東南大學 SOUTHEAST UNIVERSITY 人工智能学院

专业技能实训

授课教师：王洪松

邮箱：hongsongwang@seu.edu.cn

图像运算

图像的基本表示方法

- 二值图像

  包含黑色和白色两种颜色的图像，1为白色，0位黑色。

- 灰度图像

  采用更多的颜色灰度表现更多的细节，分为256 个灰度级，用数值区间[0, 255]来表示，0表示纯黑，255表示纯白。灰度值正好可以用一个字节（ 8 位二进制值）来表示。

- 彩色图像

  存在 R（red，红色）、G（green，绿色）和 B（blue，蓝色）三个通道。每个色彩通道值的范围都在[0, 255]之间。

  OpenCV读入的彩色图像三个通道顺序为BGR 。

获取图像属性

- 图像的属性包括：行，列，通道，图像数据类型，像素数目等
- 图像的形状

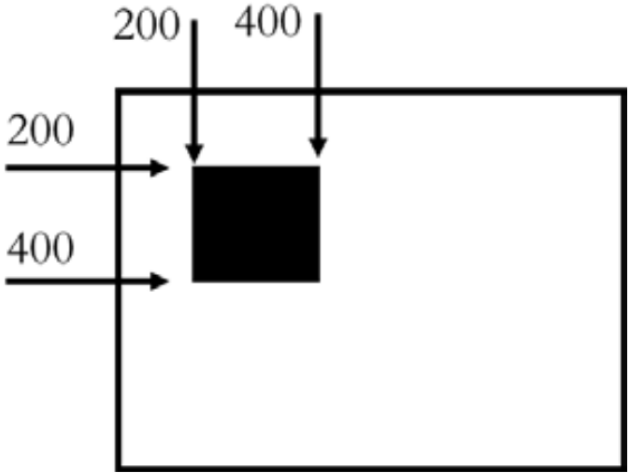
  img.shape 可以获取图像的形状，返回（height, width, channels）。

  如果图像是灰度图，返回（height, width ），通过检查这个返回值就可以知道加载的是灰度图还是彩色图。
- 图像的像素数目 img.size
- 图像的数据类型 img.dtype，通常是uint8

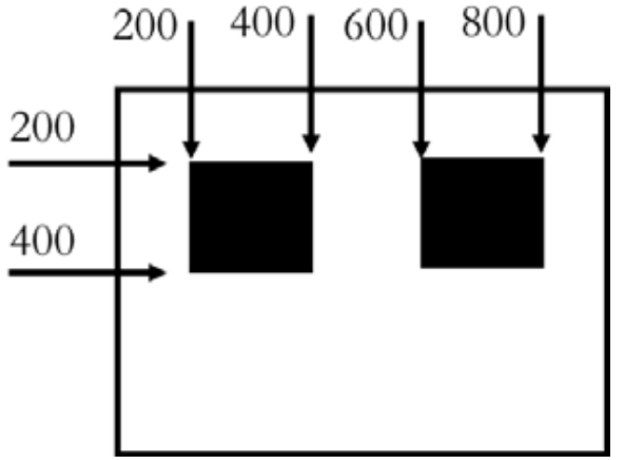
图像基础操作

- 感兴趣区域（ ROI）

在图像处理过程中，对图像的某一个特定区域感兴趣，该区域被称为感兴趣区域（Region of Interest， ROI）。



ROI 示例



复制结果

将黑色 ROI 复制到该区域右侧

```
a=img[200:400,200:400]
img[200:400,600:800]=a
```

```
img[10, 10, 2]
img.item(10,10,2)
img.itemset((10,10,2),100)
```

- 获取/修改像素值

用 Numpy 的 array.item() 和 array.itemset() 会更好，返回标量。

图像基础操作

- 选择感兴趣区域

  cv2.selectROI(windowName, img, showCrosshair, fromCenter)

  cv2.selectROI(img, showCrosshair, fromCenter)

  windowName：选择的区域被显示在的窗口的名字。
  img：要在什么图片上选择ROI 。
  showCrosshair：是否在矩形框里画十字线。fromCenter：
  是否是从矩形框的中心开始画。
  控件：使用空格或enter键完成选择，使用c键取消选择。
  返回值 [min_x,min_y,w,h]

