OpenCv

模块

- core模块实现了最核心的数据结构以及基本运算，如绘图函数、数组操作相关函数等
- highgui模块实现了视频与图像的读取、显示、存储等接口
- imgproc模块实现了图像处理的基本方法，包括图像滤波、图像的几何变换、平滑、阀值分割等
- features2d模块用于提取图像特征以及特征匹配，nonfree模块实现了一些专利算法，如sift特征
- objdetect模块实现了一些莫标检测的功能，经典的基于Haar、LBP特征的人脸检测，基于HOG的行人的等
- stitching模块实现了图像凭借功能
- FLANN模块包含口语爱苏近似最近邻搜索FLANN和聚类Clustering算法
- ml模块机器学习模块(SVM,决策树，Boosting等等)
- photo模块包含图像修复和图像去嘈两部分
- video模块针对视频处理，如背景分割、前景检测、对象跟踪等
- calib3d模块即Calibration(校准)3D,这个模块主要是相机校准和三维重建的内容
- G-API模块包含超高效的图像处理pipeline引擎

OpenCV API

图像视频的加载

创建窗口和显示窗口

- nameWindow()
- imshow()显示窗口
- destroyAllwindows()销毁窗口
- resizeWindow()改变窗口大小
- waitKey()

```
import cv2 as cv

# cv.namedWindow(window_name, cv.WINDOW_AUTOSIZE)
# WINDOW_NORMAL可以让窗口大小变得可以调节

cv.namedWINDOW('new', cv.WINDOW_NORMAL)
```

```
# cv.resizeWindow(window_name, width, height)
cv.resizeWindow('new', 800, 600)

# cv.imshow(window_name, mat)
cv.imshow('new', 0)

# cv.waitKey()
key = cv.waitKey()
if key == 'q': # key & 0xFF == 'q' ASCII是八位的，因此取出后八位来比较
    cv.destroyAllWindows()
# cv.destroyAllWindows()
```

- 注意：
  - **cv.WINDOW_AUTOSIZE不允许使用cv.resizeWindow修改窗口大小**
  - **当窗口退出时，会卡顿，使用cv.waitKey来防止卡顿**
  - cv.waitKey(delay)
    - 设置为0,就是等待键盘按键，设置为其他整数，就是等待时间

加载显示图片

- imread(path, flag)

```
import cv2 as cv
import matplotlib.pyplot as plt

img = cv.imread('path')
# plt.imshow(img)
cv.imshow('pircture', img)
```

- 注意：
  - **OpenCV读取图片的通道不是RBG,而是BGR通道，因此，如果使用plt.imshow,则颜色与真实的有差异， 因此使用cv.imshow**

保存图片

- imwrite(filename, img)

```
cv.nameWindow('img', cv.WINDOW_NORMAL)
cv.resizeWindow('img', 320, 240)

img = cv.imread('path')
```

```
# 利用while优化退出

while True:
    cv.imshow('img', img)
    key = cv.waitKey(0)
    if key & 0xFF == 'q':
        break
    elif key & 0xFF == 's':
        cv.imwrite('filename', img)
    else :
        print(key)
```

视频采集

> *视频其实也是图片，只不过是一组连续不断的图片快速切换。视频的一帧，就是一幅图片*

- VideoCapture
  - 可以捕获图片或者摄像头，用数字来代表不同的设备，比如0,1
  - **如果打开摄像头失败，不会报错**

```
# 打开视频文件
vc = cv.VideoCapture('path')

# 打开摄像头
vc = cv.VideoCapture(0)
```

- read
  - 获取到对象的每一帧数据
  - **会返回bool以及这一帧的数据,True就是读到了数据，Flase就是没有读到**
  - 此处的True,可以用**isOpened**来代替

```
while vc.isOpened() # True:
    ret, frame = vc.read()

    # 根据ret做判断
    if not ret:
        break
    # 显示数据
    cv.imshow(window_name, frame)

    key = cv.waitKey(0)
    if key & 0xFF == ord('q'):
```

```
        break
cap.release()
cv.destroyAllWindows()
```

