图像入门

- **读取图像**：既然想要cv2处理图像那么第一件事就是读取图像，说白了就是创建图片实例
  - 使用imread方法
    - 第一个参数指定图片文件路径
    - 第二个参数是可选的，指定我们想要的图像格式。这可能是：
      - IMREAD_COLOR 以 BGR 8 位格式加载图像。这是此处使用的**默认值。**
      - IMREAD_UNCHANGED 按原样加载图像（包括 alpha 通道，如果存在）
      - IMREAD_GRAYSCALE 将图像加载为强度图像
  - 读取后图像数据将存储在cv::Mat对象中。
- 使用cv2的窗口来**显示图像**：
  - imshow方法
    - 第一个参数是窗口的标题，第二个参数是将显示的cv::Mat对象，也就是上边读取的图片实例
- **保存图片**：
  -
- 窗口什么时候关？可以用waitKey方法
  - 该函数的唯一参数就是等待用户输入的时间（以毫秒为单位）。零表示永远等待。返回值是按下的键。
- 用于关闭所有窗口：
  - cv2.destroyAllWindows()
- 图像写入文件
  - 为此，将调用cv::imwrite函数，该函数以文件路径和cv::Mat对象作为参数。
- 图像大小
  - 对图像实例用shape方法，它返回行数、列数和通道数（如果图像是彩色的）的元组

重点在下边！

图像处理

- 改变图像颜色色彩空间
  - 用cvtColor(input_image, flag)，其中 flag 决定转换的类型。
    - BGR→灰度转换，使用的flag参数为标志cv.COLOR_BGR2GRAY
    - 对于 BGR 类似→HSV，我们使用标志cv.COLOR_BGR2HSV。
      -

1. **色调（Hue）**：表示颜色的种类，通常用角度来表示，范围从0°到360°，代表颜色在色轮上的不同位置。例如，红色大约在0°，绿色大约在120°，蓝色大约在240°。
2. **饱和度（Saturation）**：表示颜色的纯度，即颜色中灰色成分的多少。饱和度越高，颜色越鲜艳；饱和度越低，颜色越接近灰色。在HSV模型中，饱和度的范围通常是从0%到100%。
3. **亮度（Value）或明度（Brightness）**：表示颜色的明暗程度。亮度越高，颜色越亮；亮度越低，颜色越暗。在HSV模型中，亮度或明度的范围通常是从0%到100%。

- HSV色彩空间的优点在于，它允许用户独立地调整颜色的这三个属性，这在进行颜色选择和图像编辑时非常有用。例如，你可以改变一个图像的色调而不改变其亮度和饱和度，或者你可以增加图像的饱和度来使颜色更加鲜艳。
- 在图像处理软件和编程库中，HSV色彩空间常用于颜色检测、颜色过滤和颜色转换等任务。例如，OpenCV是一个流行的计算机视觉库，它提供了将图像从RGB色彩空间转换到HSV色彩空间的功能，这使得基于颜色的图像分析变得更加容易。%%

- inRange方法：
  - 在OpenCV中，`cv2.inRange`函数用于根据指定的上下界（范围）来创建一个二值化图像，即掩膜。这个函数会检查输入图像中每个像素的值是否在指定的范围内，如果在范围内，对应的掩膜像素值会被设置为最大值（通常是255），否则设置为0。这个函数在颜色空间转换和颜色过滤中非常有用。
  - 函数原型

