图像的运算

电子信工程系

袁羽

目录

CONTENTS

图像的运算

图像是由像素组成的，像素又是由具体的正整数表示的，因此图像也可以进行一系列数学运算，通过运算可以获得截取、合并图像等效果。OpenCV提供了很多图像运算方法，经过运算的图像可以呈现出很多有趣的视觉效果。

一 掩模

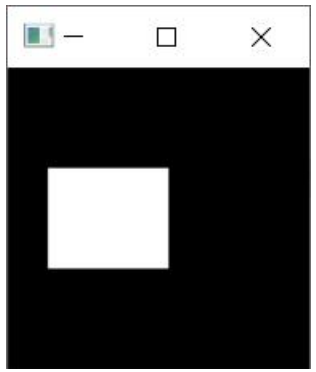
当计算机处理图像时，图像如同一名“患者”一样，有些内容需要处理，有些内容不需要处理。通常计算机处理图像时会把所有像素都处理一遍，但如果想让计算机像外科大夫那样仅处理某一小块区域，那就要为图像盖上一张仅暴露一小块区域的“手术洞巾”。像“手术洞巾”那样能够覆盖原始图像、仅暴露原始图像“感兴趣区域”（ROI）的模板图像就被叫作掩模。

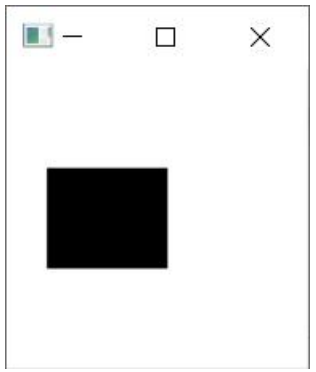
一 掩模

掩模，也叫作掩码，英文为mask，在程序中用二值图像来表示：0值（纯黑）区域表示被遮盖的部分，255值（纯白）区域表示暴露的部分（某些场景下也会用0和1当作掩模的值）。

在使用OpenCV处理图像时，通常使用numpy库提供的方法创建掩模图像

例：创建3通道掩模图像：利用numpy库的zeros()方法创建一幅掩模图像，感兴趣区域为在该图像中横坐标为20、纵坐标为50、宽为60、高为50的矩形，展示该掩模图像。调换该掩模图像的感兴趣区域和不感兴趣区域之后，再次展示掩模图像。



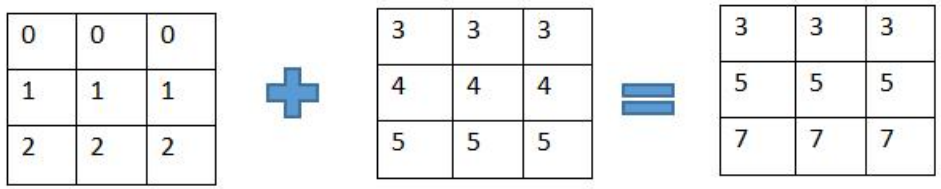


例：创建3通道掩模图像：利用numpy库的zeros()方法创建一幅掩模图像，感兴趣区域为在该图像中横坐标为20、纵坐标为50、宽为60、高为50的矩形，展示该掩模图像。调换该掩模图像的感兴趣区域和不感兴趣区域之后，再次展示掩模图像。

```
import cv2
import numpy as np
# 创建宽150、高150、3通道，像素类型为无符号8位数字的零值图像
mask = np.zeros((150, 150, 3), np.uint8)
mask[50:100, 20:80, :] = 255;  # 50~100行、20~80列的像素改为纯白像素
cv2.imshow("mask1", mask)  # 展示掩模
mask[:, :, :] = 255;  # 全部改为纯白像素
mask[50:100, 20:80, :] = 0;  # 50~100行、20~80列的像素改为纯黑像素
cv2.imshow("mask2", mask)  # 展示掩模
cv2.waitKey()  # 按下任何键盘按键后
cv2.destroyAllWindows()  # 释放所有窗体
```

二 图像的加法运算

图像中每一个像素都是用整数表示的像素值，2幅图像相加就是让相同位置像素值相加，最后将计算结果按照原位置重新组成一幅新图像。



二 图像的加法运算

在开发程序时通常不会使用“+”运算符对图像做加法运算，而是用OpenCV提供的add()方法，该方法的语法如下：dst = cv2.add(src1, src2, mask, dtype)

参数说明：

src1：第一幅图像。

src2：第二幅图像。

mask：可选参数，掩模，建议使用默认值。

dtype：可选参数，图像深度，建议使用默认值。

返回值说明：

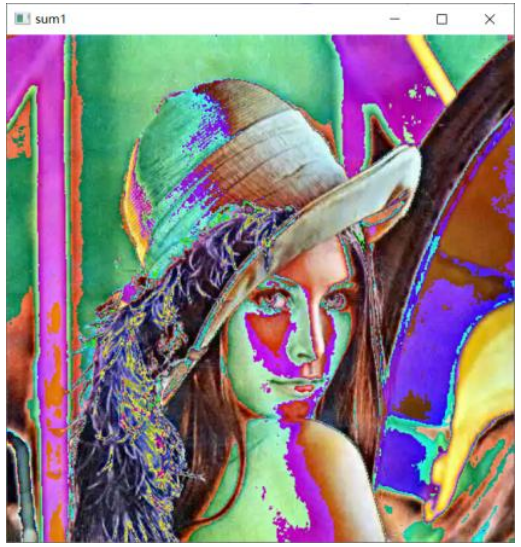
dst：相加之后的图像。如果相加之后值的结果大于255，则取255。

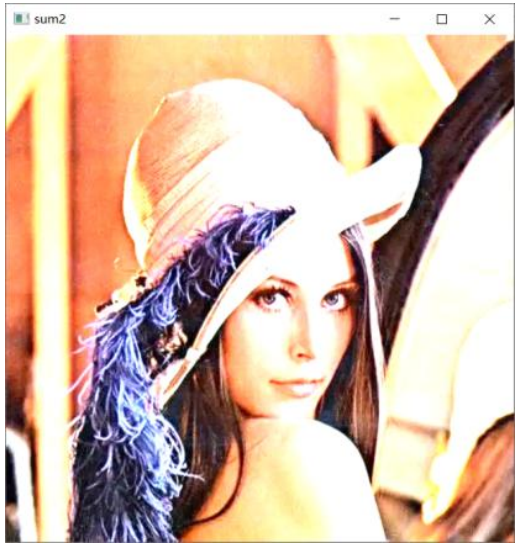
下面通过一个实例演示“+”运算符和add()方法处理结果的不同。

例：分别使用“+”和add()方法计算图像和：读取一幅图像，让该图像自己对自己做加法运算，分别使用“+”运算符和add()方法，观查两者相加结果的不同。

```
import cv2
img = cv2.imread("lenna.jpg") # 读取原始图像
sum1 = img + img # 使用运算符相加
sum2 = cv2.add(img, img) # 使用方法相加
cv2.imshow("img", img) # 展示原图
cv2.imshow("sum1", sum1) # 展示运算符相加结果
cv2.imshow("sum2", sum2) # 展示方法相加结果
cv2.waitKey() # 按下任何键盘按键后
cv2.destroyAllWindows() # 释放所有窗体
```

