数字信号处理中卷积

卷积一词最开始出现在信号与线性系统中，信号与线性系统中讨论的就是信号经过一个线性系统以后发生的变化。由于现实情况中常常是一个信号前一时刻的输出影响着这一时刻的输出，所在一般利用系统的单位响应与系统的输入求卷积，以求得系统的输出信号（当然要求这个系统是线性时不变的）。

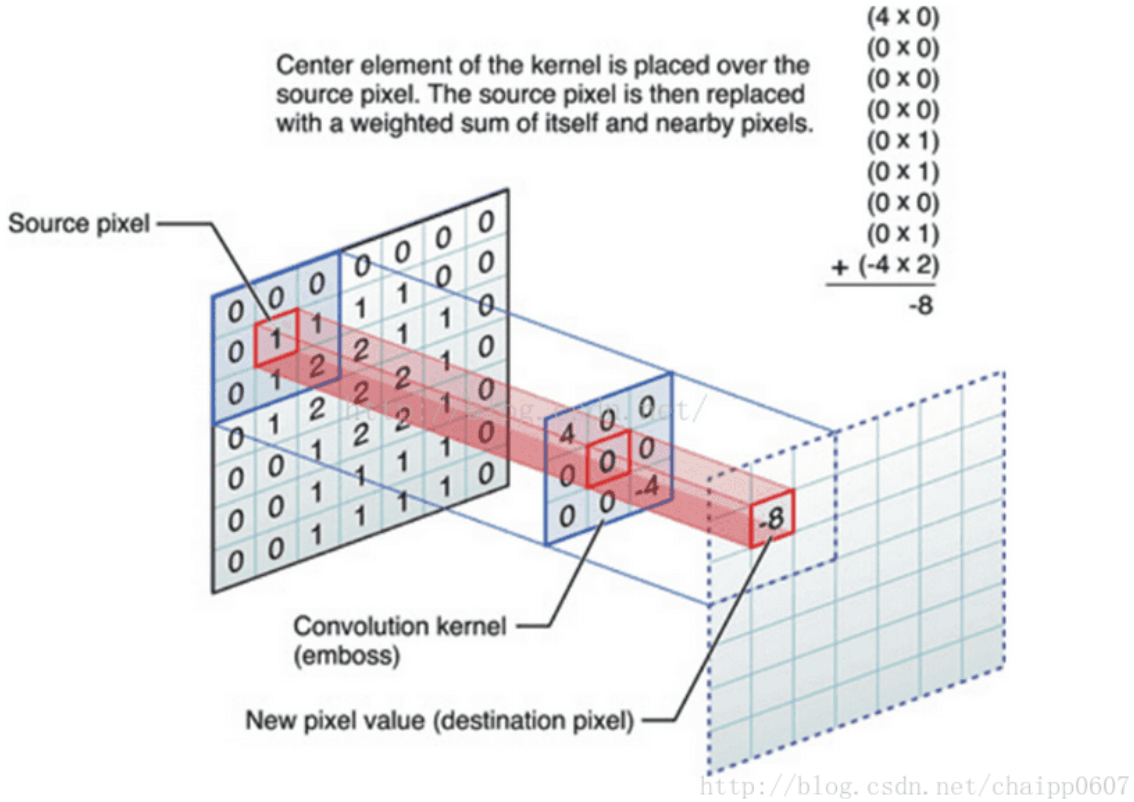
卷积的定义：

卷积是两个变量在某范围内相乘后求和的结果。如果卷积的变量是序列x(n)和h(n)，则卷积的结果：

$$y(n)=\sum_{i=-\infty}^{\infty}x(i)h(n-i)=x(n)*h(n)$$

数字图像处理中卷积

数字图像是一个二维的离散信号，对数字图像做卷积操作其实就是利用卷积核（卷积模板）在图像上滑动，将图像点上的像素灰度值与对应的卷积核上的数值相乘，然后将所有相乘后的值相加作为卷积核中间像素对应的图像上像素的灰度值，并最终滑动完所有图像的过程。



这张图可以清晰的表征出整个卷积过程中一次相乘后相加的结果：该图片选用3*3的卷积核，卷积核内共有九个数值，所以图片右上角公式中一共有九行，而每一行都是图像像素值与卷积核上数值相乘，最终结果-8代替了原图像中对应位置处的1。这样沿着图片一步长为1滑动，每一个滑动后都一次相乘再相加的工作，我们就可以得到最终的输出结果。除此之外，卷积核的选择有一些规则：

1）卷积核的大小一般是奇数，这样的话它是按照中间的像素点中心对称的，所以卷积核一般都是3x3，5x5或者7x7。有中心了，也有了半径的称呼，例如5x5大小的核的半径就是2。

