实验二　图像的代数运算

一、 实验目的

1．了解图像的算术运算在数字图像处理中的初步应用。

2．体会图像算术运算处理的过程和处理前后图像的变化。

二、 实验原理

图像的代数运算是图像的标准算术操作的实现方法，是两幅输入图像之间进行的点对点的加、减、乘、除运算后得到输出图像的过程。如果输入图像为A(x,y)和B(x,y)，输出图像为C(x,y)，则图像的代数运算有如下四种形式：

C(x,y) = A(x,y) + B(x,y)

C(x,y) = A(x,y) - B(x,y)

C(x,y) = A(x,y) * B(x,y)

C(x,y) = A(x,y) / B(x,y)

图像的代数运算在图像处理中有着广泛的应用，它除了可以实现自身所需的算术操作，还能为许多复杂的图像处理提供准备。例如，图像减法就可以用来检测同一场景或物体生产的两幅或多幅图像的误差。

代数运算的结果很容易超出数据类型允许的范围。例如，uint8数据能够存储的最大数值是255，各种代数运算尤其是乘法运算的结果很容易超过这个数值，有时代数操作（主要是除法运算）也会产生不能用整数描述的分数结果。图像的代数运算函数使用以下截取规则使运算结果符合数据范围的要求：超出数据范围的整型数据将被截取为数据范围的极值，分数结果将被四舍五入。例如，如果数据类型是uint8，那么大于255的结果（包括无穷大inf）将被设置为255。

注意：无论进行哪一种代数运算都要保证两幅输入图像的大小相等，且类型相同。

三、 实验步骤

1．图像加法

可以使用函数 cv2.add() 将两幅图像进行加法运算，当然也可以直接使用 numpy，result=img1+img2。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。

注意：OpenCV 中的加法与 Numpy 的加法是有所不同的。OpenCV 的加法是一种饱和操作，而 Numpy 的加法是一种取模操作。

例如下面的两个例子：

```
x = np.uint8([250])
y = np.uint8([10])
```

```
print cv2.add(x,y) # 250+10 = 260 => 255
 # [[255]]
print x+y # 250+10 = 260 % 256 = 4
 # [4]
```

这种差别在对两幅图像进行加法时会更加明显。OpenCV 的结果会更好一点。所以尽量使用 OpenCV 中的函数。

2 图像混合

这其实也是加法，但是不同的是两幅图像的权重不同，这就会给人一种混合或者透明的感觉。图像混合的计算公式如下：

$$g(x)=(1-a)f_0(x)+af_1(x)$$

通过修改α的值（0→1），可以实现非常酷的混合。

现在把两幅图混合在一起。第一幅图的权重是 0.7，第二幅图的权重是 0.3。函数 cv2.addWeighted() 可以按下面的公式对图片进行混合操作。

$$\text{dst}=\alpha\bullet\text{img1}+\beta\bullet\text{img2}+\lambda$$

这里λ的取值为 0。

```
import cv2
import numpy as np
img1=cv2.imread('diamond2.jpg')
img2=cv2.imread('flower2.jpg')
dst=cv2.addWeighted(img1,0.7,img2,0.3,0)
cv2.imshow('dst',dst)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

叠加结果如图2.2所示。

图2.1 待叠加的两幅图像

图2.2 叠加后的图像效果

给图像的每一个像素加上一个常数可以使图像的亮度增加。例如，以下代码将增加下图所示的图像的亮度，加亮后的结果如图所示。

```
import cv2
import numpy as np
iimg=cv2.imread('woman.tif',0)
img1 = cv2.add(img,80)
img2=cv2.subtract(img,80)
cv2.imshow('img',img)
cv2.imshow('img2',img2)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

原图

加80

减80

图2.3 亮度增加与变暗

2．图像的减法运算

图像减法也称为差分方法，是一种常用于检测图像变化及运动物体的图像处理方法。图像减法可以作为许多图像处理工作的准备步骤。例如，可以使用图像减法来检测一系列相同场景图像的差异。图像减法与阈值化处理的综合使用往往是建立机器视觉系统最有效的方法之一。在利用图像减法处理图像时往往需要考虑背景的更新机制，尽量补偿由于天气、光照等因素对图像显示效果造成的影响。

