山东大学 计算机科学与技术 学院

计算机视觉 课程实验报告

|  |  |  |
| --- | --- | --- |
| 学号：201900161140 | 姓名： 张文浩 | 2021.10.8 |
| 实验题目：图像滤波 | | |
| 实验过程中遇到和解决的问题：  （记录实验过程中遇到的问题，以及解决过程和实验结果。可以适当配以关键代码辅助说明，但不要大段贴代码。） 3.1：高斯滤波   二维高斯滤波：  根据输入的k\_size高斯核的大小，做一个k\_size \* k\_size维的高斯矩阵。  首先需要根据k\_size定义center的大小，即k\_size/2，然后根据高斯矩阵中与center的距离计算每个位置的权重。  arr[i][j] = exp(-((i - center) \* (i - center) + (j - center) \* (j - center)) / (sigma \* sigma \* 2));  为保证高斯矩阵所有权重的和为1，最后要对arr二维数组进行归一化操作。  然后就要利用这个高斯矩阵对原图像进行处理，对处理边界情况，选择了边界扩充的方法，利用copyMakeBorder函数对原图像进行边界扩充，保证高斯核在扫描原图像的时候不会越界。  然后就可以利用三个for loops求结果图像  C:\WINDOWS\TEMP\WeChat Files\30fcccae2f9a28f308c9990ce10ce9a.jpg    一维高斯滤波  利用行列可分离性，先对每一行的像素进行一维高斯滤波，在对每一列进行高斯滤波。所以这里建立的高斯矩阵是一维的数组。  arr[i] = exp(-((i - center) \* (i - center)) / (sigma \* sigma \* 2));  然后进行归一化即可。  在获得一维高斯矩阵后，就先对每一行的像素进行计算新的像素值，图像暂时存在temp中。  C:\WINDOWS\TEMP\WeChat Files\abc58ead05ddceb71abb1599c1bd980.jpg  然后在y方向上做相同的处理即可。    结果：  sigma=1时：      可以看到，对比原图，无论是二维的高斯滤波还是行列分离的一维的高斯滤波，都与opencv自带的高斯滤波效果一样，都起到了模糊的效果。  下面对比不同的sigma效果：  Sigma=1： sigma=3 sigma=5    速度对比  对比二维高斯滤波和一维高斯滤波的执行时间：  Sigma=1 sigma=3 sigma=5   3.2快速均值滤波   思路：  利用积分图进行加速，只需要扫描一遍图像，效率不受滤波器窗口大小影响。先扫描一遍图像，构造一个三维数组，所谓图像像素的前缀和，a[i][j][k]。其中前两位分别是图像的横纵坐标，第三维大小为3，表示图像的是三个通道。    现在有了二维前缀和数组后就可以给目标图像赋值了  C:\WINDOWS\TEMP\WeChat Files\612921b48a34a0838a269a70d41d701.jpg  考虑边界情况的话，就将x1，x2，y1，y2，的不要小于1，不要大于rows和cols即可    结果：    W=1:    W=2    W=3    根据图像结果可以看到，自己的写的均值滤波方法与opencv自带的boxFilter没有什么区别，并且随着滤波窗口大小w变大，图像去椒盐化效果更好，但图像也更加模糊。  速度对比  W=1： W=2    W=3 W=5    可以看到因为采用了快速均值滤波的方法，自己的写的函数的耗时并没有因为滤波窗口大小的变化发生明显的变化。  但是也可以看到比opencv自带的boxFilter慢了非常多。  boxFilter解析  经过查阅资料和里阅读opencv中boxFilter实现的源代码，我找到了boxFilter速度如此之很快的原因。  boxFilter中并没有使用二维前缀和的思想。