实验二：图像处理基本操作

一、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）。

二、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。

1. Python OpenCV库的安装与使用

```
# 使用pip安装
pip install opencv-python
```

```
# 代码中引用opencv库
import cv2
```

2. MATLAB自带图像处理库，无需额外安装OpenCV

三、实验过程

1. **图像读取、写入、平移、旋转、缩放等操作**

   参考文档：

   Python：https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_tutorials.html

   MATLAB：https://ww2.mathworks.cn/help/images/index.html

   （1）图像读取

Python: cv2.**imread**(filename[, flags]) → retval

C: IplImage* **cvLoadImage**(const char* **filename**, int **iscolor**=CV_LOAD_IMAGE_COLOR)

C: CvMat* **cvLoadImageM**(const char* **filename**, int **iscolor**=CV_LOAD_IMAGE_COLOR)

Parameters:
- **filename** – Name of file to be loaded.
- **flags** –

  Flags specifying the color type of a loaded image:

  - CV_LOAD_IMAGE_ANYDEPTH - If set, return 16-bit/32-bit image when the input has the corresponding depth, otherwise convert it to 8-bit.
  - CV_LOAD_IMAGE_COLOR - If set, always convert image to the color one
  - CV_LOAD_IMAGE_GRAYSCALE - If set, always convert image to the grayscale one
  - **>0** Return a 3-channel color image.

    Note: In the current implementation the alpha channel, if any, is stripped from the output image. Use negative value if you need the alpha channel.

  - **=0** Return a grayscale image.
  - **<0** Return the loaded image as is (with alpha channel).

The function imread loads an image from the specified file and returns it. If the image cannot be read (because of missing file, improper permissions, unsupported or invalid format), the function returns an empty matrix (Mat::data==NULL). Currently, the following file formats are supported:

MATLAB：https://ww2.mathworks.cn/help/matlab/ref/imread.html

(2) 图像写入

Python: cv2.**imwrite**(filename, img[, params]) → retval

C: int **cvSaveImage**(const char* **filename**, const CvArr* **image**, const int* **params**=0)

Parameters:
- **filename** – Name of the file.
- **image** – Image to be saved.
- **params** –

  Format-specific save parameters encoded as pairs paramId_1, paramValue_1, paramId_2, paramValue_2, ... The following parameters are currently supported:

  - For JPEG, it can be a quality (CV_IMWRITE_JPEG_QUALITY) from 0 to 100 (the higher is the better). Default value is 95.
  - For WEBP, it can be a quality (CV_IMWRITE_WEBP_QUALITY) from 1 to 100 (the higher is the better). By default (without any parameter) and for quality above 100 the lossless compression is used.
  - For PNG, it can be the compression level (CV_IMWRITE_PNG_COMPRESSION) from 0 to 9. A higher value means a smaller size and longer compression time. Default value is 3.
  - For PPM, PGM, or PBM, it can be a binary format flag (CV_IMWRITE_PXM_BINARY), 0 or 1. Default value is 1.

The function imwrite saves the image to the specified file. The image format is chosen based on the filename extension (see imread() for the list of extensions). Only 8-bit (or 16-bit unsigned (CV_16U) in case of PNG, JPEG 2000, and TIFF) single-channel or 3-channel (with 'BGR' channel order) images can be saved using this function. If the format, depth or channel order is different, use Mat::convertTo() , and cvtColor() to convert it before saving. Or, use the universal FileStorage I/O functions to save the image to XML or YAML format.

MATLAB：https://ww2.mathworks.cn/help/matlab/ref/imwrite.html?s_tid=doc_ta

(3) 图像旋转、平移、缩放

Python: cv2.**getRotationMatrix2D**(center, angle, scale) → retval

C: CvMat* **cv2DRotationMatrix**(CvPoint2D32f **center**, double **angle**, double **scale**, CvMat* **map_matrix**)

Parameters:
- **center** – Center of the rotation in the source image.
- **angle** – Rotation angle in degrees. Positive values mean counter-clockwise rotation (the coordinate origin is assumed to be the top-left corner).
- **scale** – Isotropic scale factor.
- **map_matrix** – The output affine transformation, 2x3 floating-point matrix.