- 选择多个感兴趣区域

  cv2.selectROIs(windowName, img, showCrosshair, fromCenter)

  使用空格或回车键完成当前选择并开始新的选择，使用esc终止选择过程。

图像基础操作

- 拆分/合并图像通道

  有时需要对 BGR 三个通道分别进行操作，把 BGR 拆分成单个通道，并把独立通道的图片合并成一个 BGR 图像。

```
b,g,r=cv2.split(img)          或者
img=cv2.merge(b,g,r)          b=img[:,:,0]
```

- 图像扩边（填充）

  cv2.copyMakeBorder(src, top, bottom, left, right, borderType)

图像基础操作

- 图像扩边（填充）

cv2.copyMakeBorder(src, top, bottom, left, right, borderType, value)
src 输入图像
top, bottom, left, right 对应边界的像素数目。
borderType 要添加那种类型的边界，常见类型如下
– cv2.BORDER_CONSTANT 添加有颜色的常数值边界，还需要下一个参数（value）。
– cv2.BORDER_REFLECT 边界元素的镜像。比如: cba|abcdefgh|hgf
– cv2.BORDER_REPLICATE 重复最后一个元素。例如: aaa| abcdefgh|hhh
– cv2.BORDER_WRAP, 就像这样: fgh|abcdefgh|abc
value 边界颜色，如果边界的类型是 **cv2.BORDER_CONSTANT**

```
wrap =cv2.copyMakeBorder(img1,10,10,10,10,cv2.BORDER_WRAP)
constant=cv2.copyMakeBorder(img1,10,10,10,10,
cv2.BORDER_CONSTANT,value=BLUE)
```

图像运算：算术运算

- 图像加法运算

可以通过加号运算符“+”对图像进行加法运算，也可以通过cv2.add()函数对图像进行加法运算。

两幅图像的大小，类型必须一致，或者第二个图像可以使一个简单的标量值。

使用加号运算符“+”对图像 a（像素值为 a）和图像 b（像素值为 b）进行求和运算时，遵循以下规则：

$$a+b=\begin{cases} a+b, & a+b\leqslant 255 \\ \mathrm{mod}(a+b,256), & a+b>255 \end{cases}$$