2）卷积核所有的元素之和一般要等于1，这是为了原始图像的能量（亮度）守恒。其实也有卷积核元素相加不为1的情况，下面就会说到。

3）如果滤波器矩阵所有元素之和大于1，那么滤波后的图像就会比原图像更亮，反之，如果小于1，那么得到的图像就会变暗。如果和为0，图像不会变黑，但也会非常暗。

4）对于滤波后的结构，可能会出现负数或者大于255的数值。对这种情况，我们将他们直接截断到0和255之间即可。对于负数，也可以取绝对值。

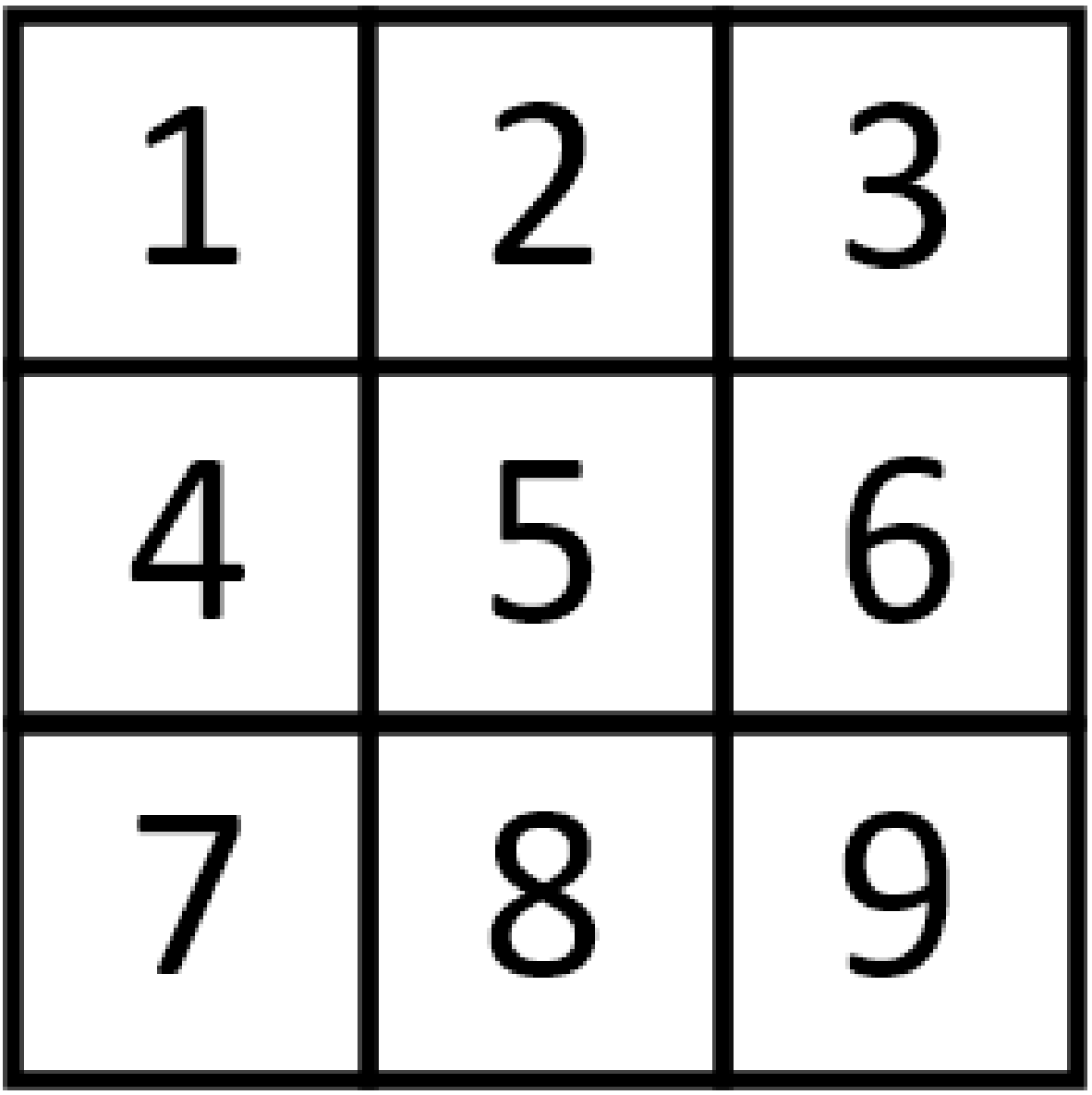
边界补充问题

上面的图片说明了图像的卷积操作，但是他也反映出一个问题，如上图，原始图片尺寸为7*7，卷积核的大小为3*3，当卷积核沿着图片滑动后只能滑动出一个5*5的图片出来，这就造成了卷积后的图片和卷积前的图片尺寸不一致，这显然不是我们想要的结果，所以为了避免这种情况，需要先对原始图片做边界填充处理。在上面的情况中，我们需要先把原始图像填充为9*9的尺寸。

