1.13: Connected-component labeling

(https://gurus.pyimagesearch.com/wp-content/uploads/2015/08/connected_components_final.jpg)

Connected-component labeling (also known as *connected-component analysis*, *blob extraction*, or *region labeling*) is an algorithmic application of graph theory (https://en.wikipedia.org/wiki/Graph_theory) that is used to determine the connectivity of "blob"-like regions in a binary image.

We often use connected-component analysis in the same situations that contours (https://gurus.pyimagesearch.com/lessons/contours/) are used; however, connected-component labeling can often give us a more *granular filtering* of the blobs in a binary image. When using contour analysis, we are often restricted by the hierarchy of the outlines (i.e. one contour contained within another), but with connected-component analysis we can more easily segment and analyze these structures.

Once we have extracted the blob using connected-component labeling, we can still apply contour properties to quantify the region. A great example usage of connected-component analysis is to compute the connected-components of a binary (i.e. thresholded) license plate image and filter the blobs based on their properties, such as width, height, area, solidity, etc.

Objectives:

In this lesson, we will:

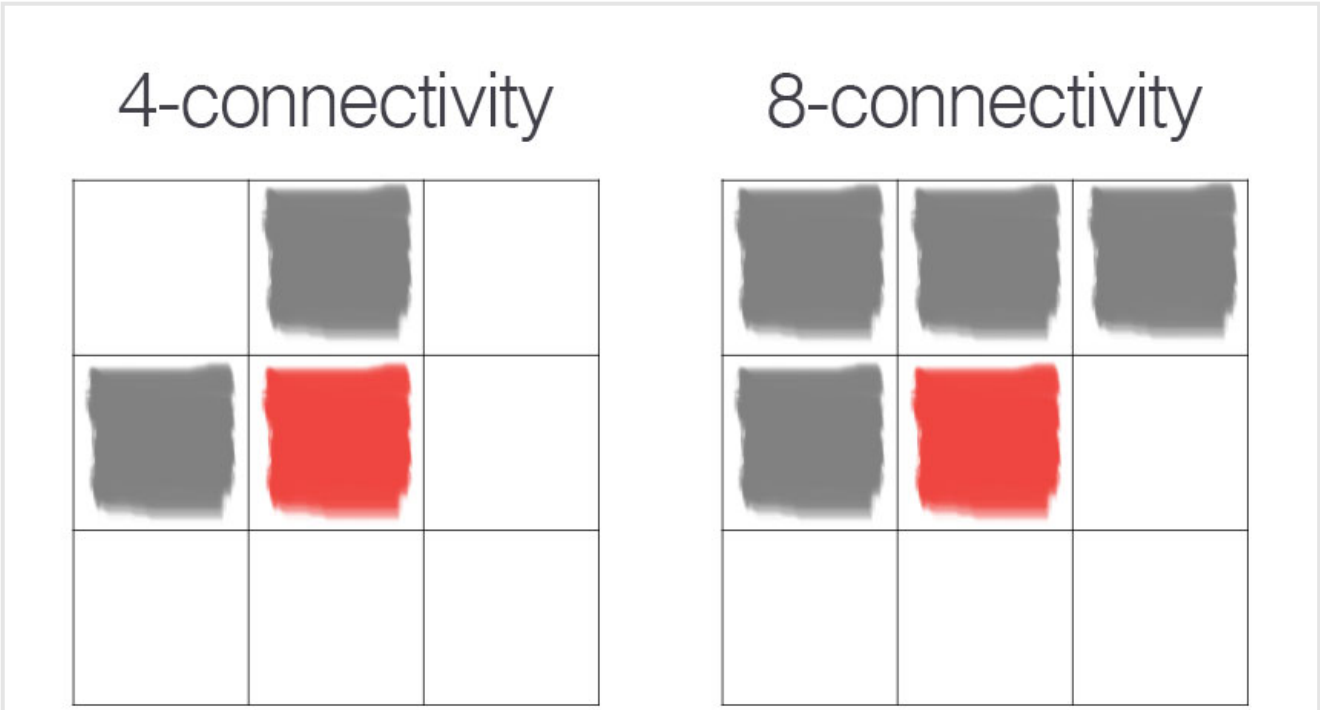
- Review the classical two-pass algorithm used for connected-component analysis.
- Apply connected-component analysis to detect characters and blobs in a license plate image.

