Goals

In this chapter we will learn :
* How to resize and image and apply basic filtering operations.
* We will see : **cv2.resize()**, **cv2.pyrDown()** , **cv2.pyrUp()**

Theory

Resizing()

Scaling or resizing is basically changing the dimensions of the image according to the specified scaling factor. Different interpolation methods are used. Preferable interpolation methods are: * `cv2.INTER_AREA` for shrinking
* `cv2.INTER_CUBIC` (slow)
* `cv2.INTER_LINEAR` for zooming.

By default, interpolation method used is `cv2.INTER_LINEAR` for all resizing purposes.
As the size of an image is reduced or enlarged, the pixels that form the image become increasingly visible, making the image appear "soft" if pixels are averaged, or jagged if not.

Function

The function is used in the following way:
`cv2.resize(src, dsize[, dst[, fx[, fy[, interpolation]]]])`

Parameters:

- **src** – input image.
- **dst** – output image; it has the size dsize (when it is non-zero) or the size computed from src.size(),fx, and fy; the type of dst is the same as of src.
- **dsize** – output image size; if it equals zero, it is computed as: **dsize = Size(round(fx*source.cols), round = (fy*src.rows)** Either dsize or both fx and fy must be non-zero.
- **fx** –scale factor along the horizontal axis; when it equals 0, it is computed as
  (double) dsize.width/src.cols
- **fy** – scale factor along the vertical axis; when it equals 0, it is computed as ***(double) dsize.height/src.rows***
- **interpolation** –interpolation method:
  1. `INTER_NEAREST` - a nearest-neighbor interpolation
  2. `INTER_LINEAR` - a bilinear interpolation (used by default)

3. `INTER_AREA` - re-sampling using pixel area relation. It may be a preferred method for image decimation, as it gives moire'-free results. But when the image is zoomed, it is similar to the `INTER_NEAREST` method.
4. `INTER_CUBIC` - a bicubic interpolation over 4x4 pixel neighborhood
5. `INTER_LANCZOS4` - a Lanczos interpolation over 8x8 pixel neighborhood

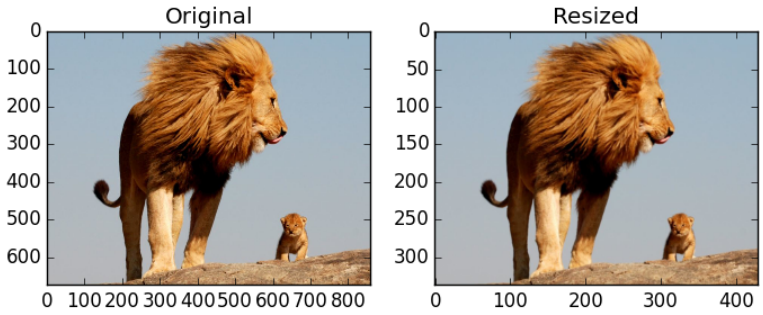
Code

```
#Import OpenCV, numpy and matplotlib
import numpy
import cv2
import matplotlib.pyplot as plt

#Read the image
img = cv2.imread('lion.jpg')

'''
OpenCV represents RGB images as multi-dimensional NumPy arrays...but in reverse
order!This means that images are actually represented in BGR order rather than
RGB!
'''
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)   #Convert to RGB

#Resize()
res = cv2.resize(img,None,fx=0.5, fy=0.5, interpolation = cv2.INTER_LINEAR)

plt.figure(0)     #Create a new window
plt.subplot(121),plt.imshow(img), plt.title('Original')
plt.subplot(122), plt.imshow(res), plt.title('Resized')

plt.show()        #Display the image
```

Input image:

Output Image:

For better comparison, look at the table:



pyrUp() and pyrDown()

OpenCV also offers inbuilt functions `pyrUp()` and `pyrDown()`. Normally, we used to work with an image of constant size. But in some occasions, we need to work with images of different resolution of the same image. Set of images with different resolution are called Image Pyramids .
There are two kinds of image pyramids:
1. Gaussian pyramid
2. Laplacian pyramid

Low resolution image is obtained by removing consecutive rows and columns. Then each pixel in higher level is formed by the contribution from 5 pixels in underlying level with Gaussian weights. By doing so, a M x N image becomes M/2 x N/2 image.in this way area is reduced by 4 times. High resolution image is obtained by adding rows and columns similarly. Area increases by four times.
For downsizing the image we use the function:

```
cv2.pyrDown(src[, dst[, dstsize[, borderType]]])
```

For upsizing the image we use the function:

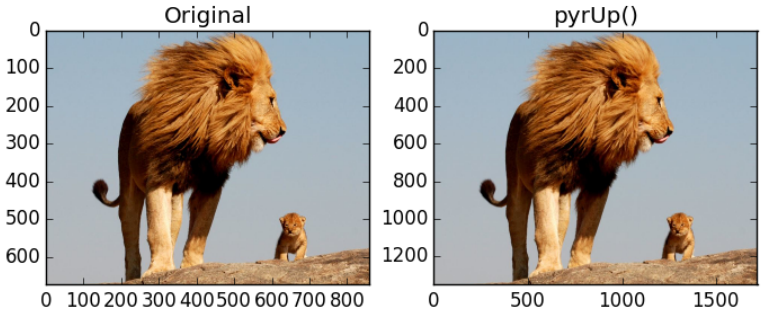
```
cv2.pyrUp(src[, dst[, dstsize[, borderType]]])
```