式中，mod()是取模运算，“mod(a+b, 256)”表示计算“a+b 的和除以 256 取余数”。

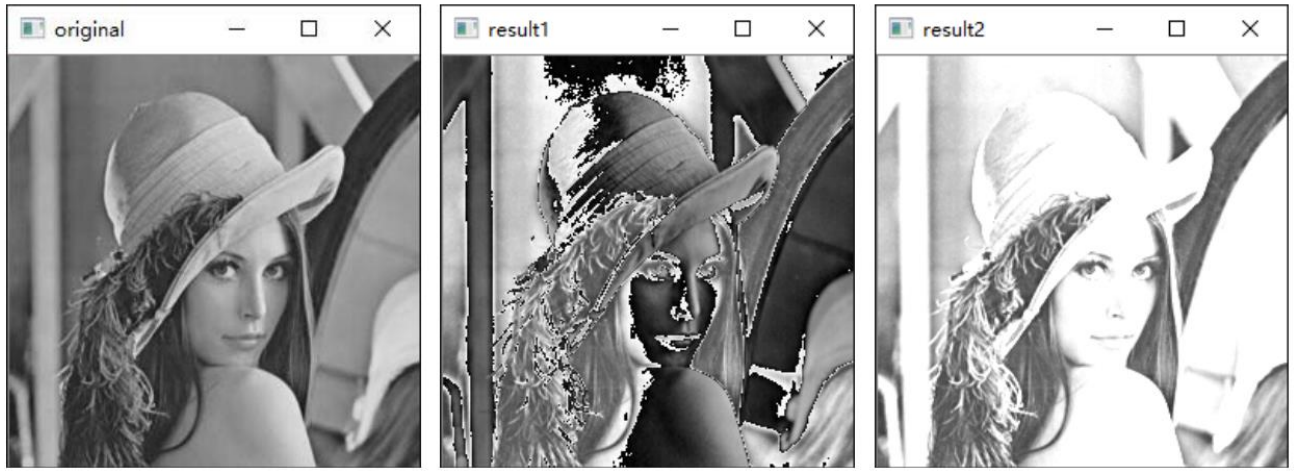
图像运算：算术运算

- 图像加法运算

计算结果=cv2.add(像素值 a,像素值 b)

用函数cv2.add() 进行加法运算，会得到像素值对应图像的饱和值（最大值）。

$$a+b=\begin{cases} a+b, & a+b\leqslant 255 \\ 255, & a+b>255 \end{cases}$$




```
a=cv2.imread("lena.bmp",0)
b=a
result1=a+b
result2=cv2.add(a,b)
```

使用加号运算符求和时，取模后大于 255 的大于255的值变得更小了，导致本来应该更亮的像素点变得更暗了，相加所得的图像看起来不自然。

图像运算：算术运算

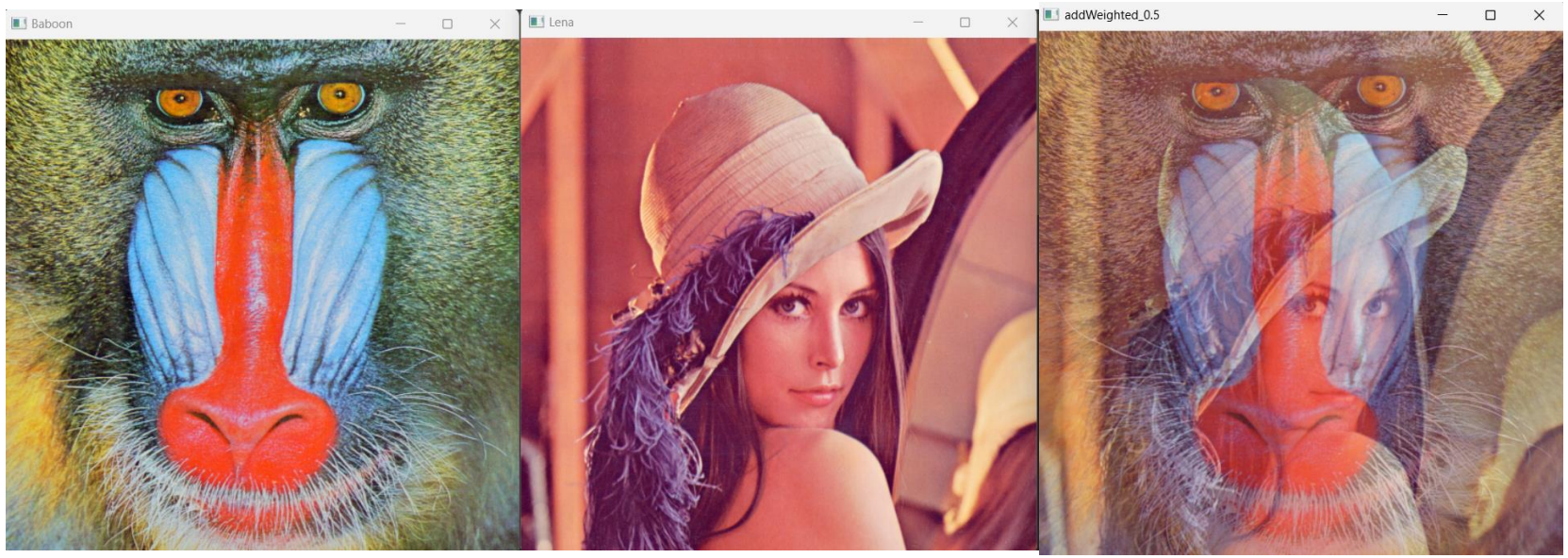
- 图像加权和

  dst=cv2.addWeighted(src1, alpha, src2, beta, gamma)

