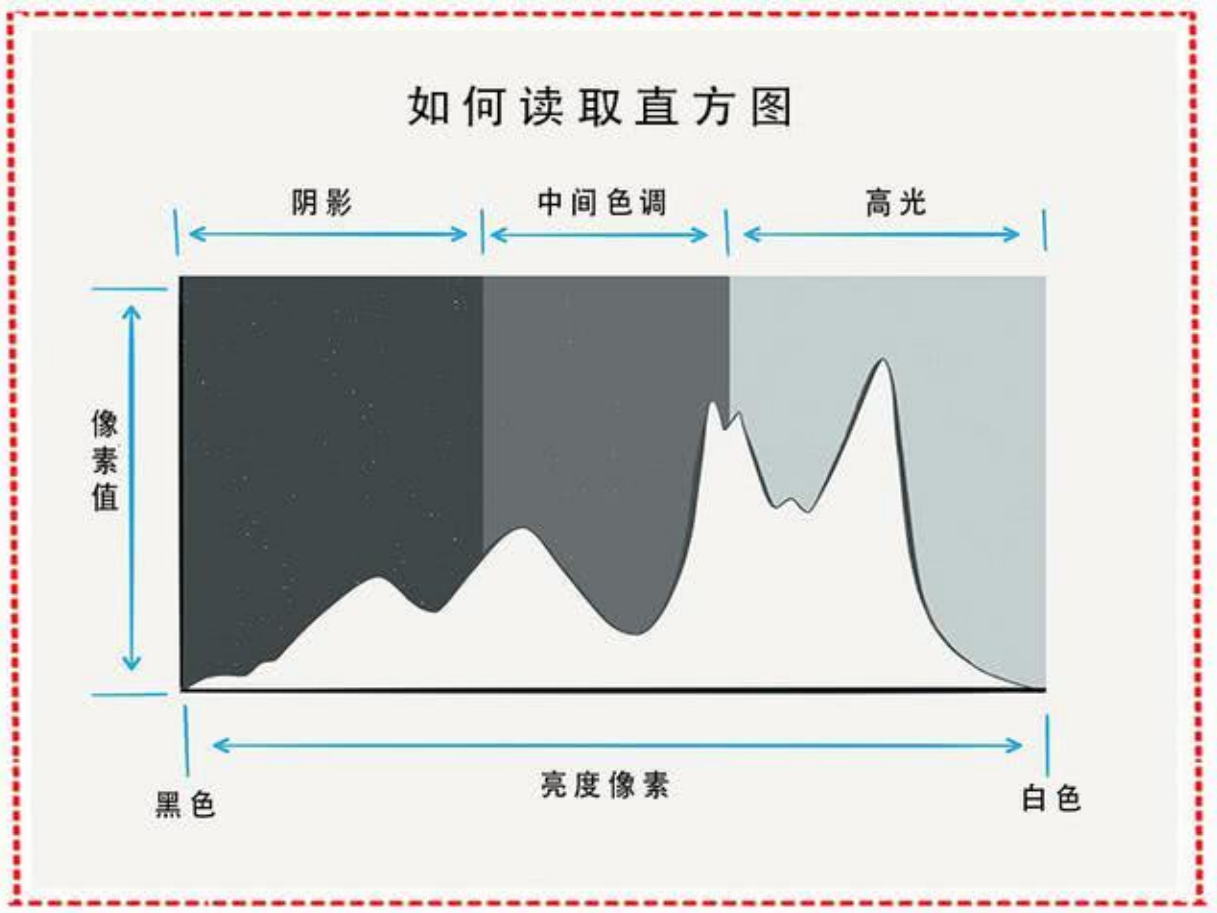
1、连通域&直方图

- 〖概念〗：图像是由不同数值（颜色）的像素构成，直方图是一个简单的二维表格，用来表示一幅（组）图像中具有某个值的像素的数量，借以描述像素值在整幅图像中的分布情况。



  - 灰度图像的直方图有256个项目（灰度等级，色阶，色域），也叫箱子（bin）。
  - 将直方图归一化，即所有箱子的累加和（像素总数）等于1，那么每个箱子的数值就表示对应的像素数量占总数的百分比。
- 【应用】：
  - 用直方图修改图像的外观；
  - 用直方图标识图像的内容；
  - 用直方图检测图像中的物体或纹理；