例：分别使用“+”和add()方法计算图像和：读取一幅图像，让该图像自己对自己做加法运算，分别使用“+”运算符和add()方法，观查两者相加结果的不同。



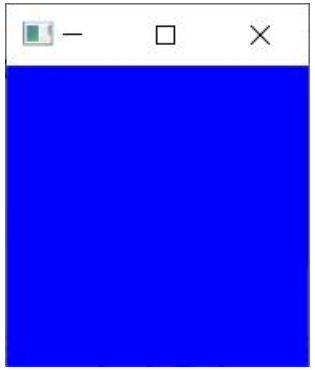
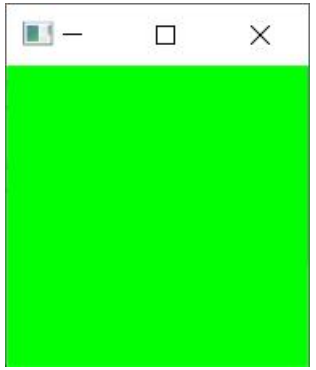
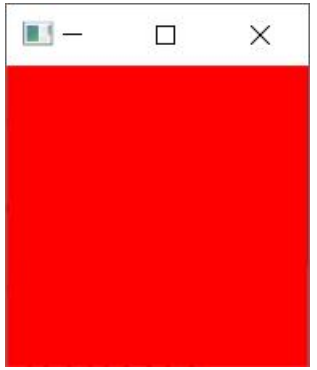
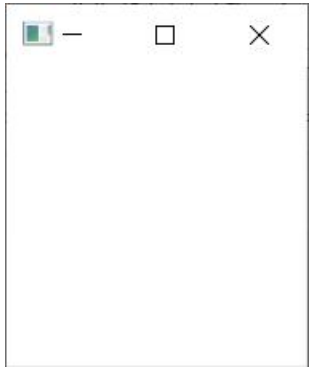


“+”运算符的计算结果如果超出了255，就会取相加和除以255的余数，也就是取模运算，像素值相加后反而变得更小，由浅色变成了深色；而add()方法的计算结果如果超过了255，就取值255，很多浅颜色像素彻底变成了纯白色。

例：模拟三色光叠加得白光：分别创建纯蓝、纯绿、纯红3种图像，取3幅图像的相加和，查看结果。

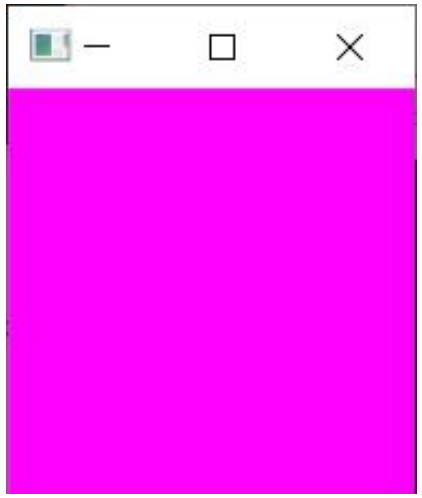
```
import cv2
import numpy as np
img = np.zeros((200, 200, 3), np.uint8)
blue = img.copy()  # 复制图像
blue[:, :, 0] = 255  # 1通道所有像素都为255
green = img.copy()
green[:, :, 1] = 255  # 2通道所有像素都为255
red = img.copy()
red[:, :, 2] = 255  # 3通道所有像素都为255
img1 = blue + green + red
img2 = cv2.add(blue,green)
img3 = cv2.add(img2,red)
cv2.imshow("blue", blue)  # 展示图像
cv2.imshow("green", green)
cv2.imshow("red", red)
cv2.imshow("img1", img1)
cv2.imshow("img3", img3)
cv2.waitKey()  # 按下任何键盘按键后
cv2.destroyAllWindows()  # 释放所有窗体
```

例：模拟三色光叠加得白光：分别创建纯蓝、纯绿、纯红3种图像，取3幅图像的相加和，查看结果。

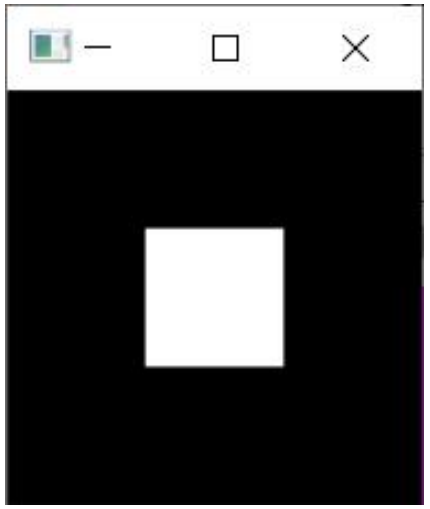





两种加法计算结果符合光学三原色的叠加原理。

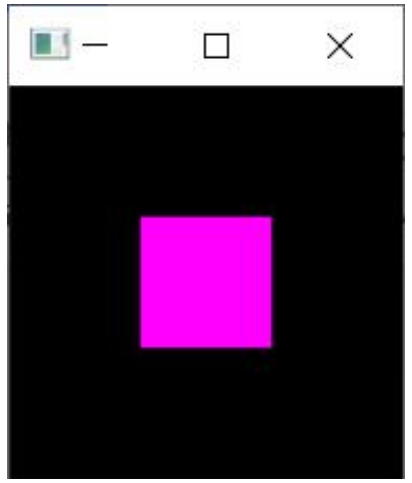
例：创建纯蓝和纯红2幅图像，使用add()方法对2幅图像进行加法运算，然后利用掩模遮盖相加。



蓝色和红色相加的结果



掩模



通过掩模相加的结果

从结果可以看出，add()方法中如果使用了掩模参数，
相加的结果只会保留掩模中白色覆盖的区域。

例：创建纯蓝和纯红2幅图像，使用add()方法对2幅图像进行加法运算，然后利用掩模遮盖相加。

```
import cv2
import numpy as np
img1 = np.zeros((150, 150, 3), np.uint8)
img1[:, :, 0] = 255  # 蓝色通道赋予最大值
img2 = np.zeros((150, 150, 3), np.uint8)
img2[:, :, 2] = 255  # 红色通道赋予最大值
img = cv2.add(img1, img2)  # 蓝色 + 红色 = 洋红色
cv2.imshow("no mask", img)  # 展示相加的结果
m = np.zeros((150, 150, 1), np.uint8)  # 创建掩模
m[50:100, 50:100, :] = 255  # 掩模中央位置为纯白色
cv2.imshow("mask", m)  # 展示掩模
img = cv2.add(img1, img2, mask=m)  # 相加时使用掩模
cv2.imshow("use mask", img)  # 展示相加的结果
cv2.waitKey()  # 按下任何键盘按键后
cv2.destroyAllWindows()  # 释放所有窗体
```