- 注意：
  - **当你调用VideoCapture的时候，会占用系统资源，因此，需要在最后使用release释放掉占用的资源**
  - **同理，开启窗口namedWindow也会占用资源，因此也要destroyAllWindows**
  - **此时的waitKey不能设置为0,不然只能获取到第一帧的数据**

视频录制

- VideoWriter
  - 参数一为输出文件，参数二为多媒体文件格式(VideoWriter_fourcc)，参数三为帧率，参数四为分辨率
- wirte
  - 编码并写入缓存
- release
  - 缓存内容写入磁盘，且释放资源

```
cap = cv2.VideoCapture(0)
fourcc = cv.VideoWriter_fourcc(*'mp4v')
# *'mp4v'解包操作，等同于'm' 'p' '4' 'v'

vm = cv.VideoWriter('output.mp4', fourcc, 20, (640, 480))

while cap.isOpende():
    ret, frame = cap.imread()
    if not ret:
        break
    vm.write(frame)
    cv.imshow('frame', frame)
    if cv.waitKay(1) == ord('q'):
        break;
cap.release()

vm.release()
cv.destroyAllWindows()
```

- 注意：
  - **对于VideoWirter的分辨率，一般就用(640, 480)**
  - write操作实际上**并没有把数据写入硬盘，而是写入缓冲区，release操作才是真正的写入到磁盘上**

- 视频格式
  - mp4v就是MP4
  - XVID就是AVI

控制鼠标

- setMouseCallBack(winname, callback, userdata)
  - winname就是窗口的名字
  - callback是回调函数
  - userdata是回调函数的参数
- callback(event, x, y, flags, userdata)
  - event是鼠标的事件
  - x,y是鼠标的坐标点
  - flags主要用于组合键
  - userdata就是setMouseCallBack

event

鼠标事件	代码	含义
EVENT_MOUSEMOVE	0	鼠标移动
EVENT_LBUTTONDOWN	1	按下鼠标左键
EVENT_RBUTTONDOWN	2	按下鼠标右键
EVENT_MBUTTONDOWN	3	按下鼠标中键
EVENT_LBUTTONUP	4	左键释放
EVENT_RBUTTONUP	5	右键释放
EVENT_MBUTTONUP	6	中键释放
EVENT_LBUTTONDBLCLK	7	左键双击
EVENT_RBUTTONDBLCLK	8	右键双击
EVENT_MBUTTONDBLCLK	9	中键双击
EVENT_MOUSEWHEEL	10	鼠标滚轮上下滚动
EVNETN_MOUSEHWHEEL	11	鼠标左右滚动

flags

flags	代码	含义
EVENT_FLAG_LBUTTON	1	按下左键
EVENT_FLAG_RBUTTON	2	按下右键

flags	代码	含义
EVENT_FLAG_MBUTTON	4	按下中键
EVENT_FLAG_CRTLKEY	8	按下ctrl
EVENT_FLAG_SHIFTKEY	16	按下shift
EVENT_FLAG_ALTKEY	32	按下alt

```
def mouse_callback(event, x, y, flags, userdata):
    print(event, x, y, flags, userdata)

cv.namedWindow('mouse', cv.WINDOW_NORMAL)
cv.resizeWindow('mouse', 640, 360)

# 设置鼠标回调函数
cv.setMouseCallBack('mouse', mouse_callback, '123')

img = np.zeros((360, 640, 3), np.uint8)

while True:
    cv.imshow('mouse', img)
    key = cv.waitKey(1)
    if key & 0xFF == ord('q'):
        break

cv.destroyAllWindow()
```

- 注意：
  - numpy的是行和列，而resizeWindow是宽和高，**是刚好相反的，并且，np还要指出是三维(因为BGR三个通道)，并且注意后面的np.uint8**
  - **使用回调函数，可以在内部定义鼠标事件所代表的意义**

