《生物医学图像处理实验》实验报告

(2019-2020 学年第 2 学期)

实验三：图像的空域滤波

课程号：310025010　　**课序号**：01　　**任课教师**：林江莉　　**成绩**：

组长：徐广玄　　**小组成员**：无

实验日期：　2023 年 4 月 6 日

一、实验内容：

编写程序对灰度和彩色图像进行空域滤波相关的各种操作，熟悉对图像的各种滤波操作，添加噪声、滤波除噪、边缘检测等。

1.编程给图片加入椒盐噪音和脉冲噪音。（选作：加入高斯噪音、高斯白噪音）

2.编程实现均值、中值、最大值滤波，（注意：滤波器模板大小可调）并研究相应算法去除图 1 所示的图片噪音。

3.编程实现两种边缘检测的算法（注意：滤波器模板大小可调）。并研究相应实验图 2 的边沿检测方法。

4.综合滤波、边源检测和灰度变换等算法，按图 3 所示的步骤增强图像。

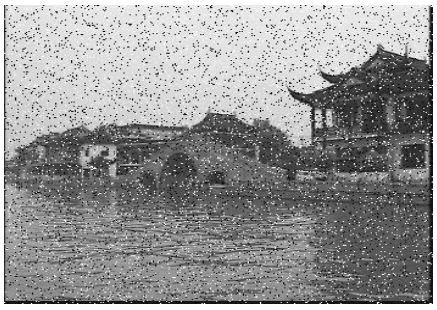


图 1

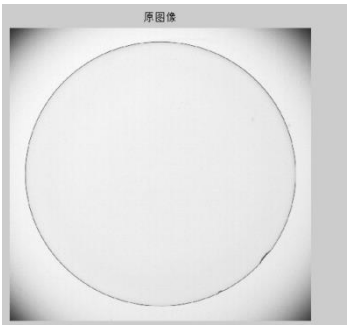


图 2

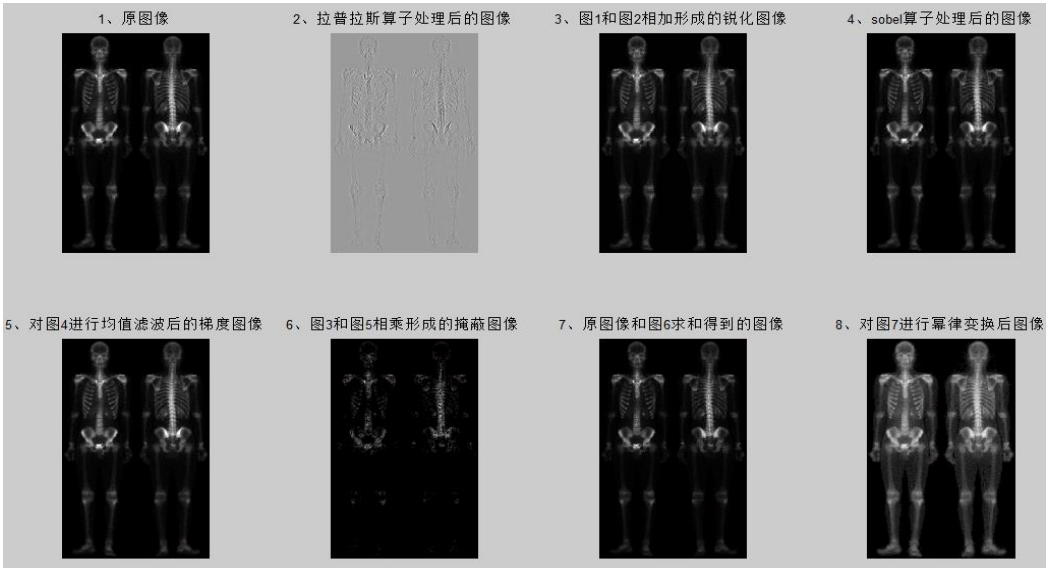


图 3

二、编程软件：

Pycharm 环境下 Python3.9，不使用 openCV。

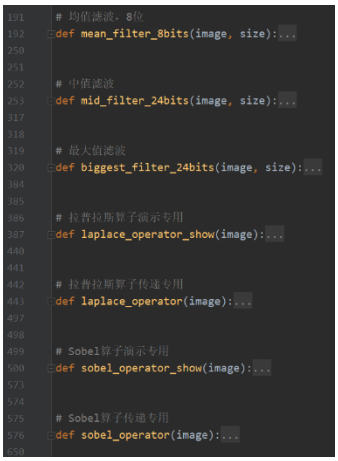
三、 实验步骤、结果及关键代码：

概况：

本实验的整体框架为如下所示：

```
# -*- coding: utf-8 -*-
import matplotlib.pyplot as plt  # plt 用于显示图片
import matplotlib.image as mpimg  # mpimg 用于读取图片
import numpy as np
import random  # 用于产生高斯噪声

# 读取目标路径
image_filter = mpimg.imread('./noise.bmp')
image_noise = mpimg.imread('./Baboon.bmp')
image_edge = mpimg.imread('./contact_lens_original.bmp')
image_skeleton = mpimg.imread('./Fig0343(a)(skeleton_orig).bmp')


# 加入椒盐噪声
def add_salt_and_pepper_noise(image, SNR):...


# 脉冲噪声
def add_pulse_noise(image, SNR, alpha):...


# 增加高斯噪音
def add_gaussian_noise(image, mean, var):...


# 均值滤波，24位
def mean_filter_24bits(image, size):...
```

图1 整体框架1



```
# 均值滤波，8位
def mean_filter_8bits(image, size):...


# 中值滤波
def mid_filter_24bits(image, size):...


# 最大值滤波
def biggest_filter_24bits(image, size):...


# 拉普拉斯算子演示专用
def laplace_operator_show(image):...


# 拉普拉斯算子传递专用
def laplace_operator(image):...


# Sobel算子演示专用
def sobel_operator_show(image):...


# Sobel算子传递专用
def sobel_operator(image):...
```

图2 整体框架2

图3 整体框架3

椒盐噪声代码：

随机取0或255的噪声点，核心代码如下：

```
# 判断噪声是否合理
if 0 <= SNR <= 1:
    print("这是一个合理的椒盐函数噪声范围")
else:
    print("Error:这不是一个合理的椒盐函数噪声范围")
    exit()

# 算法
noise_num = int((1 - SNR) * img_width * img_height)

