

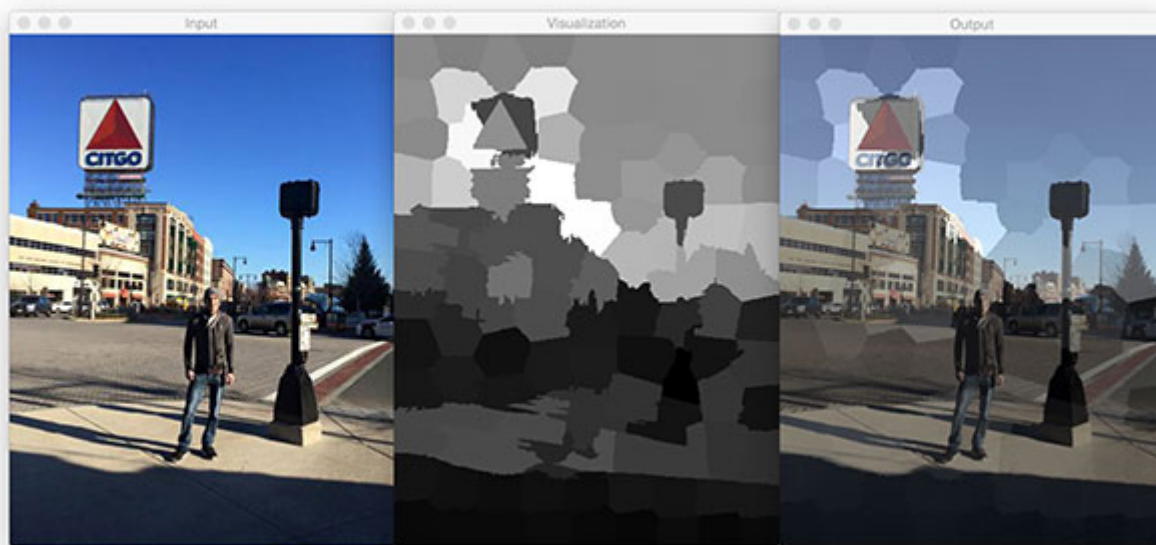
PYIMAGESEARCH

[IMAGE PROCESSING \(HTTPS://WWW.PYIMAGESEARCH.COM/CATEGORY/IMAGE-PROCESSING/\)](https://www.pyimagesearch.com/category/image-processing/)

[TUTORIALS \(HTTPS://WWW.PYIMAGESEARCH.COM/CATEGORY/TUTORIALS/\)](https://www.pyimagesearch.com/category/tutorials/)

Labeling superpixel colorfulness with OpenCV

by [Adrian R](#)



(https://pyimagesearch.com/wp-content/uploads/2017/06/example_output_02.jpg) After our previous post on [computing image colorfulness](#)

(<https://pyimagesearch.com/2017/06/05/computing-image-colorfulness-with-opencv-and-python/>) was published, Stephan, a PyImageSearch reader, left a comment on the tutorial asking if there was a method to compute the colorfulness of *specific regions* of an image (rather than the entire image).

There are multiple ways of attacking this problem. The first could be to apply a [sliding window](#) (<https://pyimagesearch.com/2015/03/23/sliding-windows-for-object-detection-with-python-and-opencv/>) to loop over the image and compute the colorfulness score for each ROI. An [image](#)

👋 Hey there, do you need help learning Computer Vision, Deep Learning, and OpenCV? Start by telling me about yourself.

Tap to answer

However, a better approach would be to use [superpixels](https://pyimagesearch.com/2014/07/28/a-slic-superpixel-tutorial-using-python/) (<https://pyimagesearch.com/2014/07/28/a-slic-superpixel-tutorial-using-python/>). Superpixels are extracted via a segmentation algorithm that groups pixels into (non-rectangular) regions based on their local color/texture. In the case of the popular SLIC superpixel algorithm, image regions are grouped based on a local version of [k-means clustering algorithm](https://pyimagesearch.com/2014/05/26/opencv-python-k-means-color-clustering/) (<https://pyimagesearch.com/2014/05/26/opencv-python-k-means-color-clustering/>) in the L*a*b* color space.

Given that superpixels will give us a much more natural segmentation of the input image than sliding windows, we can compute the colorfulness of specific regions in an image by:

- 1 Applying superpixel segmentation to the input image.
- 2 Looping over each of the superpixels individually and computing their respective colorfulness scores.
- 3 Maintaining a mask that contains the colorfulness score for each superpixel.

Based on this mask we can then visualize the most colorful regions of the image. Regions of the image that are more colorful will have larger colorful metric scores, while regions that are less colorful will smaller values.

To learn more about superpixels and computing image colorfulness, *just keep reading*.



Looking for the source code to this post?

[JUMP RIGHT TO THE DOWNLOADS SECTION](#) →

👋 Hey there, do you need help learning Computer Vision, Deep Learning, and OpenCV? Start by telling me about yourself. ✕

Tap to answer

In the first part of this blog post we will learn how to apply the SLIC algorithm to extract superpixels from our input image. The original 2010 publication by Achanta et al., [SLIC Superpixels](http://www.kev-smith.com/papers/SLIC_Superpixels.pdf), (http://www.kev-smith.com/papers/SLIC_Superpixels.pdf) goes into the details of the methodology and technique. We also briefly covered SLIC superpixels in this [blog post](https://pyimagesearch.com/2014/07/28/a-slic-superpixel-tutorial-using-python/) (<https://pyimagesearch.com/2014/07/28/a-slic-superpixel-tutorial-using-python/>) for readers who want a more concise overview of the algorithm.

Given these superpixels, we'll loop over them individually and compute their colorfulness score, taking care to compute the colorfulness metric for the *specific region* and not the *entire image* (as we did in our [previous post](https://pyimagesearch.com/2017/06/05/computing-image-colorfulness-with-opencv-and-python/) (<https://pyimagesearch.com/2017/06/05/computing-image-colorfulness-with-opencv-and-python/>)).

After we implement our script, we'll apply our combination of superpixel + image colorfulness to a set of input images.