三 图像的位运算

位运算是二进制数特有的运算操作。图像由像素组成，每个像素可以用十进制整数表示，十进制整数又可以转化为二进制数，所以图像也可以做位运算，并且位运算是图像数字化技术中一项重要的运算操作。

方法	含义
cv2.bitwise_and()	按位与运算(有0出0，全1出1)
cv2.bitwise_or()	按位或运算(有1出1，全0出0)
cv2.bitwise_not()	按位取反运算(将0变为1，1变为0)
cv2.bitwise_xor()	按位异或运算(相同出0，相异出1)

二 图像的位运算

1.按位与运算：图像做与运算时，会把每一个像素值都转为二进制数，然后让两幅图像相同位置的两个像素值做与运算，最后把运算结果保存在新图像的相同位置上。

按照二进制位进行判断，如果同一位的数字都是1，则运算结果的相同位数字取1，否则取0。

该方法的语法如下：dst = cv2.bitwise_and(src1, src2, mask)

参数说明：

src1：第一幅图像。

src2：第二幅图像。

mask：可选参数，掩模。

返回值说明：

dst：与运算之后的图像。

三 图像的位运算

图像做与运算时，会把每一个像素值都转为二进制数，然后让两幅图像相同位置的两个像素值做与运算，最后把运算结果保存在新图像的相同位置上。按照二进制位进行判断，如果同一位的数字都是1，则运算结果的相同位数字取1，否则取0。

图像做与运算时的运算过程如下：

&	0	0	0	1	0	1	0	0
	0	0	0	1	1	1	1	0
	0	0	0	1	0	1	0	0

三 图像的位运算

与运算有两个特点。

（1）如果某像素与纯白像素做与运算，结果仍然是某像素的原值，计算过程如下：

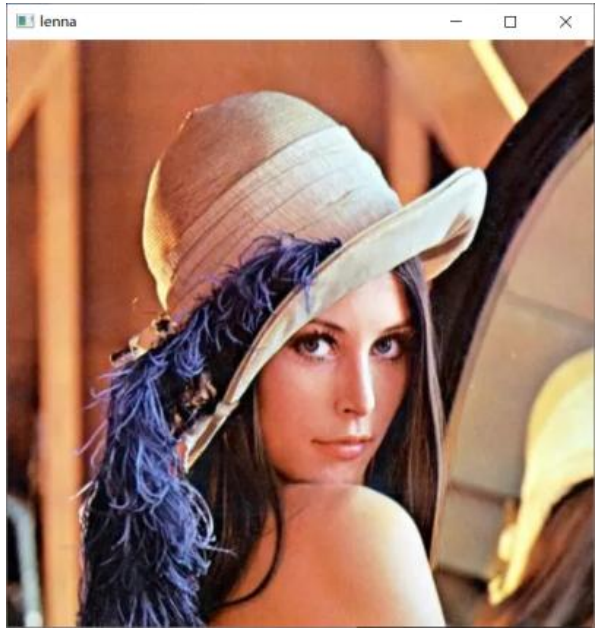
00101011 & 11111111 = 00101011

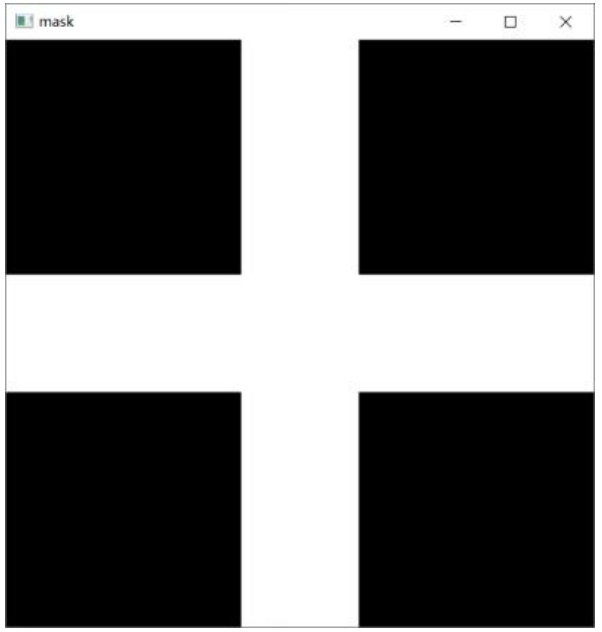
（2）如果某像素与纯黑像素做与运算，结果为纯黑像素，计算过程如下：

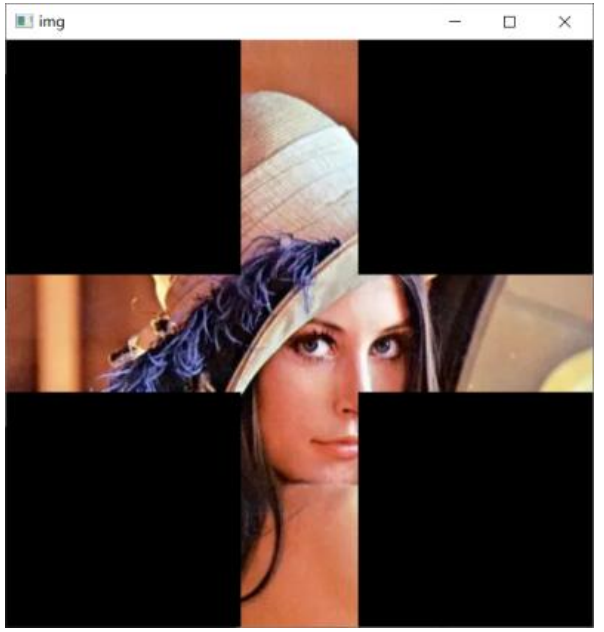
00101011 & 00000000 = 00000000

如果原图像与掩模进行与运算，原图像仅保留掩模中白色区域覆盖的内容，其他区域全部变成黑色。

例：图像与十字掩模做与运算：创建一个掩模，在掩模中央保留一个十字形的白色区域，让掩模与花图像做与运算。







例：图像与十字掩模做与运算：创建一个掩模，在掩模中央保留一个十字形的白色区域，让掩模与图像做与运算。

```
import cv2
import numpy as np
lenna = cv2.imread("lenna.jpg") # 原始图像
mask = np.zeros(lenna.shape, np.uint8) # 与图像大小相等的掩模图像
mask[200:300, :, :] = 255 # 横着的白色区域
mask[:, 200:300, :] = 255 # 竖着的白色区域
img = cv2.bitwise_and(lenna, mask) # 与运算
cv2.imshow("lenna", lenna) # 展示图像
cv2.imshow("mask", mask) # 展示掩模图像
cv2.imshow("img", img) # 展示与运算结果
cv2.waitKey() # 按下任何键盘按键后
cv2.destroyAllWindows() # 释放所有窗体
```