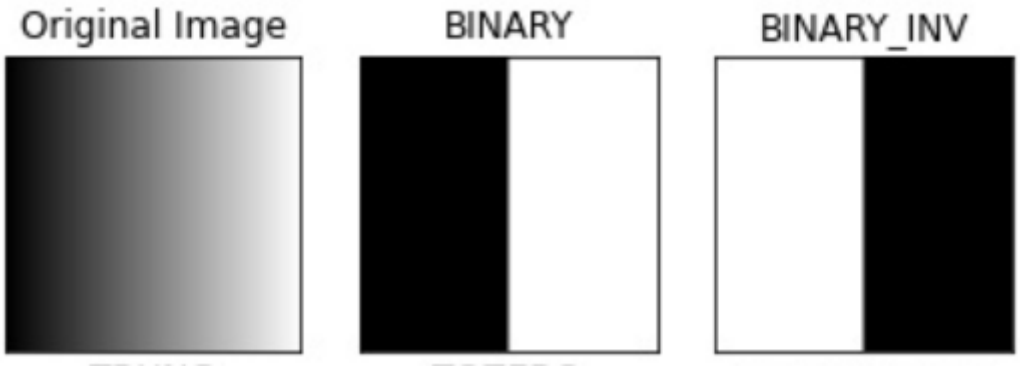
```
dst = cv2.inRange(src, lower, upper)
```

  - 参数
    - **src**：输入图像，可以是8位或32位浮点数的单通道或多通道图像。
    - **lower**：指定颜色范围的下限，是一个与输入图像通道数相同的数组。
    - **upper**：指定颜色范围的上限，也是一个与输入图像通道数相同的数组。
    - **dst**：输出图像，与输入图像大小和类型相同，但只有单通道。
  - 例子

```
import cv2 as cv
import numpy as np

# 打开摄像头
cap = cv.VideoCapture(0)

while True:
    # 读取每一帧
    ret, frame = cap.read()
```

```
    # 将BGR转换为HSV
    hsv = cv.cvtColor(frame, cv.COLOR_BGR2HSV)

    # 定义蓝色的HSV范围
    lower_blue = np.array([110, 50, 50])
    upper_blue = np.array([130, 255, 255])

    # 阈值处理HSV图像以获取蓝色区域
    mask = cv.inRange(hsv, lower_blue, upper_blue)

    # 应用掩膜和原始图像的位与操作
    res = cv.bitwise_and(frame, frame, mask=mask)

    # 显示结果