Using superpixels for segmentation

Let's get started by opening up a new file in your favorite editor or IDE, name it `colorful_regions.py`, and insert the following code:



→ [Click here to download the code](#)

Labeling superpixel colorfulness with OpenCV and Python

```
1. | # import the necessary packages
2. | from skimage.exposure import rescale_intensity
3. | from skimage.segmentation import slic
4. | from skimage.util import img_as_float
5. | from skimage import io
6. | import numpy as np
7. | import argparse
8. | import cv2
```

The first **Lines 1-8** handle our imports — as you can see we make heavy use of several [scikit-image functions](http://scikit-image.org/) (<http://scikit-image.org/>) in this tutorial.

👋 Hey there, do you need help learning Computer Vision, Deep Learning, and OpenCV? Start by telling me about yourself.

Tap to answer

[post \(https://pyimagesearch.com/2017/06/05/computing-image-colorfulness-with-opencv-and-python/\)](https://pyimagesearch.com/2017/06/05/computing-image-colorfulness-with-opencv-and-python/) where it was introduced:



→ [Click here to download the code](#)

Labeling superpixel colorfulness with OpenCV and Python

```

10. | def segment_colorfulness(image, mask):
11. |     # split the image into its respective RGB components, then mask
12. |     # each of the individual RGB channels so we can compute
13. |     # statistics only for the masked region
14. |     (B, G, R) = cv2.split(image.astype("float"))
15. |     R = np.ma.masked_array(R, mask=mask)
16. |     G = np.ma.masked_array(B, mask=mask)
17. |     B = np.ma.masked_array(B, mask=mask)
18. |
19. |     # compute rg = R - G
20. |     rg = np.absolute(R - G)
21. |
22. |     # compute yb = 0.5 * (R + G) - B
23. |     yb = np.absolute(0.5 * (R + G) - B)
24. |
25. |     # compute the mean and standard deviation of both `rg` and `yb`,
26. |     # then combine them
27. |     stdRoot = np.sqrt((rg.std() ** 2) + (yb.std() ** 2))
28. |     meanRoot = np.sqrt((rg.mean() ** 2) + (yb.mean() ** 2))
29. |
30. |     # derive the "colorfulness" metric and return it
31. |     return stdRoot + (0.3 * meanRoot)

```

Lines 10-31 represent our colorfulness metric function, which has been adapted to compute the colorfulness for a *specific region* of an image.

The region can be any shape as we take advantage of [NumPy masked arrays \(https://docs.scipy.org/doc/numpy-1.12.0/reference/maskedarray.generic.html\)](https://docs.scipy.org/doc/numpy-1.12.0/reference/maskedarray.generic.html) — only pixels part of the mask will be included in the computation.

For the specified `mask` region of a particular `image`, the `segment_colorfulness` function performs the following tasks:

- 1 Splits the image into RGB component channels (**Line 14**).

👋 Hey there, do you need help learning Computer Vision, Deep Learning, and OpenCV? Start by telling me about yourself.

Tap to answer

- 4 Uses the RGB components to compute `yb` (**Lines 23**).
- 5 Computes the mean and standard deviation of `rg` and `yb` whilst combining them (**Lines 27 and 28**).
- 6 Does the final calculation of the metric and returns (**Line 31**) it to the calling function.

Now that our key colorfulness function is defined, the next step is to parse our command line arguments:



→ [Click here to download the code](#)

Labeling superpixel colorfulness with OpenCV and Python

```
33. | # construct the argument parse and parse the arguments
34. | ap = argparse.ArgumentParser()
35. | ap.add_argument("-i", "--image", required=True,
36. |     help="path to input image")
37. | ap.add_argument("-s", "--segments", type=int, default=100,
38. |     help="# of superpixels")
39. | args = vars(ap.parse_args())
```

On **Lines 34-39** we make use of `argparse` to define two arguments:

- 1 `--image` : The path to our input image.
- 2 `--segments` : The number of superpixels. The *SLIC Superpixels* paper shows examples of breaking an image up into different numbers of superpixels. This parameter is fun to experiment with (as it controls the level of granularity of your resulting superpixels); however we'll be working with a `default=100` . The smaller the value, the fewer and larger the superpixels, allowing the algorithm running faster. The larger the number of segments, the more fine-grained the segmentation, but SLIC will take longer to run (due to more clusters needing to be computed).

Now it's time to load the image into memory, allocate space for our visualization, and compute SLIC superpixel segmentation:



Hey there, do you need help learning Computer Vision, Deep Learning, and OpenCV? Start by telling me about yourself.

Tap to answer

```
43. | orig = cv2.imread(args["image"])
44. | vis = np.zeros(orig.shape[:2], dtype="float")
45. |
46. | # load the image and apply SLIC superpixel segmentation to it via
47. | # scikit-image
48. | image = io.imread(args["image"])
49. | segments = slic(img_as_float(image), n_segments=args["segments"],
50. |                 slic_zero=True)
```

On **Line 43** we load our command line argument `--image` into memory as `orig` (OpenCV format).

We follow this step by allocating memory with the same shape (width and height) as the original input image for our visualization image, `vis`.

Next, we load the command line argument `--image` into memory as `image`, this time in scikit-image format. The reason we use scikit-image's `io.imread` here is because OpenCV loads images in BGR order rather than RGB format (which scikit-image does). The `slic` function will convert our input `image` to the L*a*b* color space during the superpixel generation process *assuming* our image is in RGB format.


Therefore we have two choices:

- 1 Load the image with OpenCV, clone it, and then swap the ordering of the channels.
- 2 Simply load a copy of the original image using scikit-image.

Either approach is valid and will result in the same output.

Superpixels are calculated by a call to `slic` where we specify `image`, `n_segments`, and the `slic_zero` switch. Specifying `slic_zero=True` indicates that we want to use the zero parameter version of SLIC, an extension to the original algorithm that does not require us to manually tune parameters to the algorithm. We refer to the superpixels as `segments` for the rest of the script.

Now let's compute the colorfulness of each superpixel:

 Hey there, do you need help learning Computer Vision, Deep Learning, and OpenCV? Start by telling me about yourself.

Tap to answer

```

54. | # construct a mask for the segment so we can compute image
55. | # statistics for *only* the masked region
56. | mask = np.ones(image.shape[:2])
57. | mask[segments == v] = 0
58. |
59. | # compute the superpixel colorfulness, then update the
60. | # visualization array
61. | C = segment_colorfulness(orig, mask)
62. | vis[segments == v] = C

```

We start by looping over each of the individual `segments` on **Line 52**.