The function calculates the following matrix:

$$\begin{bmatrix} \alpha & \beta & (1-\alpha)\cdot \mathtt{center.x} - \beta \cdot \mathtt{center.y} \\ -\beta & \alpha & \beta \cdot \mathtt{center.x} + (1-\alpha)\cdot \mathtt{center.y} \end{bmatrix}$$

where

$$\begin{aligned} \alpha &= \mathtt{scale} \cdot \cos \mathtt{angle}, \\ \beta &= \mathtt{scale} \cdot \sin \mathtt{angle} \end{aligned}$$

The transformation maps the rotation center to itself. If this is not the target, adjust the shift.

Python: cv2.**warpAffine**(src, M, dsize[, dst[, flags[, borderMode[, borderValue]]]]) → dst

C: void **cvWarpAffine**(const CvArr* **src**, CvArr* **dst**, const CvMat* **map_matrix**, int **flags**=CV_INTER_LINEAR+CV_WARP_FILL_OUTLIERS, CvScalar **fillval**=cvScalarAll(0))

C: void **cvGetQuadrangleSubPix**(const CvArr* **src**, CvArr* **dst**, const CvMat* **map_matrix**)

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 **resize()**) and the optional flag `WARP_INVERSE_MAP` that means that `M` is the inverse transformation ($\mathtt{dst} \rightarrow \mathtt{src}$).
- **borderMode** – pixel extrapolation method (see **borderInterpolate()**); 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.

The function `warpAffine` transforms the source image using the specified matrix:

$$\mathtt{dst}(x,y) = \mathtt{src}(M_{11}x + M_{12}y + M_{13}, M_{21}x + M_{22}y + M_{23})$$

when the flag `WARP_INVERSE_MAP` is set. Otherwise, the transformation is first inverted with **invertAffineTransform()** and then put in the formula above instead of `M` . The function cannot operate in-place.

Python: cv2.resize(src, dsize[, dst[, fx[, fy[, interpolation]]]]) → dst

C: void cvResize(const CvArr* **src**, CvArr* **dst**, int **interpolation**=CV_INTER_LINEAR)

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:

  - **INTER_NEAREST** - a nearest-neighbor interpolation
  - **INTER_LINEAR** - a bilinear interpolation (used by default)
  - **INTER_AREA** - resampling using pixel area relation. It may be a preferred method for image decimation, as it gives moire'-free results. But when the image is zoomed, it is similar to the INTER_NEAREST method.
  - **INTER_CUBIC** - a bicubic interpolation over 4x4 pixel neighborhood
  - **INTER_LANCZOS4** - a Lanczos interpolation over 8x8 pixel neighborhood

The function resize resizes the image src down to or up to the specified size. Note that the initial dst type or size are not taken into account. Instead, the size and type are derived from the src,``dsize``,``fx`` , and fy . If you want to resize src so that it fits the pre-created dst , you may call the function as follows:

```
// explicitly specify dsize=dst.size(); fx and fy will be computed from that.
resize(src, dst, dst.size(), 0, 0, interpolation);
```

If you want to decimate the image by factor of 2 in each direction, you can call the function this way:

```
// specify fx and fy and let the function compute the destination image size.
resize(src, dst, Size(), 0.5, 0.5, interpolation);
```

To shrink an image, it will generally look best with CV_INTER_AREA interpolation, whereas to enlarge an image, it will generally look best with CV_INTER_CUBIC (slow) or CV_INTER_LINEAR (faster but still looks OK).

