# 1.4.3: Resizing

**Topic Progress:** (https://gurus.pyimagesearch.com/topic/translation/) (https://gurus.pyimagesearch.com/topic/rotation/) (https://gurus.pyimagesearch.com/topic/resizing/) (https://gurus.pyimagesearch.com/topic/flipping/) (https://gurus.pyimagesearch.com/topic/cropping/) (https://gurus.pyimagesearch.com/topic/image-arithmetic/) (https://gurus.pyimagesearch.com/topic/bitwise-operations/) (https://gurus.pyimagesearch.com/topic/masking/) (https://gurus.pyimagesearch.com/topic/splitting-and-merging-channels/)

← Back to Lesson (https://gurus.pyimagesearch.com/lessons/basic-image-processing/)

Scaling, or simply *resizing*, is the process of increasing or decreasing the size of an image in terms of width and height.

When resizing an image, it's important to keep in mind the *aspect ratio* — which is the ratio of the width of the image to the height of an image. Ignoring the aspect ratio can lead to resized images that look compressed and distorted:

(https://gurus.pyimagesearch.com/wp-content/uploads/2015/03/resized_aspect_ratio.jpg)

**FIGURE 1:** IGNORING THE ASPECT RATIO OF AN IMAGE CAN LEAD TO RESIZED IMAGES THAT LOOK DISTORTED, CRUNCHED, AND SQUISHED.

On the *left* we have our original image. And on the *right* we have two images that have been distorted by not preserving the aspect ratio — they have been resized by ignoring the ratio of the width to the height of the image. In general, you'll want to preserve the aspect ratio of your images when resizing — especially if these images are to be presented as output to the user. Exceptions most certainly do apply, though, and as we explore machine learning techniques we'll find that our internal algorithms often *ignore* the aspect ratio of an image; but more on that when we get to machine learning.



We also need to keep in mind the *interpolation method* of our resizing function. The formal definition of interpolation is "the method of constructing new data points within the range of discrete set of known points." In this case, the "known points" are the pixels of our original image. And the goal of an interpolation function is to take these neighborhoods of pixels and use them to either increase or decrease the size of image.

In general, it's far more beneficial (and visually appealing) to decrease the size of the image. This is because the interpolation function simply has to remove pixels from an image. On the other hand, if we were to increase the size of the image the interpolation function would have to "fill in the gaps" between pixels that previously did not exist.

For example, take a look at the image below:

(https://gurus.pyimagesearch.com/wp-content/uploads/2015/03/resized_increase_decrease.jpg)

**FIGURE 2:** INCREASING AND DECREASING THE SIZE OF AN IMAGE. IN TERMS OF "QUALITY" OF THE OUTPUT IMAGE, DECREASING THE SIZE OF AN IMAGE IS ALWAYS EASIER THAN INCREASING IT.

On the *left*, we have our original image. In the *middle* we have resized the image to half its size — and other than the image being resized, there is no loss in "quality" of the image. However, on the *right* we have dramatically increased the size of the image. It now looks "pixelated" and "blown up."

As I mentioned above, you'll normally be *decreasing* the size of an image rather than *increasing* (exceptions do apply, of course). By decreasing the size of the image we have less pixels to process (not to mention less "noise" to deal with), which leads to faster and more accurate image processing algorithms.



# Objectives:

The primary objective of this topic is to understand how to resize an image using the OpenCV library.

# Resizing

So far we've covered two image transformations: translation and rotation. Now, we are going to explore how to resize an image. We'll also define another method for our `imutils` package, a series of convenience methods to help us with basic image processing.

Perhaps, not surprisingly, we will be using the `cv2.resize` function to resize our images. As I mentioned above, we'll need to keep in mind the aspect ratio of the image when we are using this function. But before we get too deep into the details, let's jump right into an example:



`resize.py` Python

```
1  # import the necessary packages
2  import argparse
3  import imutils
4  import cv2
5
6  # construct the argument parser and parse the arguments
7  ap = argparse.ArgumentParser()
8  ap.add_argument("-i", "--image", required=True, help="Path to the image")
9  args = vars(ap.parse_args())
10
11 # load the image and show it
12 image = cv2.imread(args["image"])
13 cv2.imshow("Original", image)
14
15 # we need to keep in mind aspect ratio so the image does not look skewed
16 # or distorted -- therefore, we calculate the ratio of the new image to
17 # the old image. Let's make our new image have a width of 150 pixels
18 r = 150.0 / image.shape[1]
19 dim = (150, int(image.shape[0] * r))
20
21 # perform the actual resizing of the image
22 resized = cv2.resize(image, dim, interpolation=cv2.INTER_AREA)
23 cv2.imshow("Resized (Width)", resized)
```

**Lines 1-13** should start to feel quite redundant at this point. We are importing our packages, setting up our argument parser, and finally loading our image and displaying it.