Lines 56 and 57 are responsible for constructing a `mask` for the current superpixel. The `mask` will have the same width and height as our input image and will be filled (initially) with an array of ones (**Line 56**).

Keep in mind that when using NumPy masked arrays, that a given entry in an array is *only* included in a computation if the corresponding `mask` value is set to zero (implying that the pixel is *unmasked*). If the value in the `mask` is *one*, then the value is assumed to be *masked* and is hence ignored.

Here we initially set all pixels to *masked*, then set only the pixels part of the current superpixel to *unmasked* (**Line 57**).

Using our `orig` image and our `mask` as parameters to `segment_colorfulness`, we can compute `C`, which is the colorfulness of the superpixel (**Line 61**).

Then, we update our visualization array, `vis`, with the value of `C` (**Line 62**).

At this point, we have answered PyImageSearch reader, Stephan's question — we have computed the colorfulness for different regions of an image.

Naturally we will want to see our results, so let's continue by constructing a transparent overlay visualization for the most/least colorful regions in our input image:



→ [Click here to download the code](#)

```

Labeling superpixel colorfulness with OpenCV and Python
64. | # scale the visualization image from an unrestricted floating point
65. | # to unsigned 8-bit integer array so we can use it with OpenCV and
66. | # display it to our screen

```

👋 Hey there, do you need help learning Computer Vision, Deep Learning, and OpenCV? Start by telling me about yourself.

Tap to answer

```

72. | overlay = np.dstack([vis] * 3)
73. | output = orig.copy()
74. | cv2.addWeighted(overlay, alpha, output, 1 - alpha, 0, output)

```

Since `vis` is currently a floating point array, it is necessary to re-scale it to a typical 8-bit unsigned integer `[0-255]` array. This is important so that we can display the output image to our screen with OpenCV. We accomplish this by using the `rescale_intensity` function (from `skimage.exposure`) on **Line 67**.

Now we'll overlay the superpixel colorfulness visualization on top of the original image. We've already discussed transparent overlays and the `cv2.addWeighted` (and associated parameters), so please refer to [this blog post \(https://pyimagesearch.com/2016/03/07/transparent-overlays-with-opencv/\)](https://pyimagesearch.com/2016/03/07/transparent-overlays-with-opencv/) for more details on how transparent overlays are constructed.

Finally, let's display images to the screen and close out this script:



→ [Click here to download the code](#)

Labeling superpixel colorfulness with OpenCV and Python

```

76. | # show the output images
77. | cv2.imshow("Input", orig)
78. | cv2.imshow("Visualization", vis)
79. | cv2.imshow("Output", output)
80. | cv2.waitKey(0)

```

We will display three images to the screen using `cv2.imshow` , including:

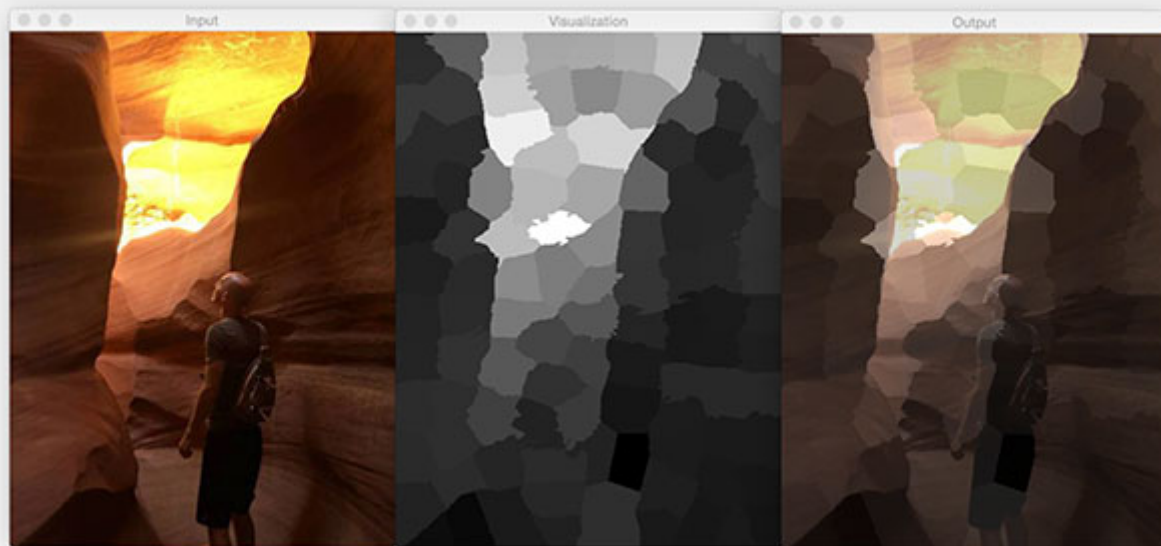
- 1 `orig` : Our input image.
- 2 `vis` : Our visualization image (i.e., level of colorfulness for each of the superpixel regions).
- 3 `output` : Our output image.

Superpixel and colorfulness metric results

Let's see our Python script in action — open a terminal, `workon` your virtual environment if you are using one (highly recommended), and enter the following command:

👋 Hey there, do you need help learning Computer Vision, Deep Learning, and OpenCV? Start by telling me about yourself.

Tap to answer



https://pyimagesearch.com/wp-content/uploads/2017/06/example_output_01.jpg

Figure 1: Computing a region-based colorfulness score using superpixel segmentation.

On the *left* you can see the original input image, a photo of myself exploring [Antelope Canyon](https://en.wikipedia.org/wiki/Antelope_Canyon) (https://en.wikipedia.org/wiki/Antelope_Canyon), arguably the most beautiful slot canyon in the United States. Here we can see a mixture of colors due to the light filtering in from above.

In the *middle* we have our computed visualization for each of the 100 superpixels. Dark regions in this visualization refer to *less colorful* regions while light regions indicate *more colorful*.

Here we can see the least colorful regions are around the walls of the canyon, closest to the camera — this is where the least light is reaching.

