# opencv-basic-demo

## 一些图像基础知识

### 图像的表示

数字图像是指由被称作**像素**的小块区域组成的矩阵

每个像素由两个属性组成：**位置**和亮度（**强度**、灰度）

# 图像的位深

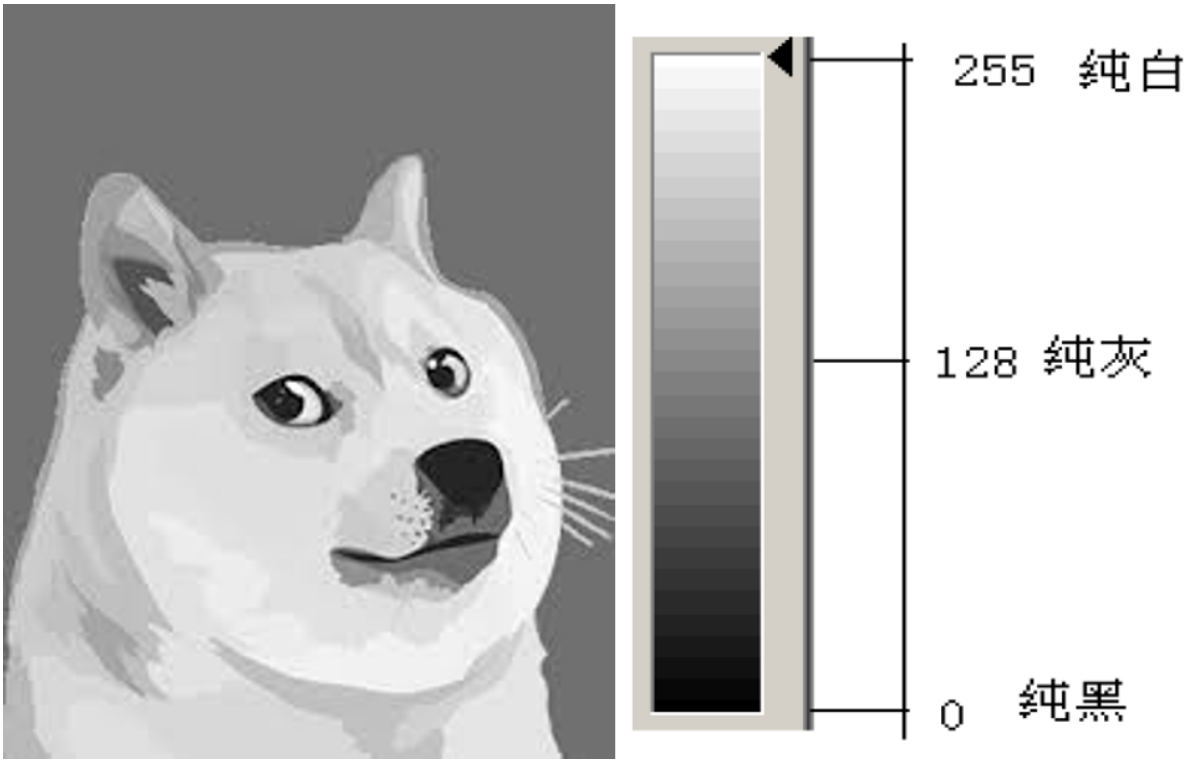
位深度是指在记录数字图像的颜色时，计算机实际上是用**每个像素**需要的**位深度**来表示的。
计算机之所以能够显示颜色，是采用了一种称作“位”( bit ) 的记数单位来记录所表示颜色的数据。

**位深度**是指用多少位来表示一个像素的颜色，位深度越高，所能表示的颜色就越多，图像的颜色就越丰富。

也可以理解为每个像素的强度就是每个像素的位深度。

一位深度的图像，每个像素只有两种可能的值，0和1，即黑和白。

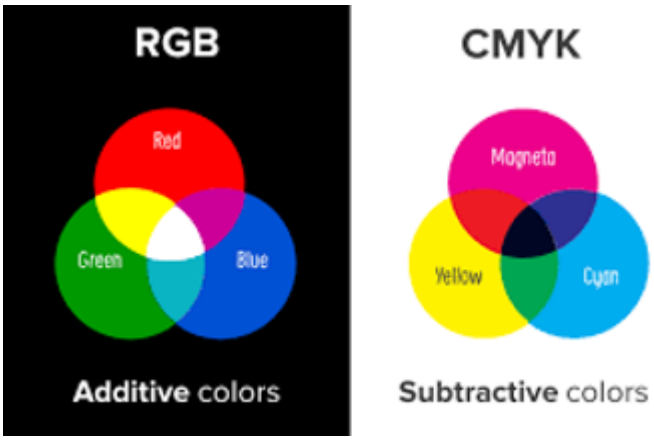
8位深度图像为灰度图，每个像素的位深度用一个8位的二进制数来表示，即0~255。0为黑色，255为白色。之间的数值即为灰度值。



