图像处理基本操作实验
实验二

1、实验介绍

本次实验进行图像处理的基本操作和底层处理算法，了解数字图像的基本形式和基本滤波操作，具体包括

1) 图像读取、写入、平移、旋转、缩放等操作

2) 图像滤波：平滑（均值滤波、高斯滤波、中值滤波、双边滤波）、边缘提取（Sobel、Canny、DOG、LOG）

3) 图像特征：灰度直方图、颜色直方图（RGB 空间、HSV 空间）、方向梯度直方图（Histogram Of Gradient）

2、实验方法

本实验利用OpenCV来完成图像的基本操作,OpenCV的全称是Open Source Computer Vision Library，是一个跨平台的计算机视觉库。

OpenCV 环境配置参见另一文档（OpenCV 环境配置）

3、实验过程

1．图像读取、写入、平移、旋转、缩放等操作

（a）图像读取和显示操作:

一个简单的例子:

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

using namespace std;
using namespace cv;
```

```
int main()
{
    // 读入一张图片
    Mat img = imread("lena.tif");
    // 创建一个名为 " test_figure "窗口
    namedWindow("test_figure");
    // 在窗口中显示lena图像
    imshow("test_figure", img);
    // 等待6000 ms后窗口自动关闭
    waitKey(6000);
}
```

（b）图像写入操作:

imwrite("save_test.png", img);将图像写入名字为“save_test.png”的文件中

(c) 图像平移，旋转，缩放操作:

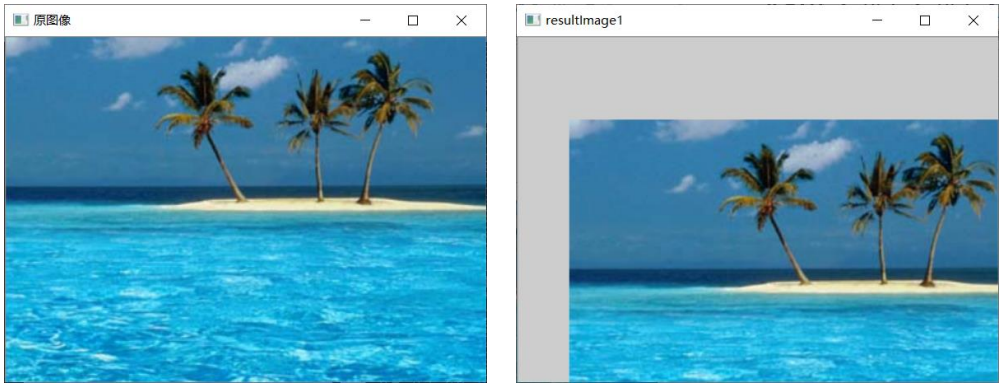
- 图像的平移操作是将图像的所有像素坐标进行水平或者垂直方向的移动，也就是所有像素点按照给定的偏移量在水平方向上沿 X 轴，垂直方向上沿 y 轴移动。

自定义函数实现:

```
Mat imageTranslation(Mat & srcImage, int xOffset, int yOffset)
{
    int nRows = srcImage.rows;
    int nCols = srcImage.cols;
    Mat resultImage(srcImage.size(), srcImage.type());
    //遍历图像
    for (int i = 0; i < nRows; i++)
    {
        for (int j = 0; j < nCols; j++)
        {
            //映射变换
            int x = j - xOffset;
            int y = i - yOffset;
            //边界判断
            if (x >= 0 && y >= 0 && x < nCols && y < nRows)
            {
                resultImage.at<Vec3b>(i, j) = srcImage.ptr<Vec3b>(y)[x];
            }
```

```
            }
        }
        return resultImage;
    }
```

结果:



使用 opencv (warpAffine)函数实现:

函数原型

§ warpAffine()

```
void cv::warpAffine ( InputArray    src,
                      OutputArray   dst,
                      InputArray    M,
                      Size          dsize,
                      int           flags = INTER_LINEAR,
                      int           borderMode = BORDER_CONSTANT,
                      const Scalar & borderValue = Scalar()
                    )
```

Python:

```
dst = cv.warpAffine( src, M, dsize[, dst[, flags[, borderMode[, borderValue]]]] )
```

Applies an affine transformation to an image.

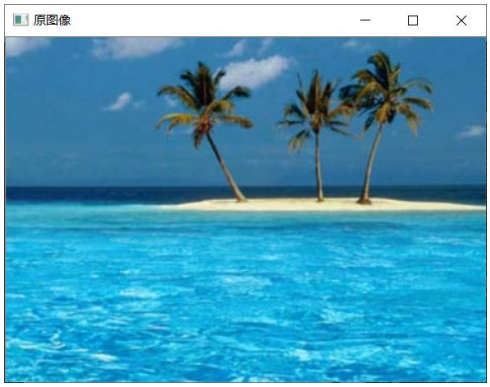
Parameters

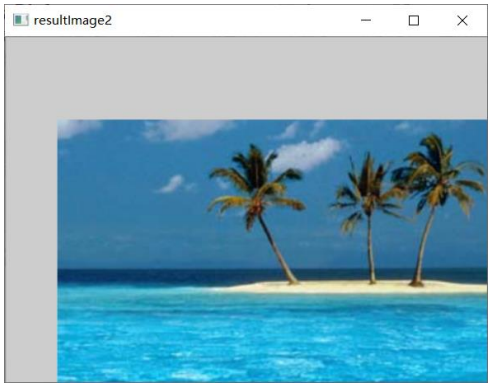
src	input image.
dst	output image that has the size dsize and the same type as src .
M	2×3 transformation matrix.
dsize	size of the output image.
flags	combination of interpolation methods (see **cv::InterpolationFlags**) and the optional flag WARP_INVERSE_MAP that means that M is the inverse transformation (`dst` $\rightarrow$ `src`).
borderMode	pixel extrapolation method (see **cv::BorderTypes**); when borderMode=BORDER_TRANSPARENT, it means that the pixels in the destination image corresponding to the "outliers" in the source image are not modified by the function.
borderValue	value used in case of a constant border; by default, it is 0.

M 是一个 2X3 的矩阵，对应在平移当中，M 矩阵定义为

$$M = \begin{vmatrix} 1 & 0 & tx \\ 0 & 1 & ty \end{vmatrix}$$

