中，敬请读者结合 Python 代码审阅。

Listing 1: Python codes for smoothing by GLPF

```python
import numpy as np
from PIL import Image
from math import exp


def logRescale(M:np.array) -> np.array:
    """先取模得到频谱，然后取对数，并映射到[0,255]区间，将映射后的频谱返回"""
    # m,n = M.shape
    M = np. abs(M) # 取模得到频谱
    M = np.log(M + 1) # 取对数
    largest = np. max(M)
    least = np. min(M)
    M = 255 * (M - least)/(largest - least)
    M = np.uint8(M) # 无符号整数可以表示0~255
    return M


def rescale(M:np.array) -> np.array:
    """映射到[0,255]区间"""
    # m,n = M.shape
    largest = np. max(M)
    least = np. min(M)
    M = 255 * (M - least)/(largest - least)
    M = np.uint8(M) # 无符号整数可以表示0~255
    return M


def smooth(pixels: np.array, sigam: float = 10) -> (np.array, np.array, np.array):
    """
    GLPF, high frequency are removed and low frequency remain,
    using the Gauss frequency filter
    :param: pixels are the indensity of all the pixels in your image
    :param: sigma is the standard deviation of your Gauss Kernel in frequency domain
    :return: three arrays: the original frequency F, the filtered frequency H, and the
        filtered figure g
    """
    # step 1: padding, double size
    m,n = pixels.shape
    P = np.zeros((2*m, 2*n))
    P[0:m, 0:n] = pixels # put the original figure on the left-top

    # step 2: preparation for centerization
```

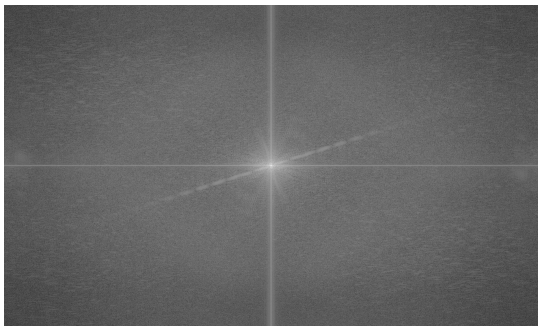
```
    S = np.ones((2*m, 2*n))
    for x in  range(1,S.shape[0],2):
        S[x,:] *= -1
    for y in  range(1,S.shape[1],2):
        S[:,y] *= -1
    P = P * S

    # step 3: DFT and make Hadarmard product with Gauss filter
    F = np.fft.fft2(P)
    H = np.empty(F.shape) # Gause filter
    center = np.array([H.shape[0]//2, H.shape[1]//2])
    for x in  range(H.shape[0]):
        for y in  range(H.shape[1]):
            D2 = (x-center[0])**2 + (y-center[1])**2
            H[x,y] = exp(-D2/(2*sigam**2))
    G = F*H

    # step 4: IDFT and decenterization
    g = np.fft.ifft2(G).real
    g = g*S
    g = g[0:m, 0:n]
    for x in  range(g.shape[0]):
        for y in  range(g.shape[1]): # truncate the intensity within [0,255]
            if g[x,y] > 255:
                g[x,y] = 255
            elif g[x,y] < 0:
                g[x,y] = 0

    # step 5: return the original frequency, the filtered frequency
    # and the filtered figure(on the left-top quadrant)
    return F, G, g


def main():
    myPicture = 'test.jpeg'
    im_raw = Image. open(myPicture)
    im = im_raw.convert('L')
    pixels = np.array(im)
    sigmas = [20,40,60,80,100]

    for sigma in sigmas:
        # 平滑处理
        orig_freq, filter_freq, smooth_pixels = smooth(pixels,sigma)

        # 对频谱取模，取对数，映射到[0,255]区间，并转换为int8类型
        orig_freq = logRescale(orig_freq)
        filter_freq = logRescale(filter_freq)

        # 将滤波后的像素映射到[0,255]，并转换为int8格式
        smooth_pixels = rescale(smooth_pixels)

        # 转成图像格式
```

```
94          freq_im = Image.fromarray(orig_freq)
95          filtered_freq_im = Image.fromarray(filter_freq)
96          smooth_im = Image.fromarray(smooth_pixels)
97
98          # 存储图像
99          freq_im.save('smooth_result_Gauss/'+'sigma='+ str(sigma)+' frequency'+'.png')
100         filtered_freq_im.save('smooth_result_Gauss/'+'sigma='+
                str(sigma)+' filtered frequency'+'.png')
101         smooth_im.save('smooth_result_Gauss/'+'sigma='+ str(sigma)+' filtered figure'+'.png')
102
103
104
105 if __name__ == '__main__':
106     main()
```