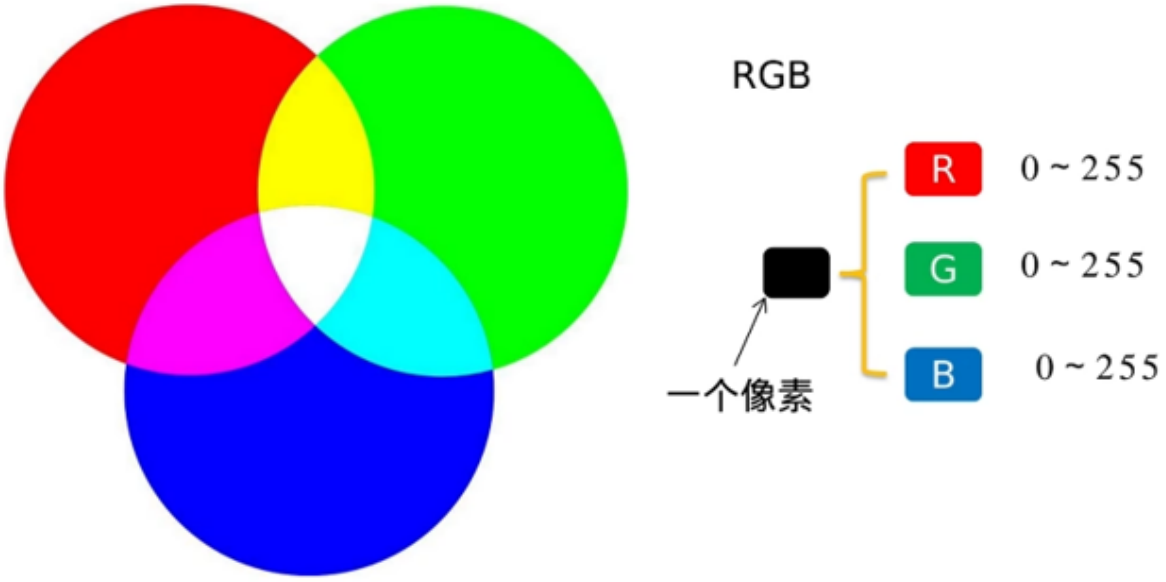

# 色彩模型

色彩模型是一种抽象数学模型，通过一组数字来描述颜色。

常见的色彩模型：三元组（RGB、HSV、YUV）、四元组（RGBa、CMYK）




我们主要了解RGB模型




RGB模型就是把每个像素的值分为三个通道，分别是红色、绿色和蓝色。

- 三个通道的值都是0~255之间的整数，0表示没有颜色，255表示最大的颜色。
- 每个像素点是 B G R 三个通道的值 。
- 255 0 0 表示红色，0 255 0 表示绿色，0 0 255 表示蓝色。
- 0 0 0 表示黑色，255 255 255 表示白色。

三通道混合之后的值就是一个像素的颜色。
RGB模型显示出的就是彩色图。

灰度图中图像像素与通道间的关系：

| | Column 0 | Column 1 | Column ... | Column m |
|---|---|---|---|---|
| Row 0 | 0,0 | 0,1 | ... | 0, m |
| Row 1 | 1,0 | 1,1 | ... | 1, m |
| Row ... | ...,0 | ...,1 | ... | ..., m |
| Row n | n,0 | n,1 | n,... | n, m |

彩色图中图像像素与通道间的关系：

| | Column 0 | | | Column 1 | | | Column ... | | | Column m | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Row 0 | 0,0 | 0,0 | 0,0 | 0,1 | 0,1 | 0,1 | ... | ... | ... | 0, m | 0, m |
| Row 1 | 1,0 | 1,0 | 1,0 | 1,1 | 1,1 | 1,1 | ... | ... | ... | 1, m | 1, m |
| Row ... | ...,0 | ...,0 | ...,0 | ...,1 | ...,1 | ...,1 | ... | ... | ... | ..., m | ..., m |
| Row n | n,0 | n,0 | n,0 | n,1 | n,1 | n,1 | n,... | n,... | n,... | n, m | n, m |

# 简单库函数的使用(图像处理c++)

## 一、图像的读取、显示和保存

```
#include<opencv2/opencv.hpp>
#include<iostream>

using namespace cv;
using namespace std;

int main()
{
    Mat image;   //创建一个空图像image
    image = imread("D://work//c++//Imageprocessing1//企鹅.jpg");  //读取文件夹中的图像

    //检测图像是否加载成功
    if (image.empty())  //检测image有无数据，无数据 image.empty()返回 真
    {
        cout << "Could not open or find the image" << endl;
        return -1;
    }

    namedWindow("IMAGE");  //创建显示窗口，不加这行代码，也能显示，默认窗口大小不能改变
    imshow("IMAGE", image);  //在窗口显示图像

    imwrite("1.png", image); //保存图像为png格式，文件名称为1

    waitKey(0);  //暂停，保持图像显示，等待按键结束

    return 0;
}
```