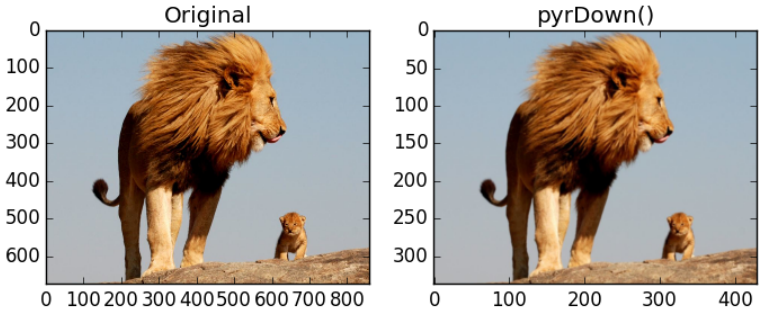
Parameters:

- **src** – input image.
- **dst** – output image; it has the specified size and the same type as src.
- **dstsize** – size of the output image.
- **borderType** – Pixel extrapolation method (BORDER_CONSTANT don't supported).

Code

```
#Import OpenCV, numpy and matplotlib
import numpy
import cv2
import matplotlib.pyplot as plt


#Read the image
img = cv2.imread('lion.jpg')

'''
OpenCV represents RGB images as multi-dimensional NumPy arrays...but in reverse
order!This means that images are actually represented in BGR order rather than
RGB!
'''
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)   #Convert to RGB

#Resize()
res1 = cv2.pyrUp(img,5)
res2 = cv2.pyrDown(img,5)


plt.figure(0)     #Create a new window
plt.subplot(121),plt.imshow(img), plt.title('Original')
plt.subplot(122), plt.imshow(res1), plt.title('pyrUp()')

plt.figure(1)     #Create a new window
plt.subplot(121),plt.imshow(img), plt.title('Original')
plt.subplot(122), plt.imshow(res2), plt.title('pyrDown()')

plt.show()        #Display the image
```


Original
pyrUp()
0
100
200
300
400
500
600
0 100 200 300 400 500 600 700 800
0
200
400
600
800
1000
1200
0 500 1000 1500


Original
pyrDown()
0
100
200
300
400
500
600
0 100 200 300 400 500 600 700 800
0
50
100
150
200
250
300
0 100 200 300 400

Excercises

1. Try to apply filtering operations on images before resizing and after resizing. Compare the results. You will find it fascinating.

Solution:

```
#Import OpenCV, numpy and matplotlib
import numpy
import cv2
import matplotlib.pyplot as plt


#Read the image
img = cv2.imread('monalisa.jpg')

'''
OpenCV represents RGB images as multi-dimensional NumPy arrays...but in reverse
order!This means that images are actually represented in BGR order rather than
RGB!
'''
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)   #Convert to RGB

#Resize()
res = cv2.resize(img,None,fx=0.2, fy=0.2, interpolation = cv2.INTER_LINEAR)

#Applying filters on image resized using resize()
r1_blur = cv2.medianBlur(res,5)

#Applying filters on original image
nr_blur = cv2.medianBlur(img,5)

#Downsizing the image
ds = cv2.pyrDown(img, 5)

#Applying filters on downsized image using pyrDown()
r2_blur = cv2.medianBlur(ds,5)


plt.figure("Compare")         #Create a new window
plt.subplot(221),plt.imshow(img), plt.title('Original image')
plt.xticks([]), plt.yticks([])
plt.subplot(222),plt.imshow(nr_blur),plt.title('Median Blur without resize')
plt.xticks([]), plt.yticks([])
plt.subplot(223),plt.imshow(r1_blur),plt.title('Median Blur after resize()')
```

```
plt.xticks([]), plt.yticks([])
plt.subplot(224),plt.imshow(r2_blur),plt.title('Median Blur after pyrDown()')
plt.xticks([]), plt.yticks([])
plt.show()
```

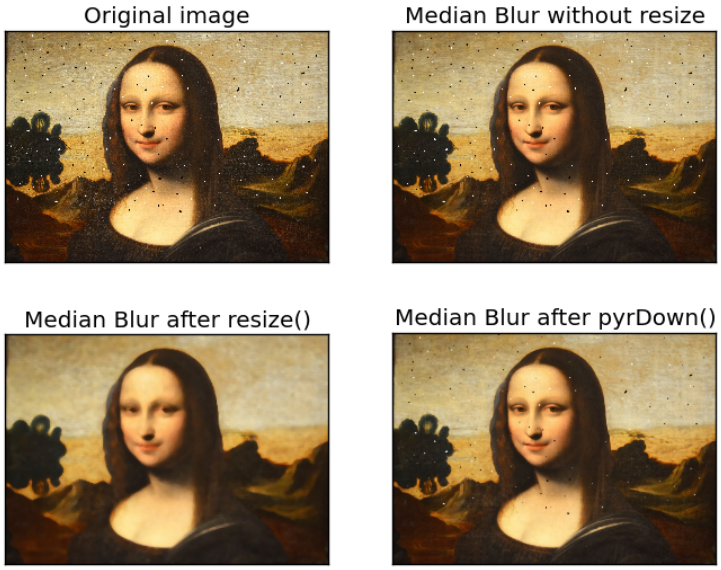


Figure 1: Output Image

References

1. http://opencv-python-tutroals.readthedocs.org/en/latest/py_tutorials/py_imgproc/py_pyramids/py_pyramids.html#pyramids
2. http://docs.opencv.org/modules/imgproc/doc/filtering.html#pyrdown