1.1 计算图像直方图

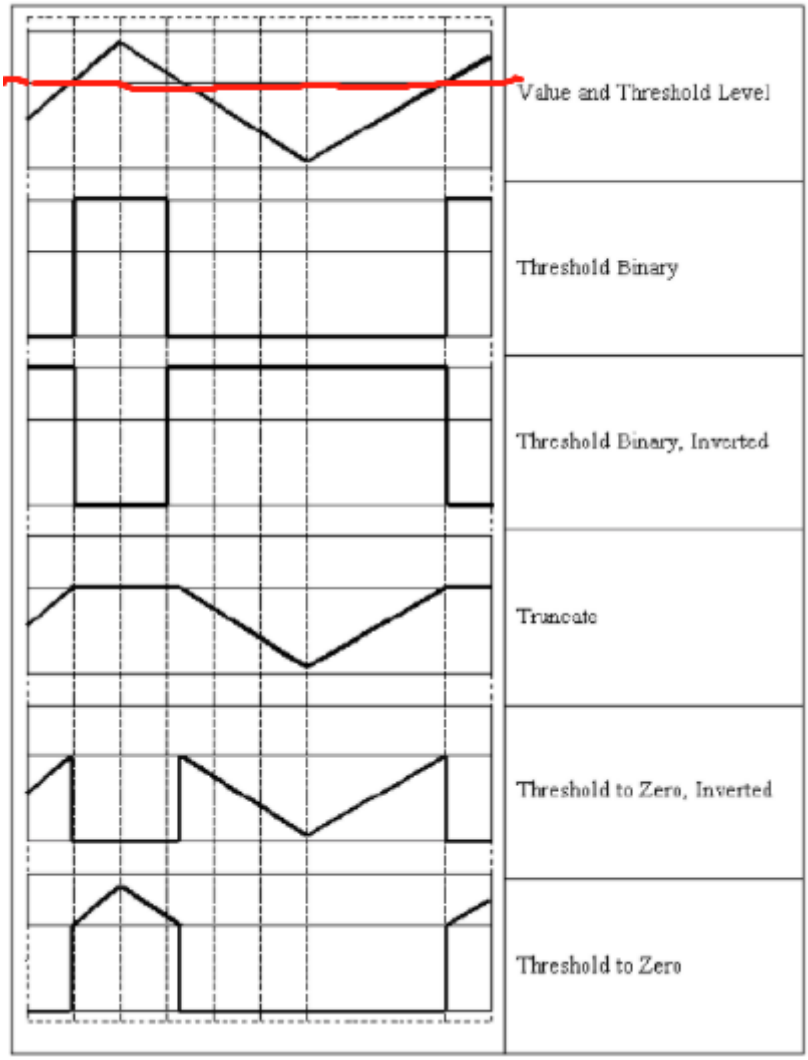
```
void cv::calcHist
(
    const Mat *      images,            //输入图像
    int      nimages,                   //源图像的个数（通常为1）
    const int *      channels,          //列出通道
    InputArray  mask,                   //输入掩码（需处理的像素）
    OutputArray      hist,              //输出直方图
    int      dims,                      //直方图的维度（通道数量）
    const int *      histSize,          //每个维度位数
    const float **  ranges,             //每个维度的范围
    bool     uniform = true,            //true表示箱子间距相同
    bool     accumulate = false         //是否在多次调用时进行累积
    )
```

1.2 二值函数

- cv::threshold()
- 〖原理〗：通过将所有像素与某个阈值（第三个参数）进行比较赋值，将图像表示为只有两种像素值的图像（例子：学生排队）。
- 阈值是图像分割的标尺。

```
#include<opencv2/imgproc.hpp>

//创建二值图像
double cv::threshold
(
    InputArray  src,            //输入图像（多通道、8位或32位浮点类型）
    OutputArray      dst,       //输出图像
    double  thresh,             //指定阈值
    double  maxval,             //设定最大值（常取255）
    int      type               //阈值类型
    )

//type，详见参数介绍
1、THRESH_BINARY：将所有大于thresh的像素赋值为maxval，将其他像素赋值为0；
2、THRESH_BINARY_INY：将所有大于thresh的像素赋值为0，将其他像素赋值为maxval；
3、THRESH_TRUNC：截断，将所有大于thresh的像素赋值为thresh，其他像素值不变；
4、THRESH_TOZERO：所有大于thresh的像素值保持不变，将其他像素赋值为0；
5、THRESH_TOZERO_INV：所有大于thresh的像素值赋值为0，其他像素值保持不变.
//******//
6、THRESH_OTSU:使用Otsu算法去寻找到最优的阈值
7、THRESH_TRIANGLE：使用三角化方法寻找到最有的阈值
```

- 【应用】：
    - 可用于区分图像的前景和背景（通常前景的像素值大于背景的像素值）；



Q:通过图像的直方图如何区分前景和背景?