### 1.1 imread函数

使用imread()读取图像，imread包含两个参数：

**imread(图像路径， 图像形式)；**

其中图像形式有三种：

**1.加载彩色图像**（默认加载形式）:imread(图像路径， IMREAD_COLOR)；

或者：imread(图像路径， 1)；

**2.加载灰度模式图像**：imread(图像路径， IMREAD_GRAYSCALE)；

或者：imread(图像路径， 0)；

**3.加载图像，包括alpha通道**:imread(图像路径，IMREAD_UNCHANGED)；

或者：imread(图像路径， -1)；

### 1.2 namedWindow函数

功能：namedWindow() 的功能就是新建一个显示窗口，用来显示图像。

namedWindow() 包含两个参数：**namedWindow(窗口名称， 窗口形式)**

窗口形式常用的两种：

1.显示的图像大小不能改变（默认形式）:**namedWindow(窗口名称， WINDOW_AUTOSIZE)**

2.图像大小能够调节：**namedWindow(窗口名称， WINDOW_NORMAL)**

### 1.3 imshow函数

功能：imshow函数用于显示图像。

imshow() 函数包含两个参数：**imshow(窗口名称，图像名称)**

### 1.4 imwrite函数

功能：imwrite函数用于显示图像。
imwrite() 函数包含两个参数：**imwrite(保存图像名称及格式，图像名称（变量名）)**

## 二、Mat创建图像（矩阵），获取图像信息，感兴趣区域（Rect）

### 1、创建图像（矩阵）：Mat

使用Mat创建图像（矩阵）的常用形式有：
1.创建一个空图像，大小为0

```
Mat image1;
```

2.指定矩阵大小，指定数据类型：

```
Mat image1(100,100,CV_8U);
```

这里包含三个参数：**矩阵行数，矩阵列数，数据类型**；

其中数据类型有很多种，常用的应该有：

**CV_8U：8位无符号型（0~255），即灰度图像；**

**CV_8UC3：三通道8位无符号型，这里三通道指B（蓝）G（绿）R（红），与matlab中的RGB正好相反。**

**这里创建矩阵时未指定矩阵的值，发现默认值的大小为205.**

注意，创建的图像的数字量指的是该图像有几行几列，100，100即指的是100行100列。共有100×100=10000个像素点。

而数据类型指的是，每个像素点的存储方式，也可以理解为每个像素点是单通道还是三通道。
CV_8U即指的是8位无符号型，即灰度图像，每个像素点只有一个通道。每个像素点的值在0~255之间。0表示黑色，255表示白色。中间的值表示灰度值。

CV_8UC3即指的是三通道8位无符号型，即彩色图像，每个像素点有三个通道，分别是B（蓝）G（绿）R（红）。每个通道的值在0~255之间。0表示该通道颜色最低，255表示该通道颜色最高。

3.指定矩阵大小，指定数据类型，设置初始值：

```
Mat image1(100,100,CV_8U, 100);
```

**这里包含四个参数：矩阵行数，矩阵列数，数据类型，初始值；**
**对于灰度图像：可以直接给出初始值，也可以使用Scalar（）；**

```
Mat image1(100,100,CV_8U, 100);
Mat image1(100,100,CV_8U, Scalar(100));
```

对于三通道图像：使用**Scalar（）；　***这是一种数据类型，一般用来通道赋值或者通道读取**

```
Mat image1(100,100,CV_8UC3, Scalar(100,100,100));
```

上面表示的是，创建一个100行100列的三通道图像，每个像素点的BGR值都为100。

## 2、获取图像信息

获取图像的宽度（列数），高度（行数），尺寸和通道数：

```
    Mat image1 = imread("lena.png");  //读取图像；
    if (image1.empty())
    {
        cout << "读取错误" << endl;
        return -1;
    }

    imshow("image1", image1);  //显示图像；

    cout << "图像的行数为： " << image1.rows << endl;  //获取图像的高度，行数；
    cout << "图像的列数为： " << image1.cols << endl;  //获取图像的宽度，列数；
    cout << "图像的通道数为： " << image1.channels() << endl;  //获取图像的通道数，彩色图=3，灰度图=1；
    cout << "图像的尺寸为： " << image1.size << endl;  //获取图像的尺寸，行*列；

    cout << "图像0的各点像素值为：" << int(image.at<uchar>(0, 0)) << endl;

    cout << "图像2的各点像素通道1为：" << int(image2.at<Vec3b>(0, 0)[0]) << endl;//第一位置的像素点的第一通道
    cout << "图像2的各点像素通道2为：" << int(image2.at<Vec3b>(0, 0)[1]) << endl;
    cout << "图像2的各点像素通道3为：" << int(image2.at<Vec3b>(0, 0)[2]) << endl;
```

**image.at<uchar>(0, 0)是读取灰度图的第一个点的像素值。**

<uchar>是数据类型，uchar是无符号字符型，即灰度图像的像素值。

**image2.at<Vec3b>(0, 0)[0]是读取彩色图的第一个点的第一通道的值。**

## 3、感兴趣区域（ROI）

通过Rect()定义一个感兴趣区域：

```
Mat imageROI(image1, Rect(0,0,10,10));  //定义感兴趣区域
```

其中Rect（）有四个参数，Rect（a,b,c,d）：

- a：感兴趣区域列(cols)的起点；
- b：感兴趣区域行(rows)的起点；
- c：感兴趣区域的列数(cols)；
- d：感兴趣区域的行数(rows)；

