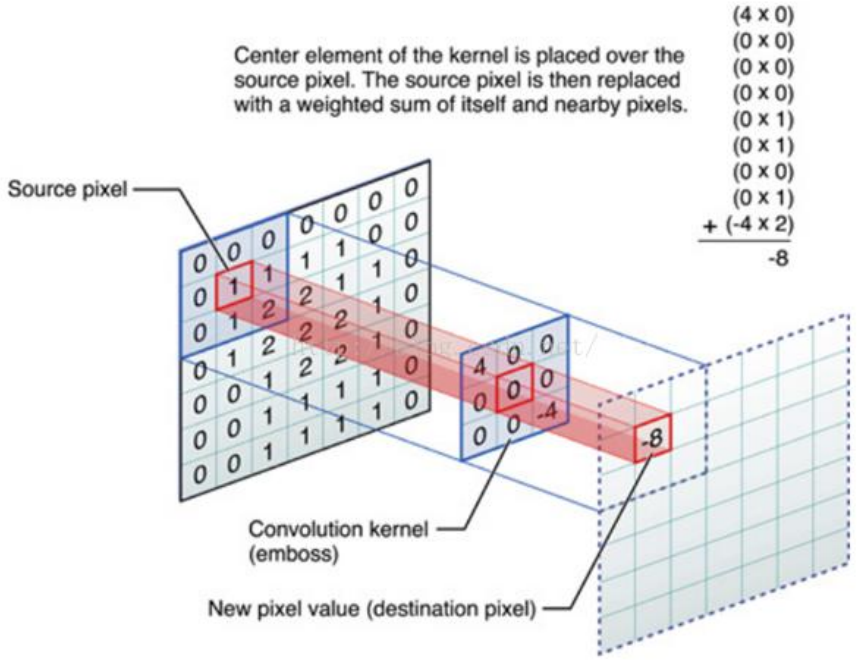
线性滤波与卷积

赵海臣

线性滤波与卷积

- 首先，我们有一个二维的滤波器矩阵（有个高大上的名字叫卷积核）和一个要处理的二维图像。
- 其次，对于图像的每一个像素点，计算它的邻域像素和滤波器矩阵的对应元素的乘积，然后加起来，作为该像素位置的值。这样就完成了滤波(卷积)过程。
- 使用不同的卷积核进行滤波(卷积)可以产生不同的图像处理效果。



线性滤波与卷积

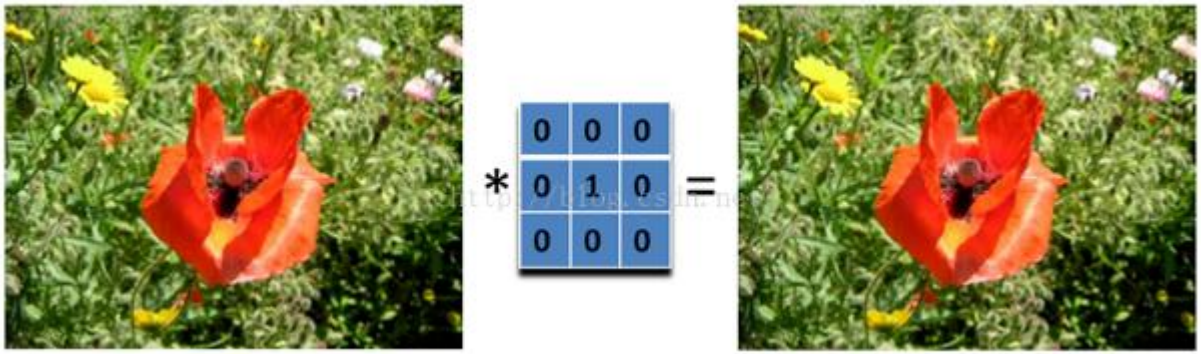
- 对图像和滤波矩阵进行逐个元素相乘再求和的操作就相当于将一个二维的函数移动到另一个二维函数的所有位置，这个操作就叫卷积或者协相关。
  - 卷积和协相关的差别：
    - 卷积需要先对滤波矩阵进行180的翻转
    - 但如果矩阵是对称的，那么两者就没有什么差别了。
  - 卷积和协相关的特点：
    - 线性：用每个像素的邻域的线性组合来代替这个像素
    - 平移不变性(shift-invariant)：在图像的每个位置都执行相同的操作

卷积核的要求

- 卷积核的大小：
  - 卷积核的大小应该是奇数，这样它才有一个中心，例如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。
- 卷积核矩阵所有元素的和：
  - =1 ： 滤波前后图像的亮度保持不变
  - >1 ： 滤波后的图像就会比原图像更亮
  - <1 ： 滤波后的图像就会比原图像更暗
  - =0 ： 图像不会变黑，但也会非常暗
- 滤波后的结构：
  - 可能会出现负数或者大于255的数值。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。