The actual interesting code doesn't start until **Lines 18 and 19**. When resizing an image, we need to keep in mind the aspect ratio of the image. The aspect ratio is the proportional relationship of the width and the height of the image:

$$AspectRatio = Width / Height$$

If we aren't mindful of the aspect ratio, our resizing will return results that look distorted (see **Figure 1** above).

Computing the resized ratio is handled on **Line 18**. In this line of code we define our new image width to be 150 pixels. In order to compute the ratio of the new height to the old height, we simply define our ratio *r* to be the new width (150 pixels) divided by the old width, which we access using `image.shape[1]` .

Now that we have our ratio, we can compute the new dimensions of the image on **Line 19**. Again, the width of the new image will be 150 pixels. The height is then computed by multiplying the old height by our ratio and converting it to an integer. By performing this operation we are able to preserve the original aspect ratio of the image.

The actual resizing of the image takes place on **Line 22**. The first argument is the image we wish to resize, and the second is our computed dimensions for the new image. The last parameter is our

`interpolation` method, which is the algorithm working behind the scenes to handle how the actual image is resized. We'll discuss the various interpolation methods that OpenCV provides later in this article.

Finally, we show our resized image on **Line 23**.

In the example we just explored, we only resized the image by specifying the width. But what if we wanted to resize the image by specifying the height? All that requires is a change to computing the resize ratio used to maintain the aspect ratio:

resize.py Python

```
25 # what if we wanted to adjust the height of the image? We can apply
26 # the same concept, again keeping in mind the aspect ratio, but instead
27 # calculating the ratio based on height -- let's make the height of the
28 # resized image 50 pixels
29 r = 50.0 / image.shape[0]
30 dim = (int(image.shape[1] * r), 50)
31
32 # perform the resizing
33 resized = cv2.resize(image, dim, interpolation=cv2.INTER_AREA)
34 cv2.imshow("Resized (Height)", resized)
35 cv2.waitKey(0)
```

On **Line 29** we redefine our ratio *r*. Our new image will have a height of 50 pixels. To determine the ratio of the new height to the old height, we divide 50 by the old height.

Then, we define the dimensions of our new image. We already know that the new image will have a height of 50 pixels. The new width is obtained by multiplying the old width by the ratio, again allowing us to maintain the original aspect ratio of the image.

We then perform the actual resizing of the image on **Line 33** and show it on **Line 34**:


(https://gurus.pyimagesearch.com/wp-content/uploads/2015/03/resized_long.jpg)

**FIGURE 3:** RESIZING AN IMAGE BY BOTH WIDTH AND HEIGHT (WHILE MAINTAINING ASPECT RATIO) USING THE CV2.RESIZE FUNCTION.

Here we can see that we have resized our original image in terms of both width and height while maintaining the aspect ratio. If we did not maintain the aspect ratio, our image would look distorted, as demonstrated in **Figure 1** at the top of this article.

Resizing an image is simple enough, but having to compute the aspect ratio, define the dimensions of the new image, and then perform the resizing takes three lines of code. These three lines of code, while they don't seem like much, can make our code quite verbose and messy. Instead, let's define a `resize` function inside our `imutils` package to help us out:

resize.py Python

```
37 # of course, calculating the ratio each and every time we want to resize
38 # an image is a real pain -- let's create a  function where we can specify
39 # our target width or height, and have it take care of the rest for us.
40 resized = imutils.resize(image, width=100)
41 cv2.imshow("Resized via Function", resized)
42 cv2.waitKey(0)
```

In this example you can see that the resizing of the image is handled by a single function: `imutils.resize` . The first argument we pass in is our image we want to resize. Then, we specify the keyword argument `width` , which is the width of our new image. The function then handles the resizing