MATLAB：

旋转：https://ww2.mathworks.cn/help/images/ref/imrotate.html?searchHighlight=imrotate&s_tid=srchtitle

平移：https://ww2.mathworks.cn/help/images/translate-an-image.html?searchHighlight=translate&s_tid=srchtitle

缩放：https://ww2.mathworks.cn/help/images/ref/imresize.html?searchHighlight=imresize&s_tid=srchtitle

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）**

（1）平滑

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，其输出像素值由输入像素的加权和决定：

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

其中N(i, j)代表像素(i, j)的邻域坐标，h(k, l)代表滤波器的系数。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：

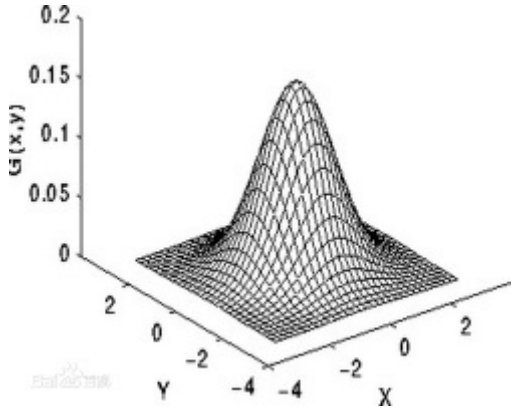
- 均值滤波器

  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。

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

- 高斯滤波器

  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。



- 中值滤波器

  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。

- 双边滤波器

  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。

Python: `cv2.blur(src, ksize[, dst[, anchor[, borderType]]]) → dst` ¶

Parameters:
- **src** – input image; it can have any number of channels, which are processed independently, but the depth should be `CV_8U`, `CV_16U`, `CV_16S`, `CV_32F` or `CV_64F`.
- **dst** – output image of the same size and type as `src`.
- **ksize** – blurring kernel size.
- **anchor** – anchor point; default value `Point(-1,-1)` means that the anchor is at the kernel center.
- **borderType** – border mode used to extrapolate pixels outside of the image.

The function smoothes an image using the kernel:

$$\mathtt{K} = \frac{1}{\mathtt{ksize.width*ksize.height}} \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 & 1 \\ 1 & 1 & 1 & \cdots & 1 & 1 \\ & & \cdots\cdots\cdots & & & \\ 1 & 1 & 1 & \cdots & 1 & 1 \end{bmatrix}$$

The call `blur(src, dst, ksize, anchor, borderType)` is equivalent to `boxFilter(src, dst, src.type(), anchor, true, borderType)`.

Python: `cv2.GaussianBlur(src, ksize, sigmaX[, dst[, sigmaY[, borderType]]]) → dst`

Parameters:
- **src** – input image; the image can have any number of channels, which are processed independently, but the depth should be `CV_8U`, `CV_16U`, `CV_16S`, `CV_32F` or `CV_64F`.
- **dst** – output image of the same size and type as `src`.
- **ksize** – Gaussian kernel size. `ksize.width` and `ksize.height` can differ but they both must be positive and odd. Or, they can be zero's and then they are computed from `sigma*` .
- **sigmaX** – Gaussian kernel standard deviation in X direction.
- **sigmaY** – Gaussian kernel standard deviation in Y direction; if `sigmaY` is zero, it is set to be equal to `sigmaX`, if both sigmas are zeros, they are computed from `ksize.width` and `ksize.height` , respectively (see `getGaussianKernel()` for details); to fully control the result regardless of possible future modifications of all this semantics, it is recommended to specify all of `ksize`, `sigmaX`, and `sigmaY`.
- **borderType** – pixel extrapolation method (see `borderInterpolate` for details).

The function convolves the source image with the specified Gaussian kernel. In-place filtering is supported.

Python: `cv2.medianBlur(src, ksize[, dst]) → dst`

Parameters:
- **src** – input 1-, 3-, or 4-channel image; when `ksize` is 3 or 5, the image depth should be `CV_8U`, `CV_16U`, or `CV_32F`, for larger aperture sizes, it can only be `CV_8U`.
- **dst** – destination array of the same size and type as `src`.
- **ksize** – aperture linear size; it must be odd and greater than 1, for example: 3, 5, 7 ...

The function smoothes an image using the median filter with the **ksize × ksize** aperture. Each channel of a multi-channel image is processed independently. In-place operation is supported.