The classical approach

The classical connected-component analysis was introduced by Rosenfeld and Pfaltz in their 1966 article, *Sequential Operations in Digital Picture Processing (https://gurus.pyimagesearch.com/wp-content/uploads/2015/08/rosenfeld-1966.pdf)*. Their clever use of graph theory to analyze connected-components in an image is very efficient and is still heavily used today.

It's important to note that we only apply connected-component analysis to ***binary*** or ***thresholded*** images. If presented with an RGB or grayscale image, we first need to threshold it based on some criterion in a manner that can segment the background from the foreground, leaving us with "blobs" in the image that we can examine. Once we have obtained the binary version of the image, we can proceed to analyze the components.

The actual algorithm consists of two passes. In the first pass, the algorithm loops over each individual pixel. For each center pixel *p*, the west and north pixels are checked:



(https://gurus.pyimagesearch.com/wp-content/uploads/2015/08/connected_components_connectivity.jpg)

FIGURE 1: EXAMPLES OF 4-CONNECTIVITY *(LEFT)* AND 8-CONNECTIVITY *(RIGHT)* FOR CONNECTED-COMPONENT LABELING.

This type of check is called *4-connectivity (left)*. Based on the west and north pixel labels, a label is assigned to the current center pixel *p* (we'll discuss this step in more detail in the **first pass** section of this lesson).

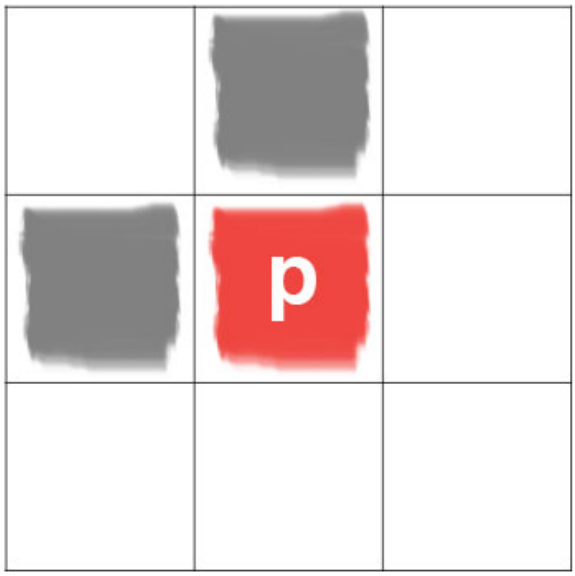
You might be wondering why only *two* pixels are being checked if we want to check the pixels surrounding *p* for *4-connectivity*. The reason is because we are looping over each pixel *individually* and *always* checking the west and north pixels. By repeating this process over the entire image, one row at a time, each pixel will actually be checked for 4-connectivity.

8-connectivity can also be performed by checking the west, north-west, north, and north-east pixels *(right)*.

Then, in the second pass, the connected-component analysis algorithm loops over the labels generated from the first pass and merges any regions together that share connected labels.

The first pass

In the first pass of our connected-component analysis algorithm, every pixel is checked. For the sake of this example, we'll use 4-connectivity (but we could just as easily use 8-connectivity) and check the west and north pixels of the central pixel *p*:



(https://gurus.pyimagesearch.com/wp-content/uploads/2015/08/connected_components_4connectivity.jpg)

FIGURE 2: WHEN PERFORMING 4-CONNECTIVITY LABELING, WE CHECK THE NORTH AND WEST PIXELS OF THE CENTRAL PIXEL *P*.

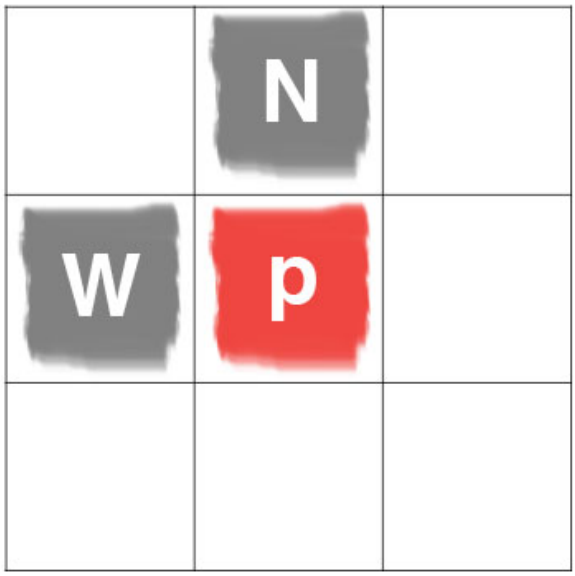
Step 1

The first step is to check if we care about the central pixel *p* or not:

- If the central pixel is a *background pixel* (normally a value of *0*, indicating black), we ignore it and move to the next pixel.
- If it is a *foreground* pixel, or if we have moved to a pixel that is in the foreground, we proceed to **Steps 2 and 3**.