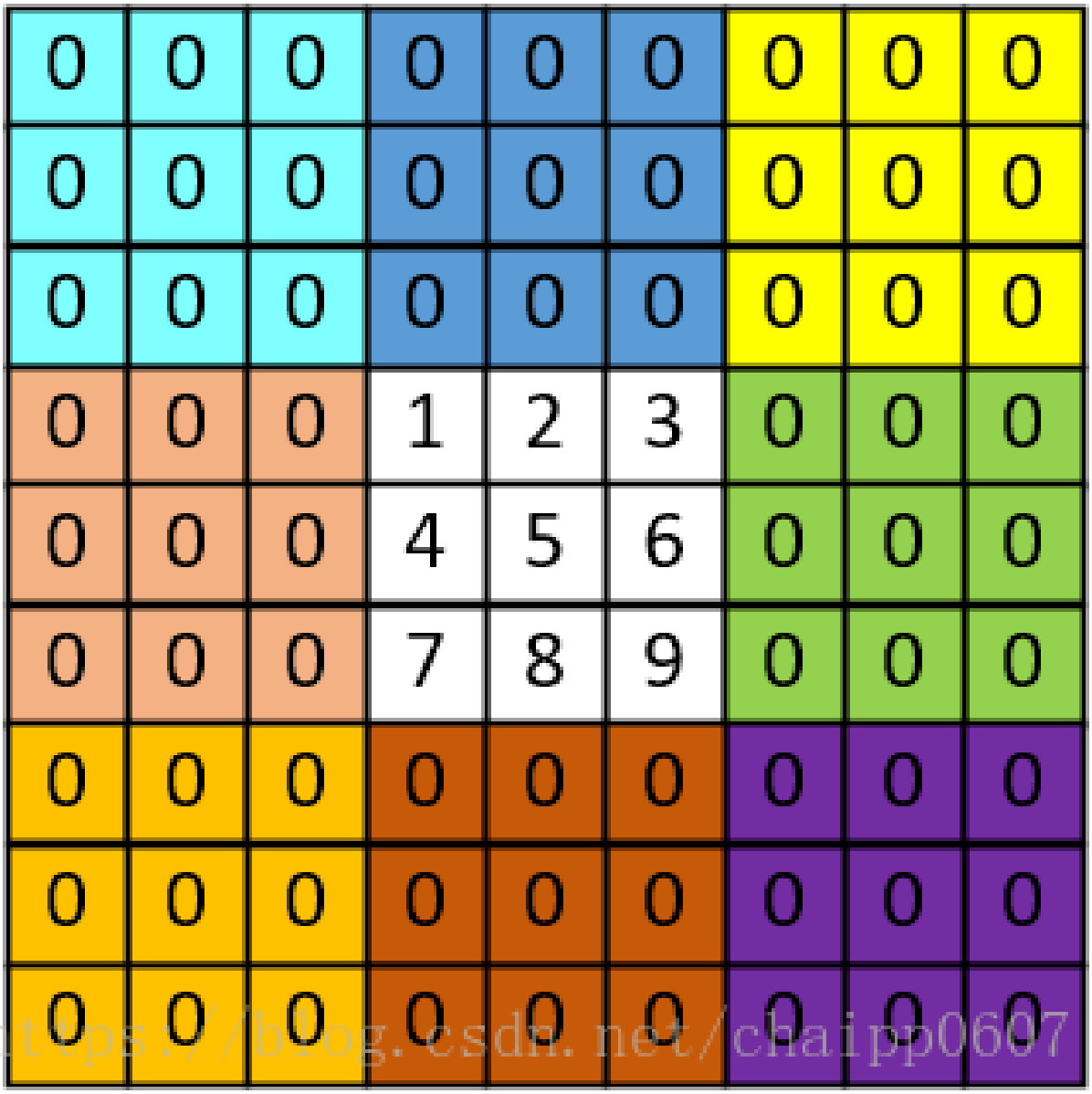
常用的区域填充方法包括：

为了画图方便，这里就不用5*5的尺寸了，用3*3定义原始图像的尺寸，补充为9*9的尺寸，**图片上的颜色只为方便观看，并没有任何其他含义。**

原始图像：



补零填充



边界复制填充


1 1 1 1 2 3 3 3 3
1 1 1 1 2 3 3 3 3
1 1 1 1 2 3 3 3 3
1 1 1 1 2 3 3 3 3
4 4 4 4 5 6 6 6 6
7 7 7 7 8 9 9 9 9
7 7 7 7 8 9 9 9 9
7 7 7 7 8 9 9 9 9
7 7 7 7 8 9 9 9 9