三 图像的位运算

2.按位或运算

或运算也是按照二进制位进行判断，如果同一位的数字都是0，则运算结果的相同位数字取0，否则取1。

该方法的语法如下： dst = cv2.bitwise_or(src1, src2, mask)

参数说明:

src1：第一幅图像。

src2：第二幅图像。

mask：可选参数，掩模。

返回值说明:

dst：或运算之后的图像。

三 图像的位运算

2.按位或运算

或运算也是按照二进制位进行判断，如果同一位的数字都是0，则运算结果的相同位数字取0，否则取1。图像做或运算时的运算过程如下：

	0	0	0	1	0	1	0	0
	0	0	0	1	1	1	1	0
	0	0	0	1	1	1	1	0

三 图像的位运算

2.按位或运算

或运算有以下两个特点。

（1）如果某像素与纯白像素做或运算，结果为纯白像素，计算过程如下：

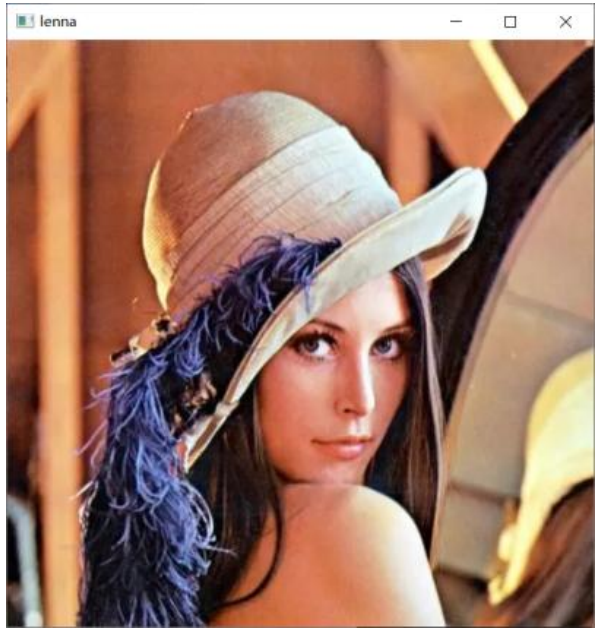
00101011 | 11111111 = 11111111

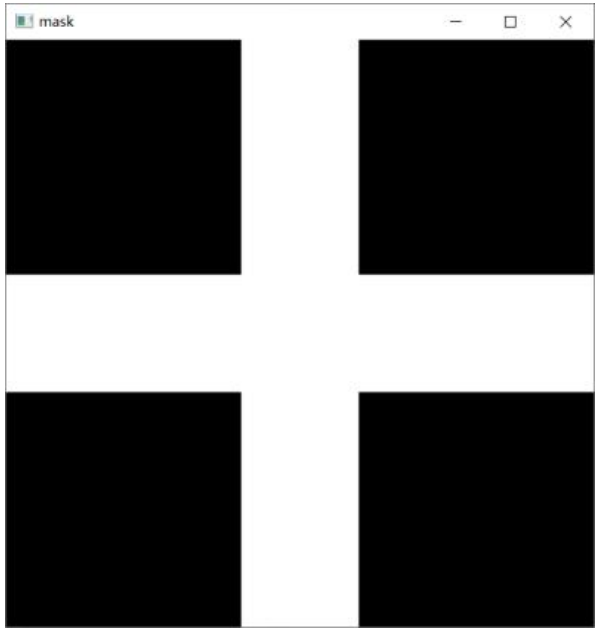
（2）如果某像素与纯黑像素做或运算，结果仍然是某像素的原值，过程如下：

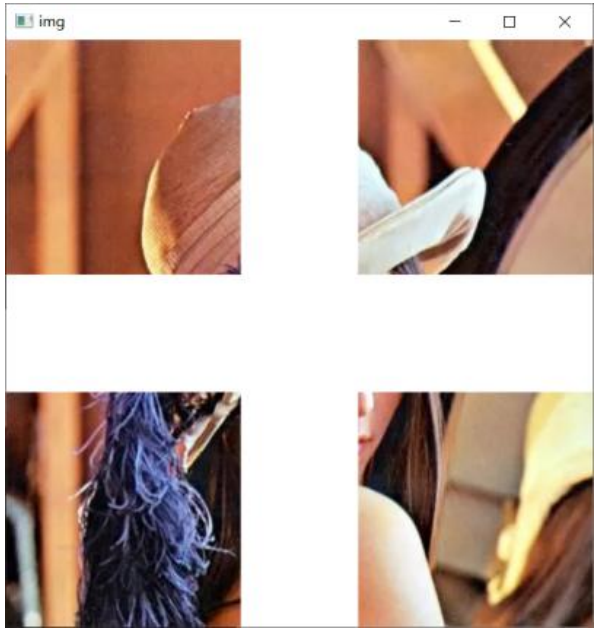
00101011 | 00000000 = 00101011

如果原图像与掩模进行或运算，原图像仅保留掩模中黑色区域覆盖的内容，其他区域全部变成白色。

例：图像与十字掩模做或运算：创建一个掩模，在掩模中央保留一个十字形的白色区域，让掩模与图像做或运算。







例：图像与十字掩模做或运算：创建一个掩模，在掩模中央保留一个十字形的白色区域，让掩模与花图像做或运算。

```
import cv2
import numpy as np
lenna = cv2.imread("lenna.jpg")  # 原始图像
mask = np.zeros(lenna.shape, np.uint8)  # 与图像大小相等的掩模图像
mask[200:300, :, :] = 255  # 横着的白色区域
mask[:, 200:300, :] = 255  # 竖着的白色区域
img = cv2.bitwise_or(lenna, mask)  # 或运算
cv2.imshow("lenna", lenna)  # 展示图像
cv2.imshow("mask", mask)  # 展示掩模图像
cv2.imshow("img", img)  # 展示或运算结果
cv2.waitKey()  # 按下任何键盘按键后
cv2.destroyAllWindows()  # 释放所有窗体
```

三 图像的位运算

3.按位取反运算

取反运算是一种单目运算，仅需一个数字参与运算就可以得出结果。如果运算数某位上数字是0，则运算结果的相同位的数字就取1，如果这一位的数字是1，则运算结果的相同位的数字就取0。

OpenCV提供bitwise_not()方法来对图像做取反运算，该方法的语法如下：

dst = cv2.bitwise_not(src, mask)

参数说明：

src：参与运算的图像。

mask：可选参数，掩模。

返回值说明：

dst：取反运算之后的图像。

三 图像的位运算

3.按位取反运算

取反运算是一种单目运算，仅需一个数字参与运算就可以得出结果。如果运算数某位上数字是0，则运算结果的相同位的数字就取1，如果这一位的数字是1，则运算结果的相同位的数字就取0。

OpenCV提供bitwise_not()方法来对图像做取反运算，该方法的语法如下：

dst = cv2.bitwise_not(src, mask)