TRACKBAR控件

- createTrackbar(trackbarname, winname, value, count, onChange)
  - value为trackbar的默认值
  - count为bar的最大值，最小为0
  - onChange是一个回调函数，**需要自己定义**
- getTrackbarPos(trackbarname, winname)
  - 获取TrackBar当前值

```
cv.namedWindow('trackbar', cv.WINDOW_NORMAL)
cv.resizeWindow('trackbar', 640, 480)
```

```python
def callback(value):
    print(value)

cv.createTrackbar('R', 'trackbar', 0, 255, callback)
cv.createTrackbar('G', 'trackbar', 0, 255, callback)
cv.createTrackbar('B', 'trackbar', 0, 255, callback)

img = np.zeros((480, 640, 3), np.uint8)

while True:
    # 获取trackbar的当前值
    r = getTrackbarPos('R', 'trackbar')
    b = getTrackbarPos('B', 'trackbar')
    g = getTrackbarPos('G', 'trackbar')

    # 改变背景图颜色
    img[:] = [b, g, r]
    cv.imshow('trackbar', img)

    key = cv.waitKey(0)
    if key == ord('q'):
        break

cv.destroyAllWindows()
```

- 注意：
  - **当waitKey的参数设置为0的时候，不会显示颜色！！！也就是说，不会跟随trackbar的改变而改变**

OpenCV的基础知识和绘制图形

色彩空间

RBG和BGR

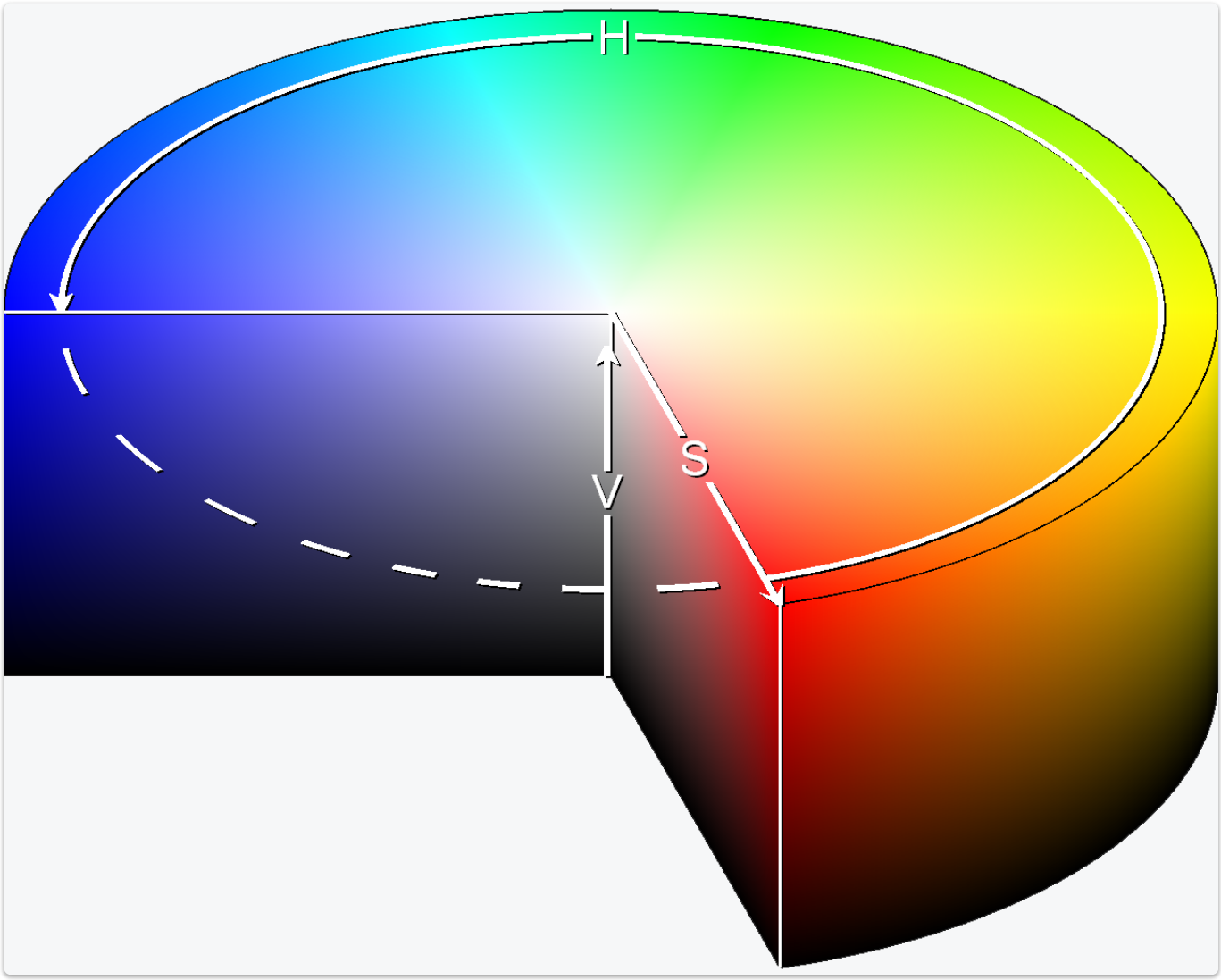
> *最常见的色彩空间就是RBG,人眼也是基于RBG的色彩空间去区分的*

- 但是，**OpenCV默认使用的是BGR, BGR和RBG区别在于图片在色彩通道上的排列顺序不同**

HSV

- OpenCV**用的最多的色彩空间是HSV**
  - H：Hue,色相，即色彩。用角度度量，**取值范围为0～360, 从红色开始逆时针计算，红色为0, 绿色为120, 蓝色为240**

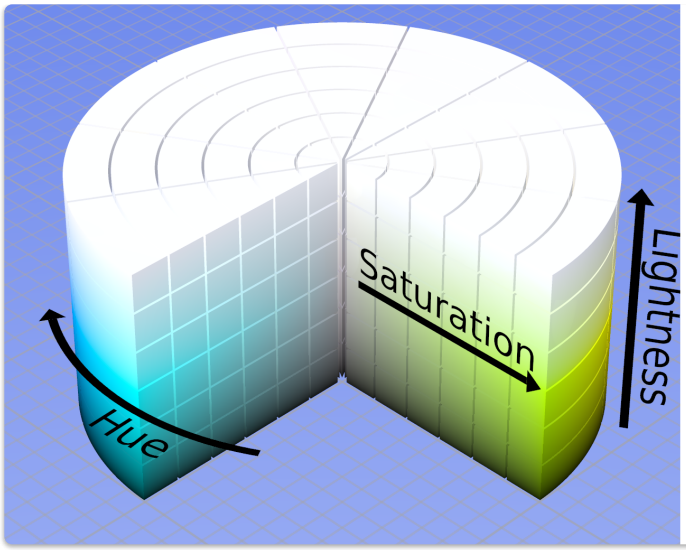
- S：Saturation，饱和度，表示**颜色接近光谱色的程度**。一种颜色也可以看成是光谱色和白色混合的结果。**光谱色的占比越大，颜色就越接近光谱色，饱和度也就越高**。饱和度越高，颜色就深而艳。通常取值0%～100%，值越大，越饱和
- V：Value,明度。**明度表示颜色明亮的程度**。对于光源色，明度值与发光体的光亮度有关;对于物体色，此值和物体的透射比或反射比有关。**通常取值范围0%(黑)～100%(白)**。