1.3 连通域

1.3.1 相邻像素

- 位于(x,y)处的像素p有4个水平和垂直的相邻像素，**4邻域**:

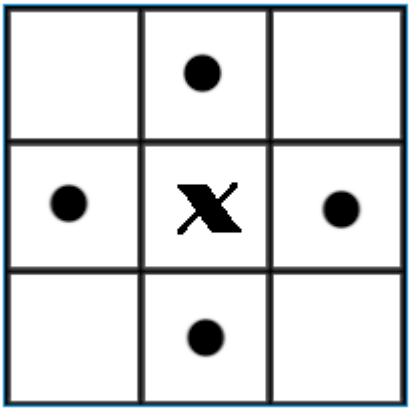
$$N4(p) = （x+1,y）,(x-1,y),(x,y+1),(x,y-1)$$

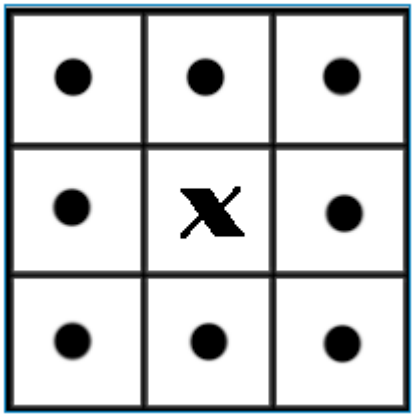
- 位于(x,y)处的像素p有4个对角邻像素:

$$(x+1,y+1),(x+1,y-1),(x-1,y+1),(x-1,y-1)$$

- 与4个邻域点一起称为p的**8邻域**

$$N8(p)$$





> *推论: 如果像素点A与B邻接，我们称A与B连通，于是我们不加证明的有如下的结论：*
>
> *如果A与B连通，B与C连通，则A与C连通。*

1.3.2 邻接性、连通性

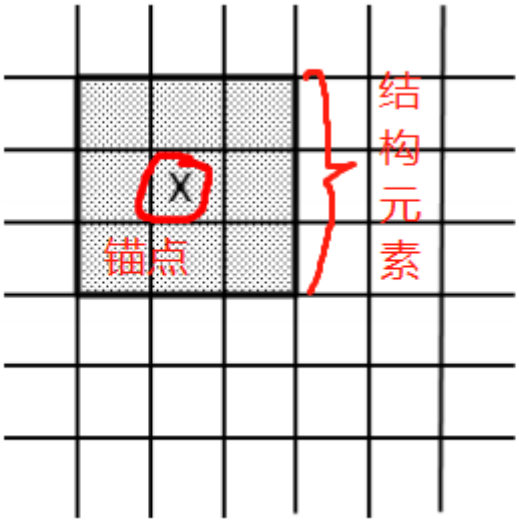
- 〖概念〗：一般是指图像中**(1)具有相同像素值、(2)且位置相邻**的前景像素点组成的图像区域（Region,Blob）。连通域分析是指将图像中的各个连通区域找出并标记。
- 【应用】：
  - OCR识别中字符分割提取（车牌/文本/字幕识别等）；
  - 视觉跟踪中的运动前景目标分割与提取（行人入侵检测、遗留物体检测等）；
- 〖常用算法〗：
  - 原理：通过(1)具有相同像素值，(2)且位置相邻，这两个条件在图像中寻找连通区域，对于找到的每个连通区域，赋予一个唯一的标识（label），以区别其他连通区域。
  - Two-Pass法
  - Seed-Filling种子填充法

```
//Two-Pass（两遍扫描法）
1、第一遍扫描：赋予每个像素位置一个label，扫描过程中同一个连通区域内的像素集合中可能会被赋予一个或多个不同label，因此需要将这些属于同一个连通区域但具有不同值的label合并，也就是记录它们之间的相等关系；

2、第二遍扫描：将具有相等关系的equal_labels所标记的像素归为一个连通区域并赋予一个相同的label（通常这个label是equal_labels中的最小值）。

//Seed-Filling（种子填充法）
1、选取一个前景像素点作为种子；
2、然后根据连通区域的两个基本条件（像素值相同、位置相邻）将与种子相邻的前景像素合并到同一个像素集合中，最后得到的该像素集合则为一个连通区域。
```

> 参考：https://blog.csdn.net/icvpr/article/details/10259577?ops_request_misc=%257B%2522request%255Fid%2522%253A%25221627485459167802640585 11%2522%252C%2522scm%2522%253A%252220140713.130102334..%2522%257D&request_id=162748545916780264058511&biz_id=0&utm_medium=distribute.pc_search_result.none-task-blog-2~all~sobaiduend~default-2-10259577.pc_search_result_control_group&utm_term=%E8%BF%9E%E9%80%9A%E5%9F%9F&spm=1018.2226.3001.4187

2、形态学运算变换图像

- 概念：
  - 形态学是一种滤波器，用结构元素探测图像中每个像素的操作过程称为形态学滤波器的应用过程；
  - 结构元素是一堆像素的组合，原则上可以是任何形状，通常是正方形、圆形或菱形，中心点为原点（锚点）。
- 作用：可用于强化或消除特殊形状。