思考题：考虑如何实现将一幅图像从另一幅图像中减去。

3. 图像的乘法运算

两幅图像进行乘法运算可以实现掩模操作，即屏蔽掉图像的某些部分。一幅图像乘以一个常数通常被称为缩放，这是一种常见的图像处理操作。如果使用的缩放因子大于1，那么将增强图像的亮度，如果因子小于1则会使图像变暗。缩放通常将产生比简单添加像素偏移量自然得多的明暗效果，这是因为这种操作能够更好地维持图像的相关对比度。

在Python中，使用multiply函数实现两幅图像的乘法。mulitply函数的调用格式如下：

Z = cv2.mulitply(X,Y)

其中，Z=X*Y。例如，以下代码将使用给定的缩放因子对图2.5(a)所示的图像进行缩放，从而得到如图2.5(b)所示的较为明亮的图像：

img3=cv2.multiply(img,1.5)

图2.5 原图和乘以因子1.5 的图像

4. 按位运算

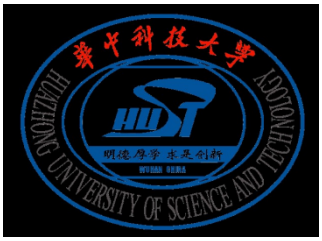
按位操作有：AND，OR，NOT，XOR 等。当我们提取图像的一部分，选择非矩形感兴趣区域（Region Of Interesting,ROI）时这些操作会很有用。

假设我们想把把 OpenCV 的 LOGO 放到另一幅图像上。如果使用加法，颜色会改变，如果使用混合，会得到透明效果。可以通过下面的按位运。

```
import cv2
import numpy as np

# 加载图像
img1 = cv2.imread('test.jpg')
img2 = cv2.imread('diamond2.jpg')
# I want to put logo on top-left corner, So I create a ROI
rows,cols,channels = img2.shape
roi = img1[0:rows, 0:cols ]
# Now create a mask of logo and create its inverse mask also
img2gray = cv2.cvtColor(img2,cv2.COLOR_BGR2GRAY)
ret, mask_front = cv2.threshold(img2gray, 175, 255,
cv2.THRESH_BINARY) #这是图像分割方法，后面讲到。
mask_inv = cv2.bitwise_not(mask_front)
# Now black-out the area of logo in ROI
# 取 roi 中与 mask 中不为零的值对应的像素的值，其他值为 0
# 注意这里必须有 mask=mask 或者 mask=mask_inv, 其中的“mask=”
不能忽略
img1_bg = cv2.bitwise_and(roi,roi,mask = mask_front)
# 取 roi 中与 mask_inv 中不为零的值对应的像素的值，其他值为 0。
# Take only region of logo from logo image.
img2_fg = cv2.bitwise_and(img2,img2,mask = mask_inv)
# Put logo in ROI and modify the main image
dst = cv2.add(img1_bg,img2_fg)
img1[0:rows, 0:cols ] = dst
cv2.imshow('img1_bg',img1_bg)
```

```
cv2.imshow('img2_fg',img2_fg)
cv2.imshow('res',img1)
cv2.waitKey(0)
cv2.destroyAllWindows()
```



img1_bg　　img2_fg

（提示:能否把logo缩小合适的比例并放置在右上角看起来比较美观的位置？）

四、 实验报告要求

1 描述实验的基本步骤，用数据和图片给出各个步骤中取得的实验结果并进行必要的讨论。

2 必须包括原始图像及其计算处理后的图像以及相应的解释。

五、 思考题

可以综合使用多种图像代数运算函数来完成一系列的操作，达到视频合成的目的。

提示:

（1） 有一个视频V1，需要作为背景；

（2） 从摄像头获取自己的视频，把背景替换为V1；

（3） 在视频下方加动态字幕；

（4） 保存视频。

六、实验图像