1.1.3 效果展示

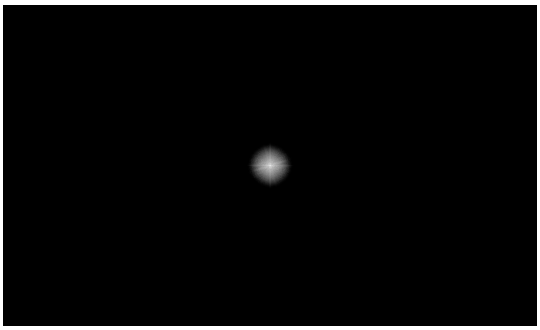
以下展示了原图的频谱、滤波后的频谱以及滤波后的效果。

(a) 原图



(b) 原图频谱（取对数）

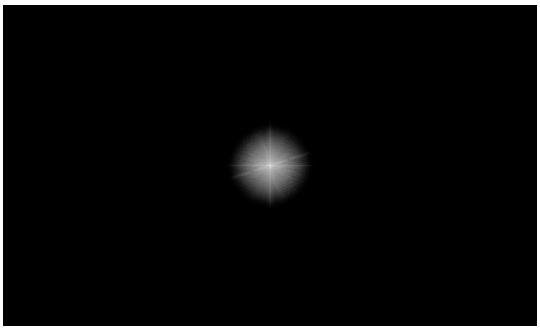
图 1: 原图与原图频谱



(a) 频域滤波结果（取对数）

(b) 滤波效果

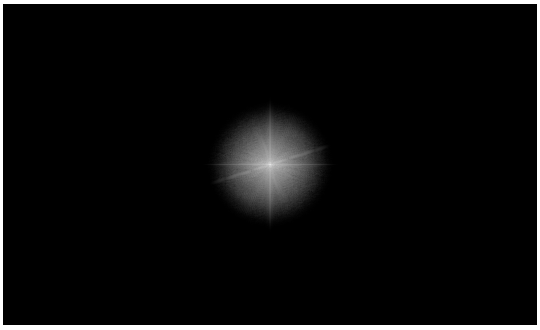
图 2: $\sigma = 20$ 的频域滤波结果与效果图



(a) 频域低通滤波结果（取对数）

(b) 滤波效果

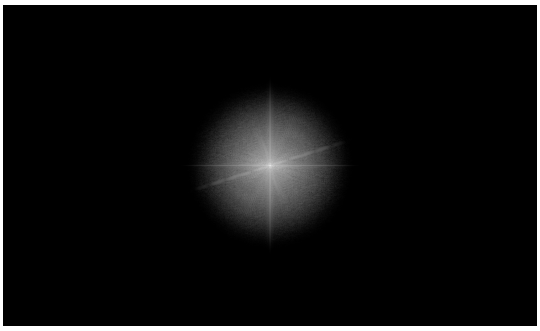
图 3: $\sigma = 40$ 的频域低通滤波结果与效果图



(a) 频域低通滤波结果（取对数）

(b) 滤波效果

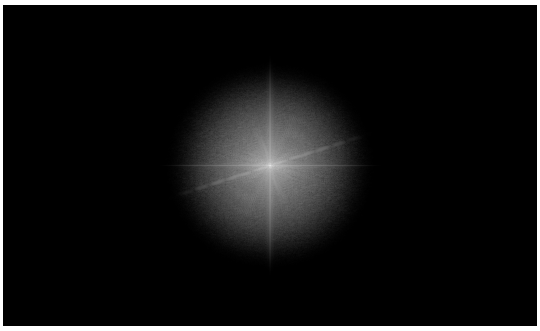
图 4: $\sigma = 60$ 的频域低通滤波结果与效果图



(a) 频域低通滤波结果（取对数）

(b) 滤波效果

图 5: $\sigma = 80$ 的频域低通滤波结果与效果图



(a) 频域低通滤波结果（取对数）

(b) 滤波效果

图 6: $\sigma = 100$ 的频域低通滤波结果与效果图

小结 可以明显看出，σ **越小，平滑的效果越好**。

1.2 锐化

1.2.1 频域滤波器：高斯高通滤波器（GHPF）

使用的高斯高通滤波器如下

$$H(u,v) = 1 - \exp[-D^2(u,v)/2\sigma^2] + c$$

其中

$$D^2(u,v) = (u-u_0)^2 + (v-v_0)^2$$

(u_0, v_0) 表示频域的中心点，σ 表示高斯滤波器的标准差，σ 越大，去除的低频信号就会越多。c 是一个常数，在本次实验中统一取 $c = 1$，该常数不会影响锐化效果，同时能保留原始的整体灰度值信息，使得滤波后的图像色调与原图尽可能一直。