参数 alpha 和 beta 是 src1 和 src2 所对应的系数，实现的功能是

  dst = src1×alpha + src2×beta + gamma

图像混合操作时gamma取0。



cv2.addWeighted()函数，对图像 Baboon 和图像 Lena 分别按照 0.5 和 0.5 的权重进行混合。

图像运算：算术运算

- 图像加权和

对比cv2.addWeighted和Numpy运算符的结果：

dst=cv2.addWeighted(src1, alpha, src2, beta, gamma)

dst = alpha *src1 + beta *src2+ gamma

```
alpha, beta = 0.5, 0.5
dst1 = cv2.addWeighted(img1,alpha,img2,beta,0)
dst2 = (alpha*img1 + beta*img2).astype(np.uint8)
```

不要用Numpy运算做图像加权和，需要注意alpha, beta是int还是float类型，是否大于1，求和的数值是否超过255等问题。

图像运算：掩模

- 掩模

OpenCV 中的很多函数都有掩模参数，使用时只会在掩模值为非空的像素点上执行，并将其他像素点的值置为0。

$$\text{img3=cv2.add(img1,img2,mask=mask)}$$

$$\text{img1} = \begin{vmatrix} 3 & 3 & 3 & 3 \\ 3 & 3 & 3 & 3 \\ 3 & 3 & 3 & 3 \\ 3 & 3 & 3 & 3 \end{vmatrix} \quad \text{mask} = \begin{vmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \end{vmatrix}$$

$$\text{img2} = \begin{vmatrix} 5 & 5 & 5 & 5 \\ 5 & 5 & 5 & 5 \\ 5 & 5 & 5 & 5 \\ 5 & 5 & 5 & 5 \end{vmatrix} \quad \text{img3} = \begin{vmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 8 & 8 \\ 0 & 0 & 8 & 8 \end{vmatrix}$$

```
import cv2
import numpy as np

img1=np.ones((4,4),dtype=np.uint8)*3
img2=np.ones((4,4),dtype=np.uint8)*5
mask=np.zeros((4,4),dtype=np.uint8)
mask[2:4,2:4]=1
img3=cv2.add(img1,img2,mask=mask)
print("求和后 img3=\n",img3)
```

图像运算：按位逻辑运算

在 OpenCV 内，常见的位运算函数如下：

按位与

dst = cv2.bitwise_and(src1, src2[, mask]])

按位或

dst = cv2.bitwise_or(src1, src2[, mask]])

按位异或

dst = cv2.bitwise_xor(src1, src2[, mask]])

按位取反

dst = cv2.bitwise_not(src[, mask]])

dst 表示与入值具有同样大小的 array 输出值。

src1 表示第一个 array 或 scalar 类型的输入值。

src2 表示第二个 array 或 scalar 类型的输入值。

mask 表示可选操作掩码，8 位单通道 array 值。

任何数值与0 进行按位与操作，都会得到 0。

任何数值 与255进行按位与操作，都会得到数值本身。

图像运算：位平面分解

将灰度图像中处于同一比特位上的二进制像素值进行组合， 得到一幅二进制值图像，该图像被称为灰度图像的一个位平面，这个过程被称为位平面分解。

例如，将一幅灰度图像内所有像素点上处于二进制位内最低位上的值进行组合，可以构成“最低有效位”位平面。

在 8 位灰度图中， 每一个像素使用 8 位二进制值来表示：

$$\text{value} = a_7 \times 2^7 + a_6 \times 2^6 + a_5 \times 2^5 + a_4 \times 2^4 + a_3 \times 2^3 + a_2 \times 2^2 + a_1 \times 2^1 + a_0 \times 2^0$$

式中， a(i)的可能值为 0 或 1。可以看出，各个a(i)的权重是不一样的，a(7)的权重最高， a(0)的权重最低。这代表a(7)的值对图像的影响最大， 而a(0)的值对图像的影响最小。