镜像填充

块填充

以上四种边界补充方法通过看名字和图片就能理解了，不在多做解释。

不同卷积核下卷积意义

我们经常能看到的，平滑，模糊，去燥，锐化，边缘提取等等工作，其实都可以通过卷积操作来完成，下面我们一一举例说明一下：

一个没有任何作用的卷积核：

卷积核：

0	0	0
0	1	0
0	0	0

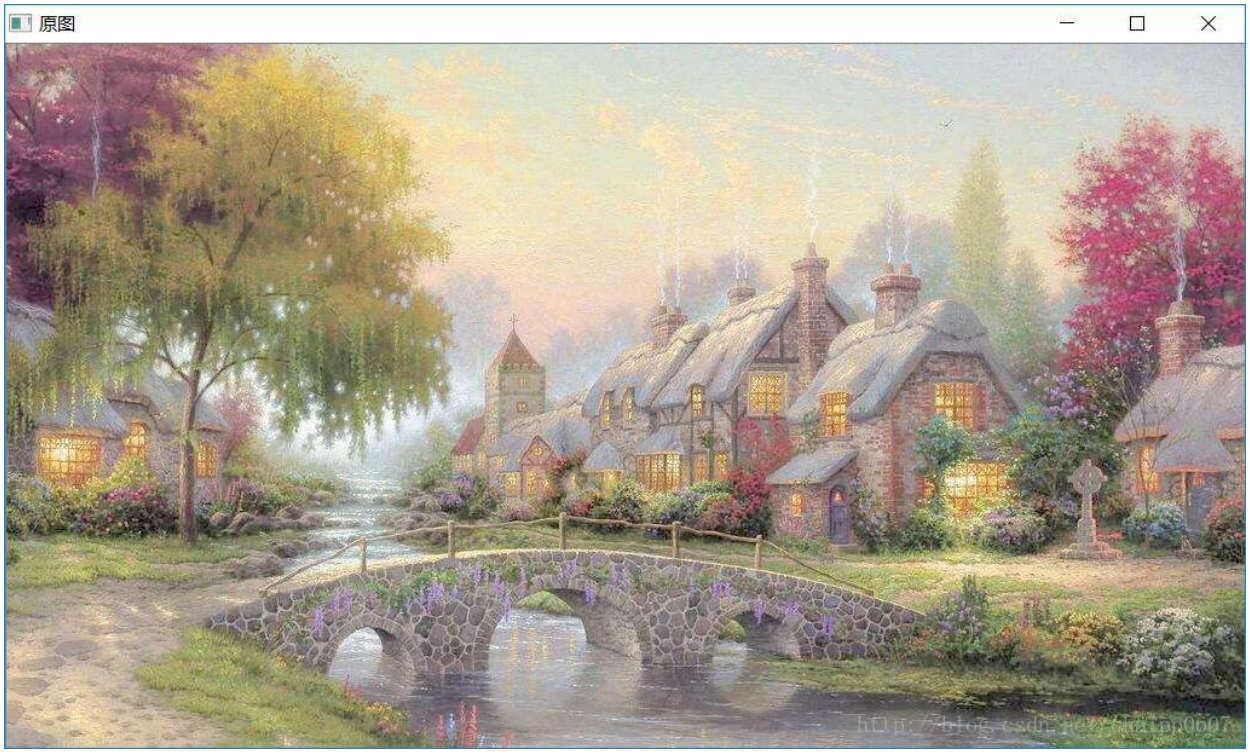
将原像素中间像素值乘1，其余全部乘0，显然像素值不会发生任何变化。

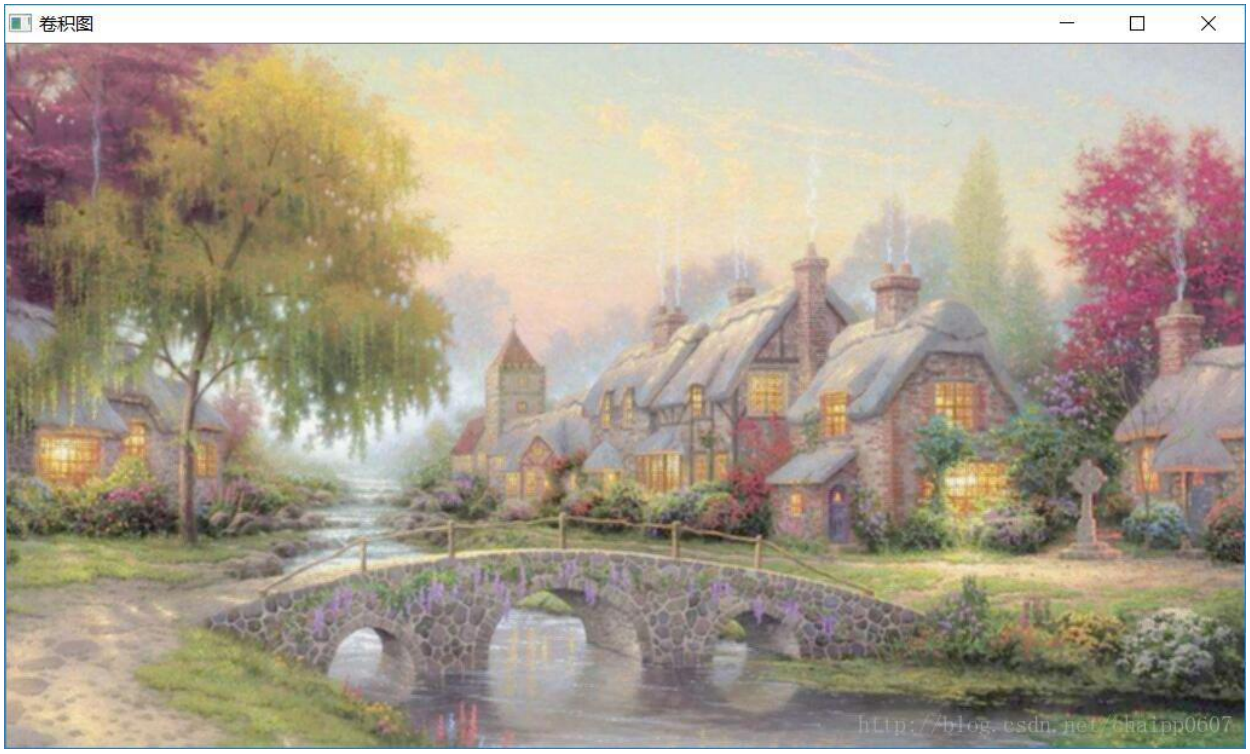
平滑均值滤波：

选择卷积核：

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