1.2.2 Python 代码实现

实现锐化的 Python 代码如下，其中，**频域滤波的五个标准步骤都已经写在 sharpen 函数的注释中**，敬请读者结合代码进行审阅。

Listing 2: Python codes for sharpening by GHPF

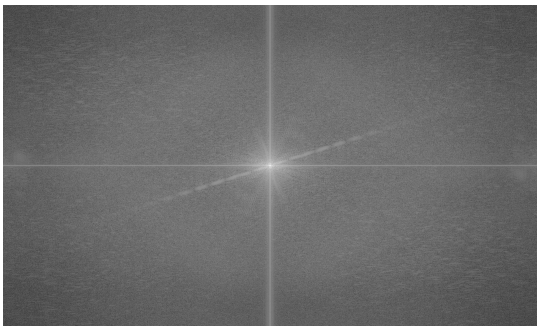
```python
import numpy as np
from numba import jit, njit  # 加速器
from PIL import Image
from math import exp


# @jit
def logRescale(M:np.array) -> np.array:
    """先取模得到频谱，然后取对数，并映射到[0,255]区间，将映射后的频谱转化为uint8后返回"""
    # m,n = M.shape
    M = np. abs(M) # 取模得到频谱
    M = np.log(M + 1) # 取对数
    largest = np. max(M)
    least = np. min(M)
    M = 255 * (M - least)/(largest - least)
```

```
16      M = np.uint8(M) # 无符号整数可以表示0~255
17      return M
18
19  # @jit
20  def rescale(M:np.array) -> np.array:
21      """映射到[0,255]区间，并转化为uint8格式返回"""
22      # m,n = M.shape
23      largest = np. max(M)
24      least = np. min(M)
25      M = 255 * (M - least)/(largest - least)
26      M = np.uint8(M) # 无符号整数可以表示0~255
27      return M
28
29  # @jit
30  def sharpen(pixels: np.array, sigam: float = 10, const = 1) -> (np.array, np.array, np.array):
31      """
32      GHPF, low frequency are removed and high frequency remain,
33      using the Gauss frequency filter
34      :param: pixels are the indensity of all the pixels in your image
35      :param: sigma is the standard deviation of your Gauss Kernel in frequency domain
36      :param: const is a constant added to the Gauss filter
37      :return: three arrays: the original frequency F, the filtered frequency H, and the
            filtered figure g
38      """
39      # step 1: padding, double size
40      m,n = pixels.shape
41      P = np.zeros((2*m, 2*n))
42      P[0:m, 0:n] = pixels # put the original figure on the left-top
43
44      # step 2: preparation for shift
45      S = np.ones((2*m, 2*n))
46      for x in  range(1,S.shape[0],2):
47          S[x,:] *= -1
48      for y in  range(1,S.shape[1],2):
49          S[:,y] *= -1
50      P = P * S
51
52      # step 3: DFT and make Hadarmard product with Gauss filter
53      F = np.fft.fft2(P)
54      H = np.empty(F.shape) # Gause filter
55      center = np.array([H.shape[0]//2, H.shape[1]//2])
56      for x in  range(H.shape[0]):
57          for y in  range(H.shape[1]):
58              D2 = (x-center[0])**2 + (y-center[1])**2
59              H[x,y] = 1 - exp(-D2/(2*sigam**2)) + const
60      G = F*H
61
62      # step 4: IDFT
63      g = np.fft.ifft2(G).real
64      g = g*S
65      g = g[0:m, 0:n]
66      for x in  range(g.shape[0]):
```

```
        for y in  range(g.shape[1]):
            # 截断法
            if g[x,y] > 255:
                g[x,y] = 255
            elif g[x,y] < 0:
                g[x,y] = 0

    # step 5: return the original frequency, the filtered frequency and the filtered figure
    return F, G, g


def main():
    myPicture = 'test.jpeg'
    im_raw = Image. open(myPicture)
    im = im_raw.convert('L')
    pixels = np.array(im)
    sigmas = [1]
    const = 1

    for sigma in sigmas:
        # 锐化处理
        orig_freq, filter_freq, sharpen_pixels = sharpen(pixels,sigma,const)

        # 对频谱取模，取对数，映射到[0,255]区间，并转换为int8类型
        orig_freq = logRescale(orig_freq)
        filter_freq = logRescale(filter_freq)

        # 将滤波后的像素映射到[0,255]，并转换为uint8格式
        sharpen_pixels = rescale(sharpen_pixels)

        # 转成图像格式
        freq_im = Image.fromarray(orig_freq)
        filtered_freq_im = Image.fromarray(filter_freq)
        sharpen_im = Image.fromarray(sharpen_pixels)

        # 存储图像
        freq_im.save('sharpen_result_Gauss/'+'sigma='+ str(sigma)+' c='+
            str(const)+' frequency'+'.png')
        filtered_freq_im.save('sharpen_result_Gauss/'+'sigma='+ str(sigma)+' c='+
            str(const)+' filtered frequency'+'.png')
        sharpen_im.save('sharpen_result_Gauss/'+'sigma='+ str(sigma)+' c='+
            str(const)+' filtered figure'+'.png')


if __name__ == '__main__':
    main()
```