Of course, we could also resize via the height of the image by changing the function call to:

resize.py Python

```
1 resized = imutils.resize(image, height=50)
```

Let's take this function apart and see what's going on under the hood:

imutils.py Python

```
29 def resize(image, width=None, height=None, inter=cv2.INTER_AREA):
30     # initialize the dimensions of the image to be resized and
31     # grab the image size
32     dim = None
33     (h, w) = image.shape[:2]
34
35     # if both the width and height are None, then return the
36     # original image
37     if width is None and height is None:
38         return image
39
40     # check to see if the width is None
41     if width is None:
42         # calculate the ratio of the height and construct the
43         # dimensions
44         r = height / float(h)
45         dim = (int(w * r), height)
46
47     # otherwise, the height is None
48     else:
49         # calculate the ratio of the width and construct the
50         # dimensions
51         r = width / float(w)
52         dim = (width, int(h * r))
53
54     # resize the image
55     resized = cv2.resize(image, dim, interpolation=inter)
56
57     # return the resized image
58     return resized
```

As you can see, we have defined our `resize` function. The first argument is the image we want to resize. Then, we define two keyword arguments, `width` and `height`. Both of these arguments cannot be `None`, otherwise we won't know how to resize the image. We also provide `inter`, which is our interpolation method and defaults to `cv2.INTER_AREA`.

On **Lines 32 and 33** we define the dimensions of our new, resized image and grab the dimensions of the original image.

We perform a quick check on **Lines 37-38** to ensure that a numerical value has been provided for either

the width or the height.

The computation of the ratio and new, resized image dimensions are handled on **Lines 41-52**, depending on whether we are resizing via width or via height.

Finally, **Line 55** handles the actual resizing of the image, then **Line 58** returns our resized image to the user.

To see the results of our `imutils.resize` function, check out Figure **Figure 4** below:


(https://gurus.pyimagesearch.com/wp-content/uploads/2015/03/resized_simple.jpg)

**FIGURE 4:** RESIZING OUR IMAGE USING OUR CONVENIENCE FUNCTION MAKES OUR CODE SUBSTANTIALLY LESS VERBOSE.

As you can see, our function was able to maintain the aspect ratio of the image while making our code cleaner and less verbose.

## Interpolation Methods

Up until now we have used only the `cv2.INTER_AREA` method for interpolation. And as I mentioned at the top of this article, the goal of an interpolation function is to examine neighborhoods of pixels and use these neighborhoods optically increase or decrease the size of image without introducing distortions (or

at least as few distortions as possible).

The first method is nearest neighbor interpolation, specified by the `cv2.INTER_NEAREST` flag. This method is the simplest approach to interpolation. Instead of calculating weighted averages of neighboring pixels or applying complicated rules, this method simply finds the "nearest" neighboring pixel and assumes the intensity value. While this method is fast and simple, the quality of the resized image tends to be quite poor and can lead to "blocky" artifacts.

Secondly, we have the `cv2.INTER_LINEAR` method with performs bilinear interpolation — this is the method that OpenCV uses by default when resizing images. The general idea behind bilinear interpolation can be found in any elementary school math textbook — *slope intercept form:*

$$y = mx + b$$

Obviously I am generalizing quite a bit, but the takeaway is that we are doing more than simply finding the "nearest" pixel and assuming the value of it (like in nearest neighbor interpolation) — now we are taking neighboring pixels and using this neighborhood to actually *calculate* what the interpolated value should be (rather than just *assuming* the nearest pixel value).

Thirdly, we have the `cv2.INTER_AREA` interpolation method. To be completely honest, I have found very little documentation about this method. As far as I can tell it is synonymous with "nearest neighbor interpolation". If I am wrong (or if anyone has any extra documentation on this method), I would really like to see it.

Finally, we have `cv2.INTER_CUBIC` and `cv2.INTER_LANCZOS4`. These methods are slower (since they no longer use simple linear interpolation and instead use *splines*) and utilize bicubic interpolation over square pixel neighborhoods. The `cv2.INTER_CUBIC` method operates on a *4 x 4* pixel neighbor and `cv2.INTER_LANCZOS4` over a *8 x 8* pixel neighborhood. In general I rarely see a the `cv2.INTER_LANCZOS4` method used in practice.