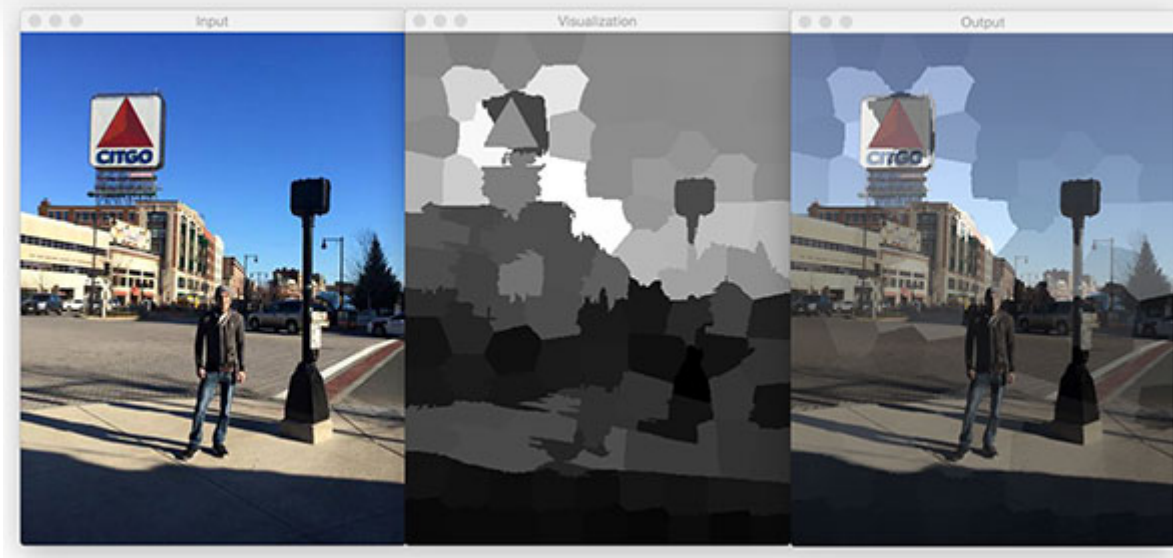
The most colorful regions of the input image are found where the light is directly reaching inside the canyon, illuminating part of the wall like candlelight.

Finally, on the *right* we have our original input image overlaid with the colorfulness visualization — this image allows us to more easily identify the most/least colorful regions of the image.

The following image is a photo of myself in Boston by the [iconic Citgo sign](https://en.wikipedia.org/wiki/Boston_Citgo_sign) (https://en.wikipedia.org/wiki/Boston_Citgo_sign) overlooking Kenmore square:

👋 Hey there, do you need help learning Computer Vision, Deep Learning, and OpenCV? Start by telling me about yourself.

Tap to answer



(https://pyimagesearch.com/wp-content/uploads/2017/06/example_output_02.jpg)

Figure 2: Using superpixels, we can first segment our image, and then compute a colorfulness score for each region.

Here we can see the least colorful regions of the image are towards the bottom where the shadow is obscuring much of the sidewalk. The more colorful regions can be found towards the sign and sky itself.

Finally, here is a photo from Rainbow Point, the highest elevation in [Bryce Canyon:](https://en.wikipedia.org/wiki/Bryce_Canyon_National_Park)



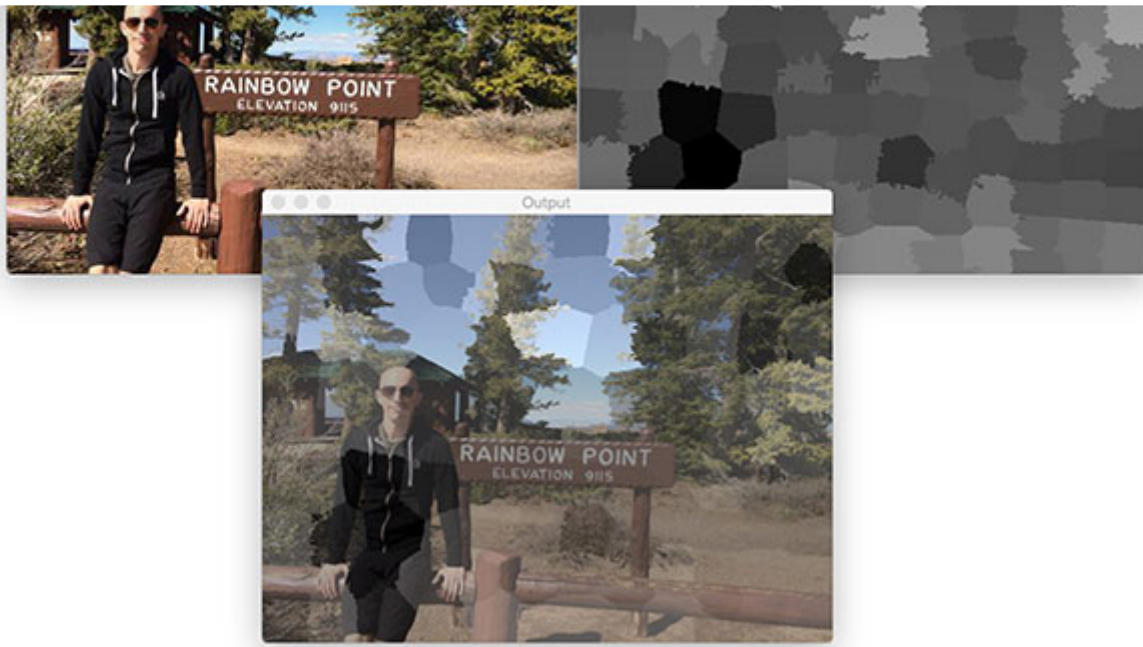
→ [Click here to download the code](#)

Labeling superpixel colorfulness with OpenCV and Python

```
1. | $ python colorful_regions.py --image images/example_03.jpg
```

👋 Hey there, do you need help learning Computer Vision, Deep Learning, and OpenCV? Start by telling me about yourself.

Tap to answer



(https://pyimagesearch.com/wp-content/uploads/2017/06/example_output_03.jpg)

Figure 3: Labeling individual superpixels in an image based on how “colorful” each region is.

Notice here that my black hoodie and shorts are the *least colorful* regions of the image, while the sky and foliage towards the center of the photo are the *most colorful*.