for i in range(noise_num):
    rand_x = random.randint(0, img_height - 1)
    rand_y = random.randint(0, img_width - 1)

    if random.randint(0, 1) == 0:
        img[rand_x][rand_y] = 0
    else:
        img[rand_x][rand_y] = 255
```

图4 椒盐噪声代码

脉冲噪声代码：

随机取自增 α 的噪声增益，核心代码如下：

```python
# 判断噪声是否合理
if 0 <= SNR <= 1:
    print("这是一个合理的冲激函数噪声范围")
else:
    print("Error:这不是一个合理的冲激函数噪声范围")
    exit()

# 算法
noise_num = int((1 - SNR) * img_width * img_height)

for i in range(noise_num):
    rand_x = random.randint(0, img_height - 1)
    rand_y = random.randint(0, img_width - 1)

    img[rand_x][rand_y] += int(img[rand_x][rand_y] * alpha)
    if img[rand_x][rand_y] < 0:
        img[rand_x][rand_y] = 0
    elif img[rand_x][rand_y] > 255:
        img[rand_x][rand_y] = 255
```

图5 脉冲噪声代码

高斯噪声代码：

随机正态分布的的噪声点，核心代码如下：

```python
# 初始化变量
img = image
mean = mean
var = var
img_height = img.shape[0]
img_width = img.shape[1]
noise = np.random.normal(mean, var ** 0.5, img.shape)

for j in range(img_height):
    for k in range(img_width):
        img[j][k] += noise[j][k]

        if img[j][k] < 0:
            img[j][k] = 0
        elif img[j][k] > 255:
            img[j][k] = 255
```

图6 高斯噪声代码

均值滤波代码：

取奇数维度的均值核，核心代码如下：

```
# 算法
range_h = img_height + 2 * extend - size + 1
range_w = img_width + 2 * extend - size + 1
cnt = 0

for c in range(3):
    for j in range(range_h):
        for k in range(range_w):
            for a in range(size):
                for b in range(size):
                    cnt += img_right[j+a][k+b][c] * val

            img_right[j+extend][k+extend][c] = int(cnt)

            if img_right[j+extend][k+extend][c] < 0:
                img_right[j+extend][k+extend][c] = 0
            elif img_right[j+extend][k+extend][c] > 255:
                img_right[j+extend][k+extend][c] = 255

            cnt = 0
```