1.2.3 效果展示

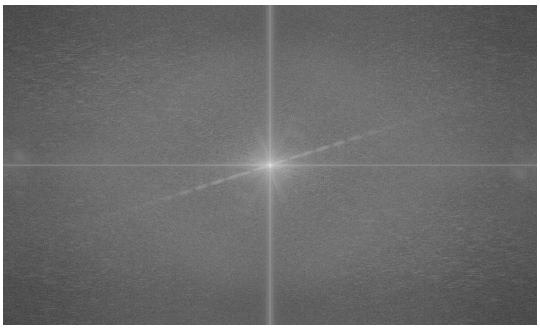
仍然使用平滑操作中的图1(a)作为测试用图。不同的 σ 对应锐化效果如下



(a) 频域高通滤波结果（取对数）

(b) 滤波效果

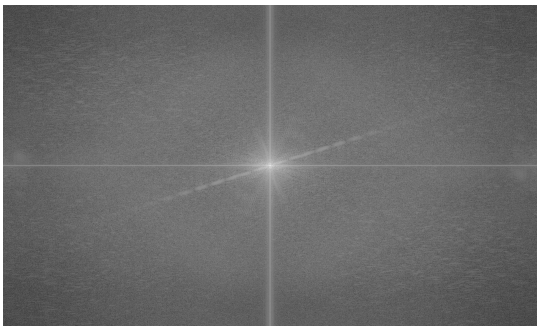
图 7: $\sigma = 1$ 的频域高通滤波结果与效果图



(a) 频域高通滤波结果（取对数）

(b) 滤波效果

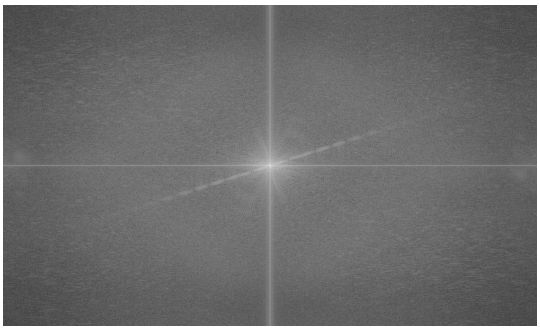
图 8: $\sigma = 2$ 的频域高通滤波结果与效果图



(a) 频域高通滤波结果（取对数）

(b) 滤波效果

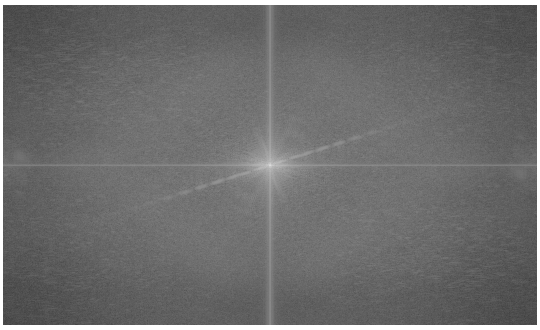
图 9: $\sigma = 5$ 的频域高通滤波结果与效果图



(a) 频域高通滤波结果（取对数）

(b) 滤波效果