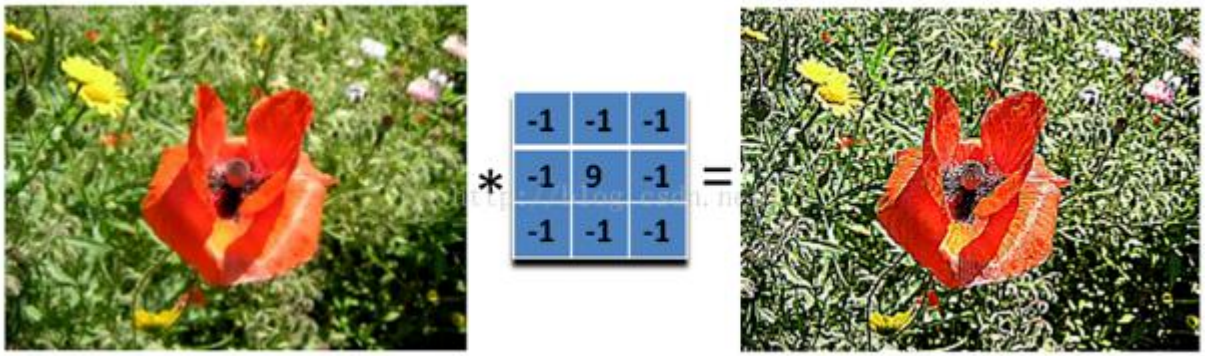
不同卷积核的效果

- 只有中心点的值是1。邻域点的权值都是0，对滤波后的取值没有任何影响



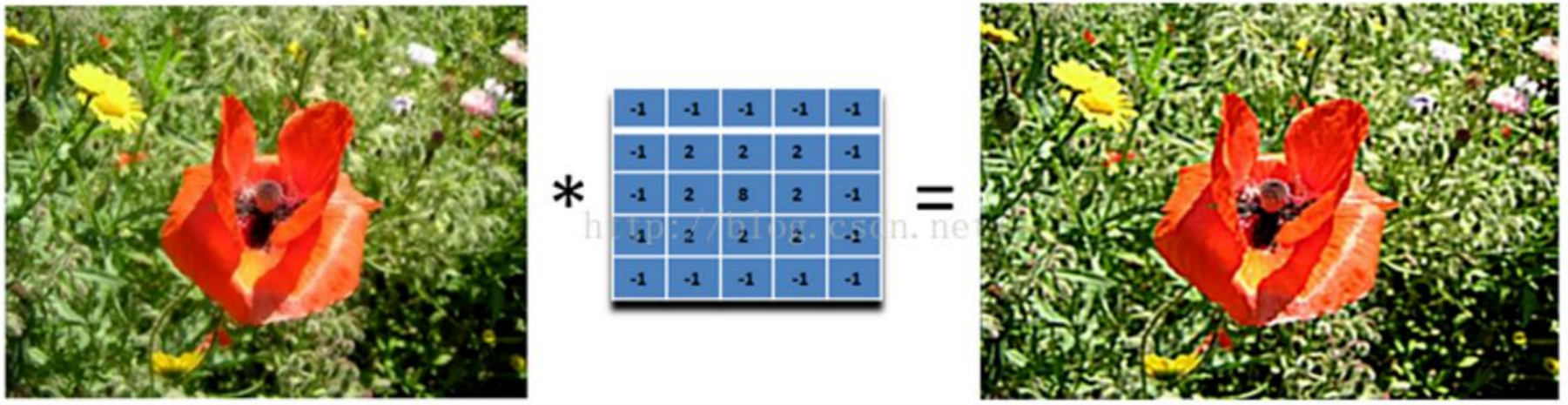
图像锐化滤波器Sharpness Filter

- 图像的锐化和边缘检测很像，首先找到边缘，然后把边缘加到原来的图像上面，这样就强化了图像的边缘，使图像看起来更加锐利了。
- 这两者操作统一起来就是锐化滤波器了，也就是在边缘检测滤波器的基础上，再在中心的位置加1，这样滤波后的图像就会和原始的图像具有同样的亮度了，但是会更加锐利。