之后可以对感兴趣区域做处理

# 三、访问图像像素、遍历图像像素

## 1、访问图像像素

### 1.1访问（j,i）处像素

以8位（0~255）灰度图像和BGR彩色图像为例，用at可以访问图像像素：

```
//灰度图像：
image.at<uchar>(j, i) //j为行数，i为列数
//BGR彩色图像
image.at<Vec3b>(j, i)[0] //B分量
image.at<Vec3b>(j, i)[1] //G分量
image.at<Vec3b>(j, i)[2] //R分量
```

### 1.2 例子：在图像中加入白色椒盐噪声

（1）创建Salt头文件

```
#pragma once
#include<iostream>
#include<opencv2/opencv.hpp>
#include <random>  //随机数头文件


using namespace cv;
using namespace std;

void Salt(Mat image, int n); //n：加入噪声点数
```

（2）创建Salt源文件

```
#include "Salt.h"
void Salt(Mat image, int n)
{
    //随机数生成器
    default_random_engine generater;
    uniform_int_distribution<int>randomRow(0, image.rows - 1);
    uniform_int_distribution<int>randomCol(0, image.cols - 1);

    int i, j;
    for (int k = 0; k < n; k++)
    {
        i = randomCol(generater);
        j = randomRow(generater);
        if (image.channels() == 1)
        {
```

```
            image.at<uchar>(j, i) = 255;
        }
        else if (image.channels() == 3)
        {
            image.at<Vec3b>(j, i)[0] = 255;
            image.at<Vec3b>(j, i)[1] = 255;
            image.at<Vec3b>(j, i)[2] = 255;
        }
    }
}
```

（3）示例

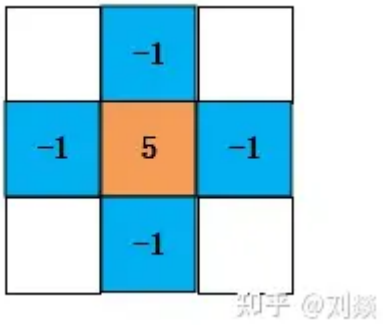
```
int main()
{

    Mat image1 = imread("lena.png");  //读取图像；
    if (image1.empty())
    {
        cout << "读取错误" << endl;
        return -1;
    }
    imshow("image1", image1);  //显示原图像；

    Salt(image1, 5000); //加入5000个噪声点
    imshow("image2", image1);  //显示噪声图像；

    waitKey(0);  //暂停，保持图像显示，等待按键结束
    return 0;
}
```