Summary

In today’s blog post we learned how to use the SLIC segmentation algorithm to compute superpixels for an input image.

We then accessed each of the individual superpixels and applied our colorfulness metric.

The colorfulness scores for each region were combined into a mask, revealing the *most colorful* and *least colorful* regions of the input image.

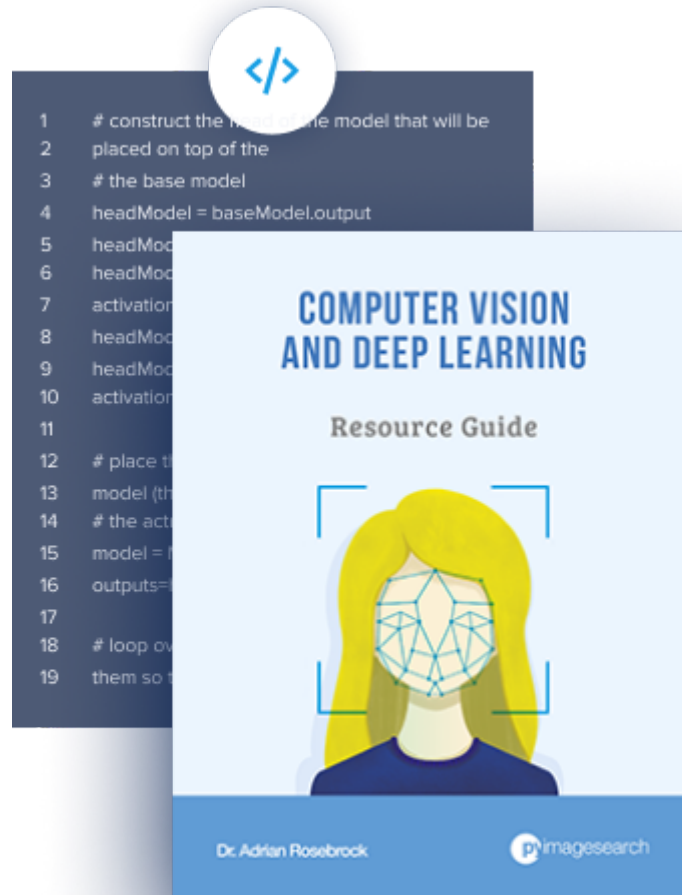
Given this computation, we were able to visualize the colorfulness of each region in two ways:

By examining the raw `vis` mask.

Hey there, do you need help learning Computer Vision, Deep Learning, and OpenCV? Start by telling me about yourself.

Tap to answer

To be notified when future tutorials are published here on PyImageSearch, be sure to enter your email address in the form below!



Download the Source Code and FREE 17-page Resource Guide

👋 Hey there, do you need help learning Computer Vision, Deep Learning, and OpenCV? Start by telling me about yourself.

Tap to answer



About the Author

Hi there, I'm Adrian Rosebrock, PhD. All too often I see developers, students, and researchers wasting their time, studying the wrong things, and generally struggling to get started with Computer Vision, Deep Learning, and OpenCV. I created this website to show you what I believe is the best possible way to get your start.

Previous Article:

Image Difference with OpenCV and Python

(<https://www.pyimagesearch.com/2017/06/19/image-difference-with-opencv-and-python/>)

Next Article:

Installing Tesseract for OCR

(<https://www.pyimagesearch.com/2017/07/03/installing-tesseract-for-ocr/>)

👋 Hey there, do you need help learning Computer Vision, Deep Learning, and OpenCV? Start by telling me about yourself.

Tap to answer

OpenCV and Python



Stephan

June 26, 2017 at 1:01 pm (<https://www.pyimagesearch.com/2017/06/26/labeling-superpixel-colorfulness-opencv-python/#comment-428093>)

you are awesome !
Stephan says THANKS



Adrian Rosebrock

June 27, 2017 at 6:21 am (<https://www.pyimagesearch.com/2017/06/26/labeling-superpixel-colorfulness-opencv-python/#comment-428246>)

Thanks Stephan 😊



Stefan van der Walt (<http://mentat.za.net>)

June 26, 2017 at 5:03 pm (<https://www.pyimagesearch.com/2017/06/26/labeling-superpixel-colorfulness-opencv-python/#comment-428108>)

Thank you for the fun post!

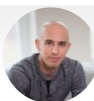
This example could be made even simpler by using the regionprops functionality in skimage.measure. Masked arrays are not very fast, unfortunately, so I tend to avoid them.

Here's the suggested implementation and output:

<https://gist.github.com/stefanv/05c57452c544b4d681e06e3e60e4d3>

👋 Hey there, do you need help learning Computer Vision, Deep Learning, and OpenCV? Start by telling me about yourself.

Tap to answer

**Adrian Rosebrock**

June 27, 2017 at 6:19 am (<https://www.pyimagesearch.com/2017/06/26/labeling-superpixel-colorfulness-opencv-python/#comment-428243>)

Thanks for the suggestion, Stefan! Part of the reason I used NumPy's masked arrays was simply re-familiarize myself with them. But you are right, `regionprops` is also a good choice.

**André Pilastrì**

June 26, 2017 at 5:33 pm (<https://www.pyimagesearch.com/2017/06/26/labeling-superpixel-colorfulness-opencv-python/#comment-428115>)

Hi Adrian,

I very enjoy your posts!

Just one question: Do you know deep learning work using graphs?

I think this is an interesting topic.

**Adrian Rosebrock**

June 27, 2017 at 6:16 am (<https://www.pyimagesearch.com/2017/06/26/labeling-superpixel-colorfulness-opencv-python/#comment-428239>)

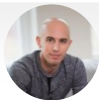
Hi André — what do you mean by “deep learning work using graphs”? Are you trying to build a graph of superpixel-like structures using deep learning? Or are you trying to apply deep learning to actual graphs (i.e., figures and plots)?

**André Pilastrì**

June 27, 2017 at 9:01 am (<https://www.pyimagesearch.com/2017/06/26/labeling-superpixel-colorfulness-opencv-python/#comment-428294>)

👋 Hey there, do you need help learning Computer Vision, Deep Learning, and OpenCV? Start by telling me about yourself.