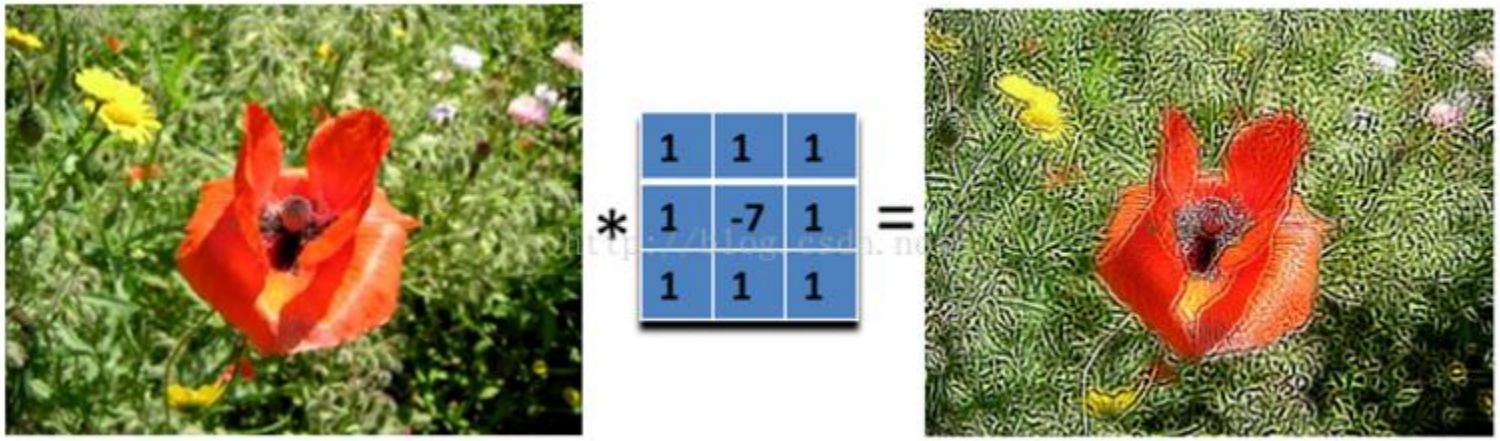
图像锐化滤波器Sharpness Filter

- 把核加大，就可以得到更加精细的锐化效果



图像锐化滤波器Sharpness Filter

- 下面的滤波器会平滑边缘，反锐化，主要是强调图像的细节



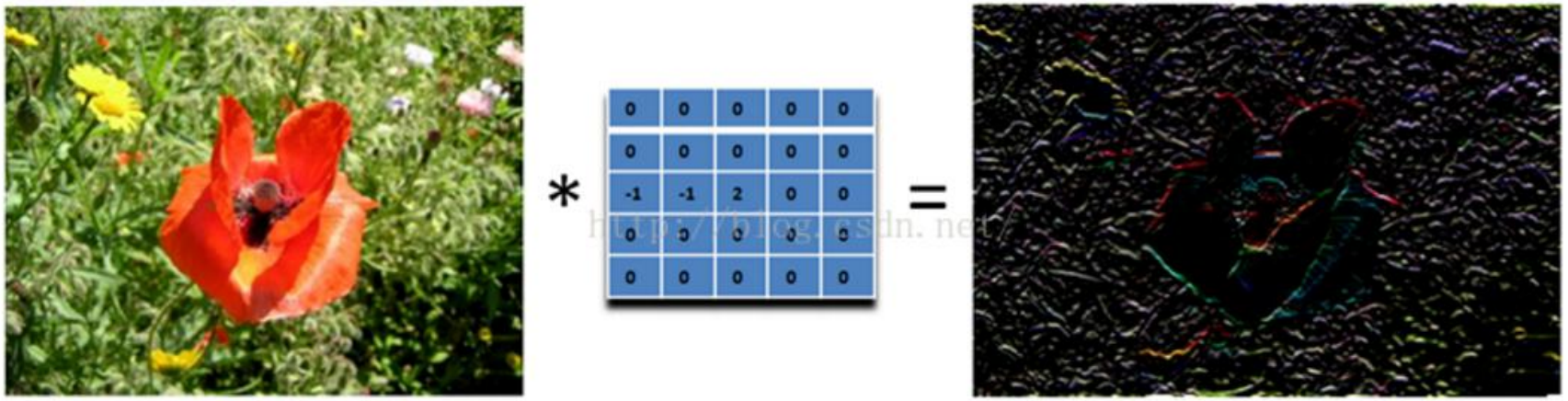
图像锐化滤波器Sharpness Filter

- 图像锐化卷积核实际上是计算当前点和周围点的差别，然后将这个差别加到原来的位置上。
- 中间点的权值要比所有的权值和大于1，意味着这个像素亮度要保持原来的值。
- 最简单的3x3的锐化滤波器如下：

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}; \begin{bmatrix} -1 & -1 & -1 \\ -1 & 9 & -1 \\ -1 & -1 & -1 \end{bmatrix}; \begin{bmatrix} -k & -k & -k \\ -k & 8k+1 & -k \\ -k & -k & -k \end{bmatrix}$$

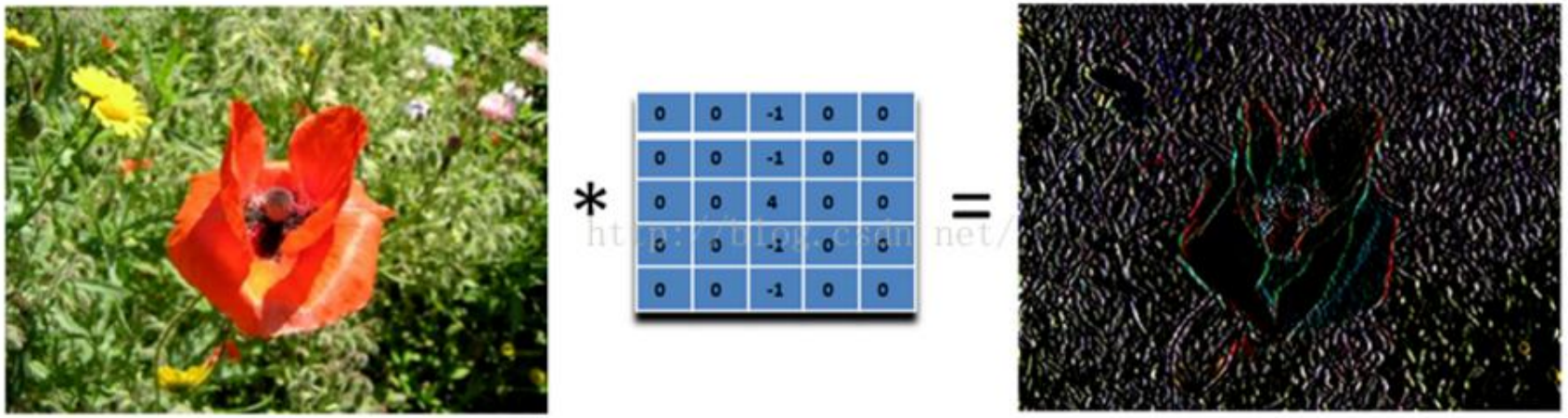
边缘检测Edge Detection

- 水平边缘检测
  - 这里矩阵的元素和是0，所以滤波后的图像会很暗，只有边缘的地方是有亮度的。
  - 用这个滤波器卷积相当于求导的离散版本：你将当前的像素值减去前一个像素值，这样你就可以得到这个函数在这两个位置的差别或者斜率。