图7 均值滤波代码

中值滤波代码：

取奇数维度的排序中值，核心代码如下：

```
# 算法
range_h = img_height + 2 * extend - size + 1
range_w = img_width + 2 * extend - size + 1
mid_num = int(size ** 2 / 2)

for c in range(3):
    for j in range(range_h):
        for k in range(range_w):
            for a in range(size):
                for b in range(size):
                    mid_list.append(img_right[j+a][k+b][c])

            mid_list.sort()
            img_right[j+extend][k+extend][c] = int(mid_list[mid_num])

            if img_right[j+extend][k+extend][c] < 0:
                img_right[j+extend][k+extend][c] = 0
            elif img_right[j+extend][k+extend][c] > 255:
                img_right[j+extend][k+extend][c] = 255

            mid_list = []
```

图8 中值滤波代码

最大值滤波代码：

取奇数维度的排序最大值，核心代码如下：

```
# 算法
range_h = img_height + 2 * extend - size + 1
range_w = img_width + 2 * extend - size + 1
biggest_num = int(size ** 2 / 2)

for c in range(3):
    for j in range(range_h):
        for k in range(range_w):
            for a in range(size):
                for b in range(size):
                    biggest_list.append(img_right[j+a][k+b][c])

            biggest_list.sort()
            img_right[j+extend][k+extend][c] = int(biggest_list[biggest_num])

            if img_right[j+extend][k+extend][c] < 0:
                img_right[j+extend][k+extend][c] = 0
            elif img_right[j+extend][k+extend][c] > 255:
                img_right[j+extend][k+extend][c] = 255

            biggest_list = []
```

图9 最大值滤波代码

拉普拉斯算子代码：

首先定义拉普拉斯算子的等效卷积核：

```
# 初始化变量
img = image
size = 3
kernel = np.array([[0, 1, 0], [1, -4, 1], [0, 1, 0]])
extend = int(size / 2)
```

图10 拉普拉斯算子的等效卷积核

然后做卷积相减二次导数的操作，核心代码如下：

```
# 算法
range_h = img_height + 2 * extend - size + 1
range_w = img_width + 2 * extend - size + 1
cnt = 0

for j in range(range_h):
    for k in range(range_w):
        for a in range(size):
            for b in range(size):
                cnt += img_right[j + a][k + b] * kernel[a][b]

        img_right[j + extend][k + extend] -= int(cnt)

        if img_right[j + extend][k + extend] < 0:
            img_right[j + extend][k + extend] = 0
        elif img_right[j + extend][k + extend] > 255:
            img_right[j + extend][k + extend] = 255

        cnt = 0
```

图11 拉普拉斯算子的核心代码

Sobel算子代码：

首先定义Sobel算子的等效卷积核（水平和竖直）：

```
# 初始化变量
img = image
size = 3
kernel_vertical = np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]])
kernel_horizontal = np.array([[-1, -2, -1], [0, 0, 0], [1, 2, 1]])
extend = int(size / 2)
```

图12 Sobel算子的等效卷积核

对卷积后的两个矩阵做权重一致的相加，核心代码如下：

```
for j in range(range_h):
    for k in range(range_w):
        for a in range(size):
            for b in range(size):
                cnt += img_right[j + a][k + b] * kernel_horizontal[a][b]

        img_right[j + extend][k + extend] += int(cnt)

        if img_right[j + extend][k + extend] < 0:
            img_right[j + extend][k + extend] = 0
        elif img_right[j + extend][k + extend] > 255:
            img_right[j + extend][k + extend] = 255

        cnt = 0

for j in range(range_h):
    for k in range(range_w):
        for a in range(size):
            for b in range(size):
                cnt += img_right_copy[j + a][k + b] * kernel_vertical[a][b]

        img_right_copy[j + extend][k + extend] += int(cnt)

        if img_right_copy[j + extend][k + extend] < 0:
            img_right_copy[j + extend][k + extend] = 0
        elif img_right_copy[j + extend][k + extend] > 255:
            img_right_copy[j + extend][k + extend] = 255

        cnt = 0
```