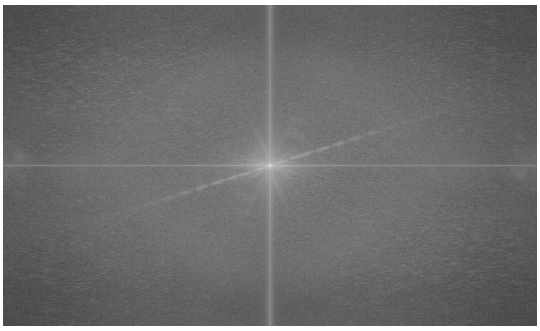
图 10: $\sigma = 10$ 的频域高通滤波结果与效果图



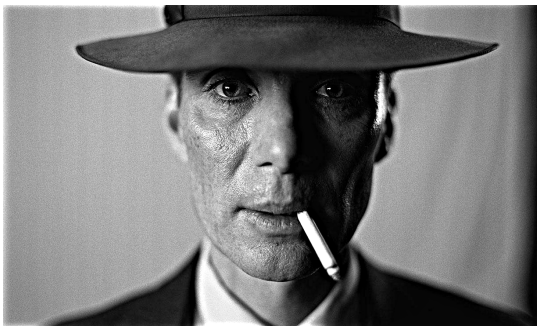
(a) 频域高通滤波结果（取对数）

(b) 滤波效果

图 11: $\sigma = 20$ 的频域高通滤波结果与效果图



(a) 频域高通滤波结果（取对数）



(b) 滤波效果

图 12: $\sigma = 40$ 的频域高通滤波结果与效果图

小结 可以看出，**随着 σ 的增大，锐化的效果会越来越好**，越来越多的细节会被展现出来，但同时，当 $\sigma > 10$ 之后，锐化增强的程度逐渐下降，边际效应显著，这是因为频域中的高频信号区域强度弱，而低频信号（频谱中心附近）区域的强度很强；当增大 σ 的时候，虽然能过滤掉更大区域内的低频信号，但是当过滤区域从低频区扩展到高频区时，由于高频区域强度小，则此时过滤和不过滤高频区的效果差异已经并不显著。