```
Mat imageTranslation_opencv(Mat & srcImage, int xOffset, int yOffset)
{
    Mat dst;

    Size dst_sz = srcImage.size();

    //定义平移矩阵
    Mat t_mat = Mat::zeros(2, 3, CV_32FC1);

    t_mat.at<float>(0, 0) = 1;
    t_mat.at<float>(0, 2) = xOffset; //水平平移量
    t_mat.at<float>(1, 1) = 1;
    t_mat.at<float>(1, 2) = yOffset; //竖直平移量

    //根据平移矩阵进行仿射变换
    cv::warpAffine(srcImage, dst, t_mat, dst_sz);

    return dst;
}
```





- 图像的旋转操作:

图像旋转最关键的就是求解旋转矩阵， opencv 里面 getRotationMatrix2D() 可以计算旋转矩阵

getRotationMatrix2D()

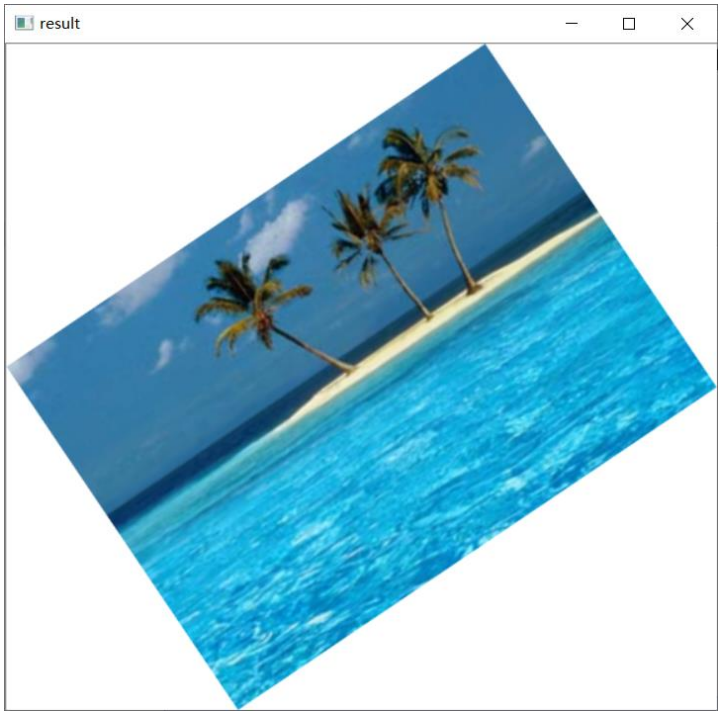
这个函数需要三个参数

1. 旋转中心

2．旋转角度

3. 旋转后图像的缩放比例

之后使用 warpAffine 函数对图像求解即可得到旋转后的图像



- 图像缩放操作:

这里介绍 opencv 里面的 resize() 函数

§ resize()

```
void cv::resize ( InputArray  src,
                  OutputArray dst,
                  Size        dsize,
                  double      fx = 0,
                  double      fy = 0,
                  int         interpolation = INTER_LINEAR
                )
```

Python:

```
dst = cv.resize( src, dsize[, dst[, fx[, fy[, interpolation]]]] )
```

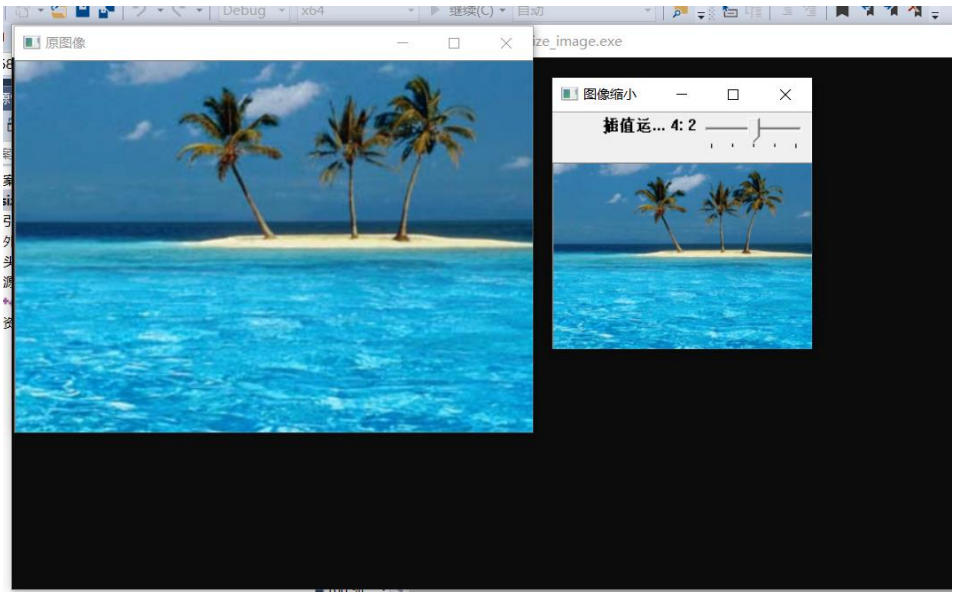
Resizes an image.

参数介绍:

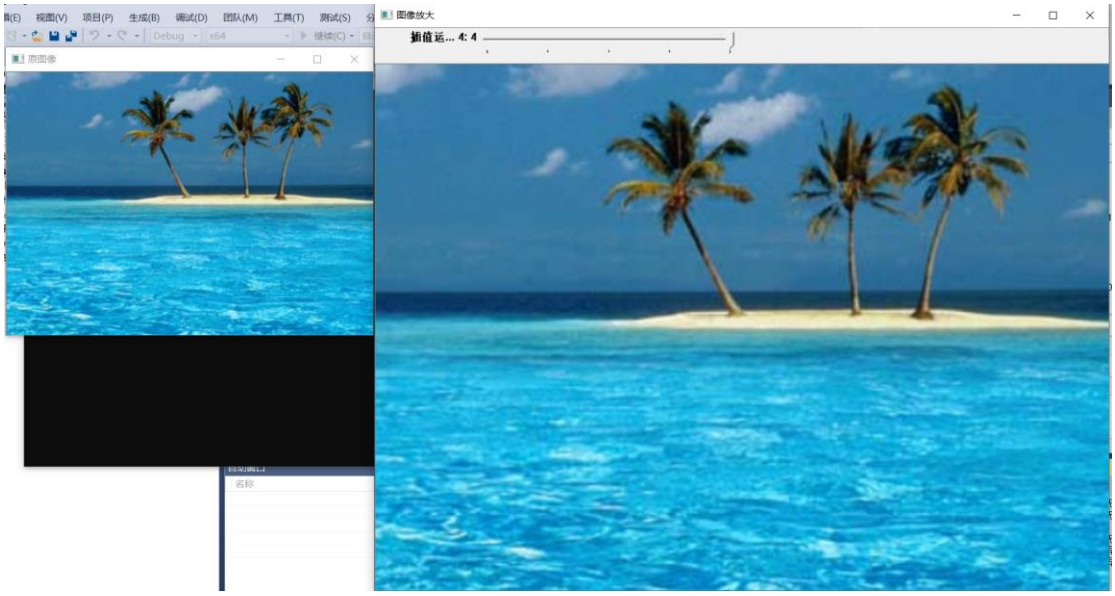
Parameters

src	input image.
dst	output image; it has the size dsize (when it is non-zero) or the size computed from src.size(), fx, and fy; the type of dst is the same as of src.
dsize	output image size; if it equals zero, it is computed as: `dsize = Size(round(fx*src.cols), round(fy*src.rows))` Either dsize or both fx and fy must be non-zero.
fx	scale factor along the horizontal axis; when it equals 0, it is computed as `(double)dsize.width/src.cols`
fy	scale factor along the vertical axis; when it equals 0, it is computed as `(double)dsize.height/src.rows`
interpolation	interpolation method, see **cv::InterpolationFlags**

图像缩小 resize 结果



对图像放大，结果：



2. 图像滤波：平滑、边缘提取（Sobel、Canny、Dog、LOG）

- 平滑，也称为模糊，是一种简单常用的图像处理操作。

通常用滤波器来实现图像的平滑运算。最常见的一类滤波器是线性滤波器，其输出像素值（如，g(i,j)）由输入像素（如，f(i+k,j+l)）的加权和决定：

$$g(i,j) = \sum_{k,l} f(i+k, j+l)h(k,l)$$

其中，h(k,l)称为核，也就是滤波器的系数。滤波过程可以视为一个由系数组成的窗口在图像上滑动。

滤波器有很多种类型，这里只列举最常见的几种：

1）均值滤波器 Normalized Box Filter

所有滤波器中最简单的。每个输出像素就是相邻像素的均值（权重相等，且为 1），即滤波器窗口中所有像素值的平均。

$$K = \frac{1}{K_{width} \cdot K_{height}} \begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & 1 & 1 & \dots & 1 \\ . & . & . & \dots & 1 \\ . & . & . & \dots & 1 \\ 1 & 1 & 1 & \dots & 1 \end{bmatrix}$$

2）高斯滤波器 Gaussian Filter

可能是最有用的滤波器(尽管不是最快的)。高斯滤波器的权重为二维高斯分布。

窗口中的权重与对应像素相乘求和作为目标像素值。

3）中值滤波器 Median Filter

对滤波器窗口中的像素值进行排序，用中间值作为目标像素的像素值。

4）双边滤波器 Bilateral Filter

上面已经解释了一些常见滤波器，主要目标是平滑图像。然而，有时滤波器不仅消除了噪声，还平滑图像中的边缘。为了避免后面情况(至少在一定程度上)，可以使用双边滤波器。

以中值滤波为例：opencv 中函数为 medianBlur()

§ medianBlur()

```
void cv::medianBlur ( InputArray  src,
                      OutputArray dst,
                      int         ksize
                    )
```

Python:

```
dst = cv.medianBlur( src, ksize[, dst] )
```

Blurs an image using the median filter.

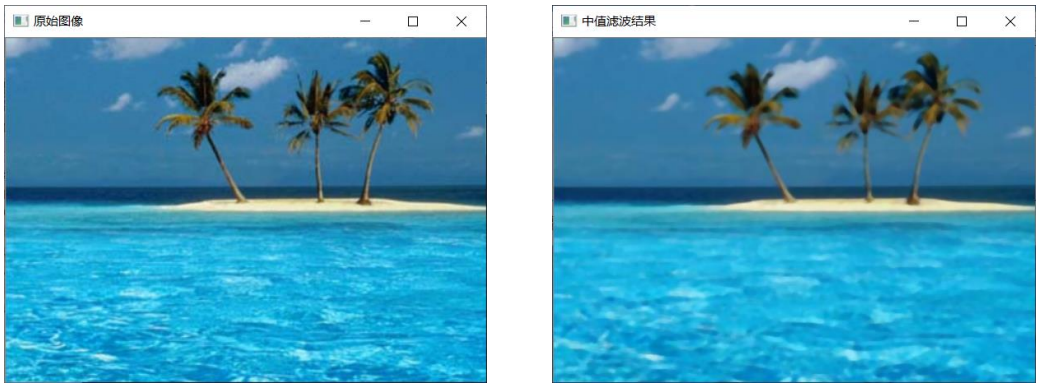
参数含义：

Src:输入图像

Dst：输出图像

Ksize：滤波器大小

实验结果：



- 边缘提取（Sobel、Canny、DoG、LOG）

边缘检测的目的是标识数字图像中亮度变化明显的点。图像边缘检测大幅度地减少了数据量，并且剔除了可以认为不相关的信息，保留了图像重要的结构属性。

1） Sobel 边缘提取：

§ Sobel()

```
void cv::Sobel ( InputArray  src,
                 OutputArray dst,
                 int         ddepth,
                 int         dx,
                 int         dy,
                 int         ksize = 3,
                 double      scale = 1,
                 double      delta = 0,
                 int         borderType = BORDER_DEFAULT
               )
```

Python:

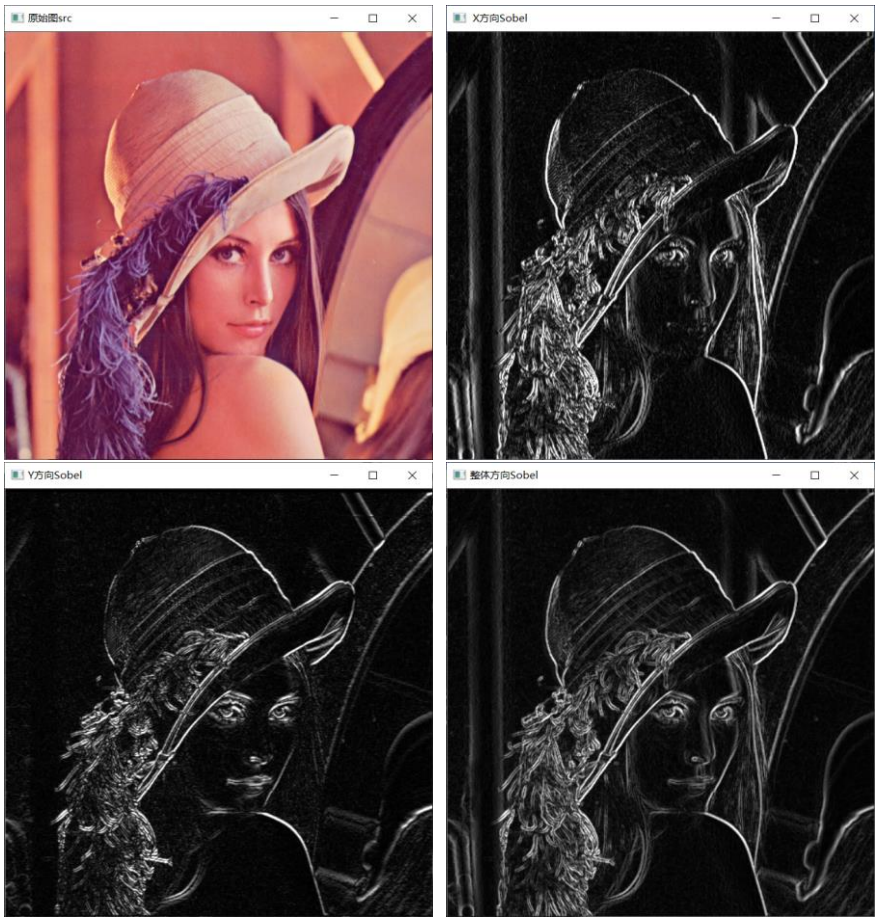
```
dst = cv.Sobel( src, ddepth, dx, dy[, dst[, ksize[, scale[, delta[, borderType]]]]] )
```

参数解释：

- 第一个参数，InputArray 类型的 src，为输入图像，填 Mat 类型即可。
- 第二个参数，OutputArray 类型的 dst，即目标图像，函数的输出参数，需要和源图片有一样的尺寸和类型。
- 第三个参数，int 类型的 ddepth，输出图像的深度，支持如下 src.depth()和 ddepth 的组合：
-
  - 若 src.depth() = CV_8U，取 ddepth =-1/CV_16S/CV_32F/CV_64F
  - 若 src.depth() = CV_16U/CV_16S，取 ddepth =-1/CV_32F/CV_64F
  - 若 src.depth() = CV_32F，取 ddepth =-1/CV_32F/CV_64F
  - 若 src.depth() = CV_64F，取 ddepth = -1/CV_64F
- 第四个参数，int 类型 dx，x 方向上的差分阶数。
- 第五个参数，int 类型 dy，y 方向上的差分阶数。

- 第六个参数，int 类型 ksize，有默认值 3，表示 Sobel 核的大小;必须取 1，3，5 或 7。
- 第七个参数，double 类型的 scale，计算导数值时可选的缩放因子，默认值是 1，表示默认情况下是没有应用缩放的。
- 第八个参数，double 类型的 delta，表示在结果存入目标图（第二个参数 dst）之前可选的 delta 值，有默认值 0。
- 第九个参数， int 类型的 borderType，我们的老朋友了（万年是最后一个参数），边界模式，默认值为 BORDER_DEFAULT。

运行结果：



2） Canny 边缘提取：

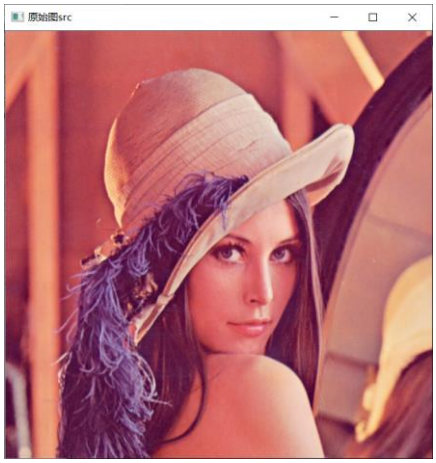
§ Canny() [1/2]

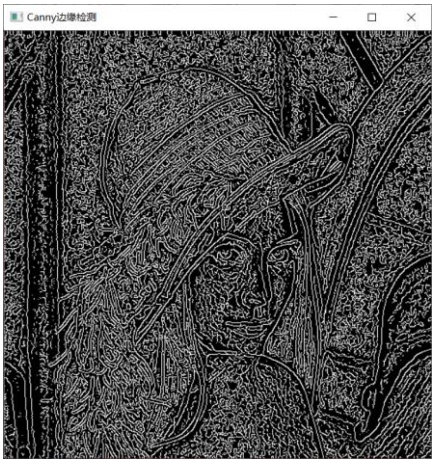
```
void cv::Canny ( InputArray  image,
                 OutputArray edges,
                 double      threshold1,
                 double      threshold2,
                 int         apertureSize = 3,
                 bool        L2gradient = false
               )
```

Python:

```
edges = cv.Canny( image, threshold1, threshold2[, edges[, apertureSize[, L2gradient]]] )
edges = cv.Canny( dx, dy, threshold1, threshold2[, edges[, L2gradient]]                 )
```

- 第一个参数，InputArray 类型的 image，输入图像，即源图像，填 Mat 类的对象即可，且需为单通道 8 位图像。
- 第二个参数，OutputArray 类型的 edges，输出的边缘图，需要和源图片有一样的尺寸和类型。
- 第三个参数，double 类型的 threshold1，第一个滞后性阈值。
- 第四个参数，double 类型的 threshold2，第二个滞后性阈值。
- 第五个参数，int 类型的 apertureSize，表示应用 Sobel 算子的孔径大小，其有默认值 3。
- 第六个参数，bool 类型的 L2gradient，一个计算图像梯度幅值的标识，有默认值 false。





3)　DoG 边缘提取：

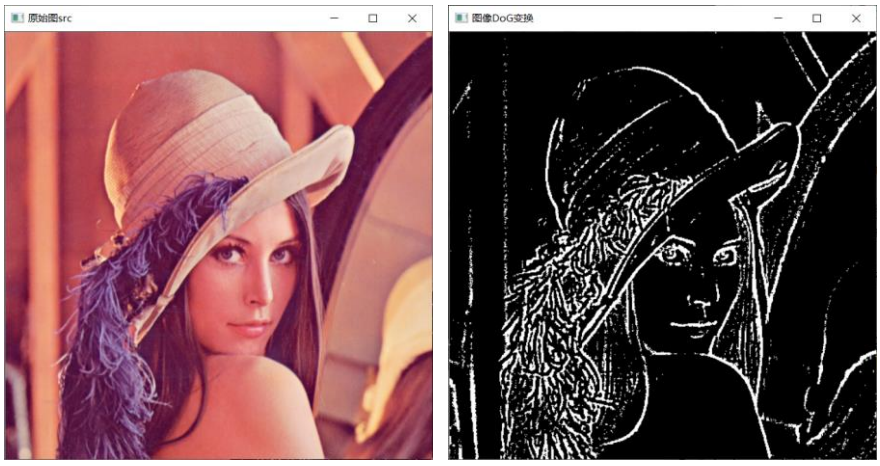
§ GaussianBlur()

```
void cv::GaussianBlur ( InputArray  src,
                        OutputArray dst,
                        Size        ksize,
                        double      sigmaX,
                        double      sigmaY = 0,
                        int         borderType = BORDER_DEFAULT
                      )
```

Python:

```
dst = cv.GaussianBlur( src, ksize, sigmaX[, dst[, sigmaY[, borderType]]] )
```

- 第一个参数，InputArray 类型的 image，输入图像，即源图像，填 Mat 类的对象即可。
- 第二个参数，OutputArray 类型的 edges，输出的边缘图，需要和源图片有一样的尺寸和通道数。
- 第三个参数，int 类型的 ksize，用于计算二阶导数的滤波器的孔径尺寸，大小必须为正奇数。
- 第四个参数，double 类型的 sigmaX，x 方向的标准差。
- 第五个参数，double 类型的 sigmaY，y 方向的标准差，有默认值 0。
- 第六个参数， int 类型的 borderType，边界模式，默认值为 BORDER_DEFAULT。这个参数可以在官方文档中 borderInterpolate()处得到更详细的信息。



4） LOG 边缘提取：

§ Laplacian()

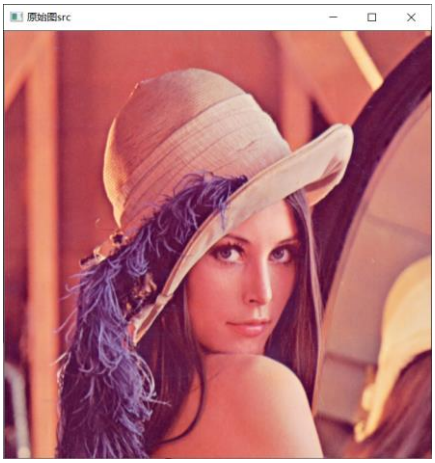
```
void cv::Laplacian ( InputArray  src,
                     OutputArray dst,
                     int         ddepth,
                     int         ksize = 1,
                     double      scale = 1,
                     double      delta = 0,
                     int         borderType = BORDER_DEFAULT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

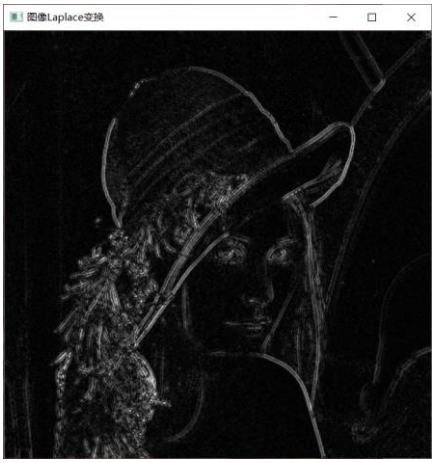
Python:

```
dst = cv.Laplacian( src, ddepth[, dst[, ksize[, scale[, delta[, borderType]]]]] )
```

- 第一个参数，InputArray 类型的 image，输入图像，即源图像，填 Mat 类的对象即可，且需为单通道 8 位图像。
- 第二个参数，OutputArray 类型的 edges，输出的边缘图，需要和源图片有一样的尺寸和通道数。
- 第三个参数，int 类型的 ddept，目标图像的深度。
- 第四个参数，int 类型的 ksize，用于计算二阶导数的滤波器的孔径尺寸，大小必须为正奇数，且有默认值 1。

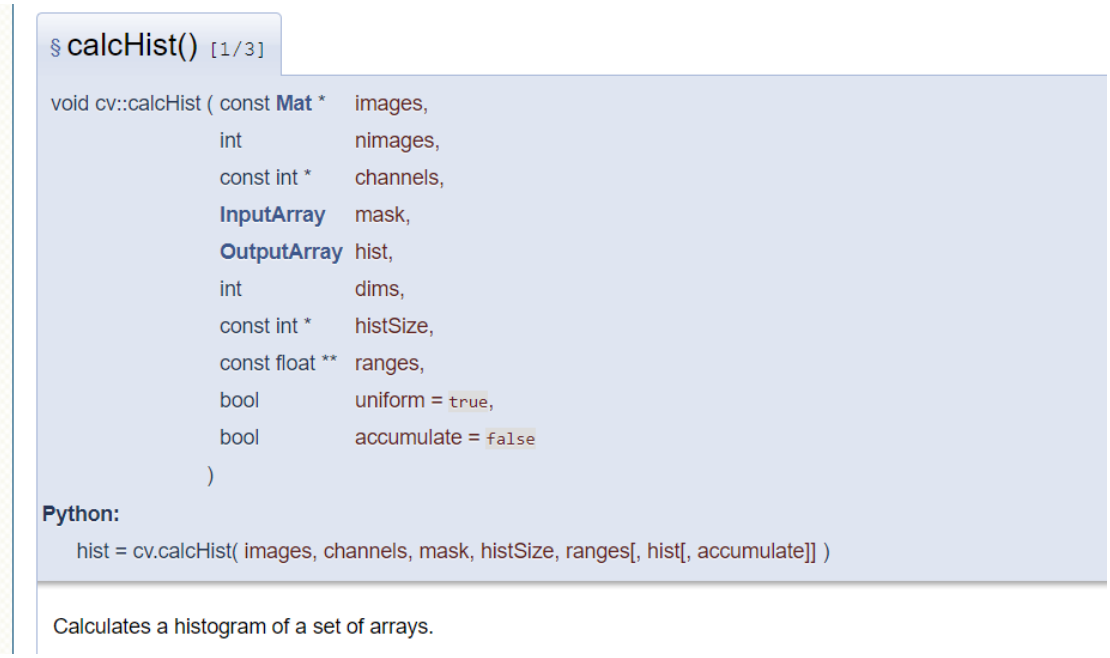
- 第五个参数，double 类型的 scale，计算拉普拉斯值的时候可选的比例因子，有默认值 1。
- 第六个参数，double 类型的 delta，表示在结果存入目标图（第二个参数 dst）之前可选的 delta 值，有默认值 0。
- 第七个参数， int 类型的 borderType，边界模式，默认值为 BORDER_DEFAULT。这个参数可以在官方文档中 borderInterpolate()处得到更详细的信息。





