# A thread for questions you never dared to ask

**tuomo** **Sep '15**

Hey everyone,

I had this idea about a thread last night for asking all sorts of seemingly stupid questions (of course, we all know they don't exist when you're learning something new) about computer vision, machine learning and everything else related to the PyImageSearch Gurus course.

I'll start right away with one question that has been bothering me:

**Do feature vectors need to be of same length in order to be able to compare them?**

**Sep 2015**

**1 / 30**
Sep 2015

**8d ago**

**Adrian** Chief PyImageSearcher **Sep '15**

Great idea **@tuomo** !

> tuomo:
>
> Do feature vectors need to be of same length in order to be able to compare them?

In general, yes, they do need to be the same size. Distance functions such as Euclidean, Manhattan/City Block, Chi-squared, Hamming, etc. all expect feature vectors to be the same size. There are a very small amount of similarity functions (which are not true distance metrics) that allow for variable size feature vectors such as the **quadratic distance** from QBIC. However, in practice, this distance method often doesn't perform as well as standard ones.

For example, let's say we extracted a *4 x 4 x 4 = 64*-dim histogram from Image A and a *2 x 2 x 2 =8*-dim histogram from Image B.

Now, let's say I wanted to compute the Euclidean distance between these two feature vectors. I would need to loop over each of pair of entries in the feature vector, compute the difference, square it, and sum up the differences.

That would work...except that the histogram from Image A is *8x larger* than the histogram from Image B. This means that when I am looping over the entries of the histograms I will run out of entries in Image B -- thus the distance cannot be computed.

A more important underlying issue is that since these histograms have different bins, the number of pixels per bin does not exactly match up. Even if you *could* compare these histograms of different sizes, the results wouldn't make any sense.

**tuomo** **Feb 11**

Back with another question!

The cv2.boundingRect function takes a contour (as a numpy array) and returns a set of coordinates: x, y, width, height.

**Is there a function that takes the x, y, width, height coordinates and turn them into a contour-like numpy array?**

**Adrian** Chief PyImageSearcher **Feb 11**

That would be a neat function to have, but unfortunately no, there is not.

In this case, you normally take the rotated bounding box (rather than a "normal" bounding box) and then call `cv2.cv.BoxPoints` which turns it into a contour:

```
box = cv2.minAreaRect(c)
c = np.int(cv2.cv.BoxPoints(box))
```

Also, keep in mind that a contour is just a list of *(x, y)*-coordinates. You can *technically* just construct a list of points and then pass it into `cv2.findContours`, although it gets quite messy with all the nested lists.

**tuomo** **Feb 14**

I seem to have a lot of questions these days ...

**Can you retrieve the user's screen resolution using OpenCV?**

**Adrian** Chief PyImageSearcher **Feb 14**

> tuomo:
>
> Can you retrieve the user's screen resolution using OpenCV?

Using strictly OpenCV? No, you cannot. This is absolutely OS dependent. You would need to use either `win32api`, `AppKit`, or `gtk` depending on your system. **This StackOverflow thread** provides a bunch of examples and more information.

**ThamNgapWei** **Feb 14**

QDesktopWidget(python support Qt5 binding too)

```
QDesktopWidget widget;
QRect const screen_size = widget.availableGeometry(widget.primaryScreen());
```

Need more info of screens?Try QScreen

**tuomo** **Mar 27**

Here we go again: **does the size of an image matter for tasks like image classification?**

I am currently sorting memes/screenshots/other clutter from photographs on Instagram, using colour statistics and Haralick texture to train a Random Forest Classifier.

So far, I've gotten better results using the 150 x 150px thumbnails than the 640 x 640px standard resolution.

**mattsoftware** **Mar 27**

Sounds like you have answered your own question. I would also like to say the likely answer is 'it depends'. I have found downsizing images smaller works better when piping them through the raspberry pi when detecting frames from video. The larger the image the more work has to be done, and as long as the resolution is enough to detect the thing your looking for, the smaller the image the less work needs to be done so more can be done in real time.

Of course it makes no sense if you are trying to detect letters and numbers in a frame when the frame has been shrunk down so much the letters become just a mear pixel in the image. So 'it depends' is the answer as so many other questions to do with image processing.

Matt.

