桂林电子科技大学

**实验3 图像平滑**  实验报告

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 实验名称 | **图像平滑** | | | | | | | |  | 辅导员意见：  成绩 辅导员  签 名 |
| 院 系 | 计算机与信息安全学院 | | | 专业 | | 计算机科学与技术 | | |
| 学 号 | 2000301320 | | | 姓名 | | 靳志凌 | | |
| 实验日期 | 2022 | 年 | 10 | | 月 | | 28 | 日 |
|  |  | | | | | | | |

## 一、实验目的

掌握图像平滑算法的基本原理。

1. 能根据提供的图像数据，利用图像平滑算法的基本原理，对图像进行平滑处理；

2. 能去掉所给图像上的粒状噪声

## 二、实验内容与要求

1. 实现图像平滑的算法。
2. 调入图像，对添加噪声的图像用均值滤波、中值滤波和高斯滤波处理，进行比较。
3. 调出MATLAB自带均值、中值滤波和高斯滤波函数，与你编写代码的效果进行比较。

## 三、实验环境

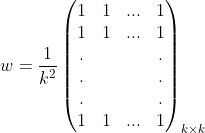
在实验室进行实验，学生可选Matlab、Python等等语言实现。

## 四、实验步骤

1. 描述算法的原理或实现流程

该实验我通过三个python文件分别实现对噪声图像的均值滤波、中值滤波和高斯滤波处理。   所谓平滑，主要目的是为了减少和抑制图像中的噪声，在空间域中使用邻域平均的方法来实现。

最简单的就是平滑模板，是下方公式中描述的一个矩阵，我们通常称之为模板、滤波器或者过滤器。



首先是均值滤波器函数：当k=3时，w是一个3x3的矩阵，输出图像中的某个像素点的数值，是输入图像中对应位置上也是一个3x3的空间与这个平滑模板的点乘的结果。由于我们加上了，所以确保结果不至于超过灰度值上限的255。为了计算的直观，通常k都是取奇数而不是偶数。那么最小的奇数是1，1实际上就是表示不进行平滑处理。

另外在事先的过程中还有一个问题，就是在图像的边界处如何处理。如果要求图像处理前后尺寸不变，那么最简单的方法就是填充，根据我们平滑模板的大小，如果平滑模板的边长为3，那么就在图像的上下左右各添加一层灰度值为0的区域，以实现尺寸不变；如果平滑模板的边长为5，那么上下左右就需要各添加两层，这种简称为“全0填充”。在本次的均值滤波器的实现中，我采用全0填充的方式。相关的代码是：



为了避免因为卷积运算导致输出图像缩小和图像边缘信息丢失，常常采用图像边缘填充技术，即在图像四周边缘填充0，使得卷积运算后图像大小不会缩小，同时也不会丢失边缘和角落的信息。在Python的numpy库中，常常采用numpy.pad()进行填充操作。

首先是椒盐噪声图像：

均值滤波器实现的效果：

其次是中值滤波器的函数实现：

使用一个3x3尺寸的中值滤波来处理含有突发性的椒盐噪声的代码如下。其中我们选择的填充方式是无填充，也就是对图像边缘，上下左右处忽略掉不进行滤波，只对可以容纳下一个滤波模板的区域滤波。这样子的做法好处是代码编写起来比较简单，而一般情况下图像边缘出的信息不是那么重要所以这样子做的风险比较小。另外，执行寻找中值的用numpy自带的median方法。

中值滤波导致了清晰度的下降，可以在对像素点赋予邻域的中位数之前添加一个判断条件。因为我们知道，噪声，往往都是数值与附近的区域相差很大，接近0或者是255，也就是很大可能是附近邻域内的极值。所以我们的判断条件就是，该像素是否是滤波窗口覆盖下邻域的极大值或极小值，如果是就进行中值滤波；如果不是就不予处理。在小区域内有可能有许多的噪声点，导致算法所寻找到的中值仍然是噪声的数值范围。这种情况下扩大滤波模板的大小并不会有多大的改变。此时可以将已经执行过一次中值滤波的结果图像再执行一次中值滤波，效果就会更好一些。

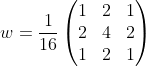
中值滤波效果：

可以看出自己编写的中值滤波器处理后的图像清晰度要比opencv库自带的函数处理的效果好一些。

最后是高斯滤波函数：

从平均模板中我们可以看出，这种平滑方式，对于邻域内所有的点的权重都是相同的，得到的结果在处理了图像中局部的亮点或者暗点的同时，也是图片变得模糊。所以很自然会想到，我们调节模板中的权重，加大模板中心的权重，而远离模板中心的权重越来越小，以确保中心点看起来更接近与它距离更近的点。基于这种考虑，得到的模板称为高斯模板。它的名称由来就是二维的正太分布密度函数，即高斯函数的形式，和我们的模板的数值有点相似，故得名高斯模板。

