【图像处理】-014 空域滤波处理-均值滤波

在上一篇中，我们对图像进行了频率域中的滤波处理，通过在频率域中设计合适的滤波器，对图像的不同频率的分量进行不同处理，比如低通滤波时将低频分量通过高频分量截止，高通滤波时对高频分量通过低频分量截止。以后我们还会遇到带通、带阻、陷波滤波器等不同形式的滤波器。频域滤波是在频率域中将滤波器与图像相乘得到的。由于频域相乘等空域相乘，那么，在空域中肯定也是可以进行滤波处理的。　　本文将对空间域中图像的滤波处理进行讨论。

1 理论依据

1.1 空间滤波器的机理

空间滤波器由两个要素组成：

- (1) 一个邻域(典型地时一个较小的矩形)；
- (2) 对该邻域包围的图像像素执行的预定义操作。

滤波产生一个新像素，新像素的坐标等于邻域中心的坐标，像素的值是滤波操作的结果。如果在图像像素上执行的是线性操作，则该滤波器称为**线性空间滤波器**，否则，滤波器就成为**非线性空间滤波器**。

对于一个大小为$m \times n$的模版，通常假设$m = 2a + 1, n = 2b + 1$，其中a, b为正整数。使用大小为$m \times n$的滤波器对于大小为$M \times N$的图像进行线性空间滤波，可以由下式表示：

$$g(x,y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s,t) f(x+s, y+t) \tag{1}$$

其中，x, y是可变的，以便w中的每个像素可访问f中的每个像素。

1.2 空间相关与卷积

在执行线性空间滤波时，必须弄清楚两个相近的概念：相关与卷积。**相关**是滤波器模版移过图像并计算每个位置乘积之和的处理。卷积的机理相似，但滤波器首先要旋转180°。

一个大小为$m \times n$的滤波器$w(x,y)$与一幅图像$f(x,y)$做相关操作，可表示为$w(x,y) \star f(x,y)$,

$$w(x,y) \star f(x,y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s,t) f(x+s, y+t) \tag{2}$$

这一等式对所有位移变量x和y求值，以便w的所有元素访问f的每一个像素，其中我们假设f已被适当的填充，即$a = (m-1)/2, b = (n-1)/2$,这里，假设m, n是奇数。　　类似的，$w(x,y)$与$f(x,y)$的卷积表示为$w(x,y) \star f(x,y)$,

$$w(x,y) \star f(x,y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s,t) f(x-s, y-t) \tag{3}$$

其中，等式右侧的减号表示翻转f（即旋转180°）

2 均值滤波

平滑线性空间滤波器的输出响应是包含在滤波器模版领域内的像素的简单平均值，这些滤波器有时也成为均值滤波器。使用滤波器模版确定的领域内像素的平均灰度值代替图像中的每个像素的值，这种处理的结果降低了图像灰度的尖锐变化，由于典型地随机噪声由灰度级的急剧变化组成的，因此，常见的平滑处理就是降低噪声。