上述为在图像中加入5000个白色椒盐噪声点。

## 2、遍历图像像素

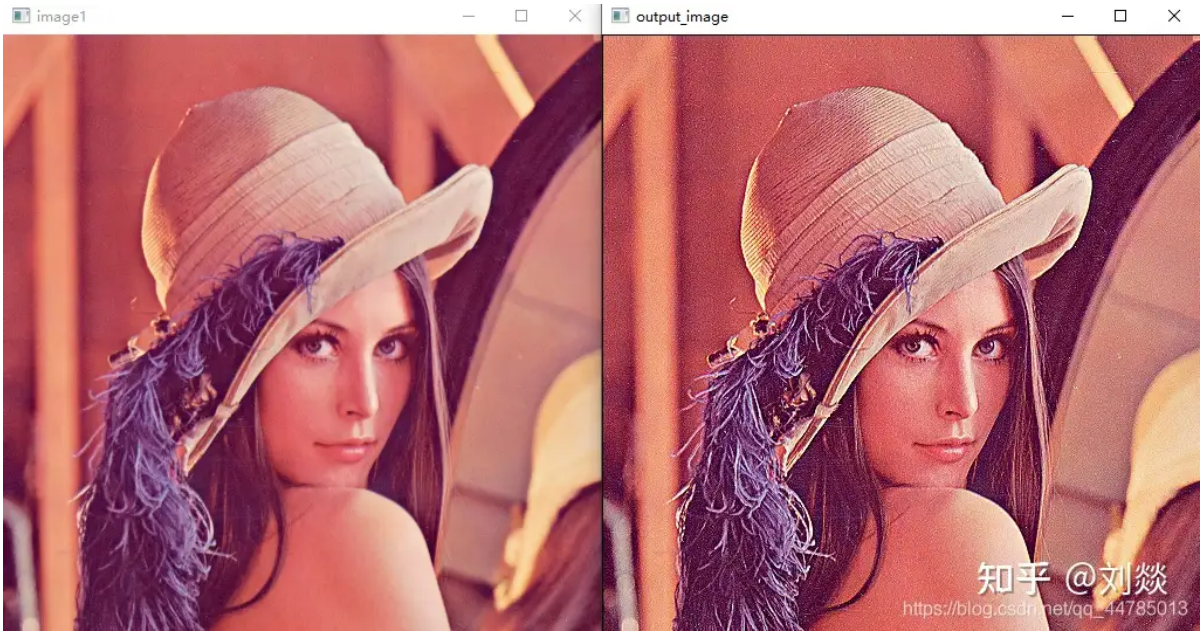
### 2.1指针扫描

以一个卷积操作为例：




```
int main()
{
    Mat image1, output_image;   //定义输入图像和输出图像
    image1 = imread("lena.png");  //读取图像；
    if (image1.empty())
```

```
    {
        cout << "读取错误" << endl;
        return -1;
    }
    output_image = Mat(image1.size(), image1.type());  //定义输出图像大小 和原图像一样
    output_image = image1.clone();   //克隆原图像素值

    int rows = image1.rows;    //原图行数
    int stepx = image1.channels();   //原图通道数
    int cols = (image1.cols) * image1.channels();  //矩阵总列数，在BGR彩色图像中，每个像素的BGR通道按顺序排列，因此总列数=像素宽度*通道数

    for (int row =1 ; row < (rows - 1); row++)   //对行遍历
    {
        const uchar* previous = image1.ptr<uchar>(row - 1);  //原图上一行指针
        const uchar* current = image1.ptr<uchar>(row);       //原图当前行指针
        const uchar* next = image1.ptr<uchar>(row + 1);      //原图下一行指针
        uchar* output = output_image.ptr<uchar>(row);        //输出图像当前行指针

        for (int col = stepx; col < (cols- stepx); col++)  //对列遍历
        {
            output[col] = saturate_cast<uchar>(5*current[col] - (previous[col]+ current[col- stepx]+ current[col + stepx]+ next[col]));
            //saturate_cast<uchar>(a)，当a在0—255时输出a，当a小于0输出0，当a大于255输出255，保证a的值在0~255之间
        }
    }

    imshow("image1", image1);
    imshow("output_image", output_image);

    waitKey(0);  //暂停，保持图像显示，等待按键结束
    return 0;
}
```



### 2.2 opencv自带的卷积运算：filter2D

上面的方法可以简化为：

```
int main()
{
    Mat image1, output_image;   //定义输入图像和输出图像
    image1 = imread("lena.png");  //读取图像；
    if (image1.empty())
    {
        cout << "读取错误" << endl;
        return -1;
    }

    Mat kernel = (Mat_<char>(3,3) << 0, -1, 0, -1, 5, -1, 0, -1, 0);  //创建滤波器
锐化
    filter2D(image1, output_image, image1.depth(), kernel);  //卷积

    imshow("image1", image1);
    imshow("output_image", output_image);
}
```

**根据卷积核的选取不同，处理效果会不一样。**

# 四、图像的灰度化、二值化

## 灰度化

灰度化有多种方法，这里只介绍一种针对绝大多数的BGR图像：

```
    Mat gray;
    cv::cvtColor(img, gray, COLOR_BGR2GRAY);
    imshow("img", gray);
```

其中img为原图，gray为灰度图的存储位置。

这里是直接调用了openc库的cvtColor函数，对原图的像素直接做灰度化处理。

其他灰度化处理方法参考：[（二）对图像进行预处理（灰度化，二值化） 灰度化和二值化-CSDN博客]

[OpenCV图像灰度化的六种方法_opencv灰度化-CSDN博客]

## 二值化

```
    cv::Mat  gray_binary;
    cv::Mat  gray_binary2;
    //对先灰度化的图进行二值化
    //cv::threshold(gray, gray_binary, 125, 255, cv::THRESH_BINARY_INV);
    //cv::threshold(gray, gray_binary2, 125, 255, cv::THRESH_BINARY);//这个是灰度大
于阈值为最大值，其他为0

    //对原图进行二值化
    int height = img.rows;
    int width = img.cols;
    // 灰度值总和
    int px_t = 0;
```

```
    for (int i = 0; i < height; i++) {
        for (int j = 0; j < width; j++) {
            px_t += int(gray.at<uchar>(i, j));//像素点的灰度值
        }
    }
    // 求像素平均值
    int avg_thresh = px_t / (height * width);

    cout << "阈值:" << avg_thresh << endl;

    cv::threshold(gray, gray_binary, avg_thresh, 255, cv::THRESH_BINARY_INV);// 大于阈值为0，其他为最大值
    cv::threshold(gray, gray_binary2, avg_thresh, 255, cv::THRESH_BINARY);//这个是灰度大于阈值为最大值，其他为0

    cv::imshow("gray_binary", gray_binary);//grar_binary为二值化后图片
    cv::imshow("gray_binary2", gray_binary2);
```

对于灰度图，直接

**cv::threshold(gray, gray_binary, avg_thresh, 255, cv::THRESH_BINARY_INV);**

**参数分别为：灰度图，二值化图，灰度阈值，灰度最大值，二值化方法标志**

**二值化方法标志主要有几种，表示二值化的遵循原则。如上的方法是，大于阈值的灰度置零，其余为最大值**

对于彩色图，一般

**先遍历图像像素值，求出像素平均值，然后拿这个平均值做阈值，再进行二值化**

当然遍历求像素平均值只是一种求阈值的方法，实际针对不同场景还有其他获取阈值的方法，这个可以自行学习。

```
double cv::threshold(InputArray src,OutputArray dst,double  thresh,double
 maxval,int  type)
```