参数说明：

src：参与运算的图像。

mask：可选参数，掩模。

返回值说明：

dst：取反运算之后的图像。

三 图像的位运算

3.按位取反运算

图像做取反运算的过程如下：

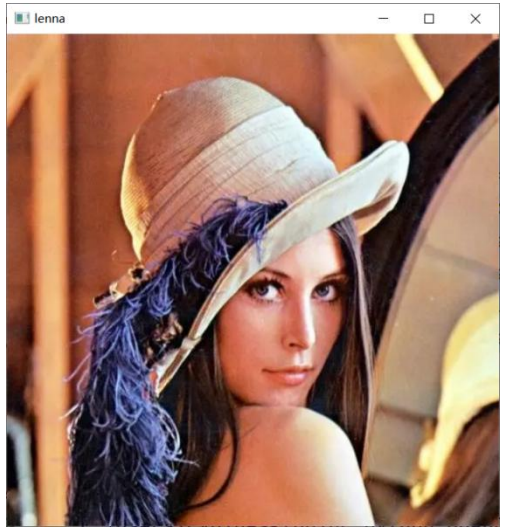
~	0	0	0	1	0	1	0	0
	1	1	1	0	1	0	1	1

例：对图像进行取反运算。

```
import cv2
import numpy as np
lenna = cv2.imread("lenna.jpg")  # 原始图像
img = cv2.bitwise_not(lenna)  # 取反运算
cv2.imshow("lenna", lenna)  # 展示图像
cv2.imshow("img", img)  # 展示取反运算结果
cv2.waitKey()  # 按下任何键盘按键后
cv2.destroyAllWindows()  # 释放所有窗体
```

例：对图像进行取反运算。

图像经过取反运算后呈现与原图颜色完全相反的效果。



三 图像的位运算

4.按位异或运算

异或运算也是按照二进制位进行判断，如果两个运算数同一位上的数字相同，则运算结果的相同位数字取0，否则取1。OpenCV提供bitwise_xor()方法对图像做异或运算，该方法的语法如下：dst = cv2.bitwise_xor(src, mask)

参数说明：

src：参与运算的图像。

mask：可选参数，掩模。

返回值说明：

dst：异或运算之后的图像。

三 图像的位运算

4.按位异或运算

^	0	0	0	1	0	1	0	0
	0	0	0	1	1	1	1	0
	0	0	0	0	1	0	1	0

异或运算有两个特点。

（1）如果某像素与纯白像素做异或运算，结果为原像素的取反结果，计算过程如下：

00101011 ^ 11111111 = 11010100

（2）如果某像素与纯黑像素做异或运算，结果仍然是某像素的原值，计算过程如下：

00101011 ^ 00000000 = 00101011

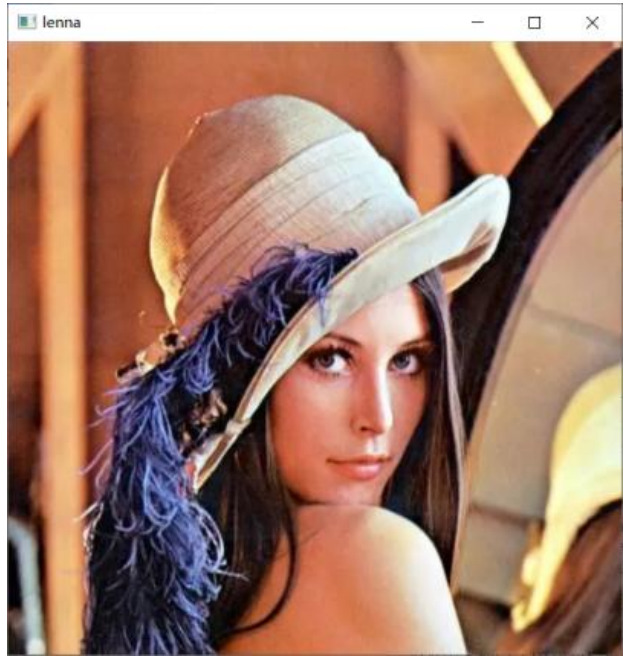
由此可以得出：如果原图像与掩模进行异或运算，掩模白色区域覆盖的内容呈现取反效果，黑色区域覆盖的内容保持不变。

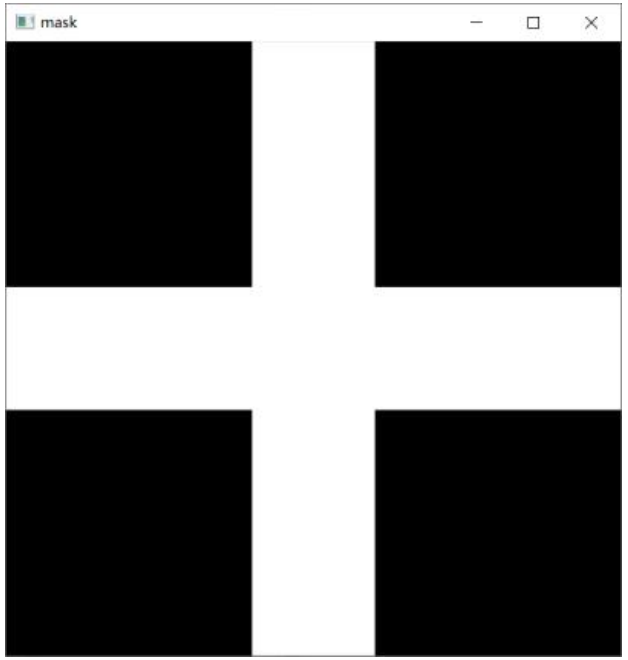
例：创建一个掩模，在掩模中央保留一个十字形的白色区域，让掩模与图像做异或运算。

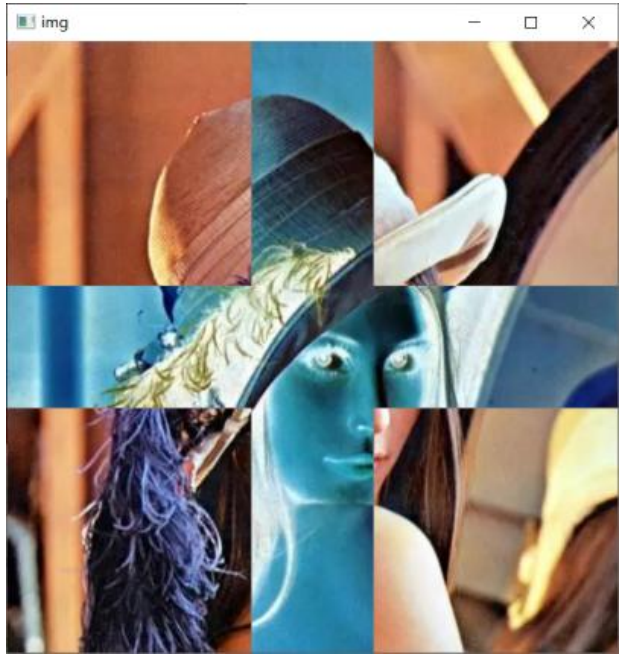
```
import cv2
import numpy as np
lenna = cv2.imread("lenna.jpg")  # 花原始图像
mask = np.zeros(lenna.shape, np.uint8)  # 与花图像大小相等的掩模图像
mask[200:300, :, :] = 255  # 横着的白色区域
mask[:, 200:300, :] = 255  # 竖着的白色区域
img = cv2.bitwise_xor(lenna, mask)  # 异或运算
cv2.imshow("lenna", lenna)  # 展示花图像
cv2.imshow("mask", mask)  # 展示掩模图像
cv2.imshow("img", img)  # 展示异或运算结果
cv2.waitKey()  # 按下任何键盘按键后
cv2.destroyAllWindows()  # 释放所有窗体
```

