# Opencv学习笔记:cv2.resize函数的介绍

## 1.函数介绍

cv2.resize(InputArray src, OutputArray dst, Size, fx, fy, interpolation)

## 2.参数意思

* InputArray src ： 输入图片
* OutputArray dst ：输出图片
* Size ： 输出图片尺寸
* fx, fy：沿x轴，y轴的缩放系数(也就是宽和高)
* interpolation ： 插入方式

## 3.插入方式选择

* INTER\_NEAREST ：最近邻插值  
  最近邻插值法的**优点**：计算量很小，算法也简单，因此运算速度较快。没考虑其他相邻像素点的影响。**缺点**：图像质量损失较大，会产生明显的马赛克和锯齿现象。
* INTER\_LINEAR：双线性插值（默认设置）  
  细节比最近邻插值更好，但是速度没有它快，复杂度更高，因此具有低通滤波器的性质, 从而导致缩放后图像的高频分量受到损失, 图像边缘在一定程度上变得较为模糊。产生图像质量受损与计算精度不高的问题。
* INTER\_AREA：使用像素区域关系进行重采样。
* INTER\_CUBIC：4x4像素邻域的双三次插值  
  细节上比双线性插值法更好，是商业图像中的标准方法。双三次插值考虑的是周围16个像素的像素值，其权重因子的计算比较复杂。
* INTER\_LANCZOS4：8x8像素邻域的Lanczos插值

## 4.实际代码

import cv2

inputpath='D:/pycharm/4kinds\_detectface\_module/1.jpg'

img=cv2.imread(inputpath)

h, w = img.shape[:2]

img1 = cv2.resize(img, None,None,fx=0.25, fy=0.25, interpolation=cv2.INTER\_CUBIC) #

img2=cv2.resize(img,(200,300),interpolation=cv2.INTER\_CUBIC)

print(img.shape)

print(img1.shape)

print(img2.shape)

cv2.imshow('image',img)

cv2.imshow('image1',img1)

cv2.imshow('image2',img2)

cv2.waitKey(0)

cv2.destroyAllWindows()

**运行结果**：  


# python-opencv-cv2.resize()函数详解

## 1.cv2.resize()参数说明？

cv2.resize(**src, dsize, dst=None, fx=None, fy=None, interpolation=None**)

| **参数** | **描述** |
| --- | --- |
| **src** | 【必需】输入原图像 |
| **dsize** | 【必需】输出图像的大小 |
| **fx** | 【可选】width方向的缩放比例 |
| **fy** | 【可选】height方向的缩放比例 |
| **interpolation**（插值） | 【可选】这个是指定插值的方式 |

dsize形参的数组的宽度在前，高度在后（output\_width,output\_height）

图像缩放之后，肯定像素要进行重新计算的，就靠这个参数来指定重新计算像素的方式，有以下几种：  
INTER\_NEAREST - 最邻近[插值](https://so.csdn.net/so/search?q=%E6%8F%92%E5%80%BC&spm=1001.2101.3001.7020)  
INTER\_LINEAR - 双[线性](https://so.csdn.net/so/search?q=%E7%BA%BF%E6%80%A7&spm=1001.2101.3001.7020)插值，如果最后一个参数你不指定，默认使用这种方法  
INTER\_CUBIC - 4x4像素邻域内的双立方插值  
INTER\_LANCZOS4 - 8x8像素邻域内的Lanczos插值

## 2.代码示例

import cv2

img = cv2.imread('图片所在路径')

# 默认使用双线性插值法

img = cv2.resize(img,(300,300))#固定长宽

img = cv2.resize(img,None,fx=0.5,fy=0.5)#固定比例

cv.imshow("img",img)

cv.waitKey(0)

cv.destroyAllWindows()

## 3.最近邻插值与双线性插值

##### 图像处理: 五种 插值法

[Syntax](https://so.csdn.net/so/search?q=Syntax&spm=1001.2101.3001.7020" \t "_blank)