Tap to answer

**Adrian Rosebrock**

June 30, 2017 at 8:29 am (<https://www.pyimagesearch.com/2017/06/26/labeling-superpixel-colorfulness-opencv-python/#comment-428591>)

What would the end goal be of supplying a machine learning model with the graph? What are you trying to accomplish?

**Tom**

June 27, 2017 at 5:58 am (<https://www.pyimagesearch.com/2017/06/26/labeling-superpixel-colorfulness-opencv-python/#comment-428229>)

I like how you go with explanations practically line-by-line in your tutorials. Can you elaborate a bit about what is going on and the technique/method you're using in line 57 "mask[segments == v] = 0" ? Specifically, what does the double equal sign do in square brackets? Thanks

**Adrian Rosebrock**

June 27, 2017 at 6:08 am (<https://www.pyimagesearch.com/2017/06/26/labeling-superpixel-colorfulness-opencv-python/#comment-428234>)

The `segments` variable is a NumPy array containing our superpixels. Each superpixel has a unique integer value. Therefore, we can find all (x, y)-coordinates in `mask` that have `segments == v` (implying that we are extracting all pixels part of the superpixel with value `v`). There is accomplished by NumPy array indexing.

**PBS (<https://blog.objectivity.co.uk/learning-with-colour/>)**

June 29, 2017 at 9:08 am (<https://www.pyimagesearch.com/2017/06/26/labeling-superpixel-colorfulness-opencv-python/#comment-428472>)

Another great blog post. Thanks very much.



Hey there, do you need help learning Computer Vision, Deep Learning, and OpenCV? Start by telling me about yourself.

Tap to answer

**stephan schulz**

September 5, 2017 at 6:05 pm (<https://www.pyimagesearch.com/2017/06/26/labeling-superpixel-colorfulness-opencv-python/#comment-434196>)

Again, thanks for taking the time to show us how to do such thing.

I implemented your code for the <http://www.openframeworks.cc> (<http://www.openframeworks.cc>) toolchain.

<https://github.com/stephanschulz/superpixels-colorfulness>
(<https://github.com/stephanschulz/superpixels-colorfulness>)

**Adrian Rosebrock**

September 7, 2017 at 7:13 am (<https://www.pyimagesearch.com/2017/06/26/labeling-superpixel-colorfulness-opencv-python/#comment-434308>)

Very cool, thanks for sharing Stephan.

**Stephan Schulz**

November 3, 2017 at 4:35 pm (<https://www.pyimagesearch.com/2017/06/26/labeling-superpixel-colorfulness-opencv-python/#comment-439575>)

I am wondering if there is a better way to derive the mean color from each super pixel cluster. Currently is adding all R,G,B values together and dividing them by the amount in each array.

I find this makes for a very muddy color.

Since this code does evaluate the colourfulness, is there a way to get a color representation of this color? My eye catches the brighter more intense colors, but the mean does not give me this feeling.

Thanks for any hints



Hey there, do you need help learning Computer Vision, Deep Learning, and OpenCV? Start by telling me about yourself.

Tap to answer



November 6, 2017 at 10:44 am (<https://www.pyimagesearch.com/2017/06/26/labeling-superpixel-colorfulness-opencv-python/#comment-439814>)

Hi Stephan — I'm not sure what you mean by a "color representation of this color"? Do you mean the name of a color?



stephan schulz

November 9, 2017 at 11:31 am (<https://www.pyimagesearch.com/2017/06/26/labeling-superpixel-colorfulness-opencv-python/#comment-440121>)

In your example you have one set where you show the source image with each super pixel cluster filled with a single color. This color seems to be the mean color of all pixels inside this cluster.

I would like this color to be the most colourful one in the cluster, not the mean. Is there a way to extract that information?

thx. 😊



Adrian Rosebrock

November 13, 2017 at 2:25 pm (<https://www.pyimagesearch.com/2017/06/26/labeling-superpixel-colorfulness-opencv-python/#comment-440491>)

Hi Stephan — can you elaborate on "I would like this color to be the most colorful one in the cluster"? The most colorful pixel?



stephan schulz

November 16, 2017 at 1:08 pm (<https://www.pyimagesearch.com/2017/06/26/labeling-superpixel-colorfulness-opencv-python/#comment-440773>)

if one of the pixel clusters has a lot of black and white in it but also has nice yellow dot in the middle

👋 Hey there, do you need help learning Computer Vision, Deep Learning, and OpenCV? Start by telling me about yourself.

Tap to answer

now can I query your code to know more about which of the colours present in this specific cluster allowed it to receive a high score?

in the end I want to represent this cluster by it's most "beautiful" / most noticeable colour, not the mean.

**Adrian Rosebrock**

November 18, 2017 at 8:19 am (<https://www.pyimagesearch.com/2017/06/26/labeling-superpixel-colorfulness-opencv-python/#comment-440935>)

The most "beautiful/noticeable" color seems a bit subjective here. The colorfulness score was developed to work on aggregate (i.e., means) and are correlated with the authors' test subjects. You can't really query a single pixel using this method.

**VS**

December 6, 2017 at 11:55 am (<https://www.pyimagesearch.com/2017/06/26/labeling-superpixel-colorfulness-opencv-python/#comment-442542>)

Hi, I have created the superpixel by using SLIC. I want to pass superpixels to the classifier. For this, I have to label the superpixel by manually. It will be great, if you suggest the way.

**Adrian Rosebrock**

December 8, 2017 at 5:07 pm (<https://www.pyimagesearch.com/2017/06/26/labeling-superpixel-colorfulness-opencv-python/#comment-442714>)

I think it would be beneficial to learn what the end goal of the project is. Is there a particular reason you need to label the superpixel manually?

👋 Hey there, do you need help learning Computer Vision, Deep Learning, and OpenCV? Start by telling me about yourself.

Tap to answer

Hi Adrian, do u have any suggestion on how to use superpixel as backgroud removal? I mean, after we have superpixel, I want to extract the object presence in the image. (Maybe by comparing individual superpixel accordingly?)

**Adrian Rosebrock**

March 7, 2018 at 9:14 am (<https://www.pyimagesearch.com/2017/06/26/labeling-superpixel-colorfulness-opencv-python/#comment-452266>)

This is potentially possible but you should instead use a pixel-wise segmentation algorithm. Segmentation embodies a large amount of literature but recent approaches use deep learning. Take a look at UNet and DeepMask.