例：创建一个掩模，在掩模中央保留一个十字形的白色区域，让掩模与图像做异或运算。

掩模白色区域覆盖的内容与原图像做取反运算的结果一致，掩模黑色区域覆盖的内容保持不变。







三 图像的位运算

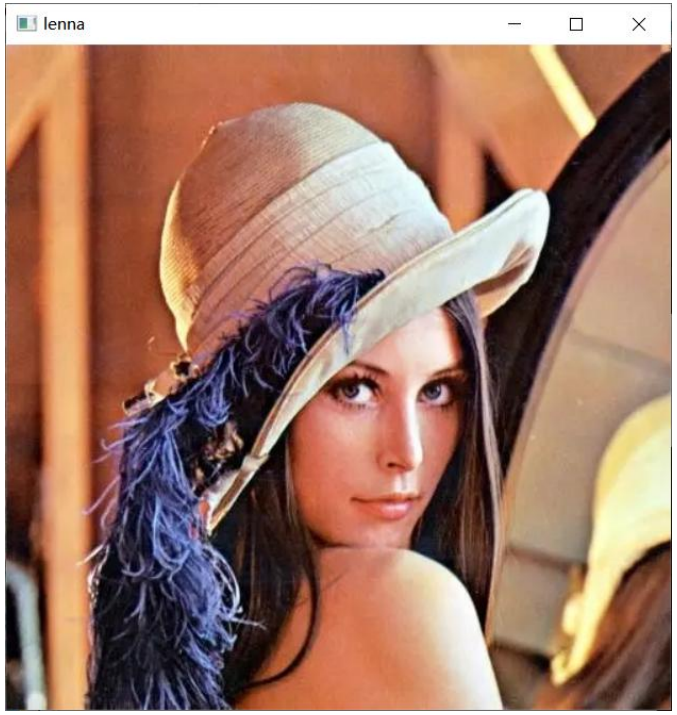
4.按位异或运算

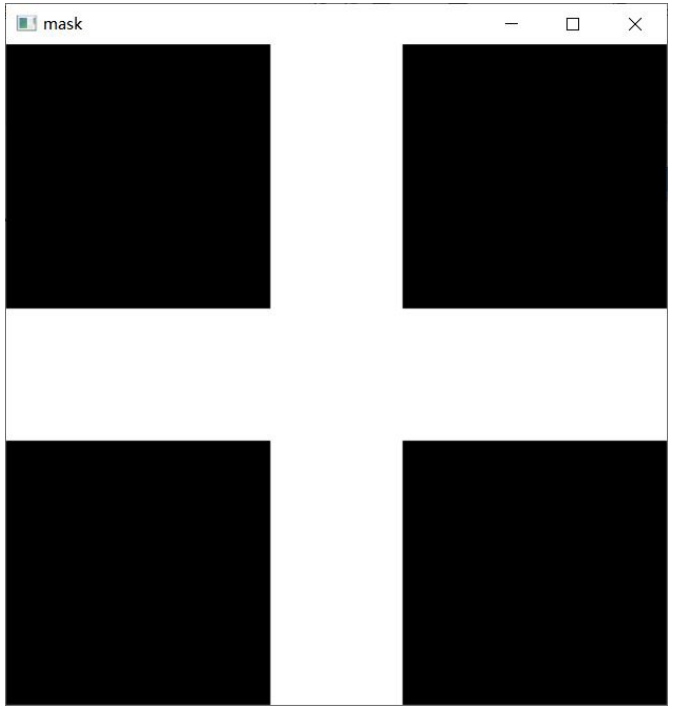
异或运算还有一个特点：执行一次异或运算得到一个结果，再对这个结果执行第二次异或运算，则还原成最初的值。利用这个特点可以实现对图像内容的加密和解密。

例：利用异或运算的特点对图像数据进行加密和解密。

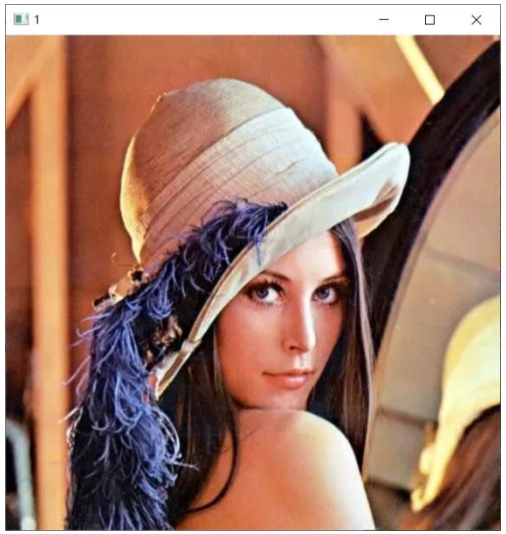
```
import cv2
import numpy as np
lenna = cv2.imread("lenna.jpg")  # 花原始图像
mask = np.zeros(lenna.shape, np.uint8)  # 与花图像大小相等的掩模图像
mask[200:300, :, :] = 255  # 横着的白色区域
mask[:, 200:300, :] = 255  # 竖着的白色区域
img = cv2.bitwise_xor(lenna, mask)  # 第一次异或运算
img = cv2.bitwise_xor(img, mask)  # 第二次异或运算
cv2.imshow("lenna", lenna)  # 展示花图像
cv2.imshow("mask", mask)  # 展示掩模图像
cv2.imshow("img", img)  # 展示两次异或运算后结果
cv2.waitKey()  # 按下任何键盘按键后
cv2.destroyAllWindows()  # 释放所有窗体
```