  常用的3x3的高斯模板为



效果图：



## 五、问题记录和实验总结（必写）

通过本次的实验，我学习到许多知识。首先，我对数字图像处理这门学科有了进一步的认识与学习，并通过这次实验的图像平滑处理，学习到了在空间域对图像进行一些操作从而使图像更加更好的满足人们关于图像所关注的地方。并且学习到了均值滤波、中值滤波以及高斯滤波这三种滤波器的不同和各自的侧重点的地方，以及适用的场所，以及各自的缺陷之处。均值滤波对邻域内的所有点的权重都相同的，得到的结果在处理了图像中局部的亮点和暗点同时，也模糊了图像；而且滤波器模板越大图像越模糊。为了调节模板的权重，加大模板中心的权重，得到的模板称为高斯模板。中值滤波可以直接将噪声点忽略，而且引起的模糊效应较低。它的典型应用就是用来消除椒盐噪声的。

实验代码：

1.中值滤波（包含添加噪声函数）：

import numpy as np

import cv2

def AddNoise(src, dst, probility = 0.05, method = "salt\_pepper"):

    """

    src:原来图像路径；

    dst:添加椒盐噪声之后图像存储路径；

     """

    imarray = cv2.imread(src,0)

    height, width = imarray.shape

    for i in range(height):

        for j in range(width):

            if np.random.random(1) < probility:

                if np.random.random(1) < 0.5:

                    imarray[i, j] = 0

                else:

                    imarray[i, j] = 255

    cv2.imwrite(dst,imarray)

    cv2.imshow('addNoise',imarray)

    cv2.waitKey(0)

    print("successfully execuite")

def means\_filter(input\_image, filter\_size):

    '''

    均值滤波器

    :param input\_image: 输入图像

    :param filter\_size: 滤波器大小

    :return: 输出图像

    注：此实现滤波器大小必须为奇数且 >= 3

    '''

    input\_image\_cp = np.copy(input\_image)  # 输入图像的副本

    filter\_template = np.ones((filter\_size, filter\_size))  # 空间滤波器模板

    pad\_num = int((filter\_size - 1) / 2)  # 输入图像需要填充的尺寸

    input\_image\_cp = np.pad(input\_image\_cp, (pad\_num, pad\_num), mode="constant", constant\_values=0)  # 填充输入图像

    """

    np.pad(array, pad\_width, mode, \*\*kwargs) 返回值为数组

    array——表示需要填充的数组；

    pad\_width——表示每个轴（axis）边缘需要填充的数值数目。

    参数输入方式为：（(before\_1, after\_1), … (before\_N, after\_N)），

    其中(before\_1, after\_1)表示第1轴两边缘分别填充before\_1个和after\_1个数值。

    取值为：{sequence, array\_like, int}

    mode——表示填充的方式（取值：str字符串或用户提供的函数）

     """

    m, n = input\_image\_cp.shape  # 获取填充后的输入图像的大小

    output\_image = np.copy(input\_image\_cp)  # 输出图像

    # 空间滤波

    for i in range(pad\_num, m - pad\_num):

        for j in range(pad\_num, n - pad\_num):

            output\_image[i, j] = np.sum(filter\_template \* input\_image\_cp[i - pad\_num:i + pad\_num + 1, j - pad\_num:j + pad\_num + 1]) / (filter\_size \*\* 2)

    output\_image = output\_image[pad\_num:m - pad\_num, pad\_num:n - pad\_num]  # 裁剪

    return output\_image

# 程序开始

img = cv2.imread("./Images/girl.jpg", 0)

# 添加椒盐噪声并显示出来，存储到文件

AddNoise('./Images/girl.jpg','./Images/girl\_noise.jpg')

# 函数实现均值滤波

out\_img1 = means\_filter(img,3)

# 均值滤波 库函数实现

img\_Blur\_3 = cv2.blur(img, (3, 3))   # 3\*3均值滤波

cv2.imshow("opencv", img\_Blur\_3)

cv2.imshow('2000301320jinzhiling',out\_img1)

cv2.waitKey(0)