该卷积核的作用在于取九个值的平均值代替中间像素值，所以起到的平滑的效果：



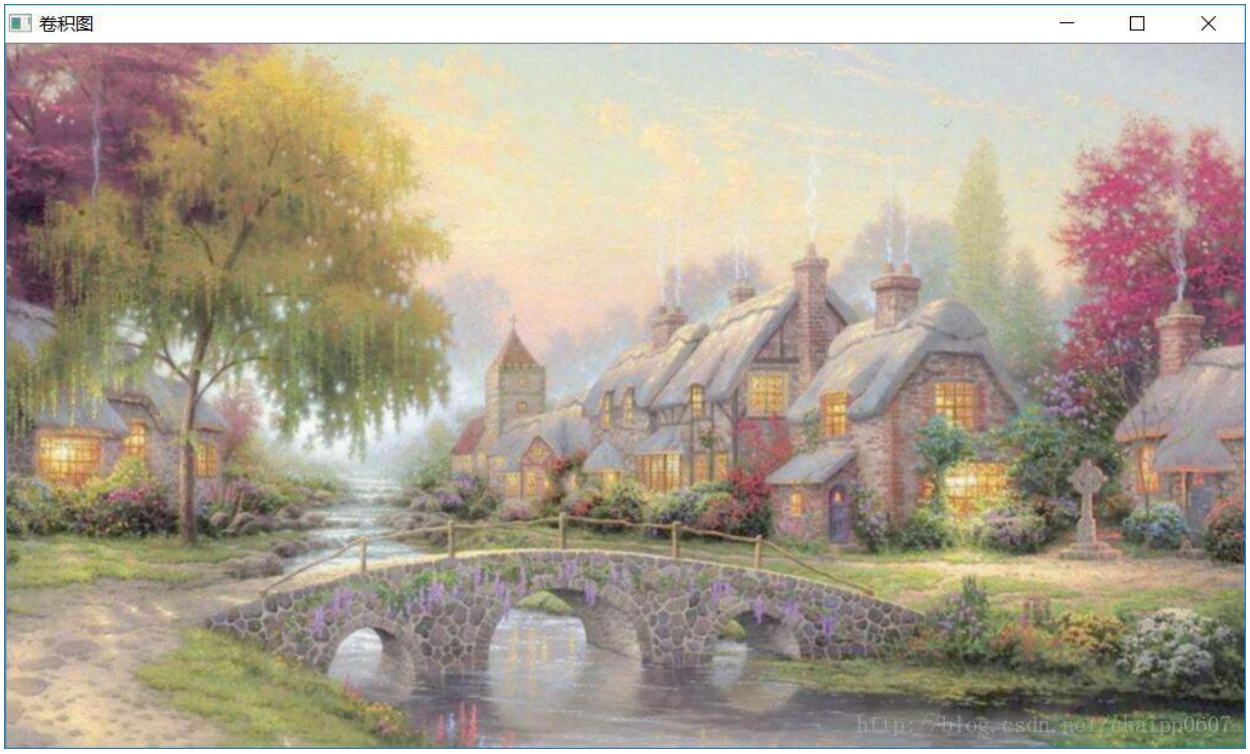


高斯平滑：

卷积核：

1/16	2/16	1/16
2/16	4/16	2/16
1/16	2/16	1/16

高斯平滑水平和垂直方向呈现高斯分布，更突出了中心点在像素平滑后的权重，相比于均值滤波而言，有着更好的平滑效果。

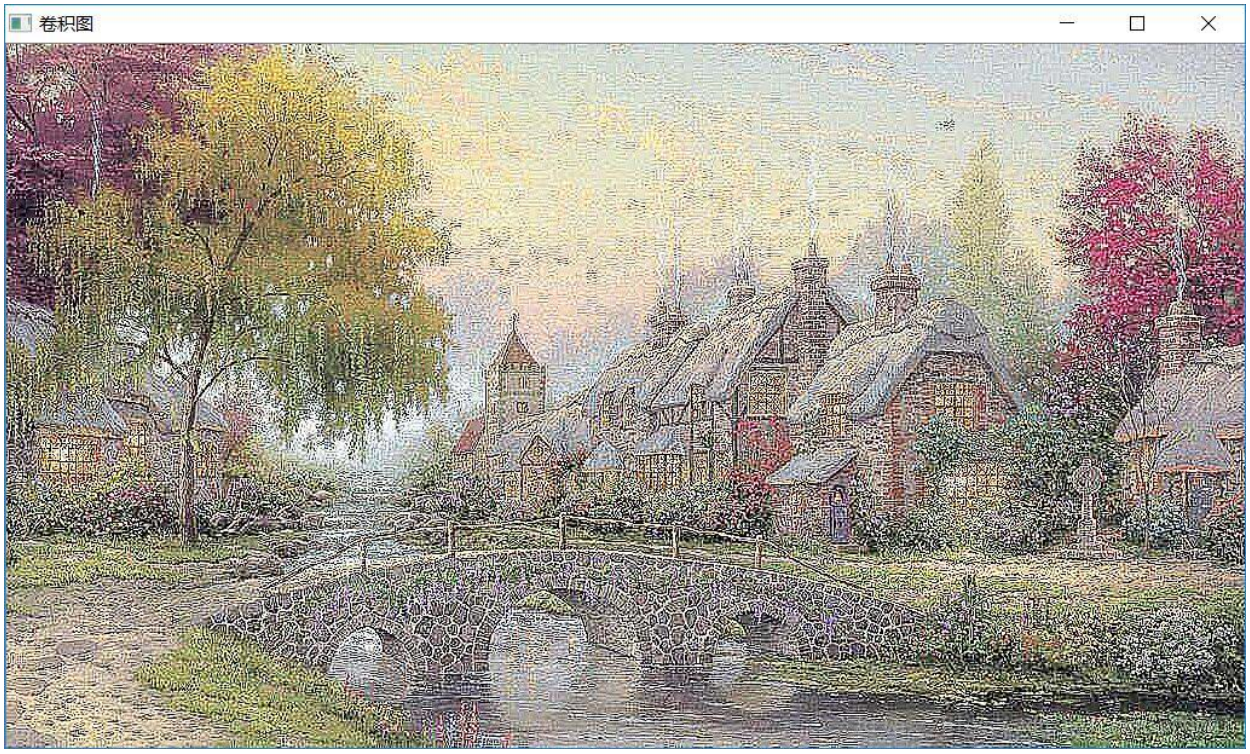


图像锐化：

卷积核：

-1	-1	-1
-1	9	-1
-1	-1	-1

