# Python-OpenCv之图像基本处理-平移，缩放，旋转，翻转，裁剪及仿射变换

### 基本图像处理

#### 1. 缩放[scale](https://so.csdn.net/so/search?q=scale&spm=1001.2101.3001.7020)

缩放通过cv2.resize()实现

**函数说明**：

cv2.resize(src, dsize[, dst[, fx[, fy[, interpolation]]]]) -> dst

**参数说明**：

* src - 原图
* dst - 目标图像。当参数dsize不为0时，dst的大小为size；否则，它的大小需要根据src的大小，参数fx和fy决定。dst的类型（type）和src图像相同
* dsize - 目标图像大小。当dsize为0时，它可以通过以下公式计算得出：  
  img

所以，参数dsize和参数(fx, fy)不能够同时为0

* fx - 水平轴上的比例因子。当它为0时，计算公式如下：  
  img
* fy - 垂直轴上的比例因子。当它为0时，计算公式如下：  
  img
* interpolation - 插值方法。共有5种：  
  INTER\_NEAREST - 最近邻插值法  
  INTER\_LINEAR - 双线性插值法（默认）  
  INTER\_AREA - 基于局部像素的重采样（resampling using pixel area relation）。对于图像抽取也叫缩小图像（image decimation）来说，这可能是一个更好的方法。但如果是放大图像时，它和最近邻法的效果类似。  
  INTER\_CUBIC - 基于4x4像素邻域的3次插值法  
  INTER\_LANCZOS4 - 基于8x8像素邻域的Lanczos插值

缩小图像 用INTER\_AREA更好，放大图像用 INTER\_CUBIC更好；

**代码示范**：

import cv2

*# 读取一张照片*

img = cv2.imread('tiger\_tibet\_village.jpg')

*# 缩放成200x200的图像*

img\_200x200 = cv2.resize(img, (200, 200))

*# 不直接指定缩放后大小，通过fx和fy指定缩放比例，0.5则长宽都为原来一半*

*# 等效于img\_200x300 = cv2.resize(img, (300, 200))，注意指定大小的格式是(宽度,高度)*

*# 插值方法默认是cv2.INTER\_LINEAR，这里指定为最近邻插值*

img\_200x300 = cv2.resize(img, (0, 0), fx=0.5, fy=0.5,

interpolation=cv2.INTER\_NEAREST)

cv2.imwrite('resized\_200x200.jpg', img\_200x200)

cv2.imwrite('resized\_200x300.jpg', img\_200x300)

#### 2. 旋转rotate

[opencv](https://so.csdn.net/so/search?q=opencv&spm=1001.2101.3001.7020)中对图像的旋转主要是先通过**getRotationMatrix2D**函数得到图像的旋转矩阵，然后再通过仿射变换函数**warpAffine**得到旋转后的图像。

**函数说明**：

cv2.getRotationMatrix2D(center, angle, scale)

cv2.warpAffine(src, M, dsize[, dst[, flags[, borderMode[, borderValue]]]]) → dst

**参数说明**：

getRotationMatrix2D:

* center–表示旋转的中心点
* angle–表示旋转的角度degrees
* scale–图像缩放因子

warpAffine:

* src – 输入的图像
* M – 2 X 3 的变换矩阵.
* dsize – 输出的图像的size大小
* dst – 输出的图像
* flags – 输出图像的插值方法
* borderMode – 图像边界的处理方式
* borderValue – 当图像边界处理方式为BORDER\_CONSTANT 时的填充值

**代码示范**：

img = cv2.imread('messi5.jpg',0)

rows,cols = img.shape

*#90度旋转*

M = cv2.getRotationMatrix2D((cols/2,rows/2),90,1)

dst = cv2.warpAffine(img,M,(cols,rows))

**图示**：  


#### 3. 裁剪crop

裁剪是利用array自身的下标截取实现

**代码示范**：

img = cv2.imread('rotation.jpg')

print img.shape

*#裁剪*

crop\_img = img[20:100, 20:160]

cv2.imwrite('crop\_img.jpg', crop\_img)

**图示**：

原图：  


裁剪后图像：



#### 4. 填充pad

填充通过函数copyMakeBorder实现：

**函数说明**：

cv2.copyMakeBorder(src, top, bottom, left, right, borderType[, dst[, value]]) → dst

**参数说明**：

* src – 输入的图像
* dst – 输出的图像
* top，bottom，left，right – 分别表示在原图四周扩充边缘的像素值
* borderType –图像边界的处理方式

常见的borderType：

* + BORDER\_REPLICATE ：复制法，复制最边缘像素
  + BORDER\_REFLECT\_101：对称法，以最边缘像素为轴，对称
  + BORDER\_CONSTANT：常量法，以一个常量参数值（自定参数value给定）填充扩充的边界
* value – 当图像边界处理方式为BORDER\_CONSTANT 时的填充值