Python: `cv2.bilateralFilter(src, d, sigmaColor, sigmaSpace[, dst[, borderType]]) → dst`

Parameters:
- **src** – Source 8-bit or floating-point, 1-channel or 3-channel image.
- **dst** – Destination image of the same size and type as `src` .
- **d** – Diameter of each pixel neighborhood that is used during filtering. If it is non-positive, it is computed from `sigmaSpace` .
- **sigmaColor** – Filter sigma in the color space. A larger value of the parameter means that farther colors within the pixel neighborhood (see `sigmaSpace`) will be mixed together, resulting in larger areas of semi-equal color.
- **sigmaSpace** – Filter sigma in the coordinate space. A larger value of the parameter means that farther pixels will influence each other as long as their colors are close enough (see `sigmaColor`). When `d>0` , it specifies the neighborhood size regardless of `sigmaSpace` . Otherwise, `d` is proportional to `sigmaSpace` .

The function applies bilateral filtering to the input image, as described in http://www.dai.ed.ac.uk/CVonline/LOCAL_COPIES/MANDUCHI1/Bilateral_Filtering.html `bilateralFilter` can reduce unwanted noise very well while keeping edges fairly sharp. However, it is very slow compared to most filters.

Sigma values: For simplicity, you can set the 2 sigma values to be the same. If they are small (< 10), the filter will not have much effect, whereas if they are large (> 150), they will have a very strong effect, making the image look "cartoonish".

Filter size: Large filters (d > 5) are very slow, so it is recommended to use d=5 for real-time applications, and perhaps d=9 for offline applications that need heavy noise filtering.

This filter does not work inplace.

MATLAB：

https://ww2.mathworks.cn/help/images/ref/fspecial.html?searchHighlight=fspecial&s_tid=srchtitle

https://ww2.mathworks.cn/help/matlab/ref/filter2.html?searchHighlight=filter2&s_tid=srchtitle

https://ww2.mathworks.cn/help/images/ref/medfilt2.html

https://blog.csdn.net/Chaolei3/article/details/88579377

（2）边缘提取

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。

- Sobel边缘提取

  该算子包含两组3x3的矩阵，分别为横向及纵向，将之与图像作平面卷积，即可分别得出横向及纵向的亮度差分近似值。如果以A代表原始图像，Gx及Gy分别代表经横向及纵向边缘检测的图像，其公式如下：

$$G_x = S_x * I,\ G_y = S_y * I,\ G = \sqrt{G_x^2 + G_y^2}$$

Python: cv2.Sobel(src, ddepth, dx, dy[, dst[, ksize[, scale[, delta[, borderType]]]]]) → dst

C: void cvSobel(const CvArr* **src**, CvArr* **dst**, int **xorder**, int **yorder**, int **aperture_size**=3)

Parameters:
- **src** – input image.
- **dst** – output image of the same size and the same number of channels as src .
- **ddepth** –

  output image depth; the following combinations of src.depth() and ddepth are supported:

  - src.depth() = CV_8U, ddepth = -1/CV_16S/CV_32F/CV_64F
  - src.depth() = CV_16U/CV_16S, ddepth = -1/CV_32F/CV_64F
  - src.depth() = CV_32F, ddepth = -1/CV_32F/CV_64F
  - src.depth() = CV_64F, ddepth = -1/CV_64F

  when ddepth=-1, the destination image will have the same depth as the source; in the case of 8-bit input images it will result in truncated derivatives.
- **xorder** – order of the derivative x.
- **yorder** – order of the derivative y.
- **ksize** – size of the extended Sobel kernel; it must be 1, 3, 5, or 7.
- **scale** – optional scale factor for the computed derivative values; by default, no scaling is applied (see getDerivKernels() for details).
- **delta** – optional delta value that is added to the results prior to storing them in dst.
- **borderType** – pixel extrapolation method (see borderInterpolate for details).

In all cases except one, the $\texttt{ksize} \times \texttt{ksize}$ separable kernel is used to calculate the derivative. When $\texttt{ksize} = 1$, the 3×1 or 1×3 kernel is used (that is, no Gaussian smoothing is done). ksize = 1 can only be used for the first or the second x- or y- derivatives.