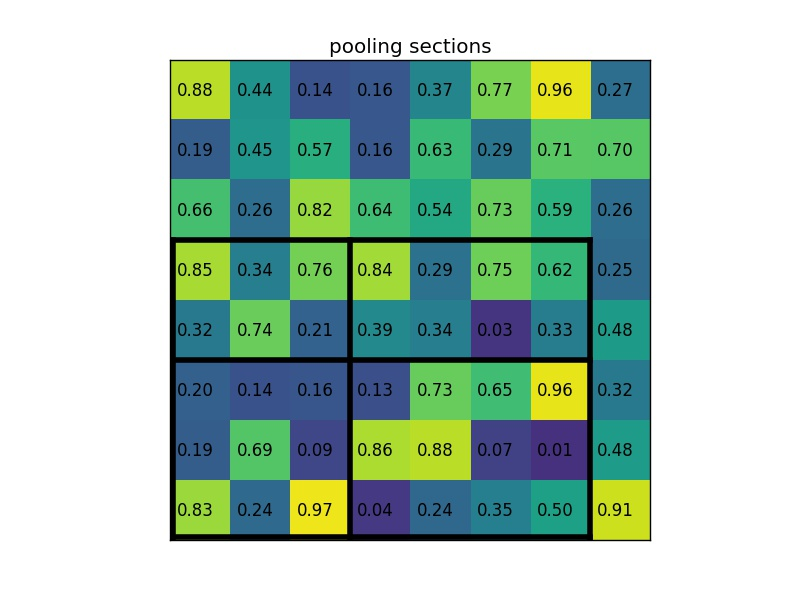
cv2.resize(src, dsize[, dst[, fx[, fy[, interpolation]]]]) → dst

| **interpolation 选项** | **所用的插值方法** |
| --- | --- |
| INTER\_NEAREST | 最近邻插值 |
| INTER\_LINEAR | 双线性插值（默认设置） |
| INTER\_AREA | 使用像素区域关系进行重采样。 它可能是图像抽取的首选方法，因为它会产生无云纹理的结果。 但是当图像缩放时，它类似于INTER\_NEAREST方法。 |
| INTER\_CUBIC | 4x4像素邻域的双三次插值 |
| INTER\_LANCZOS4 | 8x8像素邻域的Lanczos插值 |

**INTER\_NEAREST | 最近邻插值**

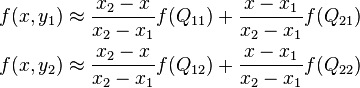
在一维空间中，最近点插值就相当于四舍五入取整。在二维图像中，[像素](https://so.csdn.net/so/search?q=%E5%83%8F%E7%B4%A0&spm=1001.2101.3001.7020)点的坐标都是整数，该方法就是选取离目标点最近的点。

会在一定程度上损失 **空间对称性（Alignment）**，在 **RoI Pooling** 中使用。



**INTER\_LINEAR | 双**[线性](https://so.csdn.net/so/search?q=%E7%BA%BF%E6%80%A7&spm=1001.2101.3001.7020)**插值（默认设置）**

在两个方向分别进行一次线性插值。



在[图像处理](https://so.csdn.net/so/search?q=%E5%9B%BE%E5%83%8F%E5%A4%84%E7%90%86&spm=1001.2101.3001.7020)的时候，我们先根据

srcX = dstX\* (srcWidth/dstWidth)

srcY = dstY \* (srcHeight/dstHeight)

来计算目标像素在源图像中的位置，这里计算的srcX和srcY一般都是浮点数，比如 f（1.2, 3.4）这个像素点是虚拟存在的，先找到与它临近的四个实际存在的像素点

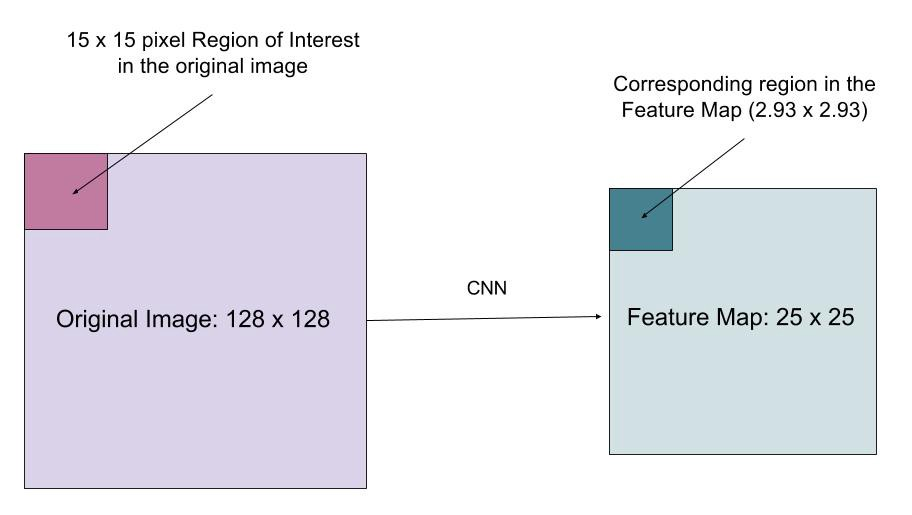
（1，3） （2，3）

（1，4） （2，4）

写成 f(i+u,j+v)的形式，则 u=0.2,v=0.4, i=1, j=3。

f(i+u,j+v) = (1-u)(1-v)f(i,j) + (1-u)vf(i,j+1) + u(1-v)f(i+1,j) + uvf(i+1,j+1)