其原理类似于二维前缀和，但速度更快，稳定性更好。  与二维前缀和一样，boxFilter也要新先遍历一遍图像，构造一个与原图像大小相同的二维数组，与二维前缀和不同的是，二维前缀和的二维数组中存的是点当前位置左上角所有像素的和，所以在计算一个方块的像素和的时候要进行两次加法和一次减法操作。但在boxFilter中的二维数组，每个位置存的就是一定范围的方块（filter）内的像素和，不需要进行加法和减法操作。这个二维数组的构造方法如下：  C:\WINDOWS\TEMP\WeChat Files\4038e90cd485bcd05728c3c95485e7c.jpg  1、给定一张图像，宽高为（M,N），确定待求矩形模板的宽高(m,n)，如图紫色矩形。图中每个黑色方块代表一个像素，红色方块是假想像素。  2、开辟一段大小为M的数组，记为buff, 用来存储计算过程的中间变量，用红色方块表示  3、将矩形模板（紫色）从左上角（0，0）开始，逐像素向右滑动，到达行末时，矩形移动到下一行的开头（0，1），如此反复，每移动到一个新位置时，计算矩形内的像素和，保存在数组A中。以(0,0)位置为例进行说明：首先将绿色矩形内的每一列像素求和，结果放在buff内（红色方块），再对蓝色矩形内的像素求和，结果即为紫色特征矩形内的像素和，把它存放到数组A中，如此便完成了第一次求和运算。  4、每次紫色矩形向右移动时，实际上就是求对应的蓝色矩形的像素和，此时只要把上一次的求和结果减去蓝色矩形内的第一个红色块，再加上它右面的一个红色块，就是当前位置的和了，用公式表示 sum[i] = sum[i-1] - buff[x-1] + buff[x+m-1]  5、当紫色矩形移动到行末时，需要对buff进行更新。因为整个绿色矩形下移了一个像素，所以对于每个buff[i], 需要加上一个新进来的像素，再减去一个出去的像素，然后便开始新的一行的计算了。  Boxfilter的初始化过程非常快速，每个矩形的计算基本上只需要一加一减两次运算。从初始化的计算速度上来说，Boxfilter比二维前缀和要快一些，大约25%。在具体求某个矩形特征时，Boxfilter比二维前缀和的方法快4倍，所谓的4倍其实就是从4次加减运算降低到1次，虽然这个优化非常渺小，但是把它放到几层大循环里面，还是能节省一些时间的。 | | |
| 结果分析与体会： 3.1：高斯滤波 在第一个实验中，我先利用二维高斯滤波进行实验，有根据高斯函数的行列可分离性进行一维高斯滤波。  随着sigma和k\_size的增大，图像更加模糊    在计算速度方面，通过对比发现，一维的高斯滤波比二维高斯滤波在速度上有明显的优势，且效果与二维高斯滤波完全相同。并且随着sigma（k\_size）的增加，二维高斯的速度越来越变得让人无法接受，而一维的高斯滤波的时间消耗则还可以。   3.2快速均值滤波 在第二个实验中，先利用积分图（二维前缀和），实现了效率与滤波窗口大小无关的快速均值滤波器。并且发现效果与opencv的boxFilter效果相同，并且随着w滤波窗口的增大，图像去椒盐化效果更好，不过图像也因此变得更加模糊。  在计算速度方面，自己编写的快速均值滤波远远慢于opencv的boxFilter，可以看到，boxFilter的执行时间与我的相比几乎可以忽略不计，执行速度非常快。    通过阅读opencv的源代码及查阅相关资料，了解了boxFilter实现的原理，以及为什么可以快这么多。了解了boxFilter实现的算法的基本原理，boxFilter的初始化方法等。  Boxfilter的初始化过程非常快速，每个矩形的计算基本上只需要一加一减两次运算。从初始化的计算速度上来说，Boxfilter比二维前缀和要快一些，大约25%。在具体求某个矩形特征时，Boxfilter比二维前缀和的方法快4倍，所谓的4倍其实就是从4次加减运算降低到1次，虽然这个优化非常渺小，但是把它放到几层大循环里面，还是能节省一些时间的。 | | |