2.1 腐蚀和膨胀

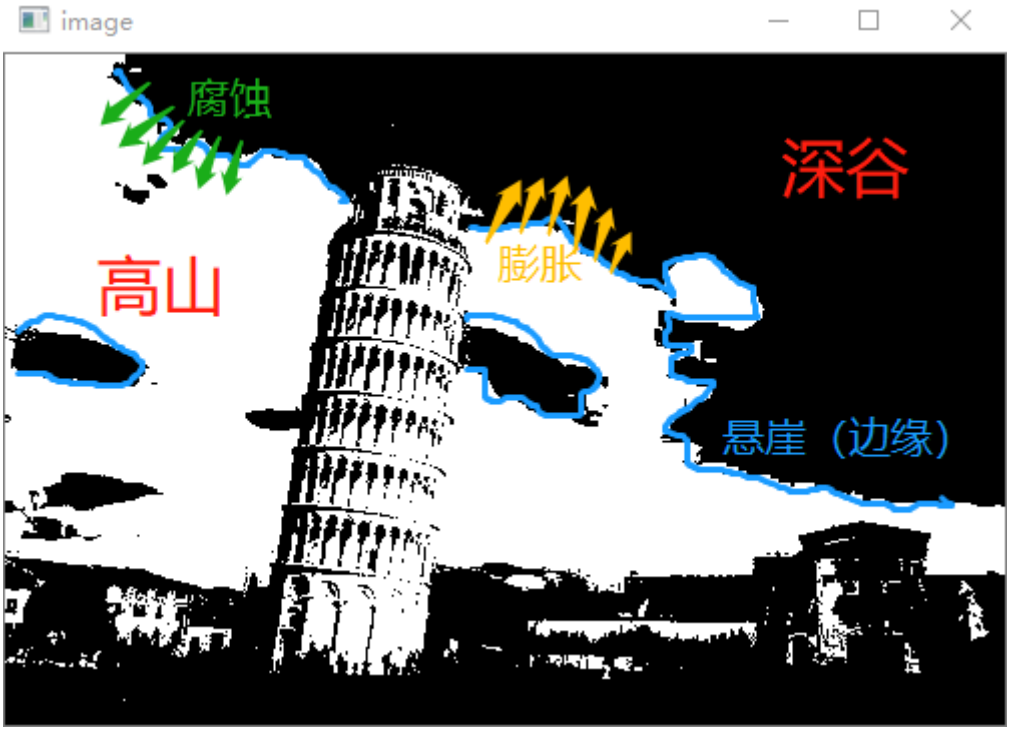
```
//腐蚀图像
//原理：在某个像素上应用结构元素时，结构元素的锚点与该像素对齐，腐蚀就是把当前像素替换成所定义像素集合中的最小像素值。
void cv::erode  (
    InputArray  src,            //输入图像：灰度图像&彩色图像
    OutputArray     dst,        //输出图像
    InputArray  kernel,             //结构元素，默认cv::Mat()，3x3的正方形
    Point   anchor = Point(-1,-1),   //结构元素的锚点位置，默认为中心
    int     iterations = 1,          //腐蚀次数
    int     borderType = BORDER_CONSTANT,   //边界类型（像素外推的方法）
    const Scalar &  borderValue = morphologyDefaultBorderValue()    //连续边界的边界值
)
```

```
//膨胀图像
//原理：把当前像素替换成所定义像素集合中的最大像素值。
void cv::dilate (
    InputArray  src,
    OutputArray     dst,
    InputArray  kernel,
    Point   anchor = Point(-1,-1),
    int     iterations = 1,
    int     borderType = BORDER_CONSTANT,
    const Scalar &  borderValue = morphologyDefaultBorderValue()
)

//增加腐蚀/膨胀次数或者使用更大的结构元素，都会增加腐蚀/膨胀的效果。
```

2.2 形态学梯度运算提取图像边缘

```
//以腐蚀和膨胀操作作为基础，执行高级的形态变换
void cv::morphologyEx   (
    InputArray  src,
    OutputArray     dst,
    int     op,             //形态学操作类型
    InputArray  kernel,
    Point   anchor = Point(-1,-1),
    int     iterations = 1,
    int     borderType = BORDER_CONSTANT,
    const Scalar &  borderValue = morphologyDefaultBorderValue()
)
```