3. 图像特征：灰度直方图、颜色直方图（RGB 空间、HSV 空间）、Histogram Of Gradient

   主要用到的函数：



§ calcHist() [1/3]

```
void cv::calcHist ( const Mat *   images,
                    int           nimages,
                    const int *   channels,
                    InputArray    mask,
                    OutputArray   hist,
                    int           dims,
                    const int *   histSize,
                    const float **ranges,
                    bool          uniform = true,
                    bool          accumulate = false
                  )
```

Python:

```
hist = cv.calcHist( images, channels, mask, histSize, ranges[, hist[, accumulate]] )
```

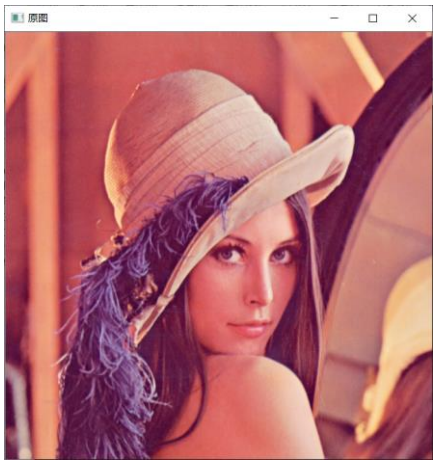
Calculates a histogram of a set of arrays.

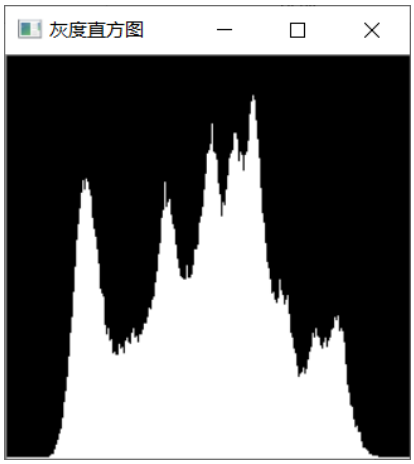
参数解释：

- 第一个参数 images，输入的图像或数组，它们的深度必须为 CV_8U, CV_16U 或 CV_32F 中的一类，尺寸必须相同。
- 第二个参数，nimages 输入数组个数，也就是第一个参数中存放了几张图像，有几个原数组。
- 第三个参数，channels 需要统计的通道 dim，第一个数组通道从 0 到 image[0].channels()-1，第二个数组从 image[0].channels()到 images[0].channels()+images[1].channels()-1，以后的数组以此类推
- 第四个参数，mask 可选的操作掩码。如果此掩码不为空，那么它必须为 8 位并且尺寸要和输入图像 images[i]一致。非零掩码用于标记出统计直方图的数组元素数据。
- 第五个参数，hist 输出的目标直方图，一个二维数组

- 第六个参数， dims 需要计算直方图的维度，必须是正数且并不大于 CV_MAX_DIMS(在 opencv 中等于 32)
- 第七个参数，histSize 每个维度的直方图尺寸的数组
- 第八个参数，ranges 每个维度中 bin 的取值范围
- 第九个参数，uniform 直方图是否均匀的标识符，有默认值 true。
- 第十个参数：accumulate 累积标识符，有默认值 false,若为 true，直方图再分配阶段不会清零。此功能主要是允许从多个阵列中计算单个直方图或者用于再特定的时间更新直方图.