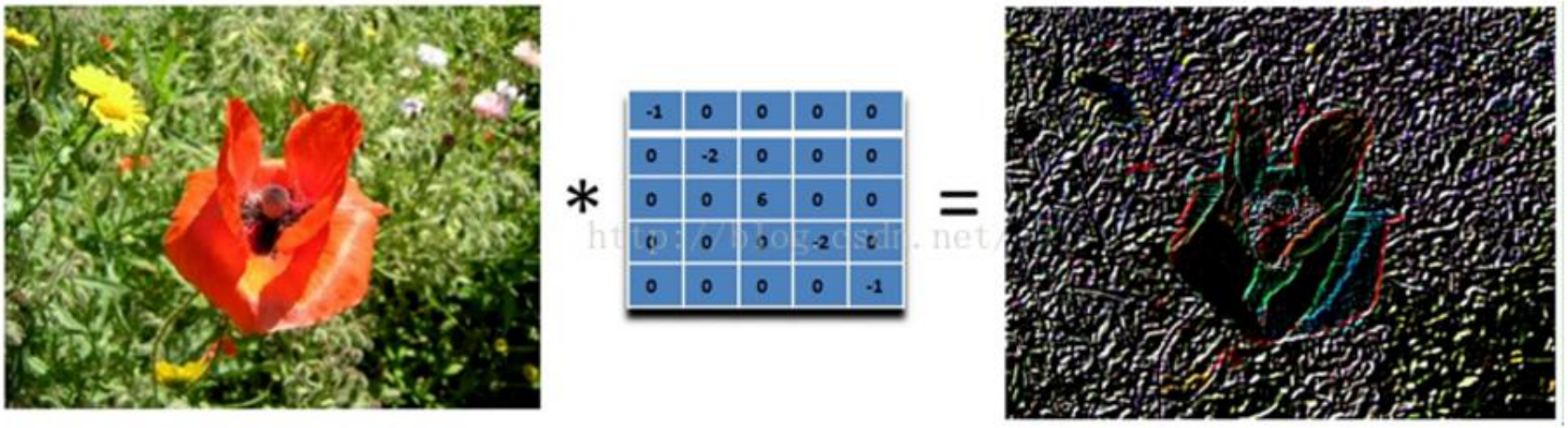
边缘检测Edge Detection

- 垂直方向的边缘检测
  - 这里像素上和下的像素值都使用



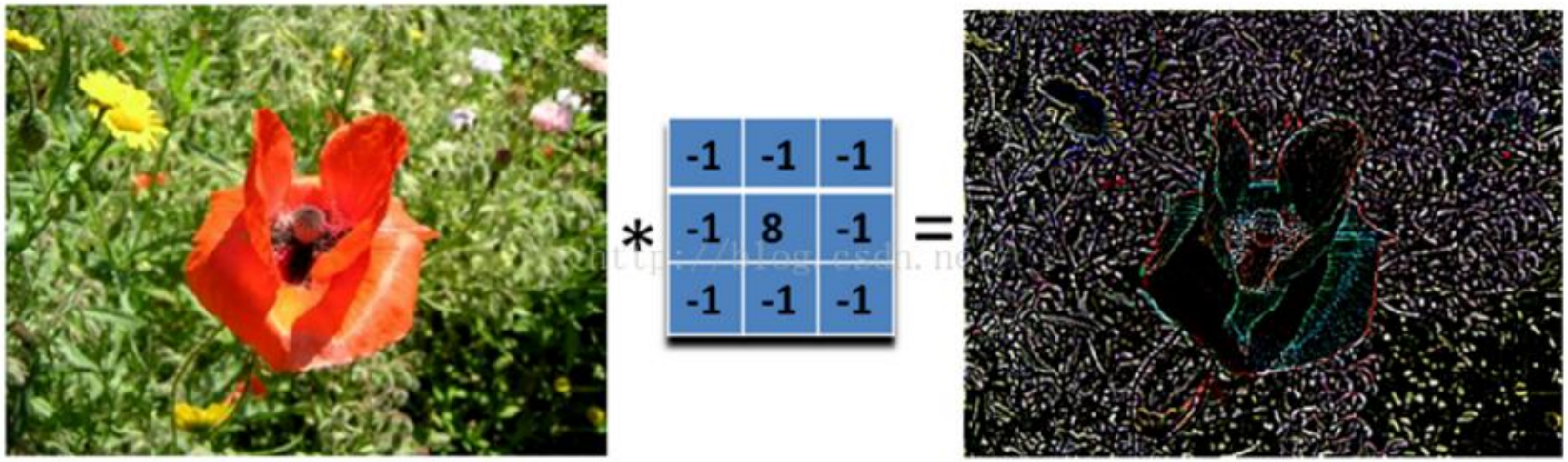
边缘检测Edge Detection

- 45度的边缘检测



边缘检测Edge Detection

- 所有方向边缘检测



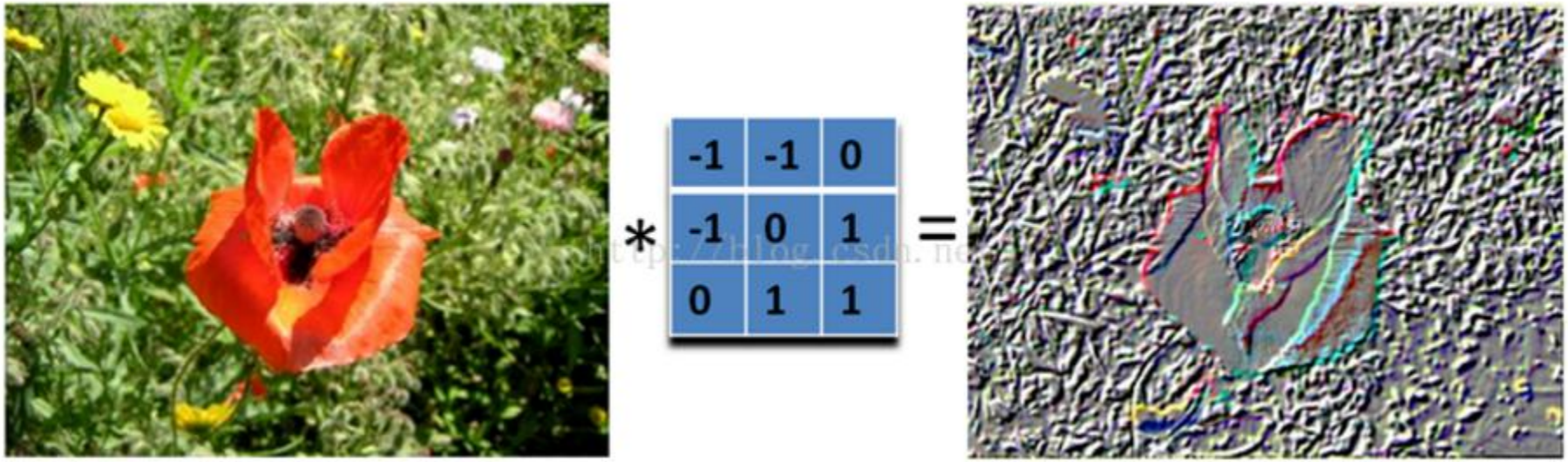
边缘检测Edge Detection

- 梯度计算
  - 这种简单的方法会把噪声也放大了
  - 矩阵所有的值加起来要是0