该卷积利用的其实是图像中的边缘信息有着比周围像素更高的对比度，而经过卷积之后进一步增强了这种对比度，从而使图像显得棱角分明、画面清晰，起到锐化图像的效果。



除了上述卷积核，边缘锐化还可以选择：

0	-1	0
-1	5	-1
0	-1	0

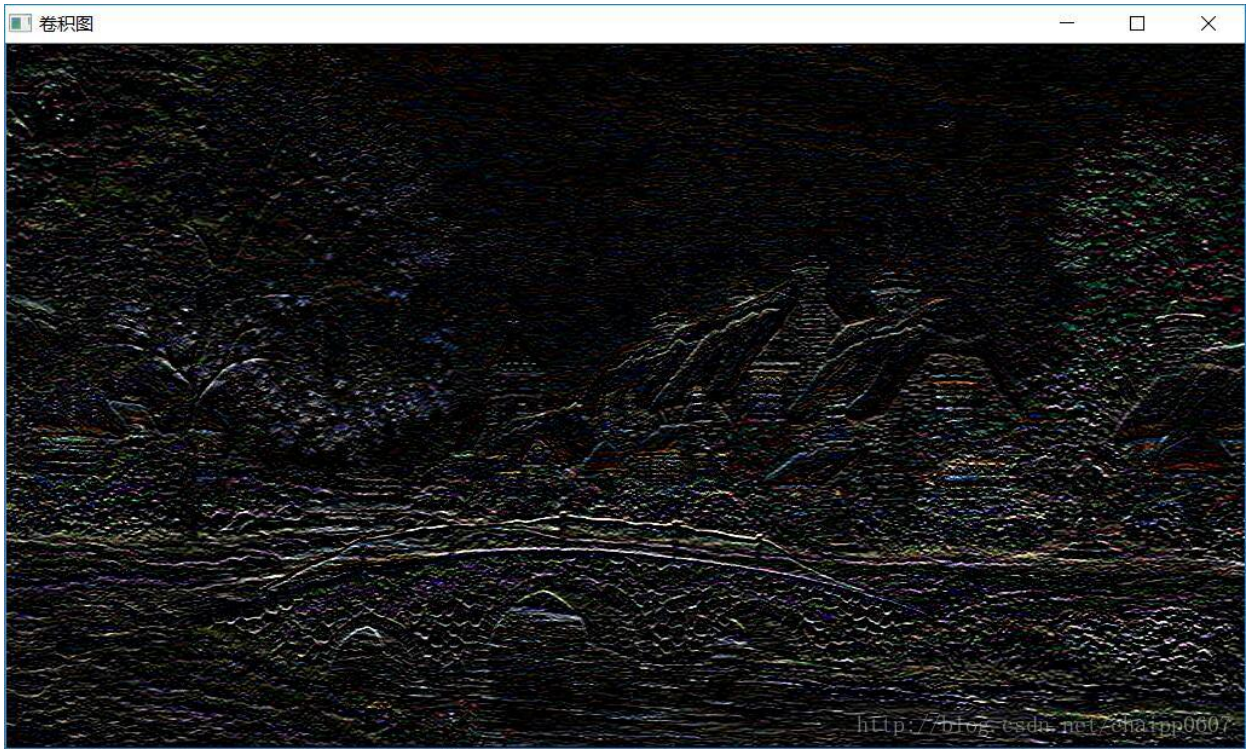
梯度Prewitt：

水平梯度卷积核：

-1	0	1
-1	0	1
-1	0	1

垂直梯度卷积核：

-1	-1	-1
0	0	0
1	1	1



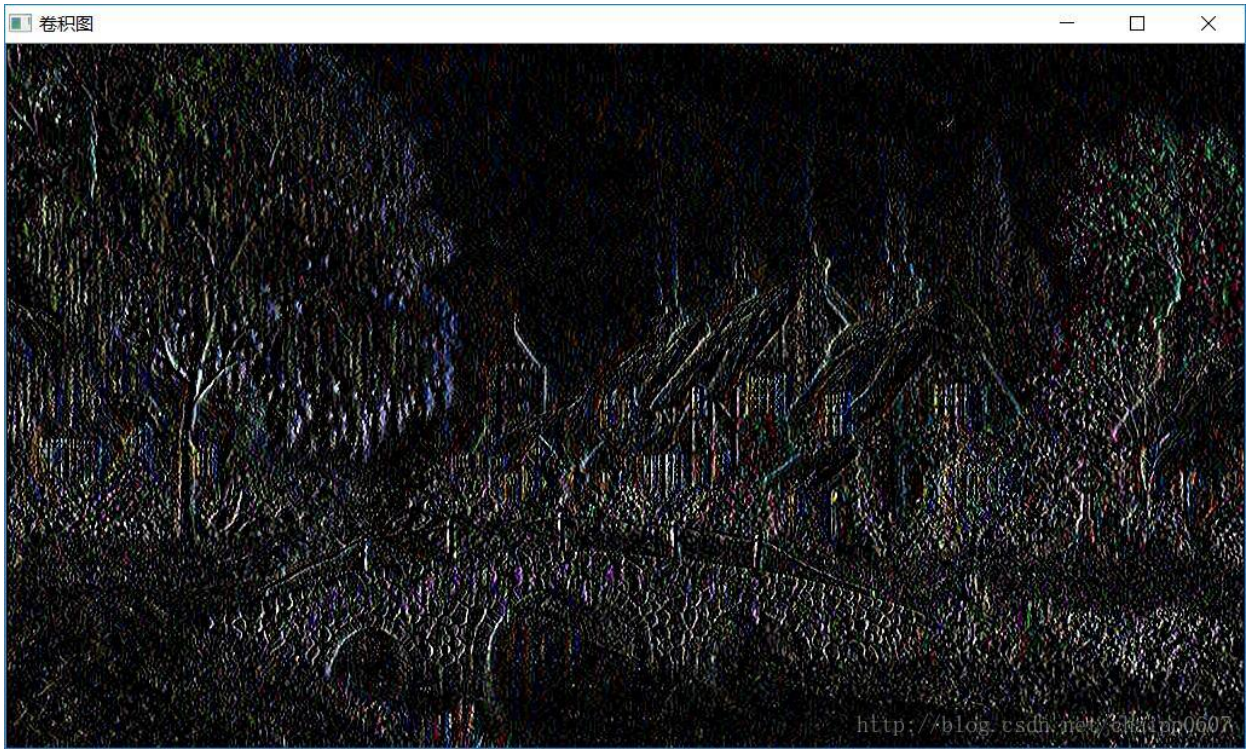
梯度Prewitt卷积核与Soble卷积核的选定是类似的，都是对水平边缘或垂直边缘有比较好的检测效果。