2 去除大脑 CT 体膜图像中的条纹

2.1 频域变换

体膜图的频域变换结果如下图13所示

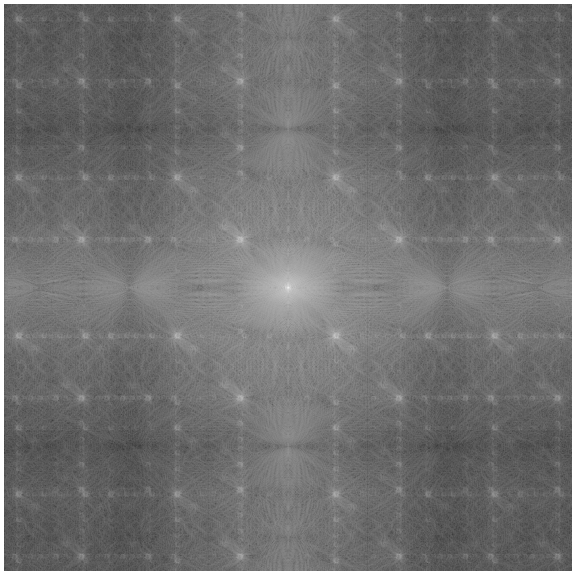


图 13: 大脑 CT 体膜图像的频域变换结果

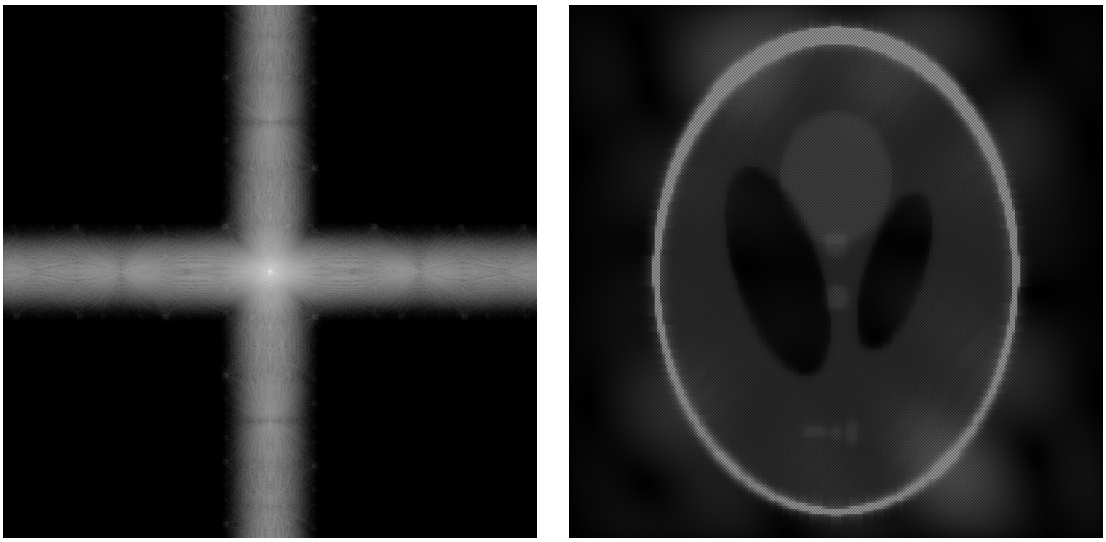
注意到在频域的四个象限中，均有对称的网格状噪点，对应着原图的周期性噪音。我考虑的方案时：仅保留中央的十字区域信号，而将四个象限中存在噪点的区域的信号全部舍弃。又考虑到如果直接用 box 类函数会出现振铃效应（ringing effect），因此考虑使用高斯类的带通滤波器进行滤波。