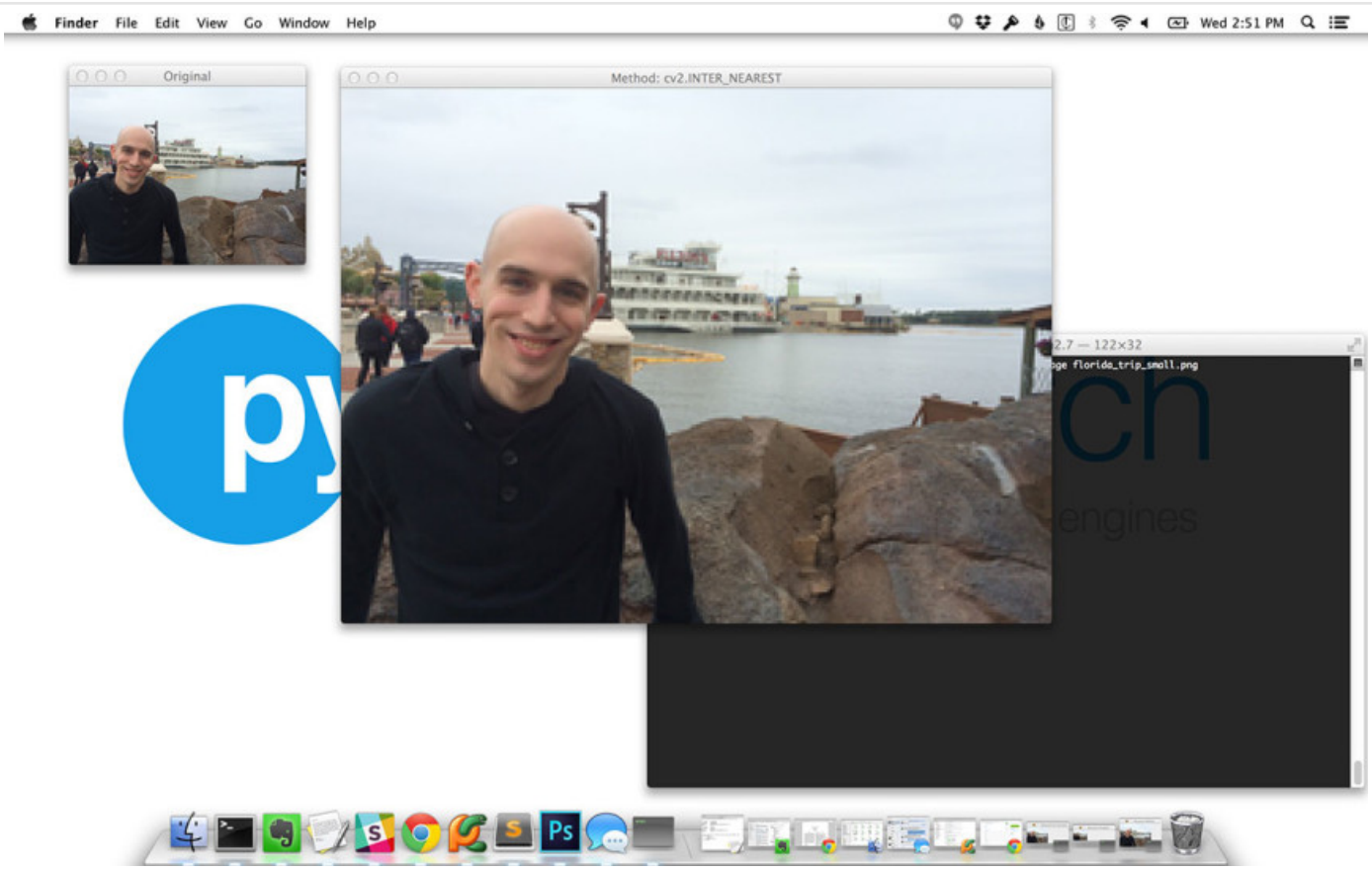
So now that we have discussed the interpolation methods that OpenCV provides, let's write some code to test them out:

resize.py — Python

```
44 # construct the list of interpolation methods
45 methods = [
46     ("cv2.INTER_NEAREST", cv2.INTER_NEAREST),
47     ("cv2.INTER_LINEAR", cv2.INTER_LINEAR),
48     ("cv2.INTER_AREA", cv2.INTER_AREA),
49     ("cv2.INTER_CUBIC", cv2.INTER_CUBIC),
50     ("cv2.INTER_LANCZOS4", cv2.INTER_LANCZOS4)]
51
52 # loop over the interpolation methods
53 for (name, method) in methods:
54     # increase the size of the image by 3x using the current interpolation
55     # method
56     resized = imutils.resize(image, width=image.shape[1] * 3, inter=method)
57     cv2.imshow("Method: {}".format(name), resized)
58     cv2.waitKey(0)
```

We start by defining our list of interpolation methods on **Lines 46-50**. And then we loop over each of the interpolation methods and resize the image (upsampling, making it 3x larger than the original image) on **Line 56**.