**Adrian** Chief PyImageSearcher **Mar 28**

> mattsoftware:
>
> The larger the image the more work has to be done, and as long as the resolution is enough to detect the thing your looking for, the smaller the image the less work needs to be done so more can be done in real time.

Matt is exactly right. This is highly dependent on your task. Just keep in mind that the less data there is to process, the faster your pipeline can ran.

Here's another way to look at it. You know how we apply blurring to smooth an image and reduce noise? Resizing can be thought as noise reduction as well. While humans love to look at HD, high resolution photographs, our computer vision algorithms don't need all that detail. So instead, we resize our images to smaller sizes to not only reduce the amount of data that needs to be processed, *but to reduce the detail as well*, which can lead to a better result (again, depending on your application).

**Jeffbass** **Mar 31**

Python/Numpy/OpenCV newbie question: Our lessons here deal with looping over one frame at a time, each of which is a numpy array that has a shape of (height, width, channels). If I wanted to have a numpy array of 100 frames, how should I use np.zeros to allocate an array that can hold 100 frames of (height, width, channels)? What would the syntax be for putting a frame in the frame array at say, frame position 7? I know I could do this with lists, but I'm sure it would be much more efficient to preallocate the array with np.zeros. There is probably a standard convention for doing this, but I did not see it in a quick google search of "array of opencv frames". Thanks in advance.

**RioDeDoro** **Mar 31**

Assuming that all of your images have the same width and height I think you could do it the following way:
- Given two images with size (h,w,3)

```
import argparse
import cv2
import numpy as np

ap = argparse.ArgumentParser()
```

```
ap.add_argument("-i1", "--image1", required = True)
ap.add_argument("-i2","--image2", required = True)
args = vars(ap.parse_args())

path1 = args["image1"]
path2 = args["image2"]

image1 = cv2.imread(path1)
image2 = cv2.imread(path2)

(h, w, chans) = image1.shape

# preallocate numpy array with zeros
all_images = np.zeros((h,w, 2*chans), dtype = "uint8")

# "Layers" (0,1,2) contain image 1
all_images[:,:, 0:3] = image1
# "Layers" (3,4,5) contain image 2
all_images[:,:, 3::] = image2

# visualization
cv2.imshow("Image1", all_images[:,:,0:3])
cv2.imshow("Image2", all_images[:,:,3::])
```

---

**Adrian** Chief PyImageSearcher **Mar 31**

As **@RioDeDoro** mentioned, you need to make sure your images all have the same width, height, and depth, otherwise you won't be able to store them in the same NumPy array. Below is a one-liner for initializing a NumPy array to store 100 frames, each with a width of `w`, height of `h` and depth of `d`:

```
images = np.zeros((100, h, w, d), dtype="uint8")
```

You can then store an image using:

```
images[i] = myImage
```

Where `i` is the current index into `images`.

---

**Jeffbass** **Mar 31**

Thanks, **@RioDeDoro** and **@Adrian** ! I had been experimenting and was using slices like your suggestion, **@RioDeDoro** , but I'll be using the one suggested by **@Adrian** , since it seems simpler and more pythonic. Although I am still such a Python newbie, I'm only beginning to understand what "pythonic" means in practical code. 🙂

---

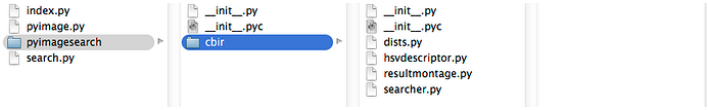
**RioDeDoro** **Apr 4**

I often run into problems if I try to structure a project into multiple files.
E.g. currently I'm working on chapter "3.2: Your first image search engine" where the structure looks as follows:

```
|--- pyimagesearch
|    |--- __init__.py
|    |--- cbir
|    |    |---- __init__.py
|    |    |--- dists.py
|    |    |--- hsvdescriptor.py
|    |    |--- resultsmontage.py
```

```
|    |    |--- searcher.py
|--- index.py
|--- search.py
```

I tried to code it by myself but I get the error message "cannot import name HSVDescriptor".
In the attachment is a picture of the folder structure .




Link to folder: **https://uni-bonn.sciebo.de/index.php/s/bGU2KVxy1DO3A7T**
Would be great if somebody could give me a hint 😉