$$\begin{bmatrix} -1/8 & -1/8 & -1/8 \\ -1/8 & 1 & -1/8 \\ -1/8 & -1/8 & -1/8 \end{bmatrix}; \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix};$$

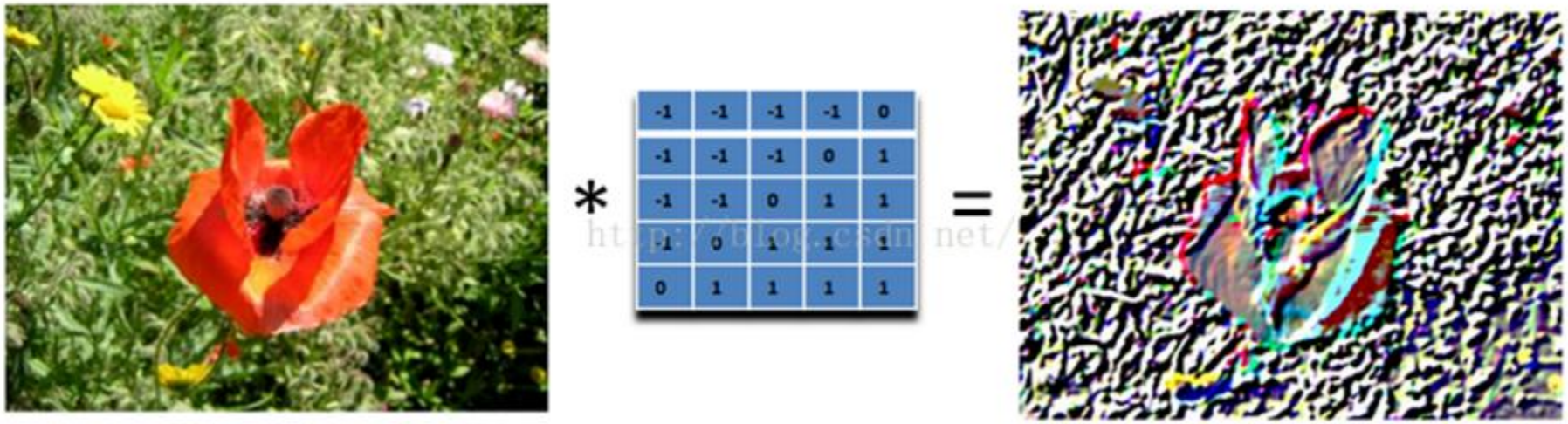
浮雕Embossing Filter

- 浮雕滤波器可以给图像一种3D阴影的效果
  - 将中心一边的像素减去另一边的像素就可以了
  - 像素值有可能是负数，对结果图像加上128的偏移
    - 将负数当成阴影
    - 将正数当成光
- 45度的浮雕滤波器：