Let's take a look at the output of the nearest neighbor interpolation:




(https://gurus.pyimagesearch.com/wp-content/uploads/2015/03/resize_inter_nearest.jpg)

**FIGURE 5:** NEAREST NEIGHBOR INTERPOLATION.

Notice in the above figure how there are "blocky" artifacts in the resized image.

From there, we can look at bilinear interpolation:

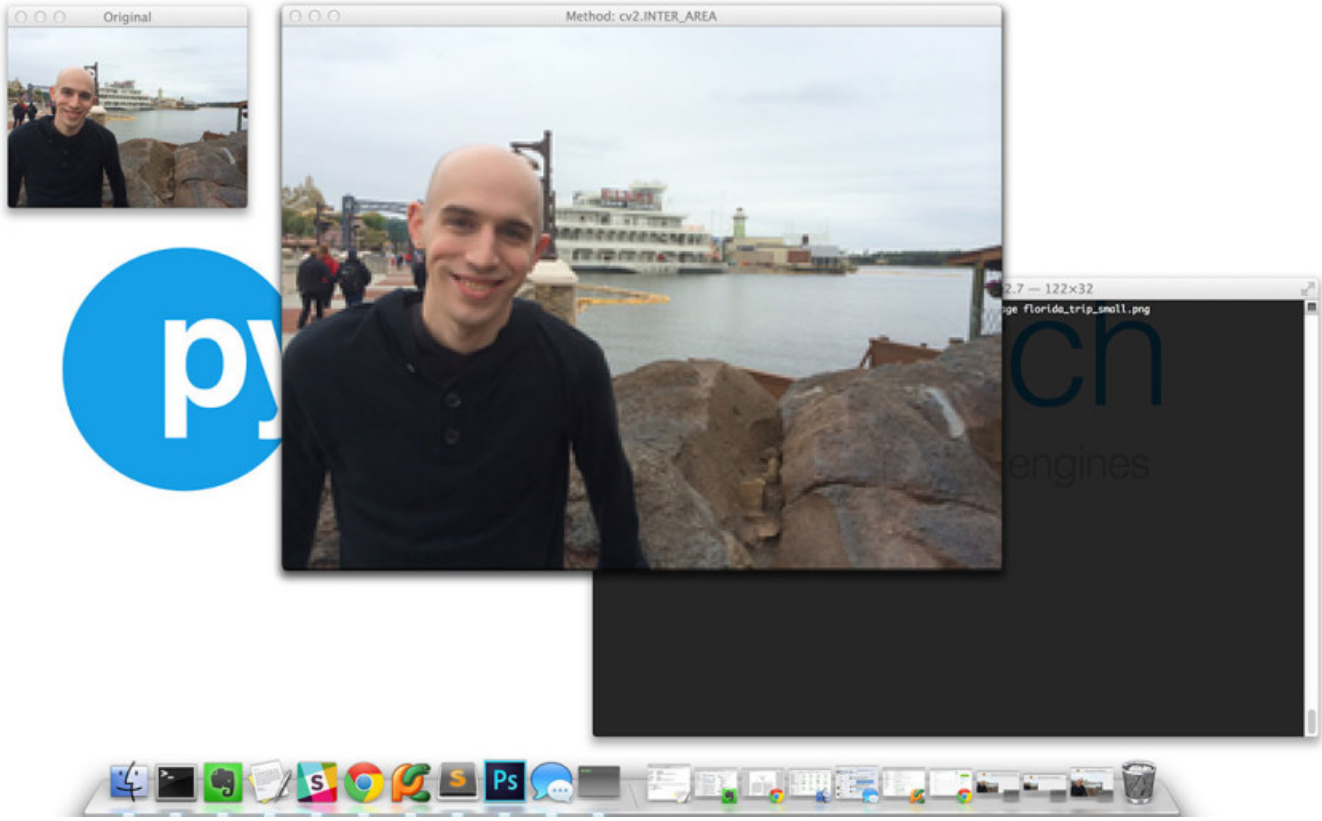

(https://gurus.pyimagesearch.com/wp-content/uploads/2015/03/resized_inter_linear.jpg)

**FIGURE 6:** THE OUTPUT OF BILINEAR INTERPOLATION.

Feedback

Notice how the block-like artifacts are gone and the image appears to be more smooth.