There is also the special value ksize = CV_SCHARR (-1) that corresponds to the 3×3 Scharr filter that may give more accurate results than the 3×3 Sobel. The Scharr aperture is

$$\begin{bmatrix} -3 & 0 & 3 \\ -10 & 0 & 10 \\ -3 & 0 & 3 \end{bmatrix}$$

for the x-derivative, or transposed for the y-derivative.

The function calculates an image derivative by convolving the image with the appropriate kernel:

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

The Sobel operators combine Gaussian smoothing and differentiation, so the result is more or less resistant to the noise. Most often, the function is called with (xorder = 1, yorder = 0, ksize = 3) or (xorder = 0, yorder = 1, ksize = 3) to calculate the first x- or y- image derivative. The first case corresponds to a kernel of:

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

The second case corresponds to a kernel of:

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

- Canny边缘提取

  Canny算子与Marr（LoG）边缘检测方法类似，也属于是先平滑后求导数的方法。算法步骤为：

  a. 灰度化；

  b. 高斯滤波去噪；

  c. 计算梯度值和方向；

  d. 非极大值抑制：沿着梯度方向，比较像素点前面和后面的梯度值。在沿其方向上邻域的梯度幅值最大，则保留；否则，抑制；

  f. 用双阈值算法检测和连接边缘。

Python: cv2.**Canny**(image, threshold1, threshold2[, edges[, apertureSize[, L2gradient]]]) → edges

C: void **cvCanny**(const CvArr* **image**, CvArr* **edges**, double **threshold1**, double **threshold2**, int **aperture_size**=3)

Parameters:
- **image** – 8-bit input image.
- **edges** – output edge map; single channels 8-bit image, which has the same size as `image` .
- **threshold1** – first threshold for the hysteresis procedure.
- **threshold2** – second threshold for the hysteresis procedure.
- **apertureSize** – aperture size for the **Sobel()** operator.
- **L2gradient** – a flag, indicating whether a more accurate L_2 norm $= \sqrt{(dI/dx)^2 + (dI/dy)^2}$ should be used to calculate the image gradient magnitude (`L2gradient=true`), or whether the default L_1 norm $= |dI/dx| + |dI/dy|$ is enough (`L2gradient=false`).

The function finds edges in the input image `image` and marks them in the output map `edges` using the Canny algorithm. The smallest value between `threshold1` and `threshold2` is used for edge linking. The largest value is used to find initial segments of strong edges. See http://en.wikipedia.org/wiki/Canny_edge_detector

- 高斯差分（DoG）

  高斯函数定义为

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

  给定一张图像，经过高斯滤波后得到

$$O(x, y) = G(x, y, \sigma) * I(x, y)$$

  对于同一张图像，采用两组不同参数下的高斯函数进行滤波，得到结果相减，即为高斯差分（DoG）图，公式表达为

$$DoG = G(\sigma_1) * I - G(\sigma_2) * I$$

  高斯滤波函数参考上一小节。

- 高斯拉普拉斯（LoG）

  拉普拉斯算子是图像二阶空间导数的二维各向同性测度。拉普拉斯算子可以突出图像中强度发生快速变化的区域，因此常用在边缘检测任务当中。在进行Laplacian操作之前通常需要先用高斯平滑滤波器对图像进行平滑处理，以降低Laplacian操作对于噪声的敏感性。

Python: cv2.**Laplacian**(src, ddepth[, dst[, ksize[, scale[, delta[, borderType]]]]]) → dst

C: void **cvLaplace**(const CvArr* **src**, CvArr* **dst**, int **aperture_size**=3)

Parameters:
- **src** – Source image.
- **dst** – Destination image of the same size and the same number of channels as `src` .
- **ddepth** – Desired depth of the destination image.
- **ksize** – Aperture size used to compute the second-derivative filters. See **getDerivKernels()** for details. The size must be positive and odd.
- **scale** – Optional scale factor for the computed Laplacian values. By default, no scaling is applied. See **getDerivKernels()** for details.
- **delta** – Optional delta value that is added to the results prior to storing them in `dst` .
- **borderType** – Pixel extrapolation method. See `borderInterpolate` for details.