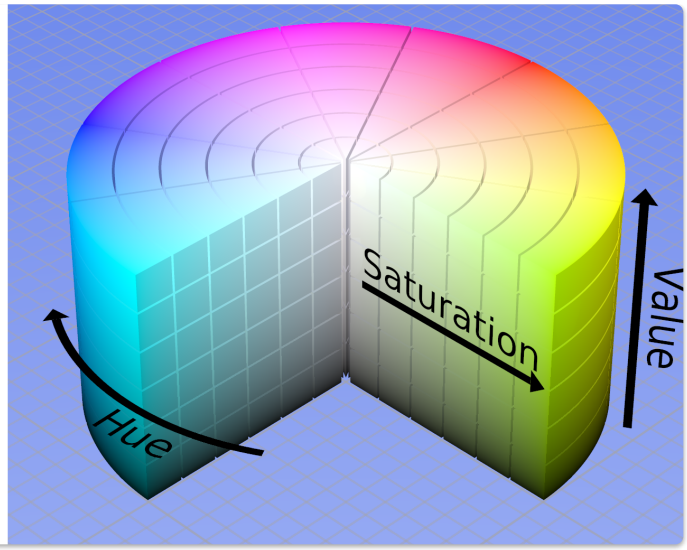


HSL

> *HSL和HSV相差不多*

- H：Hue,色相
- S：Stauration,饱和度
- L：Lightness,亮度





- **HSL在顶部是纯白的，不管是什么颜色**
- HSV和HSL的区别

  - 他们两个在字面以上是都是一样的
  - 但是，在原理和表现上，他们的H(色相)完全一致，S(饱和度)不一样，L和V(B)也不一样
    - HSV(B)中的S**控制纯色中混入白色的量**，值越大，**白色越少**，颜色越纯
    - HSV(B)中的V(B)**控制纯色中混入黑色的量**，值越大，黑色**越少**，亮度越高
    - HSL中的**S和黑白没有关系**，饱和度**不控制颜色中混入黑白的多寡**
    - HSL中的L**控制纯色中混入的黑白两种颜色**

YUV

> *YUV是一种**颜色编码方法**，常使用在各个视频处理组件上，YUV在对照片或视频编码时，**允许降低色度的带宽***

- Y, 表示**明亮度**(Luminance或者Luma)，**也就是灰度值**
- U和V表示的是色度(Chrominance或者Chroma)，**作用是描述影像色彩及饱和度，用于指定像素的颜色**

> *YUV的最大优点在于**只需要占用极少的带宽***

比值	说明
4：4：4	完全取样
4：2：2	2：1的水平取样，垂直完全采样
4：2：0	2：1的水平采样，垂直2：1采样
4：1：1	4：1的水平采样，垂直完全采样

- 4：2：0的比例是最经常使用的

色彩空间的转换

- cvtColor(img, colorspace)

```
def callback(value):
    pass

cv.namedWindow('color', cv.WINDOW_NORMAL)
cv.resizeWindow('color', 640, 480)

img = cv.imread('path')

# 常见的色彩空间转换
colorspace = [cv.COLOR_BGR2RGBA, cv.COLOR_BGR2GRAY, cv.COLOR_BGR2HSV, cv.COLOR_I

cv.createTrackbar('curcolor', 'color', 0, 4, callback)

while True:
    index = cv.getTrackbarPos('curcolor', 'color')

    # 色彩空间转换
    cvt_img = cv.cvtColor(img, colorspace[index])
    cv.imshow('color', cvt_img)
    key = cv.waitKey(10)
    if key == ord('q'):
        break

cv.destroyAllWindows()
```

MAT

> *MAT是OpenCV在C++语言中用来表示图像数据的一种数据结构，在python中转化为numpy中的ndarray*

mat属性

字段	说明	字段	说明
dims	维度	channels	通道数RGB是3
rows	行数	size	矩阵大小
cols	列数	type	dep+dt+chs CV_8UC3
depth	像素的深度	data	存放数据

mat的深浅拷贝

- view
  - 浅拷贝
- copy
  - 深拷贝

```
img = cv.imread('path')

# 浅拷贝
img1 = img.view()

# 深拷贝
img2 = img.copy()
```