一幅$M \times N$的图像经过一个大小为$m \times n$的加权均值滤波器滤波的过程可以表示如下：

$$g(x,y) = \frac{\sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s,t) f(x+s, y+t)}{\sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s,t)} \tag{4}$$

```
#include "../include/importOpenCV.h"
#include "../include/baseOps.h"
#include "../include/opencv400/opencv2/core.hpp"
#include <iostream>


int main()
{
    //将工作目录设置到EXE所在的目录。
    SetCurrentDirectoryToExePath();

    cv::Mat src = cv::imread("../images/18.jpg");
    cv::imshow("原图", src);


    cv::Mat w1 = cv::Mat::zeros(cv::Size(3, 3), CV_32FC1);
    for (int i = 0; i < w1.rows; i++)
    {
        for (int j = 0; j < w1.cols; j++)
        {
            w1.at<float>(i, j) = 1 / 9.0f;
        }
    }

    cv::Mat output;
    src.copyTo(output);
    if (src.channels() == 3)
    {
        std::vector<cv::Mat> srcbgr;
        cv::split(src, srcbgr);
        std::vector<cv::Mat> dstbgr;
        cv::split(output, dstbgr);

        for (int i = 1; i < srcbgr[0].rows-1; i ++)
        {
            for (int j = 1; j < srcbgr[0].cols-1; j++)
```

```
            {
                dstbgr[0].at<uchar>(i, j) = (uchar)(srcbgr[0].at<uchar>(i - 1, j -
1)*w1.at<float>(0, 0) + srcbgr[0].at<uchar>(i - 1, j)*w1.at<float>(0, 1) +
srcbgr[0].at<uchar>(i - 1, j + 1)*w1.at<float>(0, 2) + \
                                                    srcbgr[0].at<uchar>(i    , j -
1)*w1.at<float>(1, 0) + srcbgr[0].at<uchar>(i    , j)*w1.at<float>(1, 1) +
srcbgr[0].at<uchar>(i    , j + 1)*w1.at<float>(1, 2) + \
                                                    srcbgr[0].at<uchar>(i + 1, j -
1)*w1.at<float>(2, 0) + srcbgr[0].at<uchar>(i + 1, j)*w1.at<float>(2, 1) +
srcbgr[0].at<uchar>(i + 1, j + 1)*w1.at<float>(2, 2));

                dstbgr[1].at<uchar>(i, j) = (uchar)(srcbgr[1].at<uchar>(i - 1, j -
1)*w1.at<float>(0, 0) + srcbgr[1].at<uchar>(i - 1, j)*w1.at<float>(0, 1) +
srcbgr[1].at<uchar>(i - 1, j + 1)*w1.at<float>(0, 2) + \
                                                    srcbgr[1].at<uchar>(i    , j -
1)*w1.at<float>(1, 0) + srcbgr[1].at<uchar>(i    , j)*w1.at<float>(1, 1) +
srcbgr[1].at<uchar>(i    , j + 1)*w1.at<float>(1, 2) + \
                                                    srcbgr[1].at<uchar>(i + 1, j -
1)*w1.at<float>(2, 0) + srcbgr[1].at<uchar>(i + 1, j)*w1.at<float>(2, 1) +
srcbgr[1].at<uchar>(i + 1, j + 1)*w1.at<float>(2, 2));

                dstbgr[2].at<uchar>(i, j) = (uchar)(srcbgr[2].at<uchar>(i - 1, j -
1)*w1.at<float>(0, 0) + srcbgr[2].at<uchar>(i - 1, j)*w1.at<float>(0, 1) +
srcbgr[2].at<uchar>(i - 1, j + 1)*w1.at<float>(0, 2) + \
                                                    srcbgr[2].at<uchar>(i    , j -
1)*w1.at<float>(1, 0) + srcbgr[2].at<uchar>(i    , j)*w1.at<float>(1, 1) +
srcbgr[2].at<uchar>(i    , j + 1)*w1.at<float>(1, 2) + \
                                                    srcbgr[2].at<uchar>(i + 1, j -
1)*w1.at<float>(2, 0) + srcbgr[2].at<uchar>(i + 1, j)*w1.at<float>(2, 1) +
srcbgr[2].at<uchar>(i + 1, j + 1)*w1.at<float>(2, 2));
            }
        }

        cv::merge(dstbgr, output);

        cv::Mat dst1;
        cv::blur(src, dst1, cv::Size(3, 3));
        cv::imshow("均值滤波3*3_手动计算", output);
        cv::imshow("均值滤波3*3_cv::blur", dst1);
    }
    else
    {

        for (int i = 1; i < src.rows - 1; i++)
        {
            for (int j = 1; j < src.cols - 1; j++)
            {
                output.at<uchar>(i, j) = (uchar)(src.at<uchar>(i - 1, j -
1)*w1.at<float>(0, 0) + src.at<uchar>(i - 1, j)*w1.at<float>(0, 1) + src.at<uchar>(i -
1, j + 1)*w1.at<float>(0, 2) + \
                                                    src.at<uchar>(i    , j -
1)*w1.at<float>(1, 0) + src.at<uchar>(i    , j)*w1.at<float>(1, 1) + src.at<uchar>(i
, j + 1)*w1.at<float>(1, 2) + \
```

```
                                                src.at<uchar>(i + 1, j -
1)*w1.at<float>(2, 0) + src.at<uchar>(i + 1, j)*w1.at<float>(2, 1) + src.at<uchar>(i +
1, j + 1)*w1.at<float>(2, 2));
            }
        }

        cv::Mat dst1;
        cv::blur(src, dst1, cv::Size(3, 3));
        cv::imshow("均值滤波3*3_手动计算", output);
        cv::imshow("均值滤波3*3_cv::blur", dst1);
    }

    cv::waitKey();
    return 0;
}
```

不同尺度的均值滤波效果如下图所示。