The function calculates the Laplacian of the source image by adding up the second x and y derivatives calculated using the Sobel operator:

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

This is done when `ksize > 1` . When `ksize == 1` , the Laplacian is computed by filtering the image with the following 3×3 aperture:

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

MATLAB：https://ww2.mathworks.cn/help/images/ref/edge.html?searchHighlight=canny&s_tid=srchtitle

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)**

（1）直方图

对数据的统计集合，并将统计结果分布于一系列预定义的 bins 中。这里的数据不仅仅指的是灰度值，且统计数据可能是任何有效描述图像的特征。**灰度图和RGB图的直方图的bin数量一致设为256。HSV图对应的bin数量设为100。**

（2）RGB到HSV的转换

HSV是根据颜色的直观特性由A. R. Smith在1978年创建的一种颜色空间, 也称六角锥体模型(Hexcone Model)。这个模型中颜色的参数分别是：色调（H），饱和度（S），明度（V）。

- RGB $\leftrightarrow$ HSV (`COLOR_BGR2HSV`, `COLOR_RGB2HSV`, `COLOR_HSV2BGR`, `COLOR_HSV2RGB`)

  In case of 8-bit and 16-bit images, R, G, and B are converted to the floating-point format and scaled to fit the 0 to 1 range.

$$V \leftarrow \max(R, G, B)$$

$$S \leftarrow \begin{cases} \frac{V-\min(R,G,B)}{V} & \text{if } V \neq 0 \\ 0 & \text{otherwise} \end{cases}$$

$$H \leftarrow \begin{cases} 60(G-B)/(V-\min(R,G,B)) & \text{if } V = R \\ 120+60(B-R)/(V-\min(R,G,B)) & \text{if } V = G \\ 240+60(R-G)/(V-\min(R,G,B)) & \text{if } V = B \end{cases}$$

  If $H < 0$ then $H \leftarrow H + 360$. On output $0 \leq V \leq 1, 0 \leq S \leq 1, 0 \leq H \leq 360$.

  The values are then converted to the destination data type:

  - 8-bit images

$$V \leftarrow 255V, S \leftarrow 255S, H \leftarrow H/2 (\text{to fit to } 0 \text{ to } 255)$$

  - 16-bit images (currently not supported)

$$V < -65535V, S < -65535S, H < -H$$

  - 32-bit images

    H, S, and V are left as is

（3）方向梯度直方图（HOG）

在HOG特征描述符中，梯度方向的分布，也就是梯度方向的直方图被视作特征。
第一步，分别计算水平方向的梯度和垂直方向的梯度。
第二步，计算每个像素点处的梯度方向，注意角度的范围介于0到180度之间，而不是0到360度，这被称为“无符号”梯度，因为两个完全相反的方向被认为是相同的。
第三步，统计每个方向范围内（分为9个bin，每个bin范围为20°）的梯度数量，构造直方图。

Python: `cv2.calcHist(images, channels, mask, histSize, ranges[, hist[, accumulate]]) → hist`

C: void `cvCalcHist(IplImage** image, CvHistogram* hist, int accumulate=0, const CvArr* mask=NULL )`