**代码示范**：

*# 在上张图片的基础上，上下各填充50像素，填充值为128，生成新的的图像*

pad\_img = cv2.copyMakeBorder(crop\_img, 50, 50, 0, 0, cv2.BORDER\_CONSTANT, value=(128, 128, 128))

cv2.imwrite('pad\_img.jpg', pad\_img)

**图示**：

原图：



填充后图像：



#### 5. 平移translate

平移通过自定义平移矩阵以及函数**warpAffine**实现：

**代码示范**：

img = cv2.imread('messi5.jpg',0)

rows,cols = img.shape

*# 平移矩阵M：[[1,0,x],[0,1,y]]*

M = np.float32([[1,0,100],[0,1,50]])

dst = cv2.warpAffine(img,M,(cols,rows))

**图示**：



#### 6. 翻转flip

翻转通过函数flip实现：

**函数说明**：

cv2.flip(src, flipCode[, dst]) → dst

**参数说明**：

* src – 输入的图像
* dst – 输出的图像
* flipCode – 翻转模式，flipCode==0垂直翻转（沿X轴翻转），flipCode>0水平翻转（沿Y轴翻转），flipCode<0水平垂直翻转（先沿X轴翻转，再沿Y轴翻转，等价于旋转180°）

**代码示范**：

*# 水平翻转*

flip\_horiz\_img = cv2.flip(pad\_img, 1)

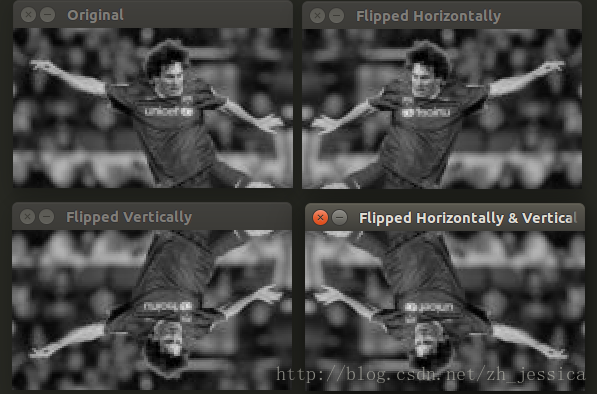
*# 垂直翻转*

flip\_verti\_img = cv2.flip(pad\_img, 0)

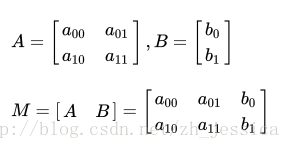
*# 水平垂直翻转*

flip\_horandver\_img = cv2.flip(pad\_img, -1)

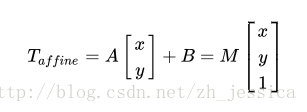
**图示**：



### 图像的仿射变换Affine transformation

图像的仿射变换涉及到图像的形状位置角度的变化，是深度学习预处理中常到的功能，在此简单回顾一下。仿射变换具体到图像中的应用，主要是对图像的**缩放scale**，**旋转rotate**，**剪切shear**，**翻转flip**和**平移translate**的组合。在OpenCV中，仿射变换的矩阵是一个2×3的矩阵，其中左边的2×2子矩阵是线性变换矩阵，右边的2×1的两项是平移项：  


对于图像上的任一位置(x,y)，仿射变换执行的是如下的操作：



需要注意的是，对于图像而言，宽度方向是x，高度方向是y，坐标的顺序和图像像素对应下标一致。所以原点的位置不是左下角而是右上角，y的方向也不是向上，而是向下。在OpenCV中实现仿射变换是通过仿射变换矩阵和cv2.warpAffine()函数的。

仿射变换矩阵：

1. 缩放scale：在x轴方向扩大 x倍，y轴方向扩大 y倍

M = [ [x, 0, 0],

​ [0, y, 0] ]

1. 旋转rotate：顺时针旋转x度

M = [ [cosx, -sinx, 0],

​ [sinx, cosx, 0] ]

1. 剪切shear

M = [ [1, shx, 0],

​ [shy, 1, 0] ]

1. 平移translate：向x方向移动 x, y方向移动 y

M = [ [1, 0, x],

​ [0, 1, y] ]

相关参考：<http://www.cnblogs.com/ghj1976/p/5199086.html>

代码：

import cv2

import numpy as np

img = cv2.imread('lanka\_safari.jpg')

*# 沿着横纵轴放大2倍，然后平移(-150,-240)，最后沿原图大小截取，等效于裁剪并放大*

M\_crop\_trans = np.array([

[2, 0, -150],

[0, 2, -240]