Enumerator	
MORPH_ERODE Python: cv.MORPH_ERODE	see **erode**
MORPH_DILATE Python: cv.MORPH_DILATE	see **dilate**
MORPH_OPEN Python: cv.MORPH_OPEN	an opening operation $$\texttt{dst} = \mathrm{open}(\texttt{src}, \texttt{element}) = \mathrm{dilate}(\mathrm{erode}(\texttt{src}, \texttt{element}))$$
MORPH_CLOSE Python: cv.MORPH_CLOSE	a closing operation $$\texttt{dst} = \mathrm{close}(\texttt{src}, \texttt{element}) = \mathrm{erode}(\mathrm{dilate}(\texttt{src}, \texttt{element}))$$
MORPH_GRADIENT Python: cv.MORPH_GRADIENT	a morphological gradient $$\texttt{dst} = \mathrm{morph\_grad}(\texttt{src}, \texttt{element}) = \mathrm{dilate}(\texttt{src}, \texttt{element}) - \mathrm{erode}(\texttt{src}, \texttt{element})$$
MORPH_TOPHAT Python: cv.MORPH_TOPHAT	"top hat" $$\texttt{dst} = \mathrm{tophat}(\texttt{src}, \texttt{element}) = \texttt{src} - \mathrm{open}(\texttt{src}, \texttt{element})$$
MORPH_BLACKHAT Python: cv.MORPH_BLACKHAT	"black hat" $$\texttt{dst} = \mathrm{blackhat}(\texttt{src}, \texttt{element}) = \mathrm{close}(\texttt{src}, \texttt{element}) - \texttt{src}$$
MORPH_HITMISS Python: cv.MORPH_HITMISS	"hit or miss" .- Only supported for CV_8UC1 binary images. A tutorial can be found in the documentation



2.3 图像分割

cv::watershed()

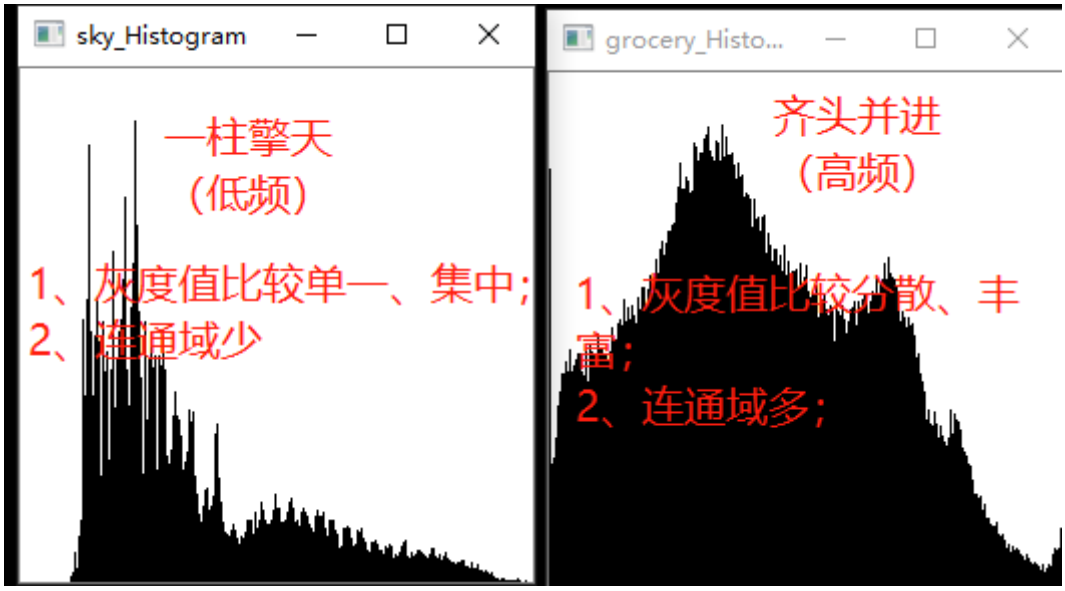
3、图像滤波

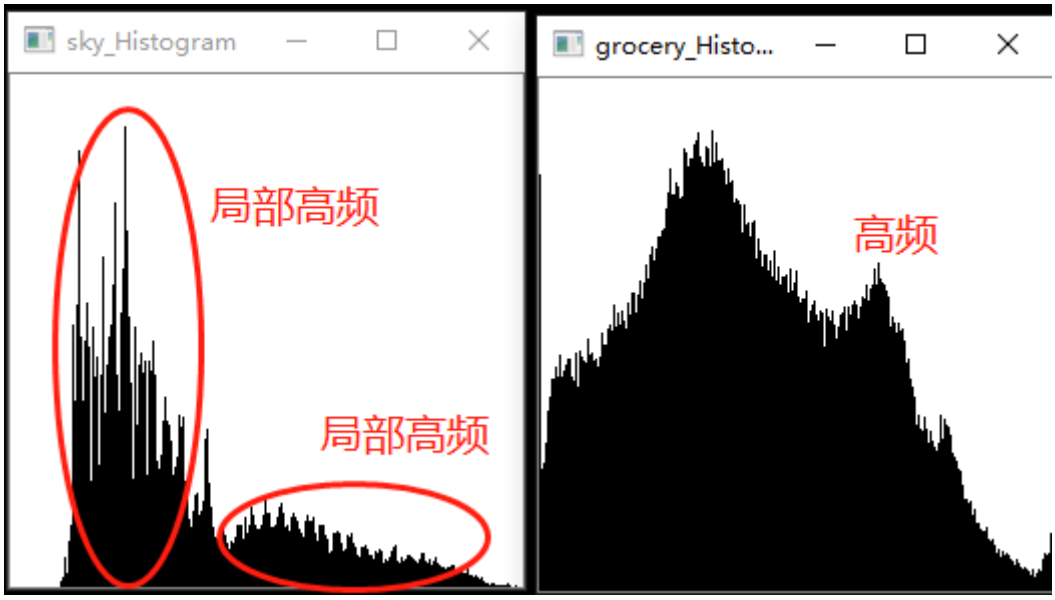
- 概念：即选择性地提取图像中某些方面的内容，这些内容通常在特定的应用环境下传达了重要信息。
- 作用：滤波器是一种放大（也可以不改变）图像中某些频段，同时滤掉（或减弱）其他频段的算子，分为低通滤波器&高通滤波器。
- 示例：去噪（噪声点）、重采样