Soble边缘检测：

Soble与上述卷积核不同之处在于，Soble更强调了和边缘相邻的像素点对边缘的影响。

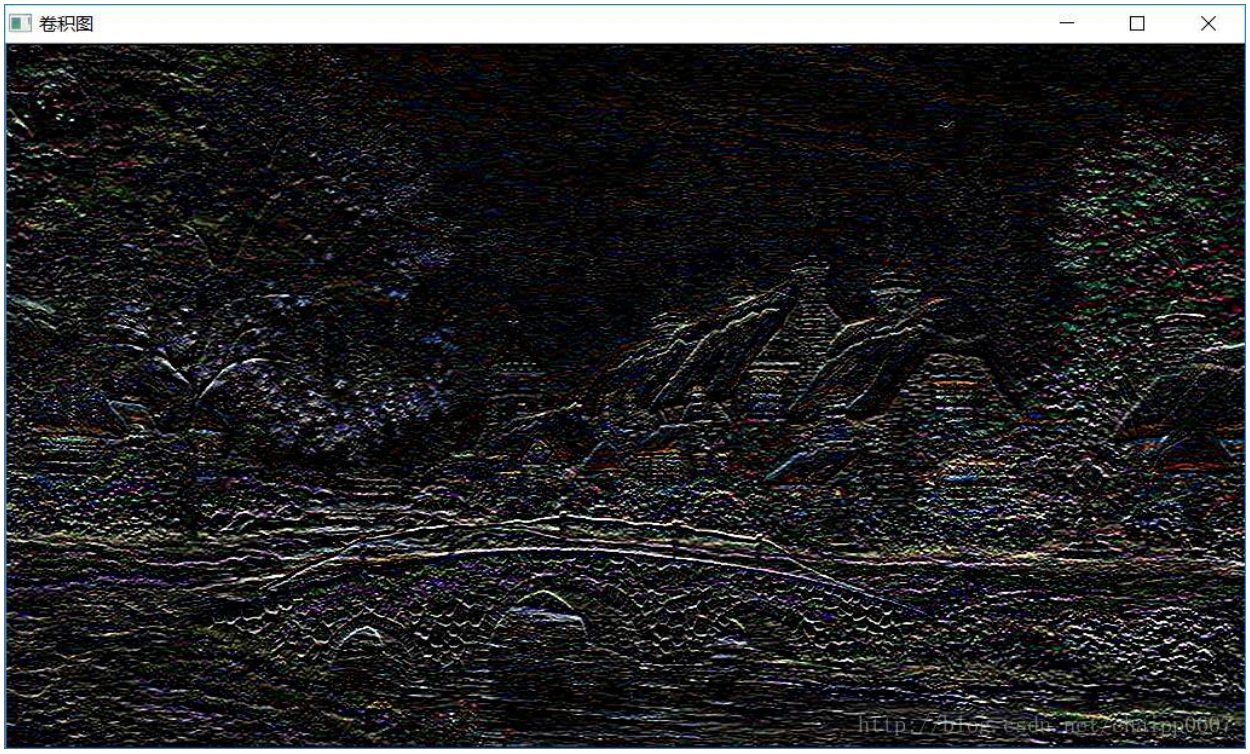
水平梯度：

-1	0	1
-2	0	2
-1	0	1



垂直梯度：

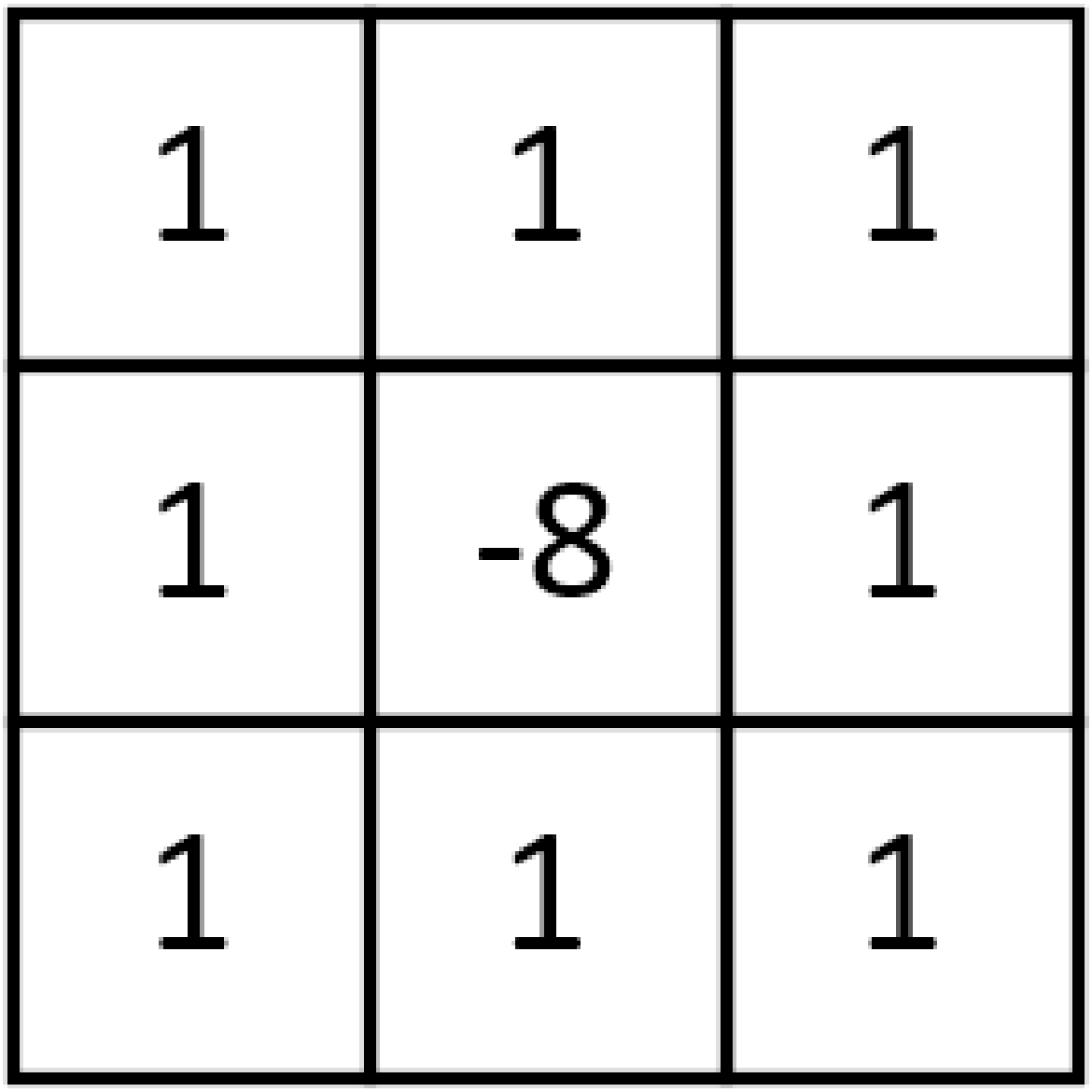
-1	-2	-1
0	0	0
1	2	1



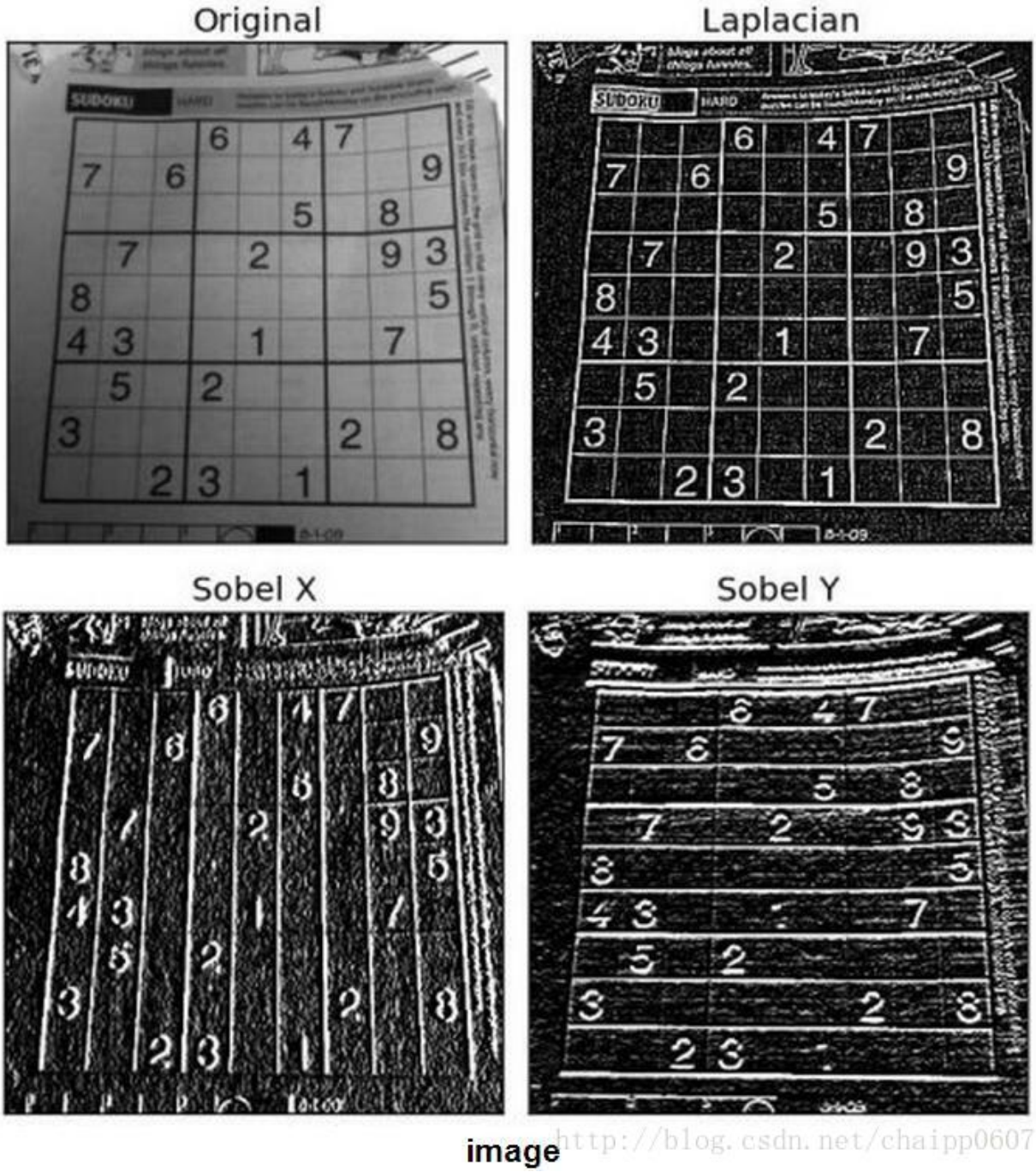
以上的水平边缘与垂直边缘检测问题可以参考：[Soble算子水平和垂直方向导数问题]