我设计的频域带通滤波器如下：

$$H(u,v) = \exp[-(u-u_0)^2/2\sigma_1^2] + \exp[-(v-v_0)^2/2\sigma_2^2]$$

其中 (u_0, v_0) 表示频域图像的中心。

经过尝试，我发现取 $\sigma_1 = 28$, $\sigma_2 = 28$ 会有比较良好的滤波效果，效果如图14所示。



(a) 带通滤波结果（取对数）　　(b) 滤波效果

图 14: 频域带通滤波结果与效果图

2.2 Python 代码实现

Listing 3: Python codes for removing noise

```python
import numpy as np
# from numba import jit, njit  # 加速器
from PIL import Image
from math import exp


# @jit
def logRescale(M:np.array) -> np.array:
    """先取模得到频谱，然后取对数，并映射到[0,255]区间，将映射后的频谱返回"""
    # m,n = M.shape
    M = np. abs(M) # 取模得到频谱
    M = np.log(M + 1) # 取对数
    largest = np. max(M)
    least = np. min(M)
    M = 255 * (M - least)/(largest - least)
    M = np.uint8(M) # 无符号整数可以表示0~255
    return M

# @jit
def rescale(M:np.array) -> np.array:
    """映射到[0,255]区间"""
    # m,n = M.shape
    largest = np. max(M)
    least = np. min(M)
    M = 255 * (M - least)/(largest - least)
    M = np.uint8(M) # 无符号整数可以表示0~255
    return M

# @jit
```

```
30 def removeNoise(pixels: np.array) -> (np.array, np.array, np.array):
31     """
32     remove noise from the target noisy picture provided by Prof. Zhuang
33     :param: pixels are the indensity of all the pixels in the noise image
34     :return: three arrays: the original frequency F, the filtered frequency G, and the
           filtered figure
35     """
36     # step 1: padding, double size
37     m,n = pixels.shape
38     P = np.zeros((2*m, 2*n))
39     P[0:m, 0:n] = pixels # put the original figure on the left-top
40 
41     # step 2: preparation for shift
42     S = np.ones((2*m, 2*n))
43     for x in  range(1,S.shape[0],2):
44         S[x,:] *= -1
45     for y in  range(1,S.shape[1],2):
46         S[:,y] *= -1
47     P = P * S
48 
49     # step 3: DFT and make Hadarmard product with a special filter
50     F = np.fft.fft2(P)
51     # print(F.shape)
52 
53     H = np.zeros(F.shape)
54     # F_log = logRescale(F)
55     H = np.empty(F.shape) # Gause filter
56     center = np.array([H.shape[0]//2, H.shape[1]//2])
57     sigma = 28
58     for x in  range(H.shape[0]):
59         for y in  range(H.shape[1]):
60             H[x,y] = exp(-(x-center[0])**2/(2*sigma**2)) + exp(-(y-center[1])**2/(2*sigma**2))
61 
62 
63     G = F*H
64 
65     # step 4: IDFT
66     g = np.fft.ifft2(G).real
67     # g = g*S
68     g = g[0:m, 0:n]
69     for x in  range(g.shape[0]):
70         for y in  range(g.shape[1]):
71             if g[x,y] > 255:
72                 g[x,y] = 255
73             elif g[x,y] < 0:
74                 g[x,y] = 0
75     # step 5: return the original frequency, the filtered frequency and the filtered figure
76     return F, G, g
77 
78 
79 def main():
80     myPicture = '作业四 图像.PNG'
```

```
    im_raw = Image. open(myPicture)
    im = im_raw.convert('L')
    pixels = np.array(im)
    # print(pixels.shape)

    # 去噪
    orig_freq, filter_freq, p = removeNoise(pixels)

    # 对频域结果取模，取对数，映射到[0,255]区间，并转换为int8类型
    orig_freq = logRescale(orig_freq)
    filter_freq = logRescale(filter_freq)

    # 频谱转成图像格式
    freq_im = Image.fromarray(orig_freq)
    filtered_freq_im = Image.fromarray(filter_freq)
    # 存储频域图像
    freq_im.save('remove_noise_result/'+'frequency'+'.png')
    filtered_freq_im.save('remove_noise_result/'+'filtered frequency'+'.png')

    # 存储去噪后的图像
    filtered_pixels = rescale(p)
    filtered_im = Image.fromarray(filtered_pixels)
    # 存储图像
    filtered_im.save('remove_noise_result/'+'result.png')


if __name__ == '__main__':
    main()
```