图像运算：位平面分解

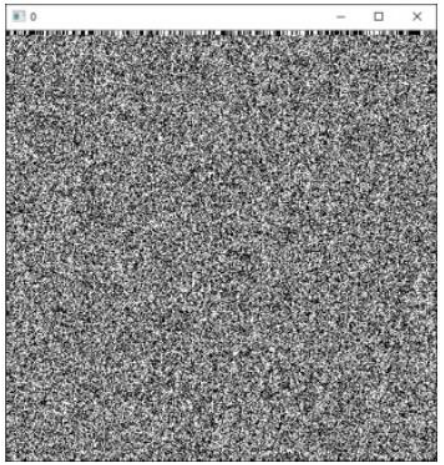
- 提取位平面

将像素值与一个值为 2^i的数值进行按位与运算，能够使像素值的第 i 位保持不变，而将其余各位均置零。因此，通过按位与运算，能够提取图像的指定位平面。

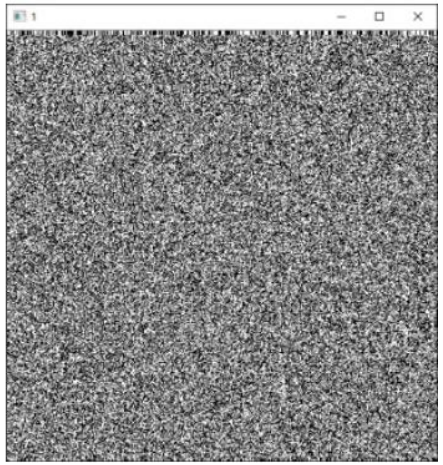
```
import cv2
import numpy as np

lena=cv2.imread("lena.bmp",0)
cv2.imshow("lena",lena)
r,c=lena.shape
x=np.zeros((r,c,8),dtype=np.uint8)
for i in range(8):
    x[:,:,i]=2**i
r=np.zeros((r,c,8),dtype=np.uint8)
for i in range(8):
    r[:,:,i]=cv2.bitwise_and(lena,x[:,:,i])
    mask=r[:,:,i]>0
    r[mask]=255
    cv2.imshow(str(i),r[:,:,i])

cv2.waitKey()
cv2.destroyAllWindows()
```

图像运算：位平面分解

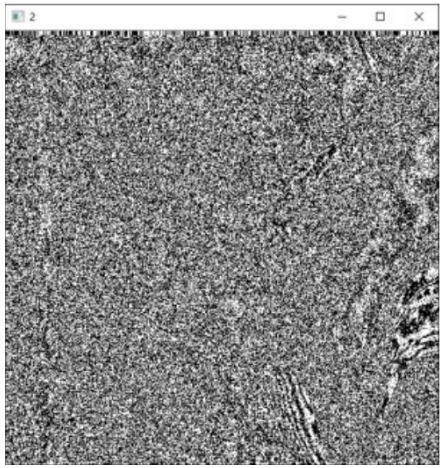
原图



第 1 个位平面



第 2 个位平面



第 3 个位平面

第 4 个位平面

第 4 个位平面

第 5 个位平面

第 6 个位平面

第 7 个位平面

按位运算应用：图像加密和解密

- 图像的加密和解密（位异或运算）

通过对原始图像与密钥图像进行按位异或， 可以实现加密；将加密后的图像与密钥图像再次进行按位异或， 可以实现解密。

按位异或运算的规则
xor(a,b)=c
xor(c,b)=a
xor(c,a)=b

```
lena=cv2.imread("lena.bmp",0)
r,c=lena.shape
key=np.random.randint(0,256,size=[r,c],dtype=np.uint8)
encryption=cv2.bitwise_xor(lena,key)
decryption=cv2.bitwise_xor(encryption,key)
```