    cv.imshow('frame', frame)
    cv.imshow('mask', mask)
    cv.imshow('res', res)

    # 等待键盘输入
    k = cv.waitKey(5) & 0xFF
    if k == 27:  # 按Esc键退出
        break

# 关闭所有窗口
cv.destroyAllWindows()
```

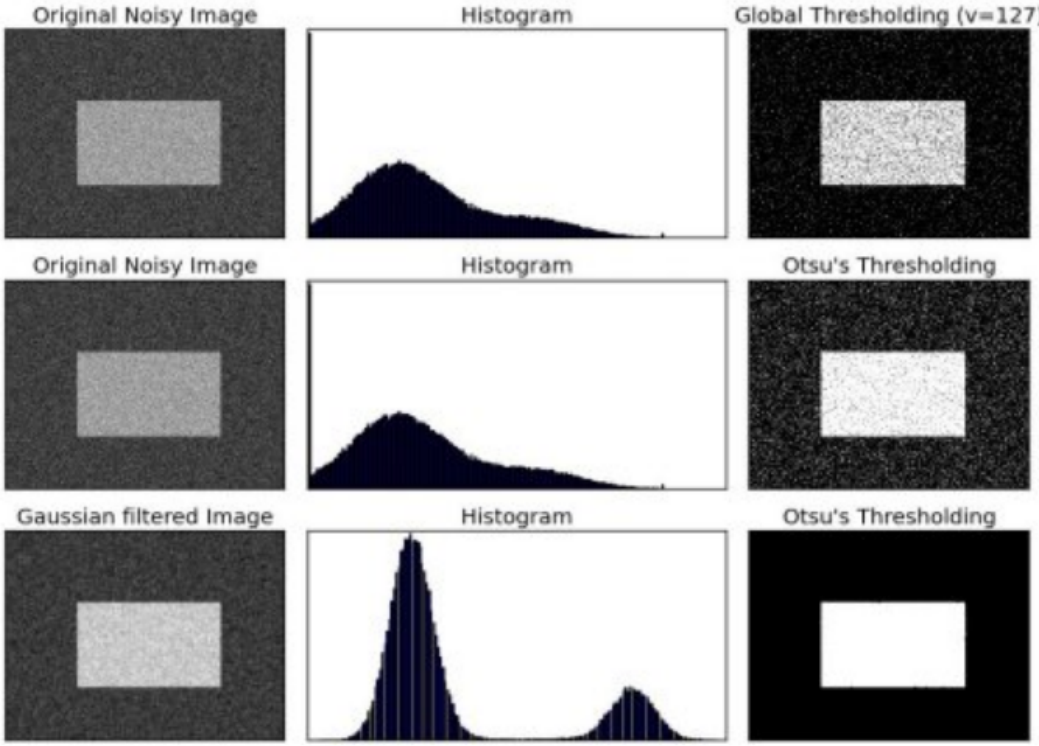
- 掩膜
  - 在图像处理中，掩膜（Mask）是一种常用的技术，它是一种特殊的数组或者图像，用于控制或修改其他图像的某些区域。掩膜通常与原图像具有相同的尺寸，但其内容由二进制值（0和1）或者灰度值组成，用于指定哪些像素应该被处理或者保留，哪些应该被忽略或者修改。
  - 掩膜的主要作用包括：
  1. **保护和隐藏**：在图像的某些区域进行操作时，可以使用掩膜来保护其他区域不被修改。例如，你可能只想对图像的某个特定区域进行模糊或增强处理，而保持其他区域不变。
  2. **提取特征**：掩膜可以用来提取图像中的特定对象或特征。通过设置掩膜，可以隔离感兴趣的区域，然后对这些区域进行进一步的分析或处理。
  3. **透明度控制**：在处理带有透明通道的图像时，掩膜可以用来控制各个像素的透明度。例如，在RGBA色彩空间中，A通道（Alpha通道）就是一个掩膜，它决定了每个像素的透明度。
  4. **图像合成**：在图像合成或叠加时，掩膜可以用来控制不同图像之间的混合方式。例如，可以使用掩膜来实现图像的局部融合或特效。
- 图像阈值处理
  - 方法threshold

