数字图像处理-作业 2

程铭

2020 年 3 月 29 日

1 三种不同模板大小的滤波算法处理结果及分析

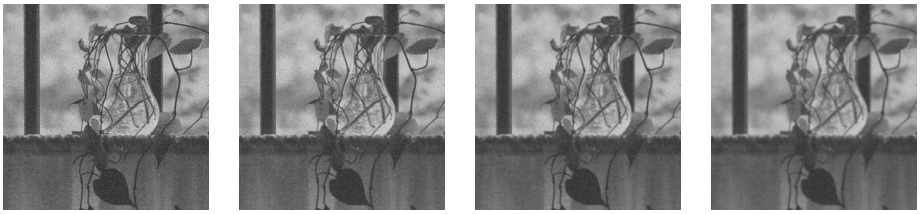
- 高斯滤波算法的实现原理在作业 1-1 中已经阐述。
- 均值滤波算法是线性滤波器，将一个窗口区域中的像素计算平均值，然后将计算得到的平均值作为锚点上的像素值。
- 中值滤波算法是非线性滤波，取当前卷积核所覆盖的像素的中值作为锚点的像素值。

对于两种噪声图像（高斯噪声、椒盐噪声），分别用三种滤波算法（高斯滤波、均值滤波、中值滤波）且大小为 3x3, 5x5, 7x7, 11x11 的模板进行滤波处理，结果如图 1, 2 所示。

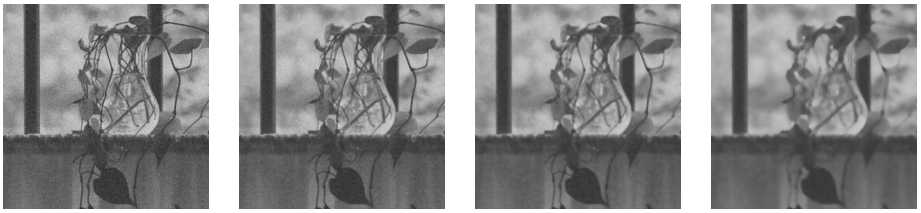
噪声的角度分析：

如图 1所示，对于高斯噪声，由于其分布遵循高斯分布，因此使用高斯滤波算法可以最有效地去除噪声，而中值滤波和均值滤波的表现相对不好。由图可以看到，当卷积模板较小 (3x3) 时，三种滤波算法的表现都不好，随着模板增大，噪声的去除效果变好，但同时，图像本身的细节和特征也变得模糊。

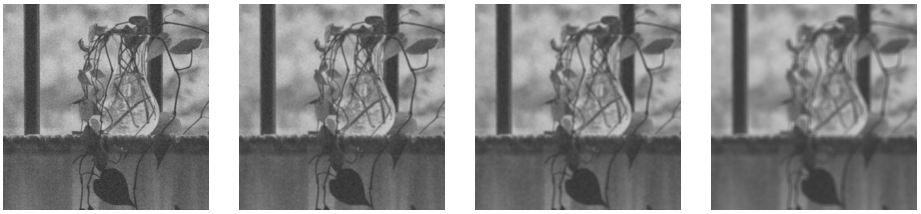
如图 2所示，对于椒盐噪声，使用中值滤波算法可以最有效地去除噪声，而高斯滤波算法和均值滤波算法均表现不好。由于椒盐噪声是在图像



(a) 高斯滤波 3x3 (b) 高斯滤波 5x5 (c) 高斯滤波 7x7 (d) 高斯滤波 11x11



(e) 中值滤波 3x3 (f) 中值滤波 5x5 (g) 中值滤波 7x7 (h) 中值滤波 11x11



(i) 均值滤波 3x3 (j) 均值滤波 5x5 (k) 均值滤波 7x7 (l) 均值滤波 11x11

图 1: 三种滤波算法对高斯噪声的滤波结果

上随机出现的黑色（椒）、白色（盐）的像素，因此噪声的均值不为 0，不能用高斯滤波和均值滤波处理。另外，由于图像中既有被黑白色污染的点，也有未被污染的干净的点，因此利用中值滤波算法采用中间点代替，可以实现将污染点用干净的点代替，从而实现较好的滤波效果。另外，对于中值滤波算法，在卷积模板较小（3x3）时，便有很好的处理效果，大部分噪声点可以得到去除；而对于另外两种算法，尽管模板较大，处理噪声的效果仍不好。