浮雕Embossing Filter

- 只要加大滤波器，就可以得到更加夸张的效果



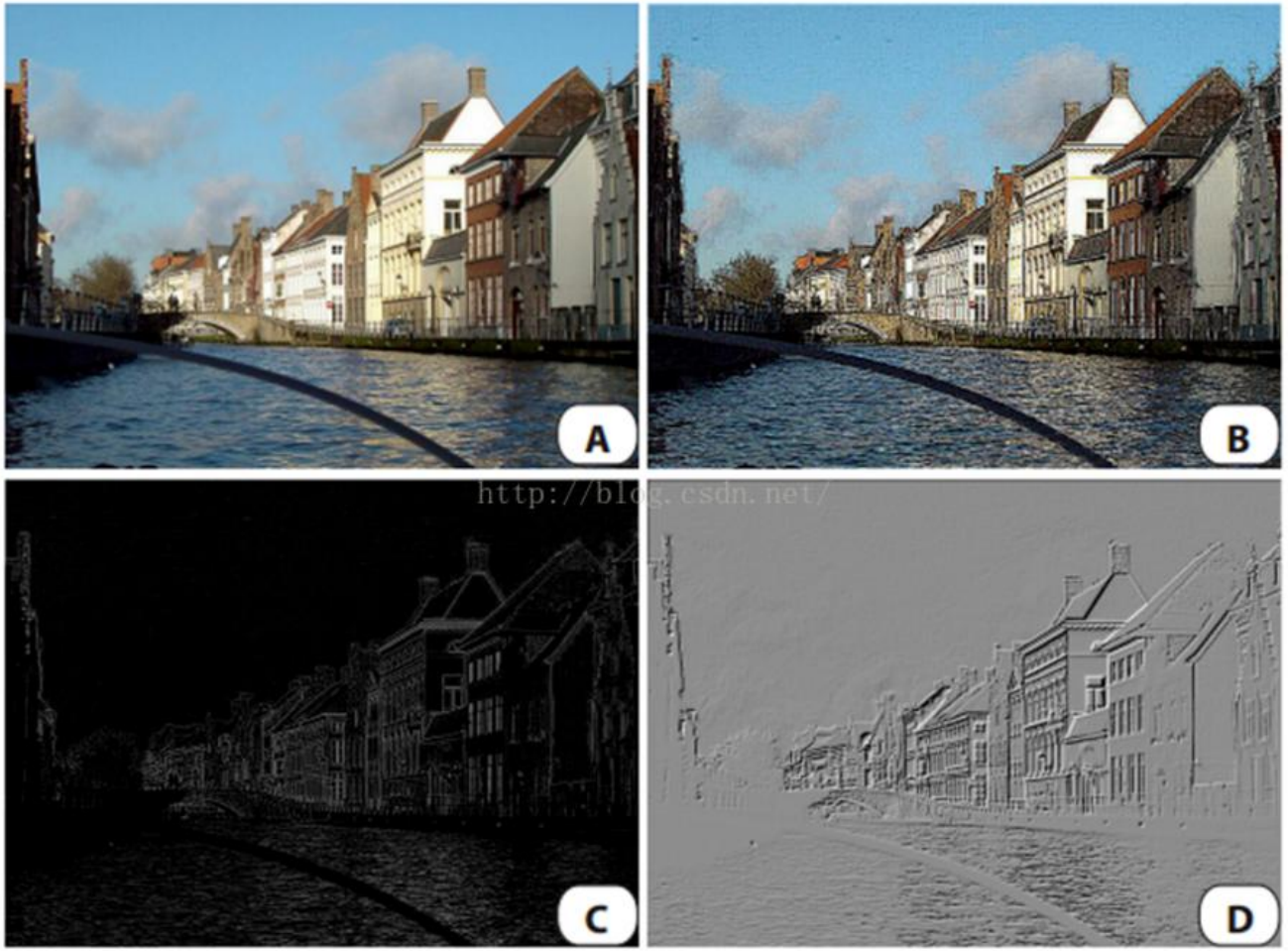
浮雕Embossing Filter

- 浮雕的效果非常的漂亮，就像是将一副图像雕刻在一块石头上面一样，然后从一个方向照亮它。
- 它和前面的滤波器不同：
  - 它是非对称的
  - 它会产生负数值，所以我们需要将结果偏移，以得到图像灰度的范围

$$\begin{bmatrix} 2 & -0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{bmatrix}$$