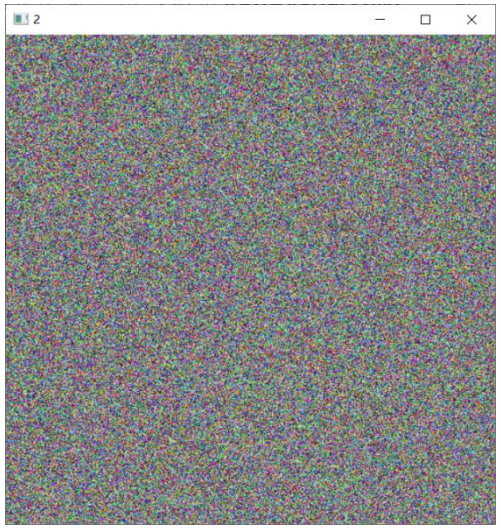
例：利用异或运算的特点对图像数据进行加密和解密。

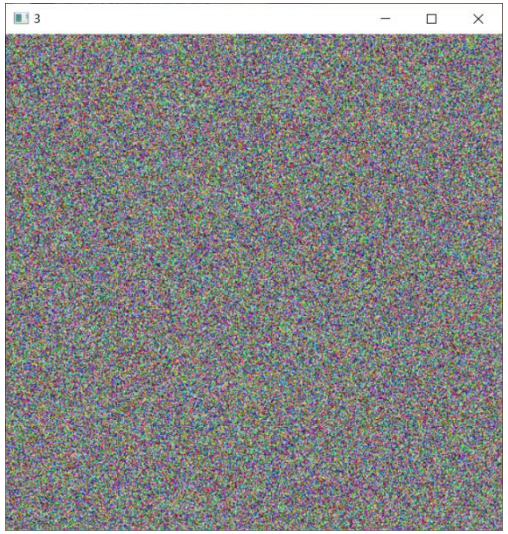




例：利用异或运算的特点对图像数据进行加密和解密：利用numpy.random.randint()方法创建一个随机像素值图像作为密钥图像，让密钥图像与原始图像做异或运算得出加密图像，再使用密钥图像对加密图像进行解密。







例：利用异或运算的特点对图像数据进行加密和解密：利用numpy.random.randint()方法创建一个随机像素值图像作为密钥图像，让密钥图像与原始图像做异或运算得出加密图像，再使用密钥图像对加密图像进行解密。

```
import cv2
import numpy as np
def encode(img, img_key): # 加密、解密方法
    result =  cv2.bitwise_xor(img, img_key)  # 两图像做异或运算
    return result
lenna = cv2.imread("lenna.jpg")  # 原始图像
rows, colmns, channel = lenna.shape  # 原图像的行数、列数和通道数
# 创建与图像大小相等的随机像素图像，作为密钥图像
img_key = np.random.randint(0, 256, (rows, colmns, 3), np.uint8)
cv2.imshow("1", lenna)  # 展示图像
cv2.imshow("2", img_key)  # 展示秘钥图像
result = encode(lenna, img_key)  # 对图像进行加密
cv2.imshow("3", result)  # 展示加密图像
result = encode(result, img_key)  # 对图像进行解密
cv2.imshow("4", result)  # 展示加密图像
cv2.waitKey()  # 按下任何键盘按键后
cv2.destroyAllWindows()  # 释放所有窗体
```

三 合并图像

在处理图像时经常会遇到需要将两幅图像合并成一幅图像，合并图像也分2种情况：

①两幅图像融合在一起；

②每幅图像提供一部分内容，将这些内容拼接成一幅图像。

OpenCV分别用加权和和覆盖两种方式来满足上述需求。

三 合并图像

1.加权和

OpenCV通过计算加权和的方式，按照不同的权重取两幅图像的像素之和，最后组成新图像。加权和不会像纯加法运算那样让图像丢失信息，而是在尽量保留原有图像信息的基础上把两幅图像融合到一起。

三 合并图像

1.加权和

OpenCV通过addWeighted()方法计算图像的加权和，addWeighted（）函数是将两张相同大小的图片融合的函数。该方法语法如下：dst = cv2.addWeighted(src1, alpha, src2, beta, gamma)

参数说明:

src1：第一幅图像。

alpha：第一幅图像的权重。

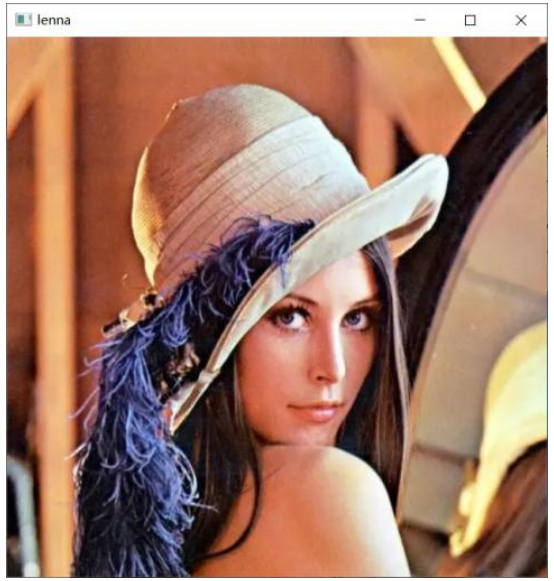
src2：第二幅图像。

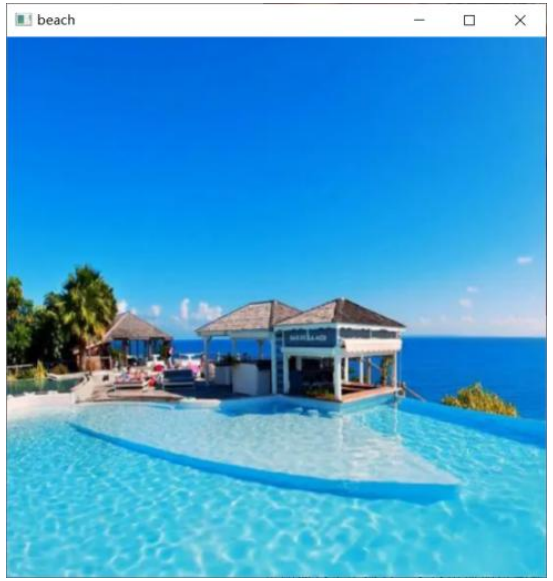
beta：第二幅图像的权重。

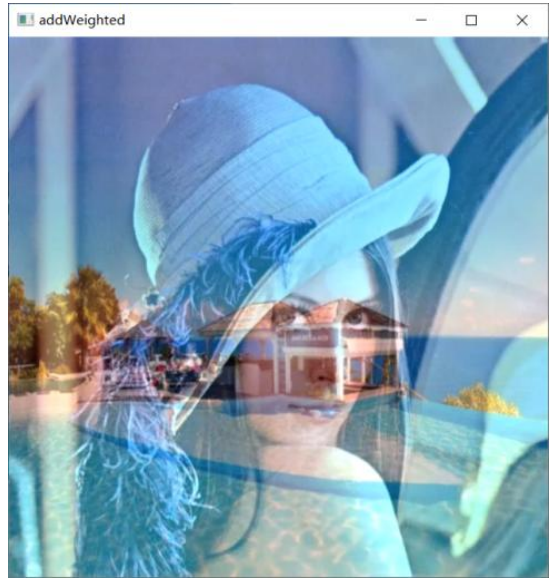
gamma：在和结果上添加的标量。该值越大，结果图像越亮，相反则越暗。可以是负数。