    - 第一个参数是源图像，应该**是灰度图像。**第二个参数是阈值，用于对像素值进行分类。第三个参数是最大值，它被分配给超过阈值的像素值。
    - OpenCV 提供了不同类型的阈值，由函数的第四个参数给出。如上所述的基本阈值是使用类型cv.THRESH_BINARY完成的。所有简单阈值类型包括：
        - THRESH_BINARY
        - THRESH_BINARY_INV（INV就是invert反转颜色）
        - THRESH_TRUNC
        - cv.THRESH_TOZERO
        - cv.THRESH_TOZERO_INV
    - 该方法返回两个输出。第一个是使用的阈值，第二个输出是**阈值化图像。**
    - 例子：ret，thresh1 = cv.threshold（img，127,255，cv.THRESH_BINARY）
    - ret，thresh2 = cv.threshold（img，127,255，cv.THRESH_BINARY_INV）
    - 

      第一个就够用了
- 还有个自适应阈值：
    - 方法cv.adaptiveThreshold接受三个输入参数：
        - adaptMethod**决定**如何计算阈值：
        - cv.ADAPTIVE_THRESH_MEAN_C：阈值是邻域区域的平均值减去常数**C。**
        - cv.ADAPTIVE_THRESH_GAUSSIAN_C：阈值是邻域值的高斯加权和减去常数**C。**
        - blockSize决定了邻域区域的大小，**C**是从邻域像素的平均值或加权和中减去的常数。
        - 例子：th2 = cv.adaptiveThreshold (img,255,cv.ADAPTIVE_THRESH_MEAN_C,\
        - cv.THRESH_BINARY，11,2）
        - th3 = cv.adaptiveThreshold (img,255,cv.ADAPTIVE_THRESH_GAUSSIAN_C,\
        - cv.THRESH_BINARY，11,2）
- Ostu的二值化：这就是用来做滑块验证的吧
    - 例子：

```
                import cv2 as cv
                import numpy as np
                from matplotlib import pyplot as plt

                # 读取图像并转换为灰度图
                img = cv.imread('noisy2.png', cv.IMREAD_GRAYSCALE)
                assert img is not None, "无法读取文件，请使用 os.path.exists() 检查"

                # 全局阈值
                ret1, th1 = cv.threshold(img, 127, 255, cv.THRESH_BINARY)

                # Otsu 的阈值
                ret2, th2 = cv.threshold(img, 0, 255, cv.THRESH_BINARY + cv.THRESH_OTSU)

                # 高斯滤波后的 Otsu 阈值处理
                blur = cv.GaussianBlur(img, (5, 5), 0)
                ret3, th3 = cv.threshold(blur, 0, 255, cv.THRESH_BINARY + cv.THRESH_OTSU)

                # 绘制所有图像及其直方图
                images = [img, 0, th1,
                          img, 0, th2,
                          blur, 0, th3]
                titles = ['原始噪声图像', '直方图', '全局阈值 (v=127)',
                          '原始噪声图像', '直方图', 'Otsu 阈值',
                          '高斯滤波图像', '直方图', "Otsu 阈值"]

                for i in range(3):
                    plt.subplot(3, 3, i * 3 + 1), plt.imshow(images[i * 3], 'gray')
                    plt.title(titles[i * 3]), plt.xticks([]), plt.yticks([])
                    plt.subplot(3, 3, i * 3 + 2), plt.hist(images[i * 3].ravel(), 256)
                    plt.title(titles[i * 3 + 1]), plt.xticks([]), plt.yticks([])
                    plt.subplot(3, 3, i * 3 + 3), plt.imshow(images[i * 3 + 2], 'gray')
                    plt.title(titles[i * 3 + 2]), plt.xticks([]), plt.yticks([])
                plt.show()
```

- 结果：



图像

- 明显第三种好很多

- 平滑图像：
  - 图像也可以用各种低通滤波器 (LPF)、高通滤波器 (HPF) 等进行滤波。LPF 有助于消除噪音、模糊图像等。HPF 滤波器有助于在图像中查找边缘。
  - 科普：二维卷积
    - ## 卷积核的作用
    - 卷积核可以设计来实现多种图像处理效果，例如：
    - **模糊**：通过使用平均化或高斯模糊核，可以减少图像的噪声或细节。
    - **锐化**：通过增强图像的边缘，使图像看起来更清晰。
    - **边缘检测**：如Sobel或Canny算子，用于检测图像中的边缘。
    - **浮雕效果**：通过突出显示图像中的纹理和轮廓。
    - ## 二维卷积的步骤

    1. **选择卷积核**：根据所需的图像处理效果选择合适的卷积核。
    2. **图像准备**：确保图像是二维数组格式，适合进行卷积操作。
    3. **边界处理**：决定如何处理图像的边界。常见的方法包括扩展边缘像素、填充零或复制边缘。
    4. **卷积操作**：将卷积核的中心与图像的第一个像素对齐，计算卷积核与图像局部区域的乘积和，得到新图像的第一个像素值。然后滑动卷积核到下一个像素位置，重复计算过程，直到覆盖整个图像。
    5. **归一化**：有时需要对结果进行归一化或缩放，以确保像素值在有效的图像显示范围内。

        - 卷积核（也称为滤波器或掩模）的大小和定义取决于你想要实现的图像处理效果。卷积核是一个小的矩阵，用于在图像上滑动并应用特定的数学运算。以下是一些选择和定义卷积核的指导原则：
        - **尺寸（Size）**：
            - **卷积核的尺寸可以是奇数或偶数，常见的尺寸有3x3、5x5、7x7等。奇数尺寸的卷积核有一个中心元素，这使得卷积操作更简单。**
            - **大尺寸的卷积核可以捕捉图像中的大特征，但计算量更大，可能会模糊细节。小尺寸的卷积核计算量小，但可能无法捕捉大的特征。**
        - **元素值（Element Values）**：
            - **卷积核的元素可以是正数、负数或零。这些值定义了在卷积过程中每个像素的贡献度。**
            - **通常，卷积核的元素值被标准化，使得所有值的总和为1或其他某个值，以控制卷积操作的强度。** %%
    - OpenCV 提供了一个函数cv.filter2D()来将内核与图像进行卷积。
    - 这个是平均模糊，直接就是blur
        - 该操作的工作原理如下：将此内核保持在像素上方，添加此内核下方的所有 25 个像素，取平均值，并用新的平均值替换中心像素。对图像中的所有像素继续此操作。
        - example