三种常见卷积核效果对比

- A：原图像。B：锐化。C：边缘检测。D：浮雕

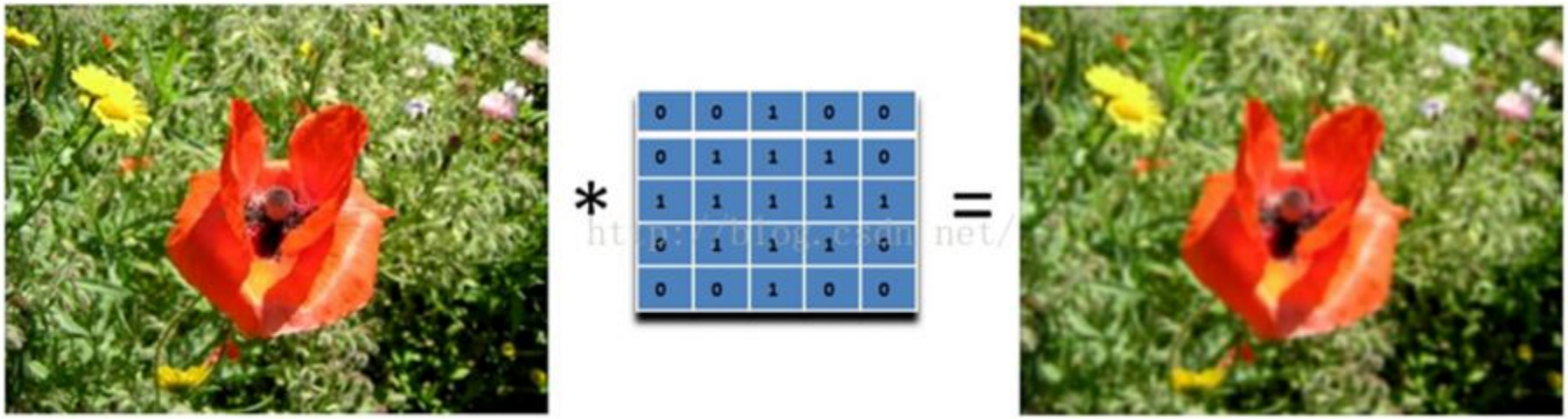


均值模糊Box Filter (Averaging)

- 我们可以将当前像素和它的四邻域的像素一起取平均，然后再除以5，或者直接在滤波器的5个地方取0.2的值即可

均值模糊Box Filter (Averaging)

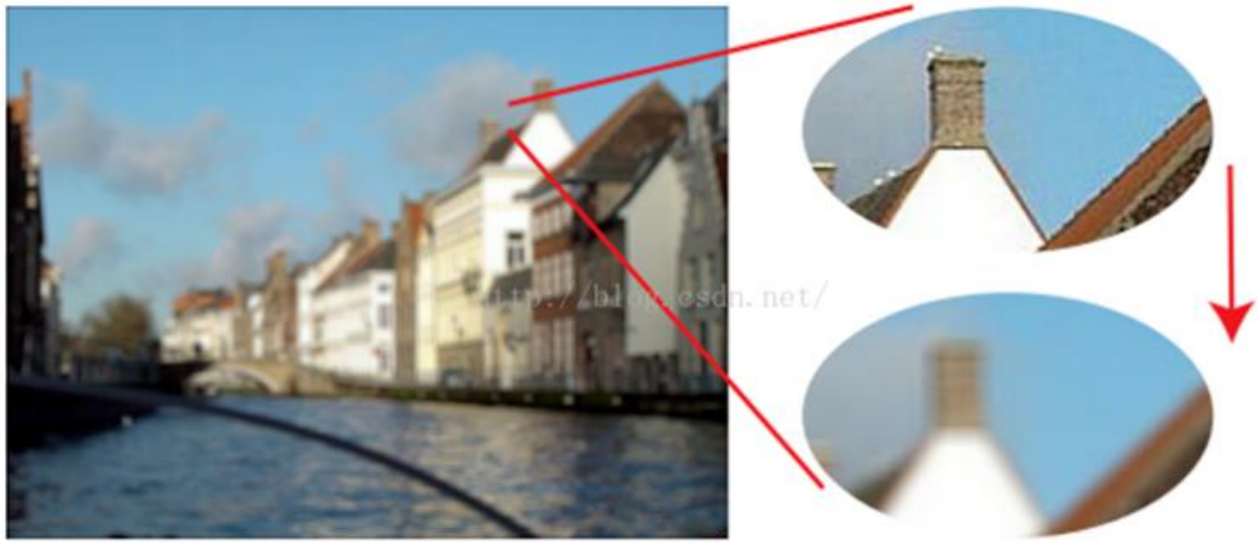
- 把滤波器变大，这样就会变得粗暴了：注意要将和再除以13.



均值模糊Box Filter (Averaging)

- 如果想要更模糊的效果，加大滤波器的大小即可。或者对图像应用多次模糊也可以。

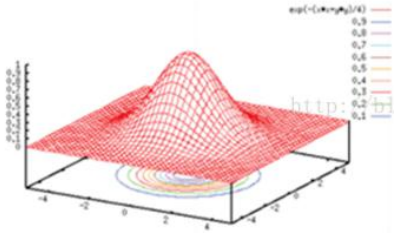
$$K_{box} = \begin{bmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{bmatrix}$$



高斯模糊

- 均值模糊很简单，但不是很平滑。
- 高斯模糊处理比较平滑，所以被广泛用在图像降噪上。特别是在边缘检测之前，都会用来移除细节。
- 高斯滤波器是一个低通滤波器，移除尖锐点。

$$G(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2}{2\sigma^2}}; G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$



$$K_{G5} = \frac{1}{273}\begin{bmatrix} 1 & 4 & 7 & 4 & 1 \\ 4 & 16 & 26 & 16 & 4 \\ 7 & 26 & 41 & 26 & 7 \\ 4 & 16 & 26 & 16 & 4 \\ 1 & 4 & 7 & 4 & 1 \end{bmatrix}$$