图像细节的角度分析：

如图 1和 2的第一行图片所示，由于高斯滤波算法在各个方向上的平滑程度相同（高斯函数的对称性），经过高斯滤波的图像的边缘走向不会被改

(a) 高斯滤波 3x3 (b) 高斯滤波 5x5 (c) 高斯滤波 7x7 (d) 高斯滤波 11x11

(e) 中值滤波 3x3 (f) 中值滤波 5x5 (g) 中值滤波 7x7 (h) 中值滤波 11x11

(i) 均值滤波 3x3 (j) 均值滤波 5x5 (k) 均值滤波 7x7 (l) 均值滤波 11x11

图 2: 三种滤波算法对椒盐噪声的滤波结果

变。另外，由于高斯滤波采用“加权平均”的思想，因此高斯滤波处理后的图片的细节会丢失，导致模糊。除此之外，由于高斯滤波器是低通滤波器，因此图像不会受到高频信号的污染和影响。

对于中值滤波算法（1和 2的第二行图片），由于其是非线性的滤波算法，因此它可以有效地保留图片的边缘线，图像的细节也可以得到较好的保留。

对于均值滤波算法，由于其对模板内的像素取平均，因此会导致图像的细节和边缘较为模糊、图像的特征信息会丢失。但均值滤波算法实现简

单（1和 2的第三行图片）。

不同模板大小的比较分析：

在图 1和 2中横向比较，我们可以发现：对于三种滤波算法，随着卷积模板不断增大，对噪声的处理效果越好，但同时图片也变得更加模糊、细节丢失更多。因此我们应该权衡卷积模板尺寸的利弊，选取合适大小的模板。

2 理想滤波子难以实现的原因

理想的噪声滤波子即为在去除噪声的同时，图像的细节、内容等不受影响。个人分析认为理想滤波子难以实现的原因有以下两点：

1. 由于在图像中，噪声和图像内容往往不是分开存在的，是互有关联的，噪声像素嵌入在原始图片当中，滤波子很难判定哪个是噪声像素哪个是图片内容。因此只消除噪声而保留图片内容不可能实现。现有的三种滤波算法，无论是均值滤波、中值滤波还是高斯滤波，都把处理点周围的像素也考虑进来，进行相应的运算，因此不可避免地会影响到周围像素点，导致图片内容模糊、细节丢失。

2. 从频域来讲，图像的噪声以及一些轮廓线、细节等，同属于高频区域，因此理想滤波子去除噪声既需要去除高频区域的噪声内容，也要保证同属于高频区域的图像细节、轮廓等不受影响，很明显这并不符合逻辑，因此理想滤波子不可实现。