- src：待二值化的图像，图像只能是CV_8U和CV_32F两种数据类型。对于图像通道数目的要求和选择的二值化方法相关。
- dst：二值化后的图像，与输入图像具有相同的尺寸、数据类型和通道数。
- thresh：二值化的阈值。
- maxval：二值化过程的最大值，此参数只在THRESH_BINARY和THRESH_BINARY_INV两种二值化方法中才使用，但是在使用其他方法是也需要输入。
- type：选择图像二值化方法的标志。

1. 该函数是众多二值化方法的集成，就是给定一个阈值，计算所有像素灰度值与这个阈值关系，得到最终的比较结果。
2. 函数中有些阈值比较方法输出结果的灰度值并不是二值的，而是具有一个取值范围，不过为了体现其最常用的功能，我们仍然称其为二值化函数或者阈值比较函数。
3. 函数的部分参数和返回值都是针对特定的算法才有用，但是即使不使用这些算法在使用函数时也需要明确的给出，不可缺省。

4. 函数的最后一个参数是选择二值化计算方法的标志，可以选择二值化方法以及控制哪些参数对函数的计算结果产生影响。

该标志的参数选择和作用：

**表3-2 二值化方法可选择的标志及含义**

| 标志参数 | 简记 | 作用 |
| --- | --- | --- |
| THRESH_BINARY | 0 | 灰度值大于阈值为最大值，其他值为0 |
| THRESH_BINARY_INV | 1 | 灰度值大于阈值为0，其他值为最大值 |
| THRESH_TRUNC | 2 | 灰度值大于阈值的为阈值，其他值不变 |
| THRESH_TOZERO | 3 | 灰度值大于阈值的不变，其他值为0 |
| THRESH_TOZERO_INV | 4 | 灰度值大于阈值的为零，其他值不变 |
| THRESH_OTSU | 8 | 大津法自动寻求全局阈值 |
| THRESH_TRIANGLE | 16 | 三角形法自动寻求全局阈值 |



接下来简单介绍每种标志对应的二值化原理和需要的参数。

**THRESH_BINARY和THRESH_BINARY_INV**

这两个标志是相反的二值化方法

- THRESH_BINARY是将灰度值与阈值（第三个参数thresh）进行比较，如果灰度值大于阈值就将灰度值改为函数中第四个参数maxval的值，否则将灰度值改成0。-
- THRESH_BINARY_INV标志正好与这个过程相反，如果灰度值大于阈值就将灰度值改为0，否则将灰度值改为maxval的值。

两种方法的计算公式：

$$\mathrm{BINARY}(x,y)=\begin{cases}\mathrm{maxval} & \text{if } \mathrm{src}(x,y)>\mathrm{thresh}\\ 0 & \text{otherwise}\end{cases}$$

$$\mathrm{BINARY\_INV}(x,y)=\begin{cases}0 & \text{if } \mathrm{src}(x,y)>\mathrm{thresh}\\ \mathrm{maxval} & \text{otherwise}\end{cases}$$



**THRESH_TRUNC**

这个标志相当于重新给图像的灰度值设定一个新的最大值

- 将大于新的最大值的灰度值全部重新设置为新的最大值
- 具体逻辑为将灰度值与阈值thresh进行比较，如果灰度值大于thresh则将灰度值改为thresh，否则保持灰度值不变。
- 这种方法没有使用到函数中的第四个参数maxval的值，因此maxval的值对本方法不产生影响。

该方法的计算公式：

$$\mathrm{TRUNC}(x,y)=\begin{cases}\mathrm{threshold} & \text{if } \mathrm{src}(x,y)>\mathrm{thresh}\\ \mathrm{src}(x,y) & \text{otherwise}\end{cases}$$

**THRESH_TOZERO和THRESH_TOZERO_INV**

这两个标志是相反的阈值比较方法

- THRESH_TOZERO表示将灰度值与阈值thresh进行比较，如果灰度值大于thresh则将保持不变，否则将灰度值改为0。
- THRESH_TOZERO_INV方法与其相反，将灰度值与阈值thresh进行比较，如果灰度值小于等于thresh则将保持不变，否则将灰度值改为0。
- 这种两种方法都没有使用到函数中的第四个参数maxval的值，因此maxval的值对本方法不产生影响。

该方法的计算公式：

$$\mathrm{TOZERO}(x,y)=\begin{cases}\mathrm{src}(x,y) & \text{if } \mathrm{src}(x,y)>\mathrm{thresh}\\ 0 & \text{otherwise}\end{cases}$$

$$\mathrm{TOZERO\_INV}(x,y)=\begin{cases}0 & \text{if } \mathrm{src}(x,y)>\mathrm{thresh}\\ \mathrm{src}(x,y) & \text{otherwise}\end{cases}$$



前面五种标志都支持输入多通道的图像，在计算时分别对每个通道进行阈值比较。
实际中进行二值化时往往先把图像转为灰度图单通道再进行二值化计算。

**THRESH_OTSU和THRESH_TRIANGLE**

这两种标志是获取阈值的方法，并不是阈值的比较方法的标志。

- 这两个标志可以和前面5种标志一起使用，例如“THRESH_BINARY| THRESH_OTSU”。
- 前面5种标志在调用函数时都需要人为的设置阈值，如果对图像不了解设置的阈值不合理，会对处理后的效果造成严重的影响。
- 这两个标志分别表示利用**大津法（OTSU）** 和 **三角形法（TRIANGLE）** 结合图像灰度值分布特性获取二值化的阈值，并将阈值以函数返回值的形式给出。
- 因此如果函数最后一个参数设置了这两个标志中的任何一个，那么函数第三个参数thresh将由系统自动给出，但是在调用函数的时候仍然不能缺省，只是程序不会使用这个数值。