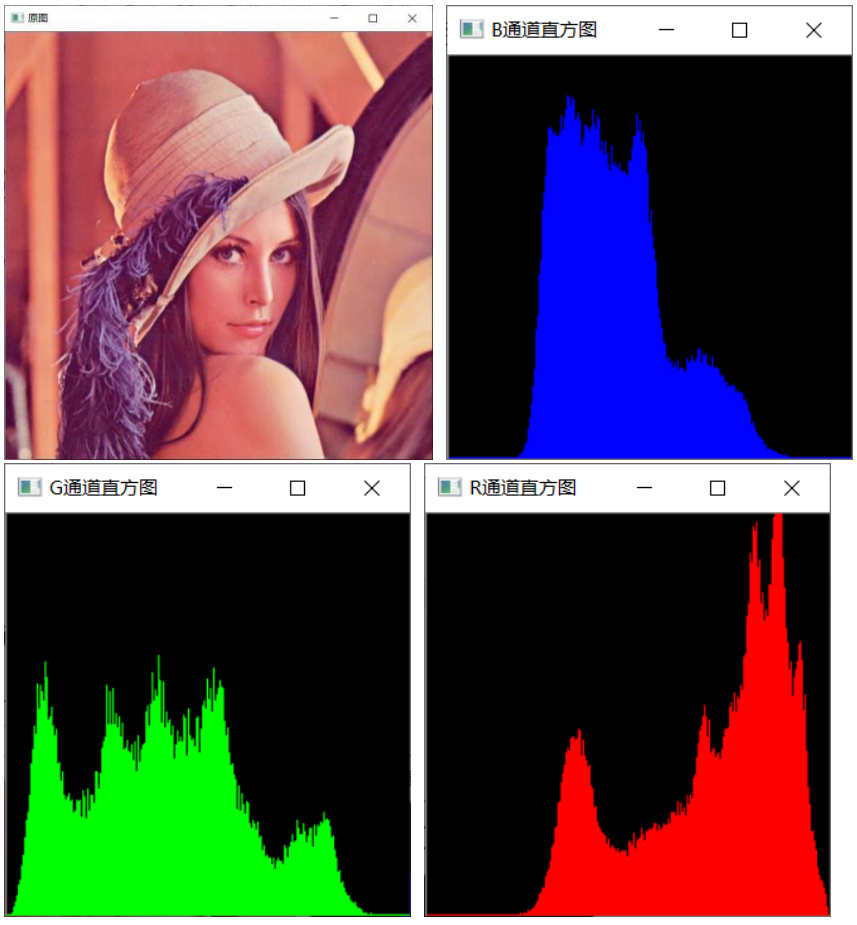
1) 灰度直方图

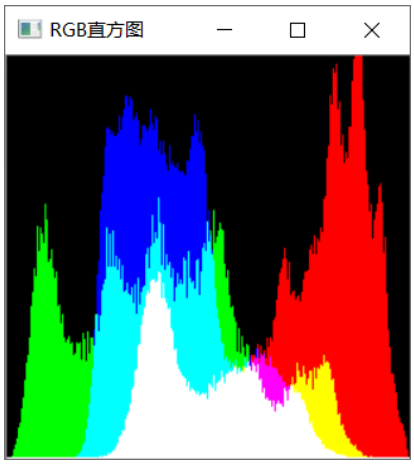




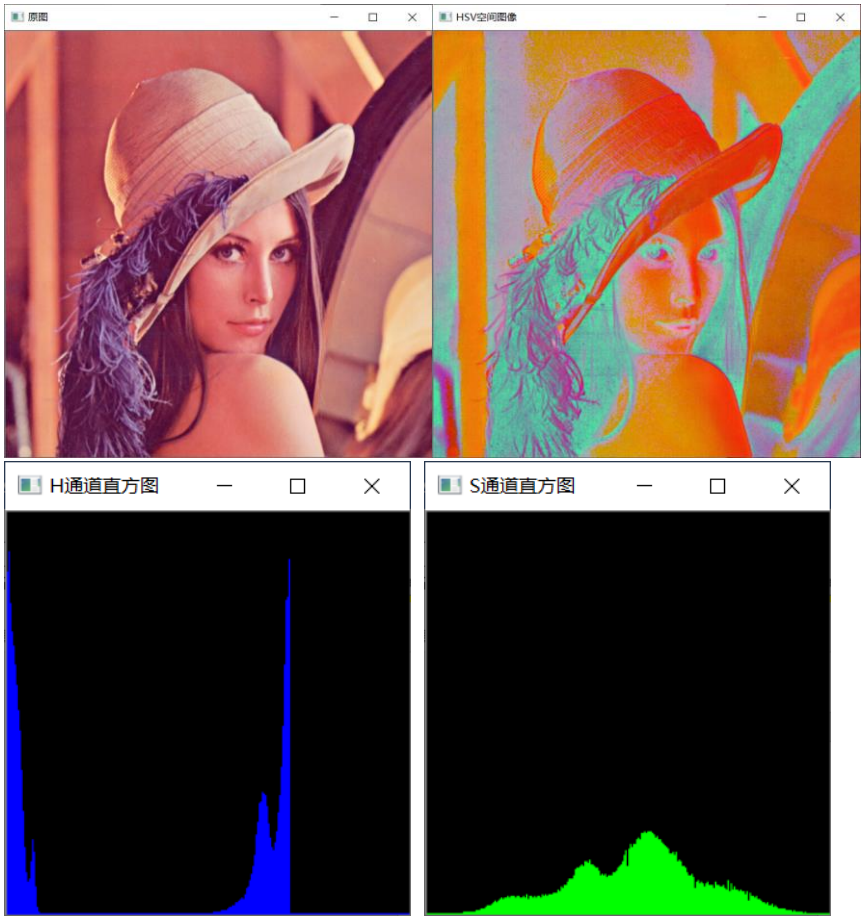
2）颜色直方图（RGB 空间、HSV 空间）

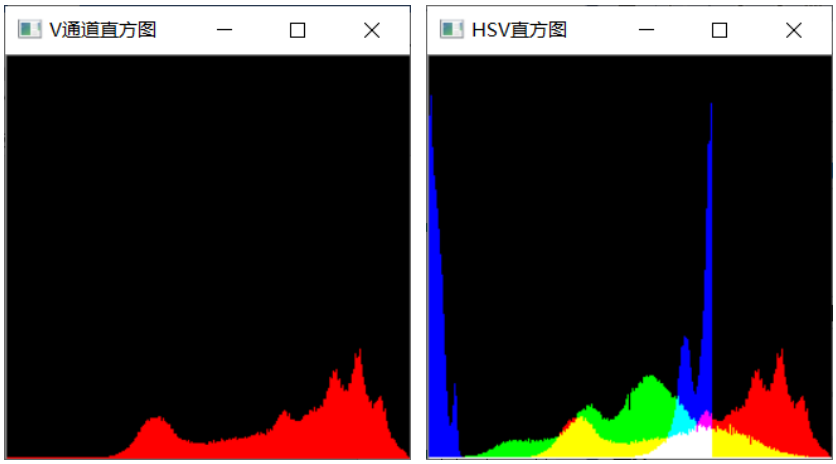
RBG 空间：





HSV 空间：





3) Histogram Of Gradient