2.均值滤波：

import numpy as np

import cv2

def MedianFilter(src, k = 3, padding = None):

    imarray = cv2.imread(src,0)

    height, width = imarray.shape

    if not padding:

        edge = int((k-1)/2)

        if height - 1 - edge <= edge or width - 1 - edge <= edge:

            print("The parameter k is to large.")

            return None

        new\_arr = np.zeros((height, width), dtype = "uint8")

        for i in range(height):

            for j in range(width):

                if i <= edge - 1 or i >= height - 1 - edge or j <= edge - 1 or j >= height - edge - 1:

                    new\_arr[i, j] = imarray[i, j]

                else:

                    #nm:neighbour matrix

                    nm = imarray[i - edge:i + edge + 1, j - edge:j + edge + 1]

                    max = np.max(nm)

                    min = np.min(nm)

                    if imarray[i, j] == max or imarray[i, j] == min:

                        new\_arr[i, j] = np.median(nm)

                        """

       np.median(a,

       axis=None,

       out=None,

       overwrite\_input=False,

       keepdims=False)

a：输入的数组；

axis：计算哪个轴上的均值，比如输入是二维数组，那么axis=0对应行，axis=1对应列；

out：用于放置求取中位数后的数组。 它必须具有与预期输出相同的形状和缓冲区长度；

overwrite\_input :一个bool型的参数，默认为Flase。如果为True那么将直接在数组内存中计算，这意味着计算之后原数组没办法保存，但是好处在于节省内存资源，Flase则相反；

keepdims：一个bool型的参数，默认为Flase。如果为True那么求取中位数的那个轴将保留在结果中；

                        """

                    else:

                        new\_arr[i, j] = imarray[i, j]

        return new\_arr

src = './Images/girl\_noise.jpg'

img = cv2.imread(src,0)

img1 = MedianFilter(src,3)

cv2.imwrite('./images/girl\_noise\_median\_filter.jpg',img1)

img2 = MedianFilter('./images/girl\_noise\_median\_filter.jpg',3)

cv2.imshow('2000301320jinzhiling',img2)

cv2.imshow('opencv',cv2.medianBlur(img,3))

cv2.waitKey(0)

3.高斯滤波：

import numpy as np

import cv2

from math import exp

def GaussianFilter(src, dst, k = 3, sigma = 1, method = "replicate"):

    imarray = cv2.imread(src,0)

    height, width = imarray.shape

    new\_arr = np.zeros((height, width), dtype = "uint8")

    filter = np.zeros((k,k))

    mid = (k-1)/2

    sigma2 = pow(sigma, 2)

    for i in range(k):

        for j in range(k):

            tmp = -(pow(i-mid,2)+pow(j-mid,2))/2/sigma2

            filter[i,j] = exp(tmp)/2/3.14/sigma2

    filtersum = filter.sum()

    for i in range(height):

        for j in range(width):

            total = 0

            for n in range(pow(k,2)):

                '''

                k = 3, n = 0, 1, 2 ..., 8, a = -1, 0, 1, b = -1, 0, 1

                k = 5, n = 0, 1, 2, 3 ..., 24, a = -2, -1, 0, 1, 2

                '''

                a, b = int(n//k - (k-1)/2), int(n%k - 1)

                #filter\_value

                aa, bb = int(n//k), int(n%k)

                f\_value = filter[aa, bb]

                if i + a <= 0:

                    if j + b <= 0:

                        total += imarray[0, 0]\*f\_value

                    elif j + b >= width - 1:

                        total += imarray[0, -1]\*f\_value

                    else:

                        total += imarray[0, j + b]\*f\_value

                elif i + a >= height - 1:

                    if j + b <= 0:

                        total += imarray[-1, 0]\*f\_value

                    elif j + b >= width - 1:

                        total += imarray[-1, -1]\*f\_value

                    else:

                        total += imarray[-1, j + b]\*f\_value

                else:

                    if j + b <= 0:

                        total += imarray[i + a, 0]\*f\_value

                    elif j + b >= width - 1:

                        total += imarray[i + a, -1]\*f\_value

                    else:

                        total += imarray[i + a, j + b]\*f\_value

            total /= filtersum

            new\_arr[i, j] = int(total)

    cv2.imwrite(dst,new\_arr)

    cv2.imshow('2000301320jinzhiling',new\_arr)

    print("Suceess.")

src = "./images/girl\_noise.jpg"

tar = "./images/girl\_noise\_gaussian\_filter.jpg"

GaussianFilter(src, tar, 3)

img = cv2.imread(src,0)

cv2.imshow('opencv',cv2.GaussianBlur(img,[3,3],0.8))

cv2.waitKey(0)