返回值说明：dst：加权和后的图像。

例：读取两幅不同的风景照片，使用addWeighted()方法计算两幅图像的加权和，两幅图像的权重分别为0.6和0.4，标量为0，查看处理之后的图像是否为多次曝光效果。







例：读取两幅不同的风景照片，使用addWeighted()方法计算两幅图像的加权和，两幅图像的权重分别为0.6和0.4，标量为0，查看处理之后的图像是否为多次曝光效果。

```
import cv2
lenna = cv2.imread("lenna.jpg")  # lenna原始图像
beach = cv2.imread("beach.png")  # 沙滩原始图像
rows, colmns, channel = lenna.shape  # lenna图像的行数、列数和通道数
beach = cv2.resize(beach, (colmns, rows))  # 沙滩图像缩放成lenna图像大小
img = cv2.addWeighted(lenna, 0.6, beach, 0.4, 0)  # 计算两幅图像加权和
cv2.imshow("sun", lenna)  # 展示lenna图像
cv2.imshow("beach", beach)  # 展示沙滩图像
cv2.imshow("addWeighted", img)  # 展示加权和图像
cv2.waitKey()  # 按下任何键盘按键后
cv2.destroyAllWindows()  # 释放所有窗体
```

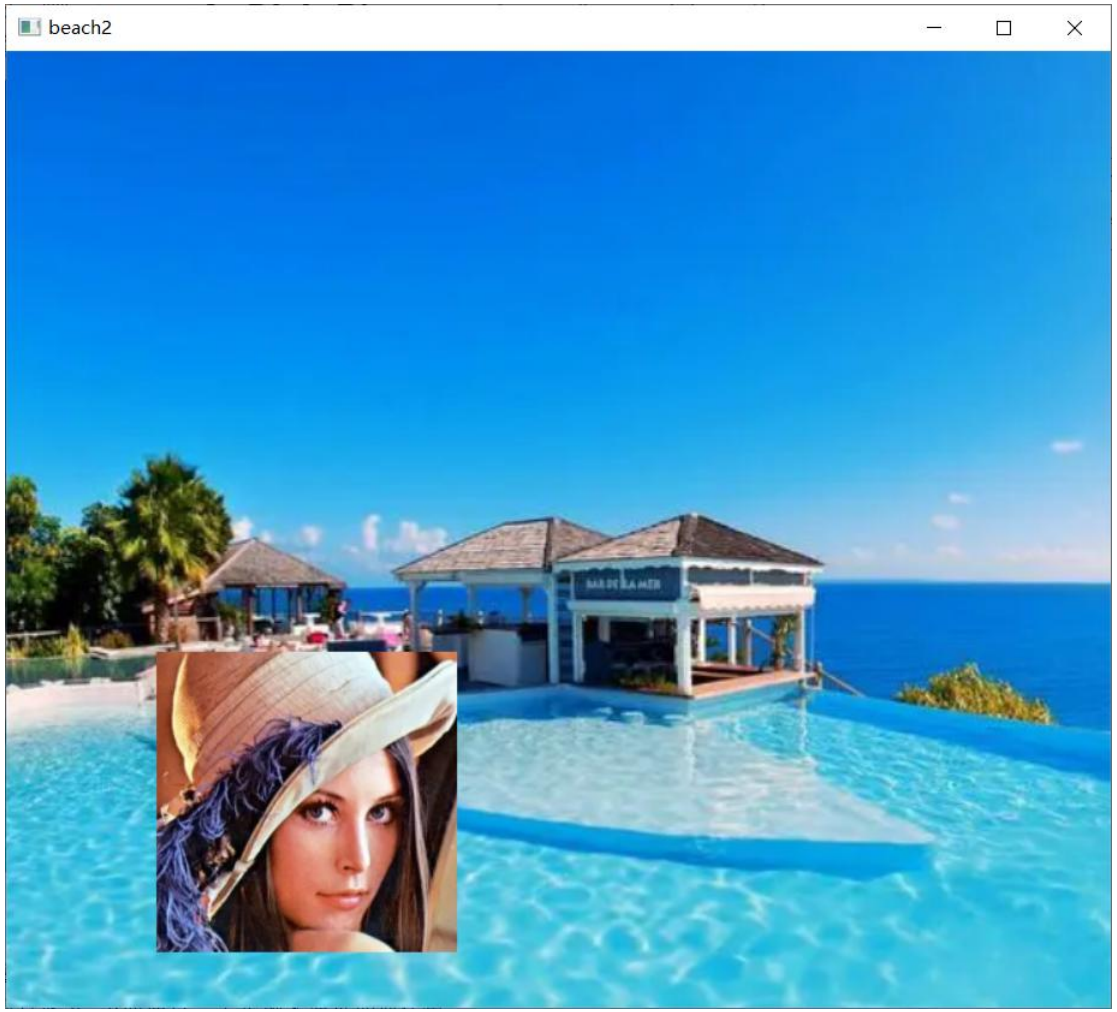
三 合并图像

2.覆盖

覆盖图像就是直接把前景图像显示在背景图像中，前景图像挡住背景图像。覆盖之后背景图像会丢失信息，不会出现加权和那样的“多次曝光”效果。

OpenCV没有提供覆盖操作的方法，可以直接用修改图像像素值的方式实现图像的覆盖、拼接效果：从A图像中取像素值，直接赋值给B图像的像素，这样就能在B图像中看到A图像的信息了。

例：从前景图像中抠图，再将抠出的图像覆盖在背景图像中。



例：从前景图像中抠图，再将抠出的图像覆盖在背景图像中。

```
import cv2
beach_img = cv2.imread("beach.png")  # 沙滩原始图像
lenna_img = cv2.imread("lenna.jpg")  # lenna原始图像
lenna = lenna_img[100:400, 100:400, :]  # 截取100行至400行、100列至400列的像素值所组成的图像
lenna = cv2.resize(lenna, (200, 200))  # 将截取出的图像缩放成200*200大小
cv2.imshow("lenna", lenna_img)  # 展示lenna原始图像
cv2.imshow("lenna2", lenna)  # 展示截取并缩放的lenna图像
cv2.imshow("beach", beach_img)  # 展示沙滩原始图像
rows, colmns, channel = lenna.shape  # 记录截取图像的行数和列数
# 将沙滩中一部分像素改成截取之后的图像
beach_img[400:400 + rows, 100:100 + colmns, :] = lenna
cv2.imshow("beach2", beach_img)  # 展示修改之后的图像
cv2.waitKey()  # 按下任何键盘按键后
cv2.destroyAllWindows()  # 释放所有窗体
```

小结

1掩模：白色是感兴趣区域，黑色是遮挡区域。

2图像加法运算：

“+”运算符取模

add()方法取纯白像素

3.图像的位运算：按位与（去白留黑）按位或（去黑留白）按位取法（颜色反转）按位异或（留黑白色反转）

4.图像合并：加权和

覆盖

THANKS