从左到右依次为:
原始图像
密钥图像key
加密图像encryption
解密图像decryption

按位运算应用：数字水印

- 最低有效位（Least Significant Bit， LSB）指的是一个二进制数中的第 0 位（即最低位）。
- 最低有效位信息隐藏是将一个需要隐藏的二值图像信息嵌入载体图像的最低有效位，即将载体图像的最低有效位层替换为当前需要隐藏的二值图像，从而实现将二值图像隐藏的目的。
- 由于二值图像处于载体图像的最低有效位上，所以对于载体图像的影响非常不明显，其具有较高的隐蔽性。
- 在必要时直接将载体图像的最低有效位层提取出来，即可得到嵌入在该位上的二值图像，达到提取秘密信息的目的。
- 这种信息隐藏也被称为数字水印，通过该方式可以实现信息隐藏、版权认证、身份认证、数字签名等功能。

按位运算应用：数字水印

```
#读取原始载体图像
lena=cv2.imread("lena.bmp",0)
#读取水印图像
watermark=cv2.imread("watermark.bmp",0)
#将水印图像内的值 255 处理为 1，以方便嵌入
w=watermark[:,:]>0
watermark[w]=1

#读取原始载体图像的 shape 值
r,c=lena.shape
#==========嵌入过程=========
#生成元素值都是 254 的数组
t254=np.ones((r,c),dtype=np.uint8)*254
#获取 lena 图像的高七位
lenaH7=cv2.bitwise_and(lena,t254)
#将 watermark 嵌入 lenaH7 内
e=cv2.bitwise_or(lenaH7,watermark)
```

```
#=======提取过程==========
#生成元素值都是 1 的数组
t1=np.ones((r,c),dtype=np.uint8)
#从载体图像内提取水印图像
wm=cv2.bitwise_and(e,t1)
#将水印图像内的值 1 处理为 255，
以方便显示
w=wm[:,:]>0
wm[w]=255
```

原始图像

水印图像
watermark

含水印
载体图像e

提取到的
水印图像 wm

按位运算应用：脸部打码及解码

- 使用掩码对图像的脸部进行打码

```
#读取原始载体图像的 shape 值
r,c=lena.shape
mask=np.zeros((r,c),dtype=np.uint8)
mask[220:400,250:350]=1
#获取一个 key,打码、解码所使用的密钥
key=np.random.randint(0,256,size=[r,c],dtype=np.uint8)
#使用密钥 key 对原始图像 lena 加密
lenaXorKey=cv2.bitwise_xor(lena, key)
#获取加密图像的脸部信息 encryptFace
encryptFace=cv2.bitwise_and(lenaXorKey,mask*255)
#将图像 lena 内的脸部值设置为 0，得到 noFace1
noFace1=cv2.bitwise_and(lena, (1-mask)*255)
#得到打码的 lena 图像
maskFace=encryptFace+noFace1
```

脸部打码图像

按位运算应用：脸部打码及解码

- 使用掩码对图像的脸部进行解码

输入：打码的图像maskFace，秘钥key，掩码mask

```
#将脸部打码的 lena 与密钥 key 进行异或运算，得到脸部的原始信息
extractOriginal=cv2.bitwise_xor(maskFace,key)
#将解码的脸部信息 extractOriginal 提取出来， 得到 extractFace
extractFace=cv2.bitwise_and(extractOriginal,mask*255)
#从脸部打码的 lena 内提取没有脸部信息的 lena 图像，得到 noFace2
noFace2=cv2.bitwise_and(maskFace,(1-mask)*255)
#得到解码的 lena 图像
extractLena=noFace2+extractFace
```

透明图像与alpha 通道

- 透明图像

透明图像是指没有背景颜色的图像，这意味着它们可以在其他图像或背景上叠加而不会影响它们。可以使用Photoshop等工具创建透明图像，并保存为支持透明度的文件格式，例如PNG。

透明图像在 RGB 色彩空间三个通道的基础上，还可以加上一个 A 通道，也叫 alpha 通道，表示透明度。alpha 通道的赋值范围是[0, 1]，或者[0, 255]，表示从透明到不透明。这种 4 个通道的色彩空间被称为 RGBA 色彩空间。

bgra = cv2.cvtColor(img, cv2.COLOR_BGR2BGRA)
将 img 从 BGR 色彩空间转换到 BGRA 色彩空间。在转换后的BGRA色彩空间中，A 是 alpha 通道，默认值为255（不透明）。