图13 Sobel算子的核心代码1

```
img_right = np.divide(np.add(img_right, img_right_copy), 2)
```

图14 Sobel算子的核心代码2

医学图像处理代码：

按照步骤调用之前所写的函数，核心代码如下：

```python
# 医学图像处理
def med_img_pro(image):
    img_ori = extend_matrix(image, (3, 3))
    img_laplace = laplace_operator(image)
    img_add1 = np.add(img_ori, img_laplace)
    img_sobel = sobel_operator(img_add1)
    img_mean_filter = mean_filter_8bits(img_sobel, (3, 3))
    img_add1 = extend_matrix(img_add1, (5, 5))
    img_mul = np.multiply(img_add1, img_mean_filter)
    img_ori = extend_matrix(img_ori, (5, 5))
    img_add2 = np.add(img_ori, img_mul)
    img_final = power_trans(img_add2, 1, 0.5)
    plt.subplot(4, 5, 13)
    plt.imshow(img_final, cmap='gray')
    plt.title("final version")
```

图15 医学图像处理的核心代码

四、实验结果：

使用 plt.subplot()方法绘制图片如下：

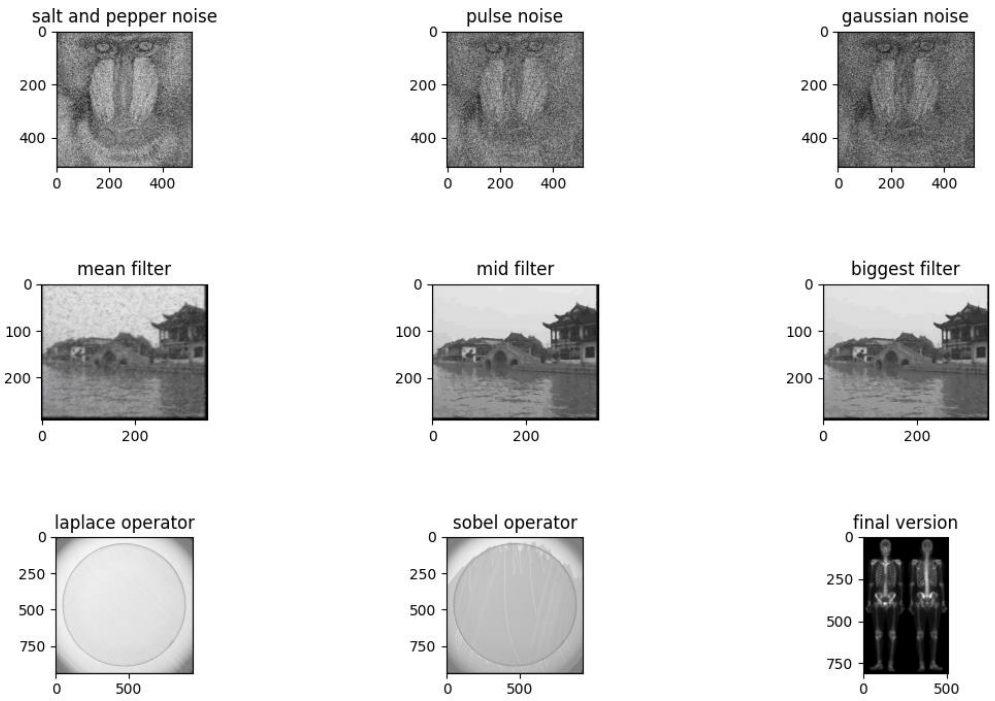


图16 实验结果的图片合集

五、结果分析及实验小结：

编程中问题及其解决：

1. 发现拉普拉斯算子需要手写二阶导数算法，但是结果发现其实可以等效为一个卷积

核操作。

2. 对图像进行卷积操作时需要扩大矩阵的边界 M-N+1 条，一开始一直报错，最后使用 numpy.insert()方法解决。

编程中的注意事项：

1. 对于最后的医学图像处理来说，每次经过卷积核处理都会自增边界，需要增加边界函数来使得二者做加减乘除运算时保持矩阵大小一致。