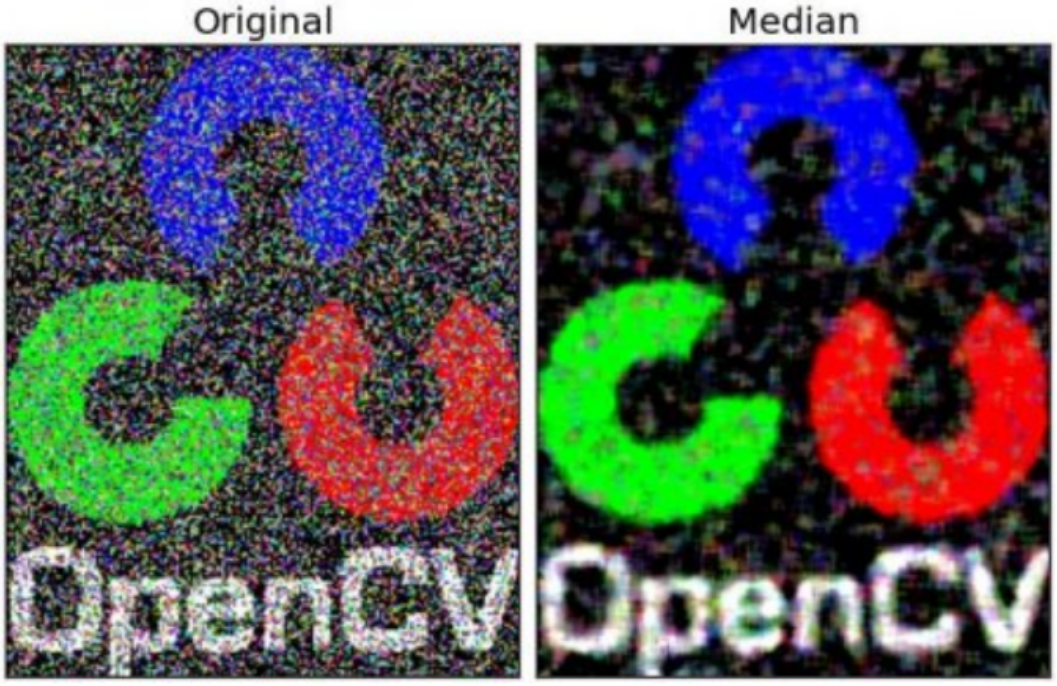
```
import numpy as np
import cv2 as cv
from matplotlib import pyplot as plt

# 读取图像
img = cv.imread('opencv_logo.png')
assert img is not None, "file could not be read, check with os.path.exists()"

# 定义一个平均模糊的卷积核
kernel = np.ones((5, 5), np.float32) / 25

# 应用二维卷积进行模糊处理
dst = cv.filter2D(img, -1, kernel)

# 使用matplotlib显示原始图像和处理后的图像
plt.subplot(121), plt.imshow(img), plt.title('Original')
plt.xticks([]), plt.yticks([])  # 隐藏坐标轴
plt.subplot(122), plt.imshow(dst), plt.title('Averaging')
plt.xticks([]), plt.yticks([])  # 隐藏坐标轴
plt.show()
```

- 然后是高斯模糊

- 在这个方法中，我们不使用盒式过滤器，而是使用高斯核。它使用函数cv.GaussianBlur()完成。我们应该指定核的宽度和高度，它们应该是正数和奇数。我们还应该分别指定 X 和 Y 方向的标准偏差 sigmaX 和 sigmaY。如果只指定了sigmaX，则 sigmaY 与 sigmaX 相同。如果两者都为零，则根据核大小计算它们。高斯模糊在从图像中去除高斯噪声方面非常有效。
- 例如：blur = cv.GaussianBlur (img,(5,5),0)