Next up: area interpolation:



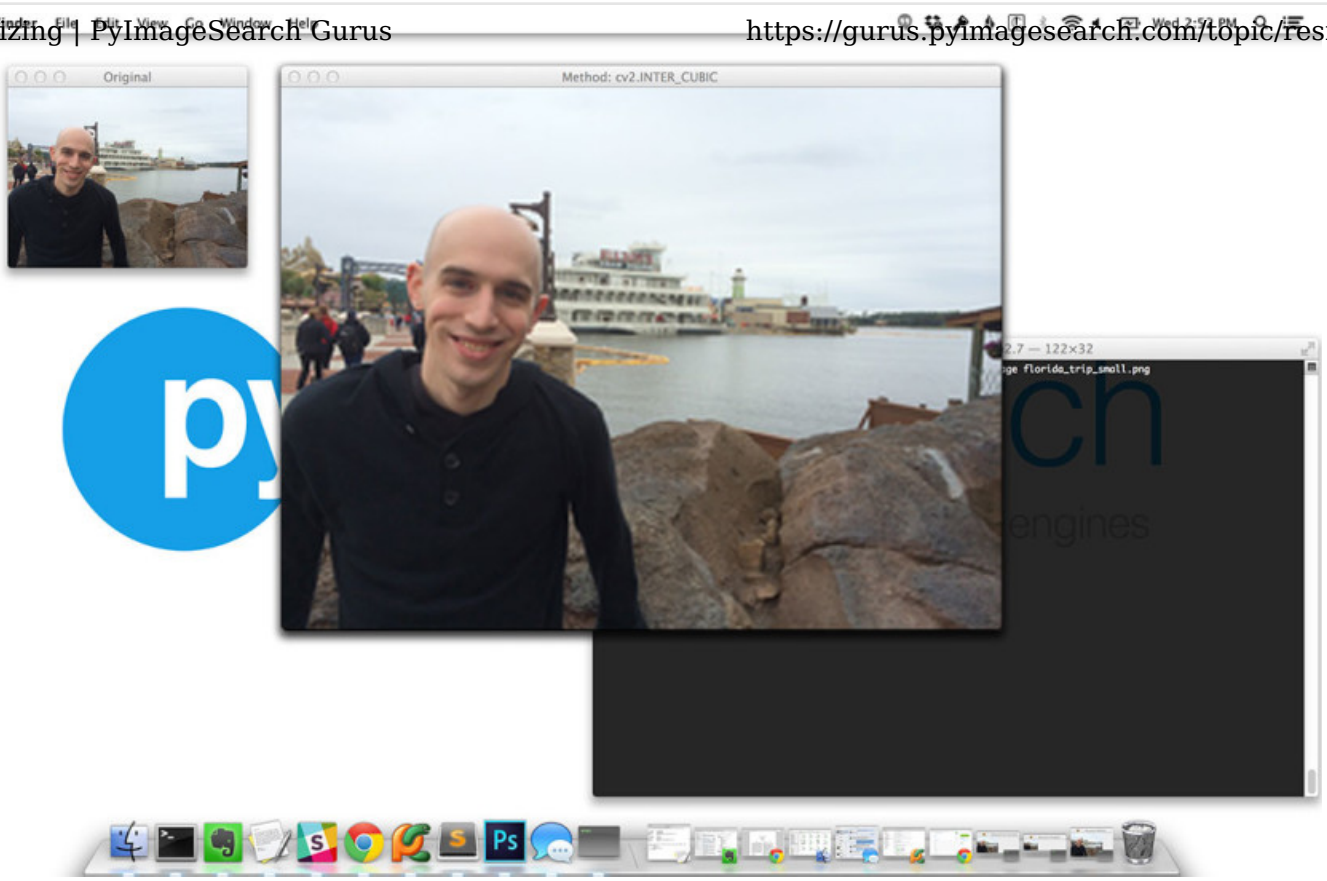

(https://gurus.pyimagesearch.com/wp-content/uploads/2015/03/resized_inter_area.jpg)

**FIGURE 7:** AREA INTERPOLATION.

Again the block-like artifacts are back. As far as I can tell, the `cv2.INTER_AREA` performs the same function as `cv2.INTER_NEAREST` .