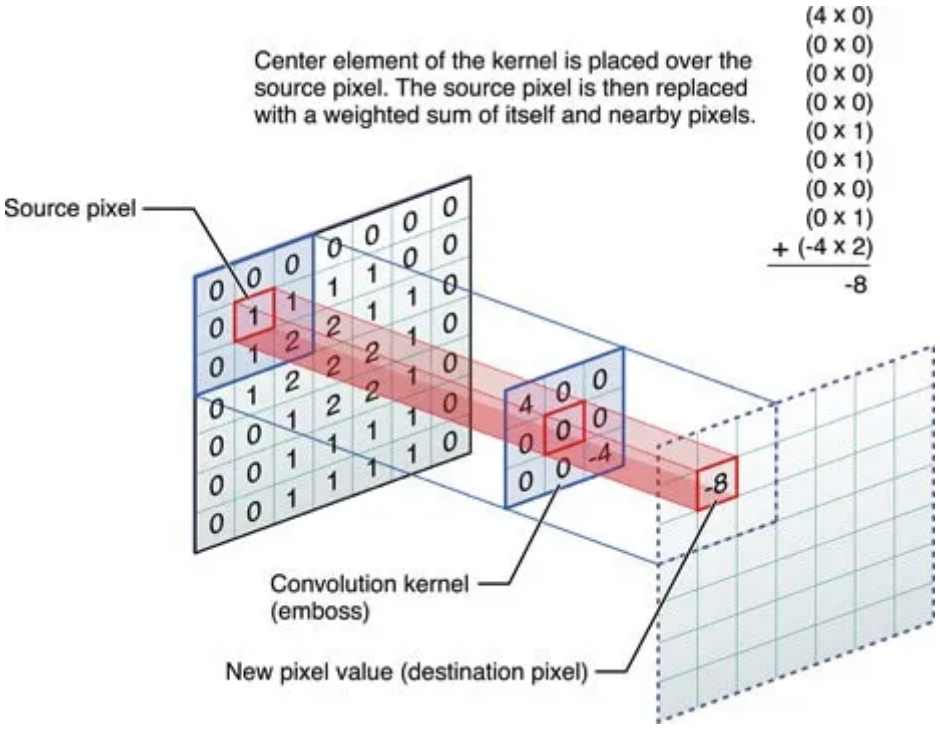
# 六、图像处理中的卷积基础

数学上，卷积表示两个函数 f(x) 和 g(x) 的混合，当一个函数滑过另一个函数。 对于每段微小的滑动位移(dx)，第一个函数的对应点值f(x)和第二个函数的镜像值g(t-x)**相乘**，最后全部的**积加**起来。 结果得到两个函数的卷积。

$$[f*g](t)$$

$$[f * g](t) = \int_0^t f(x)\, g(t-x)\, dx$$

在图像处理领域，**卷积是通过在原图数据矩阵上顺序滑动一个小型数字阵列来执行的。小阵列大小通常为 [3x3] 或 [5x5] ，称为卷积滤波器或内核。** 对于内核阵的每个位置，**与原图的对应像素值相乘最后相加**，代替原来的中心像素，如下图所示。 通过这种方式，邻近像素的值与中心像素的值混合在一起，**创建出含卷积特征的新图像矩阵。** 内核矩阵元素间的相对大小决定了“混合”将如何影响变换后的图像。




**下面是不同的卷积滤波（内核）处理效果：**

1. Identity Kernel：变换得到还是原图

| | | |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 0 | 0 | 0 |

2. Sharpen Kernel: 强调相邻像素值间的差异（锐化）

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

3. Left-to-right Sobel kernel： 强调沿水平轴向上像素之间的差异

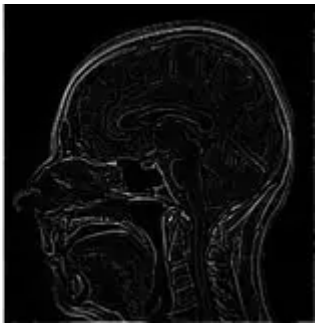
| | | |
|---|---|---|
| 1 | 0 | -1 |
| 2 | 0 | -2 |
| 1 | 0 | -1 |