3.1 频域分析

描述图像的两种方式：

1. 频域：观察图像内容强度值（灰度值）变化的频率（蓝天 VS 杂货间），图像中精致的细节对应着高频；





2. 空域：观察图像内容灰度分布来描述图像特征（——>直方图）。

频域分析：把图像分解成从低频到高频的频率成分。图像强度值变化慢的区域只包含低频率，强度值变化快的区域产生高频率。

二维图像的频率分为垂直频率和水平频率。

3.2 低通滤波器

目的：消除图像中的高频部分，减少图像变化的幅度（把前景变得光滑；把前景和背景之间的差异变小）。

常用方法：把每个像素的值替换成它周围像素的平均值，线性滤波。

3.2.1 块滤波器（box filter）——>卷积核（掩膜）

```
//典型示例一：均值滤波器
void cv::blur    (
    InputArray  src,                    //输入图像
    OutputArray     dst,                //输出图像
    Size    ksize,                      //卷积核大小（值为系统默认指定？）
    Point   anchor = Point(-1,-1),      //锚点位置
    int     borderType = BORDER_DEFAULT //边界类型
)


void cv::boxFilter   (
    InputArray  src,
    OutputArray     dst,
    int     ddepth,
    Size    ksize,
    Point   anchor = Point(-1,-1),
    bool    normalize = true,
    int     borderType = BORDER_DEFAULT
)


void cv::filter2D    (
    InputArray  src,
    OutputArray     dst,
    int     ddepth,
    InputArray  kernel,                         //卷积核值
    Point   anchor = Point(-1,-1),
    double  delta = 0,
    int     borderType = BORDER_DEFAULT
)



//典型示例二：高斯滤波器
void cv::GaussianBlur    (
    InputArray  src,
    OutputArray     dst,
    Size    ksize,                      //滤波器尺寸
    double  sigmaX,                     //控制高斯曲线水平方向形状的参数
    double  sigmaY = 0,                 //控制高斯曲线垂直方向形状的参数
    int     borderType = BORDER_DEFAULT
)
```

注意：高斯滤波（所有的滤波器）里的ksize必须是奇数，否则会引发错误；

> *引申：*
>
> - *Size(5,5), 如何将每个像素的值替换成该像素邻域的平均值?*
>   - *加权累积，求平均值，替换锚点处像素值；（均值滤波器）*
>   - *卷积运算（先360°翻转，再求内积）：*
>
> $$I_{out}(x,y)=\sum_i\sum_j I_{in}(x-i,y-j)K(i,j)$$
>
>   - *参考：https://www.zhihu.com/question/22298352*
> - *CNN——Filter/kernal，不同的滤波器可以分别检测垂直/水平边缘*

3.2.2 图像降采样

- 降低图像精度的过程称为缩减像素采样（downsampling）；
- 提升图像精度的过程称为提升像素采样（upsampling）。

难点：重采样的过程需要尽可能地保持图像质量。

3.2.3 中值滤波器

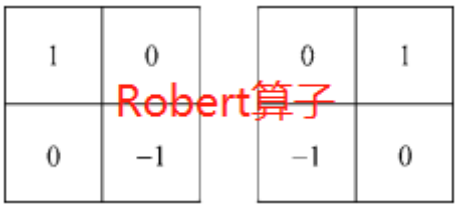
```
//原理：非线性滤波，把当前像素和它的邻域组成一个集合，然后计算出这个集合的中间值，以此作为当前像素的值（用邻域内集合的中位数代替当前像素值）
void cv::medianBlur (
    InputArray  src,
    OutputArray     dst,
    int     ksize                   //注意这里的ksize类型是 int类型
)
```