Then we move on to bicubic interpolation:

Feedback

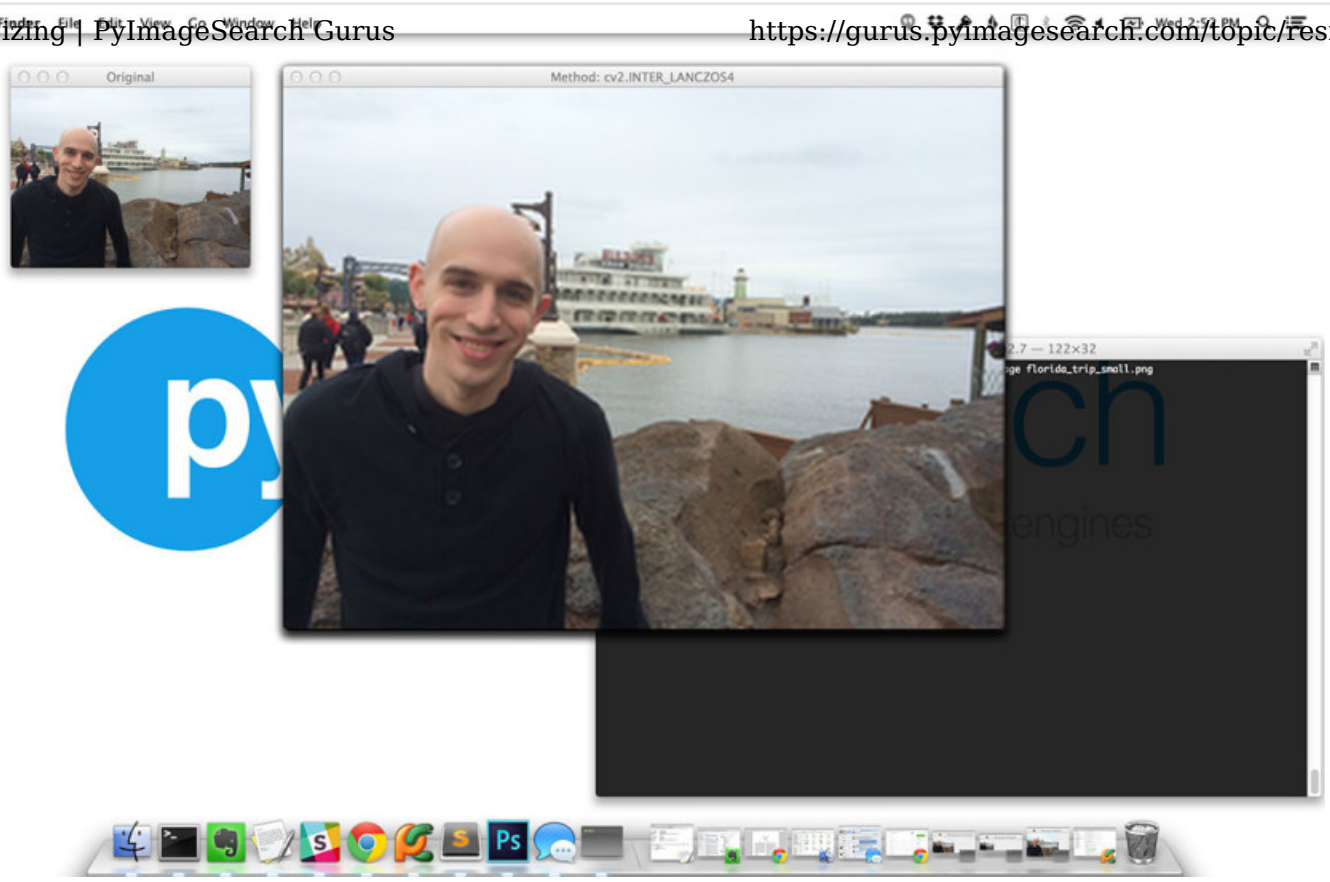



(https://gurus.pyimagesearch.com/wp-content/uploads/2015/03/resized_inter_cubic.jpg)

**FIGURE 8:** BICUBIC INTERPOLATION.

Bicubic interpolation further removes the block-like artifacts.