梯度Laplacian：

卷积核：


1 1 1
1 -8 1
1 1 1

Laplacian也是一种锐化方法，同时也可以做边缘检测，而且边缘检测的应用中并不局限于水平方向或垂直方向，这是Laplacian与soble的区别。下面这张图可以很好的表征出二者的区别：[来源于OpenCV官方文档]



image

OpenCV实现

可以利用OpenCV提供的filter2D函数完成对图像进行卷积操作，其函数接口为：

```
CV_EXPORTS_W void filter2D(
InputArray src,
OutputArray dst,
int ddepth,
InputArray kernel,
Point anchor=Point(-1,-1),
double delta=0,
int borderType=BORDER_DEFAULT );
```

第一个参数：输入图像

第二个参数：输出图像，和输入图像具有相同的尺寸和通道数量

第三个参数：目标图像深度，输入值为-1时，目标图像和原图像深度保持一致。

第四个参数：卷积核，是一个矩阵

第五个参数：内核的基准点(anchor)，其默认值为(-1,-1)说明位于kernel的中心位置。基准点即kernel中与进行处理的像素点重合的点。

第五个参数：在储存目标图像前可选的添加到像素的值，默认值为0

第六个参数：像素向外逼近的方法，默认值是BORDER_DEFAULT。

```
#include <iostream>
#include <iostream>
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/imgproc/imgproc.hpp>
using namespace std;
using namespace  cv;

int main()
{
    Mat srcImage = imread("1.jpg");
    namedWindow("srcImage", WINDOW_AUTOSIZE);
    imshow("原图", srcImage);
    Mat kernel = (Mat_<double>(3,3) <<
        -1, 0 ,1,
        -2, 0, 2,
        -1, 0, 1);
    Mat dstImage;
    filter2D(srcImage,dstImage,srcImage.depth(),kernel);
    namedWindow("dstImage",WINDOW_AUTOSIZE);
    imshow("卷积图",dstImage);
    waitKey(0);
    return 0;
}
```

所以代码的实现就非常简单了，不同的卷积操作只需要改变卷积核kernel 即可。

手写卷积操作

这个自己实现的卷积其实也依赖OpenCV，但是没有直接使用封装好的函数，这样更有利于了解图像卷积到底是如何完成的。这里面加了一个防溢出的函数，具体可以看聊一聊OpenCV的saturate_cast防溢出。

```
#include <opencv2/opencv.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/core/core.hpp>

using namespace  std;
using namespace  cv;

Mat Kernel_test_3_3 = (Mat_<double>(3,3) <<
    0,-1,0,
    -1,5,-1,
    0,-1,0);
void Convlution(Mat  InputImage,Mat  OutputImage,Mat kernel)
{
    //计算卷积核的半径
    int sub_x = kernel.cols/2;
    int sub_y = kernel.rows/2;
    //遍历图片
    for (int image_y=0;image_y<InputImage.rows-2*sub_y;image_y++)
    {
        for(int image_x=0;image_x<InputImage.cols-2*sub_x;image_x++)
        {
            int pix_value = 0;
            for (int kernel_y = 0;kernel_y<kernel.rows;kernel_y++)
            {
                for(int kernel_x = 0;kernel_x<kernel.cols;kernel_x++)
                {
                    double  weihgt = kernel.at<double>(kernel_y,kernel_x)   ;
                    int value =  (int)InputImage.at<uchar>
(image_y+kernel_y,image_x+kernel_x);
                    pix_value +=weihgt*value;
                }
            }
            OutputImage.at<uchar>(image_y+sub_y,image_x+sub_x) = (uchar)pix_value;
            //OutputImage.at<uchar>(image_y+sub_y,image_x+sub_x) =
saturate_cast<uchar>((int)pix_value);
            if ((int)pix_value!=(int)saturate_cast<uchar>((int)pix_value))
            {
                //cout<<"没有防溢出"<<(int)pix_value<<endl;
                //cout<<"防溢出"<<(int)saturate_cast<uchar>((int)pix_value)<<endl;
                //cout<<"没有防溢出写入了什么？"<<(int)OutputImage.at<uchar>
```

```
(image_y+sub_y,image_x+sub_x)<<endl;
                //cout<<endl;
            }
        }
    }
}


int main()
{
    Mat srcImage = imread("1.jpg",0);
    namedWindow("srcImage", WINDOW_AUTOSIZE);
    imshow("原图", srcImage);

    //filter2D卷积
    Mat dstImage_oprncv(srcImage.rows,srcImage.cols,CV_8UC1,Scalar(0));;
    filter2D(srcImage,dstImage_oprncv,srcImage.depth(),Kernel_test_3_3);
    imshow("filter2D卷积图",dstImage_oprncv);
    imwrite("1.jpg",dstImage_oprncv);

    //自定义卷积
    Mat dstImage_mycov(srcImage.rows,srcImage.cols,CV_8UC1,Scalar(0));
    Convlution(srcImage,dstImage_mycov,Kernel_test_3_3);
    imshow("卷积图3",dstImage_mycov);
    imwrite("2.jpg",dstImage_mycov);

    waitKey(0);
    return 0;

}
```