通道 的分离和合并

- split(mat)
  - 分割图像的通道
- merge((ch1, ch2, ch3))
  - 融合多个通道

```
img = np.zeros((480, 640, 3), np.uint8)

# 分割通道
b, g, r = cv.split(img)

# 修改
b[10:100, 10:100] = 255
g[10:100, 10:100] = 255

# 合并通道
img2 = cv.merge((b, g, r))
cv.imshow('img', np.hstack((b, g)))
cv.imshow('img', np.hstack((img, img2)))
```

- 注意：
  - **通道一定是BGR**

绘制图形

- line(img, pt1, pt2, color, thickness, lineType, shift) 画直线
  - img：在哪个图像上画线
  - pt1， pt2：开始点，结束点，指定线的开始与结束位置
  - color：颜色

    - thickness：线宽
    - lineType：线形，线形为-1, 4, 8, 16,**默认为8**，实际上就是锯齿，越高看上去越平滑
    - shift：坐标缩放比例
- rectangle(img, pt1, pt2, color, thickness, lineType, shift) 画矩形
- cicle(img, center, radius, color, [, thickness[, lineType[, shift]]])
    - 画圆，括号内表示可选参数
- ellipse(img, 中心点，长宽的一半，角度，从哪个角度开始，从哪个角度结束)
    - 画椭圆

```
# 创建纯黑的背景
img = np.zeros((480, 640, 3), np.uint8)

cv.line(img, (10, 20), (300, 400), (0, 0, 255), 5, 4)
cv.line(img, (80, 1000), (380, 480), (0, 0, 255), 5, 16)

cv.rectangle(img, (10, 20), (300, 400), (0, 255, 0), 5)

cv.circle(img, (50, 60), 10, (255, 0, 0))

cv.ellipse(img, (320, 240), (100, 50), 0, 0, 360, [0, 0, 255])
```

- 注意：
    - 给定的点，**需要用元组**
    - 给定的颜色，**需要用元组或者列表**
    - lineType越高，看上去的图形就越平滑

绘制多边形

- polylines(img, pts, isClosed, color[, thickness[, lineType[, shift]]])
    - 画多边形
- fillPoly
    - 填充多边形，参数同上，**只少了isClosed,因为一定闭合**

```
img = np.zeros((480, 640, 3), np.uint8)

pts = np.array([(250, 100), (150, 100), 450, 100], np.int32)

cv.polylines(img, [pts], True, (0, 0, 255), 5)

cv.fillPoly(img, [pts], (0, 0, 255), 5)
```

- 注意：

    - 多边形必须是**np.int32位及以上的**
    - **pts必须是三维**

绘制文本

- putText(img, text, org, fontFace, fontScale, color[, thickness[, lineType[, bottomLeftOrigin]]])
    - 绘制文本
        - text：要绘制的文本
        - org：文本在图片的左下角坐标
        - fontFace：字体
        - fontScale：字体大小

```
cv.putText(img, "Hello OpenCV", (200, 200), cv.FONT_HERSHEY_SIMPLEX, 2, (0, 0, 1
```

- 注意：
    - **OpenCV没有中文字体，因此写入中文文本会乱码**
    - 但是可以使用Pillow包来解决乱码

```
from PIL import ImageFont, ImageDraw, Image

img = np.fill((200, 200, 3), fill_value = 255, dtype = np.uint8)

# 导入字体文件
font = ImageFont.truetype('font_path', fontsize)

# 创建pillow图片
img_pil = Iamge.fromarray(img)

draw = ImageDraw.Draw(img_pil)

# 利用draw去绘制中文
draw.text((10, 150), '你好', font = font, fill = (0, 255, 0, 0))

# 重新变回ndarray
img = np.array(img_pil)

cv.imshow('img', img)
```

