

1. 像素的算术运算与逻辑运算

1．像素的算术运算涉及加减乘除运算（要进行算术运算，两张图像的shape需大小一致并且像素值的数据类型也要一致：cv2.CV\_8U…..）

1. 加运算：两图像像素相加，若相加后某点像素值>255此时该点取255。

Add\_image = cv2.add ( image1, image2 ) 该运算常用于生成锐化图像(凸显边缘)

2. 减运算：两图像像素相减，若像素值<0该点像素值取0

Subtract\_image = cv2.subtract ( image1, image2 ) 该运算常用于非锐化遮蔽mask(原图 – 模糊后图像)

3. 乘运算：两图像像素相乘，若>255此时取255

Multiply\_image = cv2.multiply ( image1, image2 ) 该运算常用于生成锐化图像(凸显边缘信息)

4. 除运算：两图像像素相除，基本是全黑

Divide\_image = cv2.divide ( image1, image2 )

2. 像素的逻辑运算( 与、或、非、异或)

这里以一个黑白的圆与矩形来实现逻辑运算

先确定画布大小

Image\_1 = np.zeros((300, 300), dtype = “uint8”) # 全黑画布1

cv2.rectangle (Image\_1, (25, 25), (275, 275), 255, -1) # 在画布1上做一个(左上角点为(25,25),右上角点(275,275))的全白(像素值为255)矩形，其中-1为做线的类型。

Image\_2 = np.zeros((300, 300), dtype = “uint8”) # 全黑画布2

cv2.circle( Image\_2, (150, 150), 150, 255, -1) # 在画布2上做一个中心点为(150, 150)半径为150像素值为255的圆，其中-1为做圆的直线类型。

1. 与运算：两图像在某点有一个为0，则该点为0, 当且仅当都不为0时，该点不为0

And\_image = cv2.bitwise\_and ( Image\_1, Image\_2) 注意：and逻辑运算与add加运算有很大区别。

2. 或运算：两图像当且仅当在某点都为0时，该点为0，否则该点不为0

Or\_image = cv2.bitwise\_or ( Image\_1, Image\_2 )

3. 非运算：一副图像取反

Not\_image = cv2.bitwise\_not ( Image\_1, Image\_2 )

4. 异或运算：两图像当在某点取值不同时，该点不为0，否则为0

Xor\_image = cv2.bitwise\_xor ( Image\_1, Image\_2 ) 实例：将一个logo嵌入到一张图像中，见add\_logo.py

1. 像素点的处理(如上图)：对像素点操作，没有区域概念

主要功能：使图像对比度增强

1. 图像像素点灰度变换：每一个像素点根据规则变换

1. 灰度线性变换：图像的每一个像素点进行线性变换,大于255的像素点取255

Image\_data = cv2.imread (“image”)

Trans\_data = Image\_data \* 3

Trans\_data[Trans\_data > 255] = 255

2. 分段线性变换：根据像素点值范围进行分段线性变换，大于255的像素点取255

例：<125: f(x) = 1.2 \* x >=125: f(x)= 2\*x

Image\_data = cv2.imread (“image”)

Trans\_data = copy.deepcopy ( Image\_data )

Trans\_data[Image\_data<125] = Trans\_data[Image\_data<125] \* 1.2

Trans\_data[Image\_data>=125] = Trans\_data[Image\_data>=125] \* 2

Trans\_data[Trans\_data>255] = 255

3．伽马变换( 幂次变换 )：像素点值先得归一化 [0,1]，后进行幂次变换(非负实数), 最后乘以255

注意：当伽马系数 = 1时为原图，

>1时的感兴趣区域较亮，降低对比度，原本两数值因为幂次后数值差值变小

<1时的感兴趣区域较暗，增强对比度，原本两数值因为幂次后数值差值变大

Image\_data = cv2.imread ( “image”)

Trans\_data = Image\_data / 255.0

Gamma\_w = 0.4

Trans\_data = np.power ( Trans\_data, Gamma\_w ) \* 255.0

4.对数变换：主要用于傅里叶频谱

S = c \* log( 1 + r ) 以2为底的对数

.

1. 灰度直方图修正：根据图像灰度分布进行整体灰度变换
2. 直方图正规化：将图像灰度范围转至0-255从而得到线性变换关系，后进行线性变换

实现如下：

Image\_data = cv2.imread ( “image” )

Min\_v, Max\_v, Min\_loc, Max\_loc = cv2.minMaxLoc ( image\_data ) # 返回当前图像最小、最大像素值以及所在位置。

trans\_w = 255.0 / ( Max\_v – Min\_v)

trans\_bias = 0 – trans\_w \* Min\_v

Trans\_data = trans\_w \* Image\_data + trans\_bias

Opencv中自带函数：

Src = cv2.imread (“image”)

Dst = np.zeros (Src.shape, np.uint8)

cv2.normalize (Src, Dst, 255, 0, cv2.NORM\_MINMAX, dtype = cv2.CV\_8U) # dtype 在opencv中统一指图像数据类型