4. Box Blur Kernel：每个像素与其相邻的8个像素做等权重平均

$$\frac{1}{9}\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix}$$

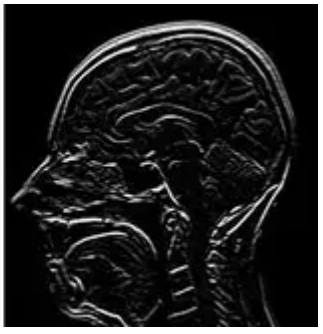
5. Outline（“Edge”）Kernel：突出显示邻近像素，亮度差异性较大

| | | |
|---|---|---|
| -1 | -1 | -1 |
| -1 | 8 | -1 |
| -1 | -1 | -1 |



6. Superior-to-inferior Sobel Kernel：强调自下而上方向上像素之间的差异

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



额外说明：

- **卷积核大小** 通过目标图像的小滑动矩阵的大小。 大小一般为 [3x3]、[5x5] 或 [7x7]。对图像中较小的特征区域， 较小的卷积核有着的更好分辨率，通常是 MRI/放射学处理的首选。 而代价是计算量增加，由于卷积更加密集。

- **填充** 当卷积核移动到图像的边缘或角落附近时，核的一部分势必落到图像之外，而没有可供乘积的像素值。 虽然可以直接简单地忽略掉这些空值，但更常用的方法是在边缘附近创建额外的像素，通常就是常量填充。
- **移动步长(stride)** 步长是指卷积核在前后运算次序之间于图像上移动过的像素数。 **通常 stride = 1**，这意味着滑动框每一步移动一个像素单位。

# 七、绘图

画线

```
void MyLines()
{
    Point p1 = Point(20, 30);
    Point p2;
    p2.x = 300;
    p2.y = 300;
    Scalar color = Scalar(0, 0, 255);
    // 六个参数分别是：原图，起始点，结束点，颜色，线宽，线类型
    line(bgImage, p1, p2, color, 1, 8);
}
```

画圆

```
void MyCircle()
{
    Scalar color = Scalar(0, 255, 255);
    Point center = Point(bgImage.cols / 2, bgImage.rows / 2);
    circle(bgImage, center, 150, color, 2, 8);
    // 六个参数：原图，圆心，半径，颜色，线宽，线类型
}
```

线类型默认是LINE_8 有的opencv版本可以简写为8

# opencv--c++--摄像头操作

## 1.打开摄像头

```
// 1.创建视频采集对象;
    //VideoCapture cap;

    // 2.打开默认相机;
    //cap.open(0);

    VideoCapture cap(0);
    //cap.open(1);

    // 3.判断相机是否打开成功;
    if (!cap.isOpened())
        return;
    // 4.显示窗口命名;

    // With webcam get(CV_CAP_PROP_FPS) does not work.
    // Let's see for ourselves.
```

```
    namedWindow("Video", 1);
    while (1)
    {
        // 获取新的一帧;
        Mat frame;
        cap >> frame;
        if (frame.empty())
            return;
        // 显示新的帧;
        imshow("Video", frame);
        // 按键退出显示;
        if (waitKey(30) >= 0) break;
    }
    // 5.释放视频采集对象;
    cap.release();
```

**在while循环里始终利用cap抓取帧图像并输出。**

## 2.打开视频

打开视频和打开摄像头十分相似，**就是将摄像头换成视频路径即可。**

```
VideoCapture cap("./test.mp4");
    // 3.判断相机是否打开成功;
    if (!cap.isOpened())
        return;
    // 4.显示窗口命名;

    // With webcam get(CV_CAP_PROP_FPS) does not work.
    // Let's see for ourselves.
    namedWindow("Video", 1);
    while (1)
    {
        // 获取新的一帧;
        Mat frame;
        cap >> frame;
        if (frame.empty())
            return;
        // 显示新的帧;
        imshow("Video", frame);
        // 按键退出显示;
        if (waitKey(30) >= 0) break;
    }
    // 5.释放视频采集对象;
    cap.release();
```

## 3.摄像头图像操作

### 灰度化

```
VideoCapture cap(0);

    if (!cap.isOpened())
        return;
```

```
    namedWindow("original", 1);
    namedWindow("gray", 1);
    namedWindow("end", 1);
    while (1)
    {
        // 获取新的一帧;
        Mat frame;
        cap >> frame;
        if (frame.empty())
            return;
        // 显示新的帧;
        imshow("original", frame);

        //对每帧图像进行灰度化
        Mat gray;
        cv::cvtColor(frame, gray, COLOR_BGR2GRAY);
        imshow("gray", gray);

        //对每帧图像膨胀二值化
        Mat dst = ExpansionBina(gray);
        imshow("end", dst);
        // 按键退出显示;
        if (waitKey(30) >= 0) break;
    }
    cap.release();
```

**对每帧读进来的图像直接做灰度化处理，循环。**

## 二值化

```
Mat  ExpansionBina(Mat& img) {
    Mat dst = img;
    blur(dst, dst, Size(1, 3)); // 沿垂直方向模糊目标图像
    int ave = pixAve(img);//这里是计算阈值的平均值
    threshold(dst, dst, ave, 255, THRESH_BINARY); // 二值化目标图像

    Mat kernel = getStructuringElement(MORPH_ELLIPSE, Size(3, 3), Point(-1, -1));
// 创建椭圆结构元素
    dilate(dst, dst, kernel); // 膨胀
    return dst;
}
```

**上面是膨胀二值化的函数封装。可以在读取视频帧后对每帧做如上处理。**

膨胀用了**卷积核**。调用opencv内部库函数，主要作用简单来说就是把每个点的像素值向周围方向扩展一部分。