图像的算术与位运算

图像的算术运算

图像的加法运算

- add
  - opencv使用add来进行执行图像的加法运算
  - **值得注意的是，图片是矩阵，因此两个执行add操作的图片的shape一定要一样，通道数也要相同**
  - add的规则就是两个图的对应位置的元素相加，**如果超过255,那么规定为255**
  - 图片也能和单个数字进行运算，每个数字和100进行加法运算，**超出255的数字，会被截断，相当于%255**

```
img1 = cv.imread('path1')
img2 = cv.imread('path2')

printf(img1.shape)
printf(img2.shape)

# 如果两张图片shape，不一致，则需要处理，假如img1(640, 480, 3), img2(480, 300, 3)
# 那么使img1变小
new_img1 = img1[0:480, 0:300]

new_img = cv.add(new_img1, img2)
```

图像的减法运算

- substract
  - 同上add一样
  - **但，如果相减位置小于0, 则规定为0**

```
img1 = cv.imread('path1')
img2 = cv.imread('path2')

printf(img1.shape)
printf(img2.shape)

# 如果两张图片shape，不一致，则需要处理，假如img1(640, 480, 3), img2(480, 300, 3)
# 那么使img1变小
new_img1 = img1[0:480, 0:300]

new_img = cv.substract(new_img1, img2)
```

图像的乘法运算

- multiply
  - 同上add一致

```
img1 = cv.imread('path1')
img2 = cv.imread('path2')

printf(img1.shape)
printf(img2.shape)

# 如果两张图片shape，不一致，则需要处理，假如img1(640, 480, 3), img2(480, 300, 3)
# 那么使img1变小
new_img1 = img1[0:480, 0:300]

new_img = cv.multiply(new_img1, img2)
```

图像的除法运算

- divide
  - 同上substract一致

```
img1 = cv.imread('path1')
img2 = cv.imread('path2')

printf(img1.shape)
printf(img2.shape)

# 如果两张图片shape，不一致，则需要处理，假如img1(640, 480, 3), img2(480, 300, 3)
# 那么使img1变小
new_img1 = img1[0:480, 0:300]

new_img = cv.divide(new_img1, img2)
```

图像的融合

> *不是简单的加法，而是相当于对图片做线性运算：new_img = img1 w1 + img2 w2 + bias*

- addWeighted(src1, alpha, src2, beta, gamma)
  - alpha：是第一个权重参数
  - beta：是第二个权重参数
  - gamma：是偏置
  - **但是，图片的运算始终是要求图片shape和维数一致的**

```
img1 = cv.imread('path1')
img2 = cv.imread('path2')

printf(img1.shape)
printf(img2.shape)

# 如果两张图片shape，不一致，则需要处理，假如img1(640, 480, 3), img2(480, 300, 3)
# 那么使img1变小
new_img1 = img1[0:480, 0:300]

res = cv.addWeighted(new_img1, 0.4, img2, 0.6, 0)
```

逻辑运算

> *对应位置进行与或非，异或等逻辑运算*

- 注意：
  - opencv中的非，**0反过来是255,255反过来是0**
  - bitwise_not
    - 非操作
  - bitwise_and
    - 与操作
  - bitwise_or
    - 或操作
  - bitwise_xor
    - 异或操作

```
img1 = cv.imread('path1')
img2 = cv.imread('path2')

printf(img1.shape)
printf(img2.shape)

# 如果两张图片shape，不一致，则需要处理，假如img1(640, 480, 3), img2(480, 300, 3)
# 那么使img1变小
new_img1 = img1[0:480, 0:300]

ref1 = cv.bitwise_not(nwe_img1, img2)
ref2 = cv.bitwise_and(nwe_img1, img2)
ref3 = cv.bitwise_ro(nwe_img1, img2)
ref4 = cv.bitwise_xor(nwe_img1, img2)
```

图形的基本变换

图像的放大与缩小

- resize(src, dsize[, dst[, fx[, fy[, interpolation]]]])
  - src：需要缩放的图片
  - dsize：缩放之后的图片大小，**元组和列表表示均可**
  - dst：可选参数，缩放之后的输出图片
  - fx, fy：x轴和y轴的缩放比，即宽度与高度的缩放比
  - imterpolation：插值算法：
    - INTER_NEAREST,邻近插值，速度快，效果差
    - INTER_LINEAR.双线性插值，使用原图的4个点进行插值，**默认**
    - INTER_CUBIC,三次插值，原图中的16个点
    - INTER_AREA,区域插值，效果最好，计算时间最长

```
img = cv.imread('path')

new_img = cv.resize(img, (640, 480))
```