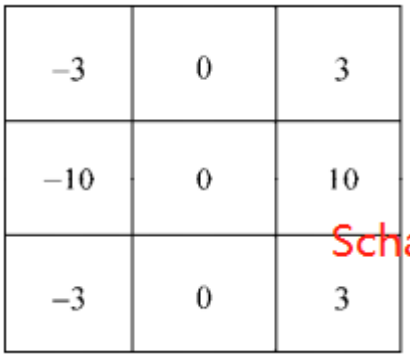
- 特点：异常的0/1值不是中位数，肯定会被邻域的值替换掉。

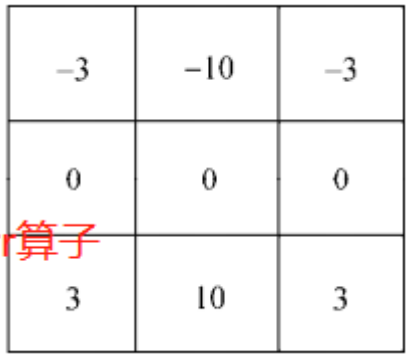
3.3 高通滤波器

3.3.1 定向滤波器（边缘检测）

- 二维图像分为水平方向和垂直方向；
- 比较经典的卷积核，称为算子；







Q:如何辨别x方向和y方向的滤波？（分别沿x/y方向去找灰度值发生急剧变化的边缘处）

```
//Sobel滤波器：只对垂直或水平方向的图像频率起作用
void cv::Sobel  (
    InputArray  src,
    OutputArray     dst,
    int     ddepth,             //位深，-1代表输出图像和源图像的位深相同
    int     dx,                 //x方向的微分，几阶导数
    int     dy,                 //y方向的微分，几阶导数
    int     ksize = 3,          //sobel内核尺寸，只能为 1,3,5或7
    double  scale = 1,
    double  delta = 0,
    int     borderType = BORDER_DEFAULT
)

//Schar滤波器
void cv::Scharr (
    InputArray  src,
    OutputArray     dst,
```

```
    int      ddepth,            //注意，此处位深最好选用CV_16S,否则会丢失很多信息
    int      dx,
    int      dy,
    double   scale = 1,
    double   delta = 0,
    int      borderType = BORDER_DEFAULT
)


//Laplacian算子
void cv::Laplacian  (
    InputArray  src,
    OutputArray     dst,
    int      ddepth,
    int      ksize = 1,
    double   scale = 1,
    double   delta = 0,
    int      borderType = BORDER_DEFAULT
)
```

Depth combinations

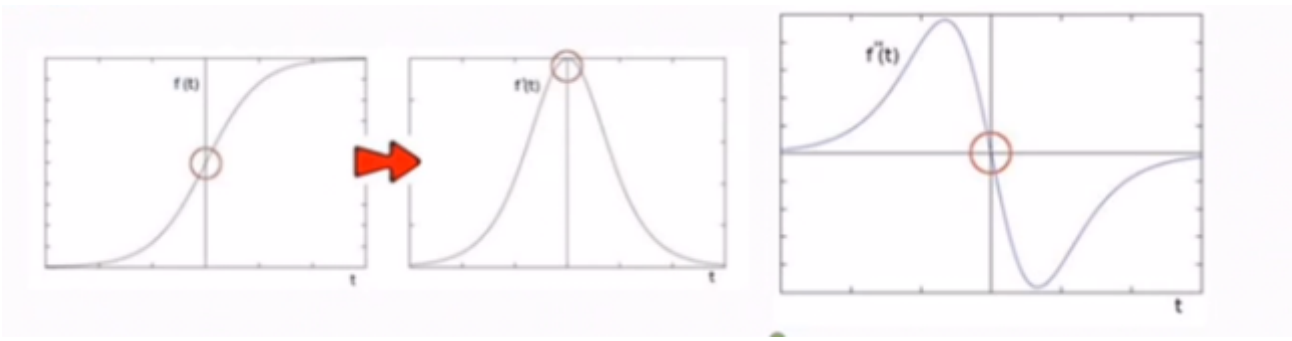
Input depth (src.depth())	Output depth (ddepth)
CV_8U	-1/CV_16S/CV_32F/CV_64F
CV_16U/CV_16S	-1/CV_32F/CV_64F
CV_32F	-1/CV_32F/CV_64F
CV_64F	-1/CV_64F

Note

when ddepth=-1, the output image will have the same depth as the source.

3.3.2 本质

- 区别：一阶导数&二阶导数；



- 解释说明：

- 一阶导数：将图像看作是一个二维函数I(x, y)，其在垂直/水平方向变化速度最快的地方，就是边缘处，此处梯度最大（一阶导数的极大值点）；

$$\mathbf{dst} = \frac{\partial^{xorder+yorder}\mathbf{src}}{\partial x^{xorder}\partial y^{yorder}}$$

- 二阶导数：对二维函数I(x, y)进行二阶求导，则图像在垂直/水平方向变化速度最快的地方，二阶导数值为零，此处即为边缘。

$$\mathbf{dst} = \Delta\mathbf{src} = \frac{\partial^2\mathbf{src}}{\partial x^2} + \frac{\partial^2\mathbf{src}}{\partial y^2}$$

- 区别：
  - （一）、滤波器（尺寸/值）越大，提取边缘越多；
  - （二）、同等尺寸的滤波器，二阶求导比一阶求导提取边缘更多。

Q：在做滤波之前，需要高斯模糊（GaussianBlur()）和灰度转换操作(cvtColor())??