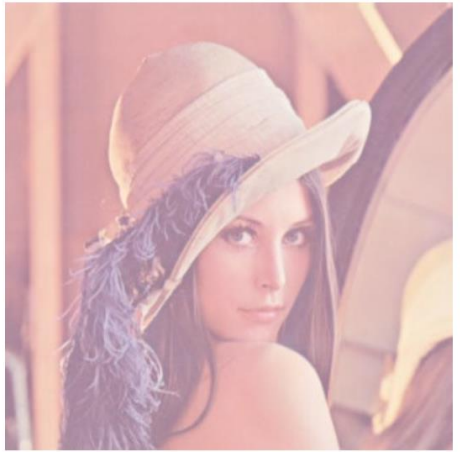
透明图像与alpha 通道

对图像的 alpha 通道进行处理

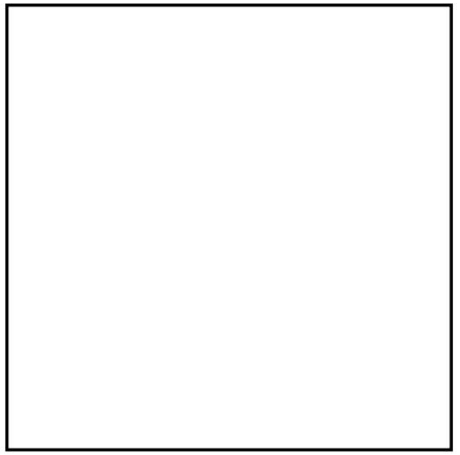
```
import cv2
img=cv2.imread( “LenaRGB.bmp")
bgra = cv2.cvtColor(img, cv2.COLOR_BGR2BGRA)
b,g,r,a= cv2.split(bgra)
a[:,:]=125
bgra125=cv2.merge([b,g,r,a])
a[:,:]=0
bgra0= cv2.merge([b,g,r,a])
cv2.imshow("img",img)
cv2.imshow("bgra",bgra)
cv2.imshow("bgra125",bgra125)
cv2.imshow("bgra0",bgra0)
cv2.waitKey()
cv2.destroyAllWindows()
cv2.imwrite("bgra.png", bgra)
cv2.imwrite("bgra125.png", bgra125)
cv2.imwrite("bgra0.png", bgra0)
```

各个图像的 alpha 通道值虽然不同，但是在显示时是没有差别的。
但打开保存的图像，能看到图像是透明的。

原图像



alpha=125



alpha=0

专业技能实训：练习题

- 制作透明图像
  - 输入图像，用鼠标选择选择前景目标像素（目标边缘不需要很精细）。
  - 将原图中除了前景目标之外的其它所有像素设置为透明。
  - 保存只包含前景目标的透明图像。

- 在图像中添加一个物体
  - 输入一张图像和另一个待插入的包含某个物体的透明图像。
  - 将待插入图像插入鼠标所在的位置。
  - 移动鼠标，该图像跟着移动。
  - 设置一个滚动条，拖动该滚动条，所插入物体相应的放大或缩小。
  - 按Enter键确认，保存合成的图像，退出。

专业技能实训：练习题

- 用多边形对图像区域打码
  - 输入图像，用鼠标按顺时针单击左键选择N个像素点，并以这N个顶点构成多边形。
  - 按Enter键确认后，对多边形所包含的图像像素打码。
- 在图像中隐藏信息
  - 用绘图函数制作一张黑白二值图像。
  - 输入新的图像，假设二值图像比输入图像小，将二值图像插入鼠标所在的位置。
  - 移动鼠标，该二值图像跟着移动。
  - 按Enter键，二值图像隐藏在输入图像中。
  - 再按Enter键，从图像中提取最低位并显示。