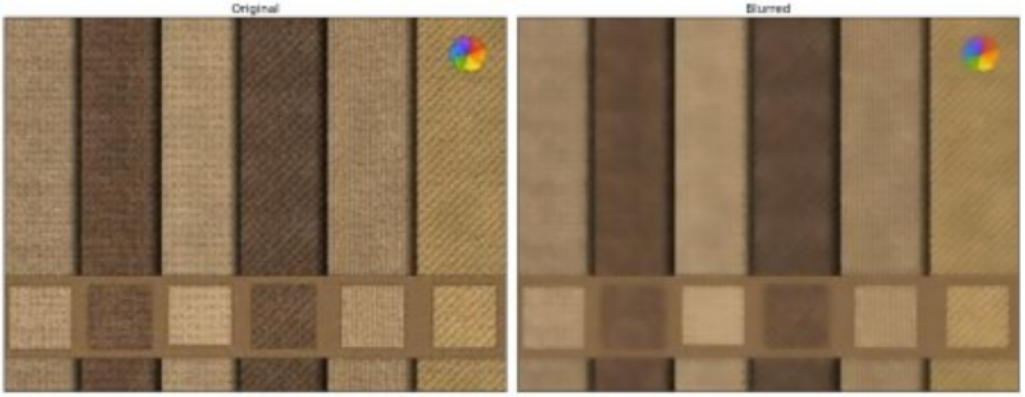
图像

- 然后是中值模糊
  - 在这里，函数 `cv.medianBlur()` 计算卷积核下所有像素的中值，并将中心元素替换为这个中值。这对于图像中的椒盐噪声非常有效。有趣的是，在上述滤波器中，中心元素是一个新计算的值，可能是图像中的像素值或一个新值。但在中值模糊中，中心元素总是被图像中的某个像素值替换。它有效地减少了噪声。其卷积核大小应该是一个正奇数。
  - median = cv.medianBlur(img,5)



图像

- 双边过滤
  - cv.bilateralFilter()在去除噪声的同时保持边缘清晰，效果非常好。但与其他过滤器相比，该操作速度较慢。我们已经看到，高斯滤波器取像素周围的邻域并找到其高斯加权平均值。此高斯滤波器仅是空间函数，即在过滤时考虑附近的像素。
  - 它不考虑像素是否具有几乎相同的强度。它不考虑像素是否是边缘像素。因此它也会模糊边缘，这是我们不想发生的。
  - 双边滤波也采用空间高斯滤波器，但多了一个高斯滤波器，它是像素差的函数。空间高斯函数确保只考虑附近的像素进行模糊处理，而强度差高斯函数确保只考虑与中心像素强度相似的像素进行模糊处理。因此，它保留了边缘，因为边缘处的像素将具有较大的强度变化。
  - blur = cv.bilateralFilter (img,9,75,75)



图像

- 形态变换
  - 简单起见，学个侵蚀就走人！
  - 侵蚀的基本思想就像土壤侵蚀一样，它会侵蚀掉前景物体的边界（始终尝试保持前景为白色）。那么它的作用是什么呢？内核在图像中滑动（就像在 2D 卷积中一

样）。只有当内核下的所有像素都为 1 时，原始图像中的像素（1 或 0）才会被视为 1，否则它会被侵蚀（变为零）。

- 因此，根据内核的大小，所有边界附近的像素都将被丢弃。因此，前景对象的厚度或大小会减小，或者图像中的白色区域会减小。它对于去除小白噪声（如我们在色彩空间章节中看到的那样）、分离两个相连的对象等很有用。

```
        import cv2 as cv
        import numpy as np

        # 读取图像，以灰度模式
        img = cv.imread('j.png', cv.IMREAD_GRAYSCALE)
        # 确保图像文件被成功读取
        assert img is not None, "file could not be read, check with
os.path.exists()"

        # 定义一个5x5的全1卷积核，用于腐蚀操作
        kernel = np.ones((5, 5), np.uint8)

        # 执行腐蚀操作，迭代1次
        erosion = cv.erode(img, kernel, iterations=1)
```

-