3.3 Canny边缘检测算法

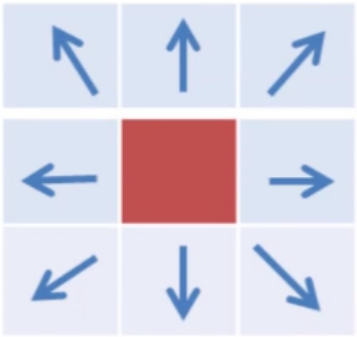
```
//Canny边缘检测算法
void cv::Canny  (
    InputArray  image,            //8位的输入图像
    OutputArray     edges,        //输出图像，一般是二值图像
    double  threshold1,           //低阈值，常取高阈值的1/2或1/3
    double  threshold2,           //高阈值
    int     apertureSize = 3,     //sobel算子的size，通常取值3
    bool    L2gradient = false    //选择true表示用L2归一化，选择false表示用L1
来归一化
)
```

> *注释：*
>
> $L1$范数：$|grad(I) = |Ix| + |Iy|$
>
> $L2$范数：$|grad(I)| = \sqrt{\left(\frac{\partial I}{\partial x}\right)^2 + \left(\frac{\partial I}{\partial y}\right)^2}$

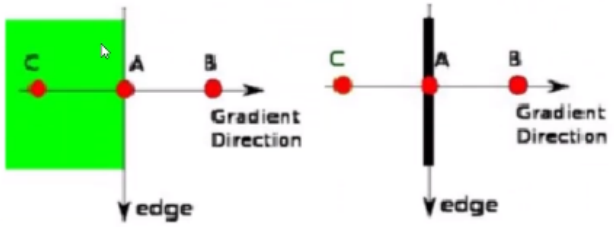
【检测流程】：

1. 高斯滤波，去噪；
2. 灰度变换；
3. 计算图像梯度：sobel算子，计算梯度大小和方向；

4. 非极大值抑制（Non-Maximum Suppression）：利用梯度方向像素来判断当前像素是否为边界点；
    - 梯度通常有四个方向（水平、垂直、两个对角线）；
    - 梯度的方向总是与边缘垂直；



为了简化计算，由于一个像素周围有八个像素，把一个像素的梯度方向离散为八个方向，这样就只需计算前后即可，不用插值了。



5. 应用双阈值（Double-Threshold）检测来确定真实的和潜在的边缘，确定最终的边界。
    - 将小于低阈值的点抛弃，赋值为0；
    - 将大于高阈值的点立即标记（这些点确定位边缘点），赋值为1或255；
    - 将小于高阈值、大于低阈值之间的点，使用8连通区域确定，即只有与高阈值像素点同属于一个连通域时才会被接受称为边缘点，赋值为1或255.
    - max_Val/min_Val通常取值为2：1或3：1

【优势】：

- 使用双阈值分别检测强边缘和弱边缘，并且仅当弱边缘与强边缘相连时，才将弱边缘包含在输出图像中，因此不易受噪声的干扰，更容易检测出真正的弱边缘。

参考资料：

1. OpenCV计算机视觉编程攻略（第3版） 第4、5、6、7章节；
2. https://www.bilibili.com/video/BV1uW411d7Wf?from=search&seid=5643963470540304150 OpenCV图像处理教程；
3. https://blog.csdn.net/icvpr/article/details/10259577?ops_request_misc=%257B%2522request%255Fid%2522%253A%252216274854591678026405851 1%2522%252C%2522scm%2522%253A%25222014071 3.130102334..%2522%257D&request_id=162748545916780264058511&biz_id=0&utm_medium=distribute.pc_search_result.none-task-blog-2~all~sobaiduend~default-2-10259577.pc

_search_result_control_group&utm_term=%E8%BF%9E%E9%80%9A%E5%9F%9F&spm=1018.2226.3001.4187 OpenCV_连通域分析;
4. https://blog.csdn.net/likezhaobin/article/details/6892176?ops_request_misc=%257B%2522request%255Fid%2522%253A%252216277984171678026542195 4%2522%252C%2522scm%2522%253A%25222014071 3.130102334.pc%255Fall.%2522%257D&request_id=162779841716780265421954&biz_id=0&utm_medium=distribute.pc_search_result.none-task-blog-2~all~first_rank_v2~hot_rank-1-6892176.pc_search_result_control_group&utm_term=canny%E8%BE%B9%E7%BC%98%E6%A3%80%E6%B5%8B%E7%AE%97%E6%B3%95&spm=1018.2226.3001.4187 Canny边缘检测算法原理及其VC实现详解