代码

```
import numpy as np
import cv2 as cv2
import os
import math


def gaussian_multi_scale():
    ## get paths of the images
    path_lis = ['./灰度图版本/gray高斯多尺度平滑' + str(i + 1) + '.jpg' for i in
range(6)]
    kernel_lis = [3, 5, 7, 9, 11, 17]
    if not os.path.exists('./gaussian_multi_scale'):
        os.mkdir('./gaussian_multi_scale')
    for i in range(len(path_lis)):
        img = cv2.imread(path_lis[i], cv2.IMREAD_UNCHANGED)
        for kernel in kernel_lis:
            sigma = kernel / (3.0 *2)
            img_gaussian = cv2.GaussianBlur(img, (kernel, kernel), sigma)
            cv2.imwrite('./gaussian_multi_scale/' + path_lis[i].split('/')[-1]
[0:-4] + \
                '_' + str(kernel) + 'x' + str(kernel) + '.jpg', img_gaussian)
            cv2.destroyAllWindows()

def filtering():
    dir_ = os.walk('./灰度图版本')
    kernel_lis = [3, 5, 7, 11]
    if not os.path.exists('./gaussian_filtering'):
        os.mkdir('./gaussian_filtering')
    if not os.path.exists('./mean_filtering'):
        os.mkdir('./mean_filtering')
    if not os.path.exists('./median_filtering'):
        os.mkdir('./median_filtering')

    for path, _, file_list in dir_:
        for fname in file_list:
            if '多尺度' in fname:
                continue
            img = cv2.imread(os.path.join(path, fname), cv2.IMREAD_UNCHANGED)
            for kernel in kernel_lis:
                '''''
                gaussian filtering
                '''''
                sigma = kernel / (3.0 * 2)
                img_gaussian = cv2.GaussianBlur(img, (kernel, kernel), sigma)
```

```
                if fname[-2] == 'n':
                    ## png format
                    cv2.imwrite('./gaussian_filtering/' + fname[0:-4] + \
                    '_' + str(kernel) + 'x' + str(kernel) + '.png',
img_gaussian)
                elif fname[-2] == 'p':
                    ## jpg format
                    cv2.imwrite('./gaussian_filtering/' + fname[0:-4] + \
                    '_' + str(kernel) + 'x' + str(kernel) + '.jpg',
img_gaussian)
                cv2.destroyAllWindows()


                '''''
                mean filtering
                '''''
                img_mean = cv2.blur(img, (kernel, kernel))
                if fname[-2] == 'n':
                    ## png format
                    cv2.imwrite('./mean_filtering/' + fname[0:-4] + \
                        '_' + str(kernel) + 'x' + str(kernel) + '.png',
img_mean)
                elif fname[-2] == 'p':
                    ## jpg format
                    cv2.imwrite('./mean_filtering/' + fname[0:-4] + \
                        '_' + str(kernel) + 'x' + str(kernel) + '.jpg',
img_mean)
                cv2.destroyAllWindows()

                '''''
                median filtering
                '''''
                img_median = cv2.medianBlur(img, kernel)
                if fname[-2] == 'n':
                    ## png format
                    cv2.imwrite('./median_filtering/' + fname[0:-4] + \
                        '_' + str(kernel) + 'x' + str(kernel) + '.png',
img_median)
                elif fname[-2] == 'p':
                    ## jpg format
                    cv2.imwrite('./median_filtering/' + fname[0:-4] + \
                        '_' + str(kernel) + 'x' + str(kernel) + '.jpg',
img_median)
                cv2.destroyAllWindows()

def gaussian_dis(sigma, dist):
    return math.exp(- (dist + 0.0) / (2 * sigma * sigma)) * (1.0 / (2 *
math.pi * sigma *sigma))
```

```
def calculate_gaussian():
    kernel_lis = [3, 5, 7, 9, 11, 17]
    for kernel in kernel_lis:
        sigma = kernel / (3.0 * 2)
        arr = np.zeros((kernel, kernel), dtype='float')
        center_x = int(kernel / 2)
        center_y = int(kernel / 2)
        sum_ = 0
        for row in range(kernel):
            for col in range(kernel):
                dist = (row - center_x) ** 2 + (col - center_y) ** 2
                weight = gaussian_dis(sigma, dist)
                arr[row, col] = weight
                sum_ += weight
        arr /= sum_
        if not os.path.exists('./weight'):
            os.mkdir('./weight')
        np.savetxt('./weight/weight' + '_' + str(kernel) + 'x' + str(kernel) +
'.txt', arr)




if __name__ == "__main__":
    gaussian_multi_scale()
    filtering()
    calculate_gaussian()
```