], dtype=np.float32)

img\_crop\_trans = cv2.warpAffine(img, M\_crop\_elephant, (400, 600))

cv2.imwrite('img\_crop\_trans.jpg', img\_crop\_trans)

*# x轴的剪切shear变换，角度45°*

theta = 45 \* np.pi / 180

M\_shear = np.array([

[1, np.tan(theta), 0],

[0, 1, 0]

], dtype=np.float32)

img\_sheared = cv2.warpAffine(img, M\_shear, (400, 600))

cv2.imwrite('img\_sheared.jpg', img\_sheared)

*# 顺时针旋转，角度45°*

M\_rotate = np.array([

[np.cos(theta), -np.sin(theta), 0],

[np.sin(theta), np.cos(theta), 0]

], dtype=np.float32)

img\_rotated = cv2.warpAffine(img, M\_rotate, (400, 600))

cv2.imwrite('img\_rotated.jpg', img\_rotated)

*# 某种变换，具体旋转+缩放+旋转组合可以通过SVD分解理解*

M = np.array([

[1, 1.5, -400],

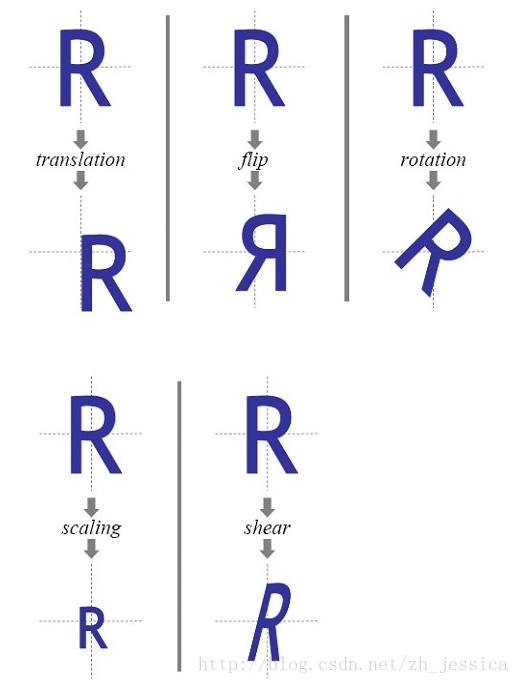
[0.5, 2, -100]

], dtype=np.float32)

img\_transformed = cv2.warpAffine(img, M, (400, 600))

cv2.imwrite('img\_transformed.jpg', img\_transformed)

图示：



## 仿射变换

在仿射变换中，原始图像中的所有平行线在输出图像中依旧平行。为了找到变换矩阵，我们需要从输入图像中得到三个点，以及它们在输出图像中的对应位置。然后cv.getAffineTransform将创建一个2x3矩阵，最后该矩阵将传递给cv.warpAffine。

参考以下示例，并查看选择的点（以绿色标记）：

**img = cv.imread('drawing.png')**

**rows,cols,ch = img.shape**

**pts1 = np.float32([[50,50],[200,50],[50,200]])**

**pts2 = np.float32([[10,100],[200,50],[100,250]])**

**M = cv.getAffineTransform(pts1,pts2)**

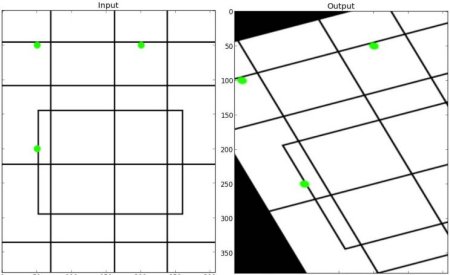
**dst = cv.warpAffine(img,M,(cols,rows))**

**plt.subplot(121),plt.imshow(img),plt.title('Input')**

**plt.subplot(122),plt.imshow(dst),plt.title('Output')**

**plt.show()**

窗口将如下图显示：



## 透视变换

对于透视变换，需要一个3x3变换矩阵。即使在转换之后，直线仍是直线。要找到此变换矩阵，需要在输入图像上找4个点，以及它们在输出图像中的对应位置。在这4个点中，其中任意3个不共线。然后可以通过函数cv.getPerspectiveTransform找到变换矩阵，将cv.warpPerspective应用于此3x3变换矩阵。

代码：

**img = cv.imread('sudoku.png')**

**rows,cols,ch = img.shape**

**pts1 = np.float32([[56,65],[368,52],[28,387],[389,390]])**

**pts2 = np.float32([[0,0],[300,0],[0,300],[300,300]])**

**M = cv.getPerspectiveTransform(pts1,pts2)**

**dst = cv.warpPerspective(img,M,(300,300))**

**plt.subplot(121),plt.imshow(img),plt.title('Input')**

**plt.subplot(122),plt.imshow(dst),plt.title('Output')**

窗口显示：