1. 全局直方图均衡化：增强图像对比度，但暗区域噪声也会变得更清晰

实现如下：1.计算图像的灰度直方图

2.计算灰度直方图的累加直方图

3.计算输入灰度级与输出灰度级的映射关系

4.根据映射关系循环输出图像的每一个像素的灰度级

函数：

Image\_data = cv2.imread ( “image” )

Trans\_data = cv2.equalizeHist ( Image\_data )

1. 限制对比度的自适应直方图均衡化

Image\_data = cv2.imread ( “image” )

Clahe = cv2.createCLAHE ( clipLimit = 2.0, tileGridSize = (8, 8) )

Trans\_data = Clahe.apply ( Image\_data )

1. 区域处理：对滤波器区域大小进行操作，有区域概念

自定义滤波器：以拉普拉斯算子滤波器为例，拉普拉斯滤波器有两种形式，其中默认的为[ [0, 1, 0], [1, -4, 1], [0, 1, 0] ] (凸显 (扩张边缘) 外部梯度，4领域) 第二种：[[1, 1, 1], [1, -8, 1], [1, 1, 1]] ( 8领域 )

也可以是 [ [0, -1, 0], [-1, 4, -1], [0, -1, 0] ] 以及 [[-1, -1, -1], [-1, 8, -1], [-1, -1, -1]] 此时边缘没有改变

Kernel = np.array( [ [1, 1, 1], [1, -8, 1], [1, 1, 1] ] ) # 指定卷积核

Image\_data = cv2.imread( “image” )

Trans\_data = cv2.filter2D (Image\_data, dtype = cv2.CV\_8U, kernel = Kernel ) # 类似神经网络中卷积操作

1. 平滑( 也叫非锐化 )滤波器（模糊）：去除噪声

滤波器的不同在于滤波器（卷积核）的不同或滤波器内操作不同

1. 均值滤波：取滤波器区域内的每个像素值的均值作为当前点的像素值

Ksize = (3, 3)时，kernel = np.array( [ [ 1/9, 1/9, 1/9 ], [ 1/9, 1/9, 1/9 ], [ 1/9, 1/9, 1/9 ] ])

Trans\_data = cv2.blur ( Image\_data, ksize=( 3, 3 ) ) # 均值滤波ksize为元组

2. 中值滤波：取滤波器区域内的像素值的中间值作为当前点的像素值

中值滤波广泛用于去除椒盐噪声，效果很好

Trans\_data = cv2.medianBlur ( Image\_data, ksize = 3 ) # 中值滤波只需指定ksize = 3,并非元组

3. 高斯滤波：取高斯分布系数作为权重，对滤波器区域内像素进行卷积操作

Trans\_data = cv2.GaussianBlur ( Image\_data, (3, 3), 1 )

Ksize = (3, 3) 指定滤波器大小，为一个元组 第三个参数：高斯分布的方差值sigma = 1

高斯滤波以后具体描述

2. 锐化滤波操作：凸显边界与轮廓信息

1. 拉普拉斯算子滤波：滤波器参数在上面已叙述过，噪声也会被放大

Trans\_data = cv2.Laplacian( Image\_data, dtype = cv2.CV\_8U, ksize = 3 )

2. 索贝尔sobel滤波：相对来说较好

同时求x、y双方向的梯度：（该方法并不同于x方向 + y方向）

Trans\_data = cv2.Sobel( Image\_data, dtype = cv2.CV\_8U, dx = 1, dy = 1, ksize = 3 )

dx参数表示x方向移动的步长 dy参数表示y方向移动的步长

求x方向上的梯度:

Trans\_data\_x = cv2.Sobel( Image\_data, dtype = cv2.CV\_8U, dx = 1, dy = 0, ksize = 3 )

求y方向上的梯度：

Trans\_data\_y = cv2.Sobel( Image\_data, dtype = cv2.CV\_8U, dx = 0, dy = 1, ksize = 3 )

求图像的梯度时需将x方向梯度与y方向梯度按各自0.5比例相加

Trans\_data = cv2.addWeighted ( Trans\_data\_x, 0.5, Trans\_data\_y , 0.5, 0, dtype=cv2.CV\_8U)

3. 非锐化遮蔽（mask）：也就是生成一个钝化（原图 – 模糊）的mask 作为钝化模板，再将模板按一定比例与原图相加

1. k = 1时为非锐化遮蔽：此时采用的模糊算法主要为高斯滤波与均值滤波

Smooth\_image = cv2.GaussianBlur ( Image\_data, (3, 3), 1 )

Template\_image = Image\_data – Smooth\_image

Trans\_data = Image\_data + k( =1 ) \* Template\_image 也可以写为 = 2 \* Image\_data – Smooth\_data

此时生成的是锐化图像

2. k > 1时为高提升滤波