And lastly, the `cv2.LANCOSZ4` method, which appears to be very similar to the bicubic method:




(https://gurus.pyimagesearch.com/wp-content/uploads/2015/03/resized_inter_lanczos4.jpg)

**FIGURE 9:** BICUBIC INTERPOLATION OVER AN *8 X 8* PIXEL NEIGHBORHOOD.

Feedback

**So which method interpolation methods should you be using?**

In general, `cv2.INTER_NEAREST` is quite fast, but does not provide the highest quality results. So in *very* resource-constrained environments, consider using nearest neighbor interpolation — otherwise you probably won't use this interpolation method much (especially if you are trying to increase the size of an image).

When increasing (upsampling) the size of an image, consider using `cv2.INTER_LINEAR` and `cv2.INTER_CUBIC`. The `cv2.INTER_LINEAR` method tends to be slightly faster than the `cv2.INTER_CUBIC` method, but go with whichever one gives you the best results for your images.

When decreasing (downsampling) the size of an image, the OpenCV documentation suggests using `cv2.INTER_AREA` — although as far as I can tell, this method is very similar to nearest neighbor interpolation. In either case, *decreasing* the size of an image (in terms of quality) is always an easier task than *increasing* the size of an image.

Finally, as a general rule, `cv2.INTER_LINEAR` interpolation method is recommended as the default for

whenever you're upsampling or downsampling — it simply provides the highest quality results at a modest computation cost.

# Summary

In this article we discovered how to resize an image using OpenCV. When resizing an image it's important to keep in mind (1) the aspect ratio of your image, so your resized image does not look distorted, and (2) the interpolation method you are using to perform the resizing. See the section entitled *Interpolation Methods* above to help you decide on which interpolation method you should use. In general, you'll find that `cv2.INTER_LINEAR` is a good default choice.

Finally, it's important to note that if you are concerned about image quality, it's almost always preferable to go from a *larger* image to a *smaller* image. Increasing the size of an image normally introduces artifacts and reduces the quality of the image.

If you find yourself in a situation where your algorithms are not performing well on a low resolution image, consider upgrading the camera you are using to capture your photos instead of trying to make poor quality images work inside your algorithm.



# Downloads:

Download the Code (https://gurus.pyimagesearch.com/protected/code/computer_vision_basics/resizing.zip)

| Quizzes | | Status |
|---|---|---|
| 1 | Resizing Quiz (https://gurus.pyimagesearch.com/quizzes/resizing-quiz/) | |

← Previous Topic (https://gurus.pyimagesearch.com/topic/rotation/) Next Topic → (https://gurus.pyimagesearch.com/topic/flipping/)

## Course Progress

## Ready to continue the course?

Click the button below to **continue your journey to computer vision guru**.

I'm ready, let's go! (/pyimagesearch-gurus-course/)

## Resources & Links

- PyImageSearch Gurus Community (https://community.pyimagesearch.com/)
- PyImageSearch Virtual Machine (https://gurus.pyimagesearch.com/pyimagesearch-virtual-machine/)
- Setting up your own Python + OpenCV environment (https://gurus.pyimagesearch.com/setting-up-your-python-opencv-development-environment/)
- Course Syllabus & Content Release Schedule (https://gurus.pyimagesearch.com/course-syllabus-content-release-schedule/)
- Member Perks & Discounts (https://gurus.pyimagesearch.com/pyimagesearch-gurus-discounts-perks/)
- Your Achievements (https://gurus.pyimagesearch.com/achievements/)
- Official OpenCV documentation (http://docs.opencv.org/index.html)

## Your Account

- Account Info (https://gurus.pyimagesearch.com/account/)
- Support (https://gurus.pyimagesearch.com/contact/)
- Logout (https://gurus.pyimagesearch.com/wp-login.php?action=logout&redirect_to=https%3A%2F%2Fgurus.pyimagesearch.com%2F&_wpnonce=5736b21cae)

Feedback

Search