**Amy**

March 8, 2018 at 9:27 pm (<https://www.pyimagesearch.com/2017/06/26/labeling-superpixel-colorfulness-opencv-python/#comment-452461>)

Thanks Adrian for your help! 😊

**PJS**

February 20, 2019 at 3:01 pm (<https://www.pyimagesearch.com/2017/06/26/labeling-superpixel-colorfulness-opencv-python/#comment-502571>)

Thanks for all your great posts. For this one, shouldn't line16 be changed from:

```
G = np.ma.masked_array(B, mask=mask)
```

to

```
G = np.ma.masked_array(G, mask=mask)
```

👋 Hey there, do you need help learning Computer Vision, Deep Learning, and OpenCV? Start by telling me about yourself.

Tap to answer



Rupesh (<https://medium.com/@rupeshthetech>)

November 21, 2019 at 2:14 am (<https://www.pyimagesearch.com/2017/06/26/labeling-superpixel-colorfulness-opencv-python/#comment-576246>)

Hi,

I really found the article to be very interesting and informative. I found a link (<https://technology.condenast.com/story/handbag-brand-and-color-detection> (<https://technology.condenast.com/story/handbag-brand-and-color-detection>)) in which the author has explained about identifying the colour of a pixel with superpixels and Machine Learning. Can you please implement the same in Python?


Before you leave a comment...

Hey, Adrian here, author of the PyImageSearch blog. I'd love to hear from you; however, I have made the decision to no longer offer free 1:1 help over blog post comments. I simply do not have the time to moderate and respond to them all.

To that end, myself and my team are doubling down our efforts on supporting our paying customers, writing new books and courses, and authoring high quality Computer Vision, Deep Learning, and OpenCV content for you to learn from.

I'd be happy to help you with your question or project, but **I have to politely ask you to purchase one of my books or courses first.** (<https://www.pyimagesearch.com/books-and-courses/>)

Why bother becoming a PyImageSearch customer?

 **Hey there, do you need help learning Computer Vision, Deep Learning, and OpenCV? Start by telling me about yourself.** ×

Tap to answer

- You'll receive a **guaranteed response** from myself and my team.
- You'll be able to **confidently and successfully** apply Computer Vision, Deep Learning, and OpenCV to your projects.

[Click here to see my full catalog of books and courses.](https://www.pyimagesearch.com/books-and-courses/)

[\(https://www.pyimagesearch.com/books-and-courses/\)](https://www.pyimagesearch.com/books-and-courses/) Take a look and I hope to see you on the other side!

Similar articles

IMAGE PROCESSING TUTORIALS

How to Display a Matplotlib RGB Image

November 3, 2014

[\(https://www.pyimagesearch.com/2014/11/03/display-matplotlib-rgb-image/\)](https://www.pyimagesearch.com/2014/11/03/display-matplotlib-rgb-image/)



Hey there, do you need help learning Computer Vision, Deep Learning, and OpenCV? Start by telling me about yourself.

Tap to answer

January 22, 2018

(<https://www.pyimagesearch.com/2018/01/22/install-dlib-easy-complete-guide/>)