- 注意：
  - **opencv的图片大小是先宽后高**

图像的翻转

- flip(src, flipCode[, dst])
  - flipCode：控制反转。**code=0, 表示上下翻转，code>0,表示左右翻转,code<0,表示上下+左右翻转**

```
cv.flipCode(img, filpCode = 0)
```

图像的旋转

- rotate(img, rotateCode)
  - ROTATE_90_CLOCKWISE 90度顺时针
  - ROTATE_180 180度
  - ROTATE_90_COUNTERCLOCKWISE 90度逆时针

```
cv.rotate(img, cv.ROTATE_90_CLOCKWISE)
```

仿射变换

> *仿射变换是图像转换，缩放，平移的总称。具体的做法是通过一个矩阵和原图片坐标进行计算，得到新的坐标，完成变换，**所以关键在于矩阵***

图像平移

- warpAffine(src, M, dsize, flags, mode, value)
  - M：变换矩阵
  - dsize：输出图片大小
  - flag：与resize的插值算法一样
  - mode：边界外推法标志
  - value：填充边界值
- 平移矩阵
  - 矩阵中的每个像素由(x, y)组成，(x, y)表示这个像素的坐标。假设沿x轴平移t_x，沿y轴平移t_y，最后得到的坐标为$(\hat{x}, \hat{y}) = (x + t_x, y + t_y)$,用矩阵表示：

$$\begin{bmatrix} \hat{x} \\ \hat{y} \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

```
img = cv.imread('path')

M = np.float([[1, 0, 200], [0, 1, 0]])
h, w, ch = img.shape

cv.warpAffine(img, M, dsize=(w, h))
```

- 注意：
  - **M变换矩阵要求最低是float32位的**
  - **M变换矩阵实际上就是二维，[[1, 0, tx], [0, 1, ty]]特定格式**

获取变换矩阵

- getRotationMatrix2D(center, angle, scale)
  - center：中心点，以图片的哪个点作为旋转时的中心点
  - angle：角度，旋转的角度，**按照逆时针旋转**
  - scale：缩放比例，想把图片进行什么样的缩放

```
img = cv.imread('path')

h, w, ch = img.shape
```

```
M = cv.getRotationMatrix2D((w/2, h/2), 15, 1.0)

new_img = warpAffine(img, M, dsize=(w, h))
```

- getAffineTransform(src[], dst[])
  - 通过三点可以确定变换后的位置，相当于解方程。三个点对应三个方程，能解出偏置的参数和旋转的角度
  - src：原目标的三个点
  - dst：变换后的三个点

```
img = cv.imread('path')

h, w, ch = img.shape

src = np.float32([[200, 100], [300, 100], [200, 300]])
dst = np.float32([[100, 150], [360, 200], [280, 120]])

M = cv.getAffineTransform(src, dst)

new_img = warpAffine(img, M, dsize=(w, h))
```

透视变换

> *透视变换就是将一种坐标系变成另一种坐标系，简单来说就是可以把“斜”变“正”*

- warpPerspective(img, M, dsize, ...)
  - **透视变换的变换矩阵一定是3*3的矩阵**
  - getPerspectiveTransform(src, dst)
    - 需要四个点，**即图片的四个角**

```
img = cv.imread('path')

src = np.float32([[100, 1100], [2100, 1100], [100, 4000], [2100, 3900]])
dst = np.float32([[0, 0], [2300, 0], [0, 3000], [2300, 3000]])

M = cv.getPerspectiveTraqnsform(src, dst)

cv.warpPerspective(img, M, (2300, 3000))
```

- 注意：
  - 四个角的顺序分别是**上左、上右、下左、下右**