Parameters:
- **images** – Source arrays. They all should have the same depth, `CV_8U` or `CV_32F` , and the same size. Each of them can have an arbitrary number of channels.
- **nimages** – Number of source images.
- **channels** – List of the `dims` channels used to compute the histogram. The first array channels are numerated from 0 to `images[0].channels()-1` , the second array channels are counted from `images[0].channels()` to `images[0].channels() + images[1].channels()-1`, and so on.
- **mask** – Optional mask. If the matrix is not empty, it must be an 8-bit array of the same size as `images[i]` . The non-zero mask elements mark the array elements counted in the histogram.
- **hist** – Output histogram, which is a dense or sparse `dims` -dimensional array.
- **dims** – Histogram dimensionality that must be positive and not greater than `CV_MAX_DIMS` (equal to 32 in the current OpenCV version).
- **histSize** – Array of histogram sizes in each dimension.
- **ranges** – Array of the `dims` arrays of the histogram bin boundaries in each dimension. When the histogram is uniform (`uniform` =true), then for each dimension `i` it is enough to specify the lower (inclusive) boundary L_0 of the 0-th histogram bin and the upper (exclusive) boundary $U_{\texttt{histSize[i]}-1}$ for the last histogram bin `histSize[i]-1` . That is, in case of a uniform histogram each of `ranges[i]` is an array of 2 elements. When the histogram is not uniform (`uniform=false`), then each of `ranges[i]` contains `histSize[i]+1` elements: $L_0, U_0 = L_1, U_1 = L_2, ..., U_{\texttt{histSize[i]}-2} = L_{\texttt{histSize[i]}-1}, U_{\texttt{histSize[i]}-1}$. The array elements, that are not between L_0 and $U_{\texttt{histSize[i]}-1}$, are not counted in the histogram.
- **uniform** – Flag indicating whether the histogram is uniform or not (see above).
- **accumulate** – Accumulation flag. If it is set, the histogram is not cleared in the beginning when it is allocated. This feature enables you to compute a single histogram from several sets of arrays, or to update the histogram in time.

The functions `calcHist` calculate the histogram of one or more arrays. The elements of a tuple used to increment a histogram bin are taken from the corresponding input arrays at the same location. The sample below shows how to compute a 2D Hue-Saturation histogram for a color image.

Python: `cv2.cvtColor(src, code[, dst[, dstCn]]) → dst`

C: void `cvCvtColor(const CvArr* src, CvArr* dst, int code)`

Parameters:
- **src** – input image: 8-bit unsigned, 16-bit unsigned (`CV_16UC...`), or single-precision floating-point.
- **dst** – output image of the same size and depth as `src`.
- **code** – color space conversion code (see the description below).
- **dstCn** – number of channels in the destination image; if the parameter is 0, the number of the channels is derived automatically from `src` and `code` .

The function converts an input image from one color space to another. In case of a transformation to-from RGB color space, the order of the channels should be specified explicitly (RGB or BGR). Note that the default color format in OpenCV is often referred to as RGB but it is actually BGR (the bytes are reversed). So the first byte in a standard (24-bit) color image will be an 8-bit Blue component, the second byte will be Green, and the third byte will be Red. The fourth, fifth, and sixth bytes would then be the second pixel (Blue, then Green, then Red), and so on.

The conventional ranges for R, G, and B channel values are:

- 0 to 255 for `CV_8U` images
- 0 to 65535 for `CV_16U` images
- 0 to 1 for `CV_32F` images

MATLAB：https://ww2.mathworks.cn/help/images/ref/imhist.html?searchHighlight=imhist&s_tid=srchtitle

四、报告要求

1. 本次实验不限编程语言，可以使用 Python， MATLAB 等语言。
2. 实验可以使用 OpenCV 函数（也可以自己实现函数）。
3. 实验报告：

   实验内容主要分为三部分：

   - 图像读取、写入、平移、旋转、缩放等操作
   - 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）
   - 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）

   具体要求如下：

- 图像读取、写入、平移、旋转、缩放等操作： 这一部分要在报告中展示实验结果即可（输入图像，输出图像）
- 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）： 这一部分要在报告中展示实验结果（输入图像，输出图像），（可选）分析每一个滤波器的使用场景，并解释原因、边缘提取要分析哪种边缘提取适用什么场景，并解释原因。
- 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））： 这一部分要在报告中展示实验结果（输入图像，输出图像），分析这几种直方图的使用场景，并解释原因。灰度直方图和RGB直方图设置bin数量为256，HSV直方图设置bin数量为100，方向梯度直方图设置bin数量为9（每个范围为20°）。
- 实验代码
- 所有实验结果放在对应的代码目录下

4. 请将以下文件打包发送至algorithm_2022@126.com

   文件名如： 张三_PB18000000_第二次实验.zip
   邮件主题：张三，PB18000000，第二次实验
   截至日期：下周三之前