保证了 **空间对称性（Alignment）**，在 **RoI Align** 中使用。



**INTER\_AREA | 使用像素区域关系进行**[重采样](https://so.csdn.net/so/search?q=%E9%87%8D%E9%87%87%E6%A0%B7&spm=1001.2101.3001.7020)**。**

略。

**INTER\_CUBIC | 4x4像素邻域的双三次插值**

略。

**INTER\_LANCZOS4 | 8x8像素邻域的Lanczos插值**

在x，y方向分别对相邻的八个点进行插值，也就是计算加权和，所以它是一个8x8的描述子。

**Code**

# coding=utf-8

import cv2

"""

INTER\_NEAREST | 最近邻插值

INTER\_LINEAR | 双线性插值（默认设置）

INTER\_AREA | 使用像素区域关系进行重采样

INTER\_CUBIC | 4x4像素邻域的双三次插值

INTER\_LANCZOS4 | 8x8像素邻域的Lanczos插值

"""

if \_\_name\_\_ == '\_\_main\_\_':

img = cv2.imread("girl.jpg")

height, width = img.shape[:2]

# 缩小图像

size = (int(width\*0.8), int(height\*0.7))

shrink\_NEAREST = cv2.resize(img, size, interpolation=cv2.INTER\_NEAREST)

shrink\_LINEAR = cv2.resize(img, size, interpolation=cv2.INTER\_LINEAR)

shrink\_AREA = cv2.resize(img, size, interpolation=cv2.INTER\_AREA)

shrink\_CUBIC = cv2.resize(img, size, interpolation=cv2.INTER\_CUBIC)

shrink\_LANCZOS4 = cv2.resize(img, size, interpolation=cv2.INTER\_LANCZOS4)

# 放大图像

fx = 1.2

fy = 1.1

enlarge\_NEAREST = cv2.resize(img, (0, 0), fx=fx, fy=fy, interpolation=cv2.INTER\_NEAREST)

enlarge\_LINEAR = cv2.resize(img, (0, 0), fx=fx, fy=fy, interpolation=cv2.INTER\_LINEAR)

enlarge\_AREA = cv2.resize(img, (0, 0), fx=fx, fy=fy, interpolation=cv2.INTER\_AREA)

enlarge\_CUBIC = cv2.resize(img, (0, 0), fx=fx, fy=fy, interpolation=cv2.INTER\_CUBIC)

enlarge\_LANCZOS4 = cv2.resize(img, (0, 0), fx=fx, fy=fy, interpolation=cv2.INTER\_LANCZOS4)

# 保存图像

cv2.imwrite("shrink\_NEAREST.jpg", shrink\_NEAREST)

cv2.imwrite("shrink\_LINEAR.jpg", shrink\_LINEAR)

cv2.imwrite("shrink\_AREA.jpg", shrink\_AREA)

cv2.imwrite("shrink\_CUBIC.jpg", shrink\_CUBIC)

cv2.imwrite("shrink\_LANCZOS4.jpg", shrink\_LANCZOS4)

cv2.imwrite("enlarge\_NEAREST.jpg", enlarge\_NEAREST)

cv2.imwrite("enlarge\_LINEAR.jpg", enlarge\_LINEAR)

cv2.imwrite("enlarge\_AREA.jpg", enlarge\_AREA)

cv2.imwrite("enlarge\_CUBIC.jpg", enlarge\_CUBIC)

cv2.imwrite("enlarge\_LANCZOS4.jpg", enlarge\_LANCZOS4)

**Demo**

原图像：  


**利用插值缩小**

肉眼基本看不出差别。

**最近邻插值**：  


**双线性插值（默认设置）**：  


**使用像素区域关系进行重采样**：  


**4x4像素邻域的双三次插值**：  


**8x8像素邻域的Lanczos插值**：  


**利用插值放大**

**最近邻插值**：  


**双线性插值（默认设置）**：  


**使用像素区域关系进行重采样**：  


**4x4像素邻域的双三次插值**：  


**8x8像素邻域的Lanczos插值**：  