Figure 4: *The 2D Gaussian function.*

运动模糊Motion Blur

- 运动模糊可以通过只在一个方向模糊达到，例如下面9x9的运动模糊滤波器。注意，求和结果要除以9。

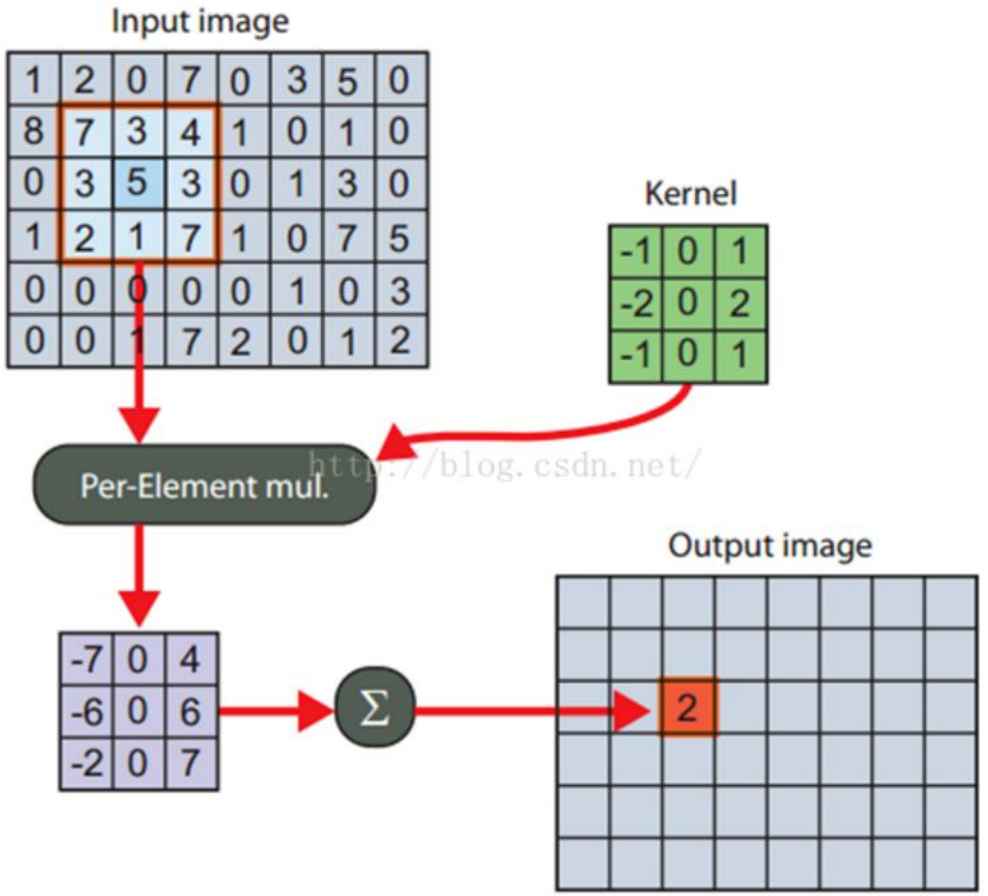
```
1, 0, 0, 0, 0, 0, 0, 0, 0
0, 1, 0, 0, 0, 0, 0, 0, 0
0, 0, 1, 0, 0, 0, 0, 0, 0
0, 0, 0, 1, 0, 0, 0, 0, 0
0, 0, 0, 0, 1, 0, 0, 0, 0
0, 0, 0, 0, 0, 1, 0, 0, 0
0, 0, 0, 0, 0, 0, 1, 0, 0
0, 0, 0, 0, 0, 0, 0, 1, 0
0, 0, 0, 0, 0, 0, 0, 0, 1
```

卷积的计算

- 对图像处理而言，存在两大类的方法
  - 空域处理
    - 直接对原始的像素空间进行计算
  - 频域处理
    - 先对图像变换到频域，再做滤波等处理。

空域计算-直接2D卷积

- 对于图像的每一个像素点，计算它的邻域像素和滤波器矩阵的对应元素的乘积，然后加起来，作为该像素位置的值。



空域计算-边界的处理

- 方法1：扩0法
  - 想象I是无限长的图像的一部分，除了我们给定值的部分，其他部分的像素值都是0
- 方法2：边界值延伸法
  - 想象I是无限图像的一部分，但没有指定的部分是用图像边界的值进行拓展。
- 方法3：周期性扩充法
  - 认为图像是周期性的。也就是I不断的重复。
- 方法4：舍弃无意义的边界值
  - 不管边界了。I之外的情况是没有定义的，所以没办法使用这些没有定义的值。
  - 输出J会比原图像I要小。

频域计算-快速傅里叶变换FFT卷积

- 这个快速实现得益于卷积定理：时域上的卷积等于频域上的乘积。所以将我们的图像和滤波器通过算法变换到频域后，直接将他们相乘，然后再变换回时域（也就是图像的空域）就可以了。

$$(p * k)[n, m] = F(p)[n, m] \cdot F(k)[n, m]$$

$$\mathbf{I} * \mathbf{K} = \mathrm{IFFT}_2(\mathrm{FFT}_2(\mathbf{I}) \circ \mathrm{FFT}_2(\mathbf{K}))$$

REFERENCE

- http://blog.csdn.net/zouxy09/article/details/49080029

The end