PYIMAGESEARCH GURUS

TUTORIALS

Implementing the Max RGB filter in OpenCV

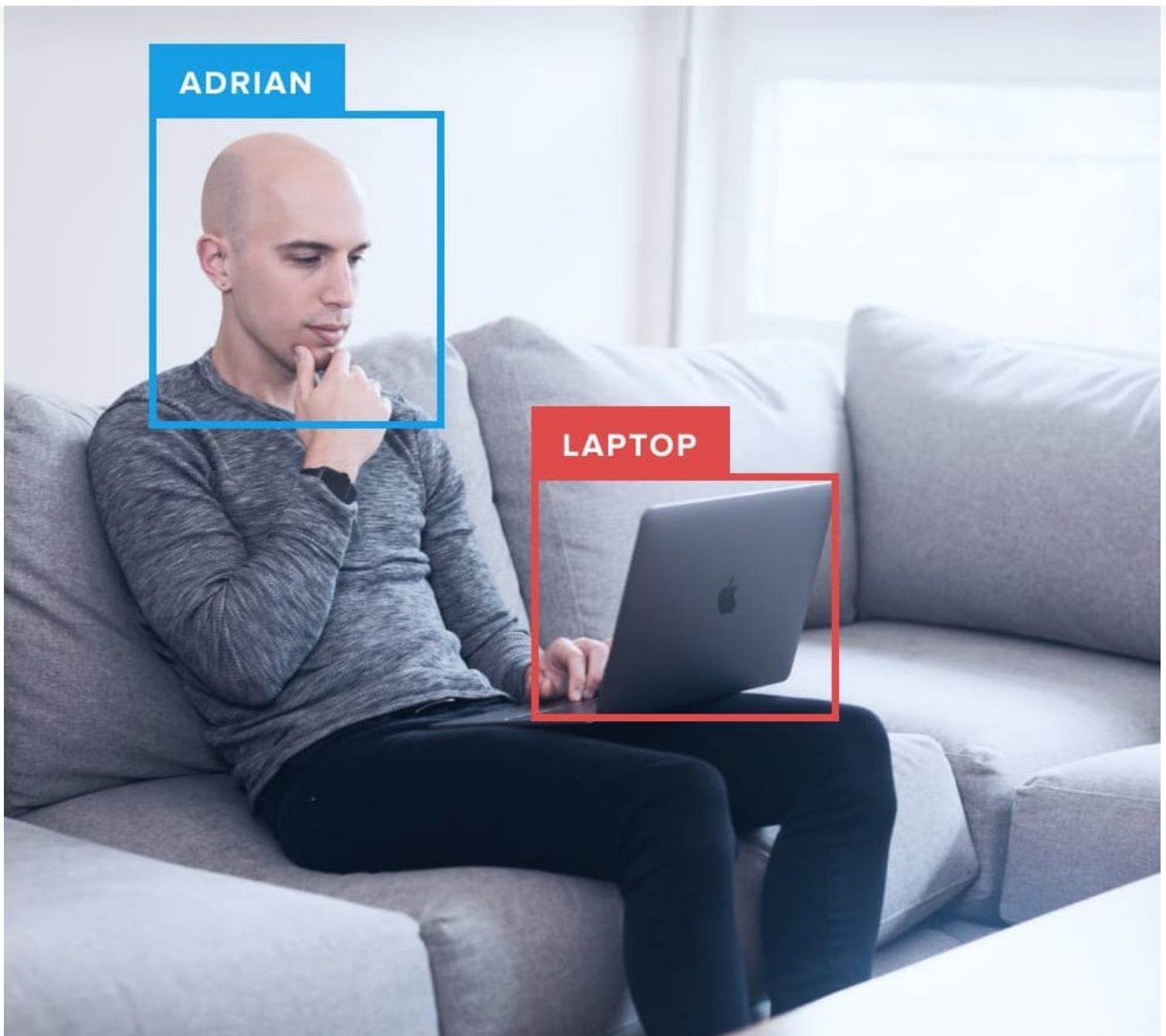
September 28, 2015

(<https://www.pyimagesearch.com/2015/09/28/implementing-the-max-rgb-filter-in-opencv/>)



Hey there, do you need help learning Computer Vision, Deep Learning, and OpenCV? Start by telling me about yourself.

Tap to answer



You can learn Computer Vision, Deep Learning, and OpenCV.

Get your FREE 17 page Computer Vision, OpenCV, and Deep Learning Resource Guide PDF. Inside you'll find my hand-picked tutorials, books, courses, and libraries to help you master CV and DL.

👋 Hey there, do you need help learning Computer Vision, Deep Learning, and OpenCV? Start by telling me about yourself.

Tap to answer

Topics

[Deep Learning](https://www.pyimagesearch.com/category/deep-learning-2/)

(<https://www.pyimagesearch.com/category/deep-learning-2/>)

[Dlib Library](https://www.pyimagesearch.com/category/dlib/)

(<https://www.pyimagesearch.com/category/dlib/>)

[Embedded/IoT and Computer Vision](https://www.pyimagesearch.com/category/embedded/)

(<https://www.pyimagesearch.com/category/embedded/>)

[Face Applications](https://www.pyimagesearch.com/category/faces/)

(<https://www.pyimagesearch.com/category/faces/>)

[Image Processing](https://www.pyimagesearch.com/category/image-processing/)

(<https://www.pyimagesearch.com/category/image-processing/>)

[Interviews](https://www.pyimagesearch.com/category/interviews/)

(<https://www.pyimagesearch.com/category/interviews/>)

[Keras](https://www.pyimagesearch.com/category/keras/)

(<https://www.pyimagesearch.com/category/keras/>)

[Machine Learning and Computer Vision](https://www.pyimagesearch.com/category/machine-learning-2/)

(<https://www.pyimagesearch.com/category/machine-learning-2/>)

[Medical Computer Vision](https://www.pyimagesearch.com/category/medical/)

(<https://www.pyimagesearch.com/category/medical/>)

[Optical Character Recognition \(OCR\)](https://www.pyimagesearch.com/category/optical-character-recognition-ocr/)

(<https://www.pyimagesearch.com/category/optical-character-recognition-ocr/>)

[Object Detection](https://www.pyimagesearch.com/category/object-detection/)

(<https://www.pyimagesearch.com/category/object-detection/>)

[Object Tracking](https://www.pyimagesearch.com/category/object-tracking/)

(<https://www.pyimagesearch.com/category/object-tracking/>)

[OpenCV Tutorials](https://www.pyimagesearch.com/category/opencv/)

(<https://www.pyimagesearch.com/category/opencv/>)

[Raspberry Pi](https://www.pyimagesearch.com/category/raspberry-pi/)

(<https://www.pyimagesearch.com/category/raspberry-pi/>)

Books & Courses

[FREE CV, DL, and OpenCV Crash Course](https://www.pyimagesearch.com/free-opencv-computer-vision-deep-learning-crash-course/)

(<https://www.pyimagesearch.com/free-opencv-computer-vision-deep-learning-crash-course/>)

[Practical Python and OpenCV](https://www.pyimagesearch.com/practical-python-opencv/)

(<https://www.pyimagesearch.com/practical-python-opencv/>)

[Deep Learning for Computer Vision with Python](https://www.pyimagesearch.com/deep-learning-computer-vision-with-python/)

(<https://www.pyimagesearch.com/deep-learning-computer-vision-with-python/>)

PyImageSearch

[Get Started](https://www.pyimagesearch.com/start-here/) (<https://www.pyimagesearch.com/start-here/>)

[OpenCV Install Guides](https://www.pyimagesearch.com/opencv-tutorials-resources-guides/)

(<https://www.pyimagesearch.com/opencv-tutorials-resources-guides/>)

[About](https://www.pyimagesearch.com/about/) (<https://www.pyimagesearch.com/about/>)

[FAQ](https://www.pyimagesearch.com/faqs/) (<https://www.pyimagesearch.com/faqs/>)



Hey there, do you need help learning Computer Vision, Deep Learning, and OpenCV? Start by telling me about yourself.

Tap to answer

[gurus/](#)

[Raspberry Pi for Computer Vision](#)

(<https://www.pyimagesearch.com/raspberry-pi-for-computer-vision/>)

[Privacy Policy](#)


(<https://www.pyimagesearch.com/privacy-policy/>)

[f](https://www.facebook.com/pyimagesearch) (<https://www.facebook.com/pyimagesearch>) [t](https://twitter.com/PyImageSearch) (<https://twitter.com/PyImageSearch>)

[in](http://www.linkedin.com/pub/adrian-rosebrock/2a/873/59b) (<http://www.linkedin.com/pub/adrian-rosebrock/2a/873/59b>) [y](https://www.youtube.com/channel/UCoQK7OVcIVy-nV4m-SMcK_Q/videos)

(https://www.youtube.com/channel/UCoQK7OVcIVy-nV4m-SMcK_Q/videos)

© 2020 [PyImageSearch](https://www.pyimagesearch.com) (<https://www.pyimagesearch.com>). All Rights Reserved.

 Hey there, do you need help learning Computer Vision, Deep Learning, and OpenCV? Start by telling me about yourself. ✕

Tap to answer