Steps 2 and 3

If we have reached this step, then we must be examining a foreground pixel, so we grab the north and west pixels, denoted as *N* and *W*, respectively:



(https://gurus.pyimagesearch.com/wp-content/uploads/2015/08/connected_components_step2.jpg)

FIGURE 3: GRABBING THE NORTH AND WEST PIXELS SURROUNDING THE CENTER PIXEL.

Now that we have *N* and *W*, there are two possible situations:

1. Both *N* and *W are* background pixels, so there are no labels associated with these pixels. In this case, create a new label (normally by incrementing a unique label counter) and store the label value in the current pixel. Then move on to **Steps 4 and 5**.
2. *N* and/or *W are not* background pixels. If this is the case, we can already proceed to **Steps 4 and 5** since at least one pixel already has a label associated with it.

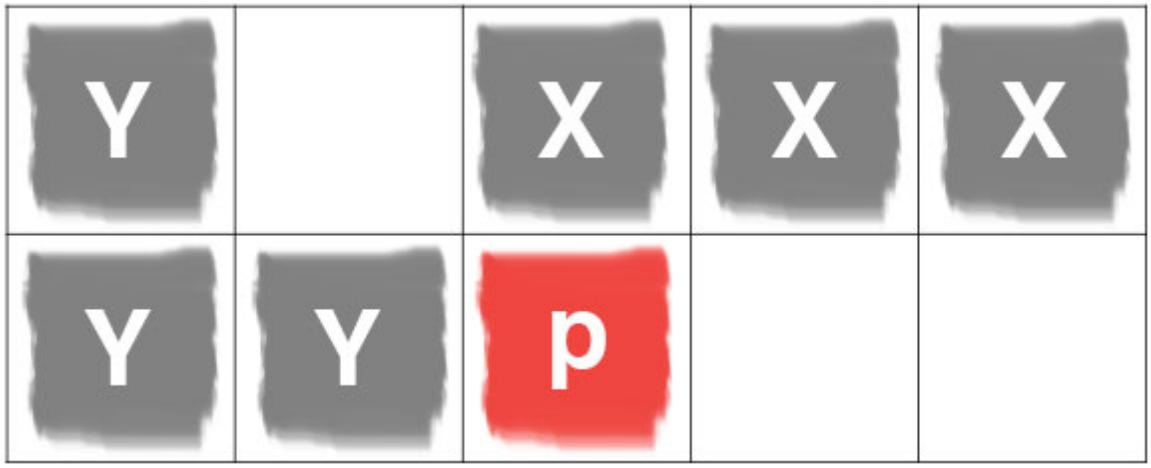
Steps 4 and 5

This is an easy step. All we need to do is set the center pixel *p* by taking the minimum of the label value:

$$p = min(N, W)$$

Step 6

Suppose that, in the following figure, the north pixel has label *X* and the west pixel has label *Y*:



(https://gurus.pyimagesearch.com/wp-content/uploads/2015/08/connected_components_labels.jpg)

FIGURE 4: DESPITE THE GRAY STRUCTURE IN THE IMAGE CONTAINING TWO SEPARATE LABELS, WE KNOW THEY ARE PART OF THE SAME COMPONENT SINCE PIXELS OF THE TWO LABELS "TOUCH" AND ARE THUS A SINGLE OBJECT.

Even though these pixels have two separate labels, we know they are actually connected and part of the same blob. To indicate that the *X* and *Y* labels are part of the same component, we can leverage the **union-find data structure (https://en.wikipedia.org/wiki/Disjoint-set_data_structure)** to indicate that *X* is a child of *Y*. We'll insert a node in our union-find structure to indicate that *X* is a child of *Y* and that the pixels are actually *connected* even though they have different label values.

The second pass of our connected-components algorithm will leverage the union-find structure to connect any blobs that have different labels but are actually part of the same blob.

Step 7

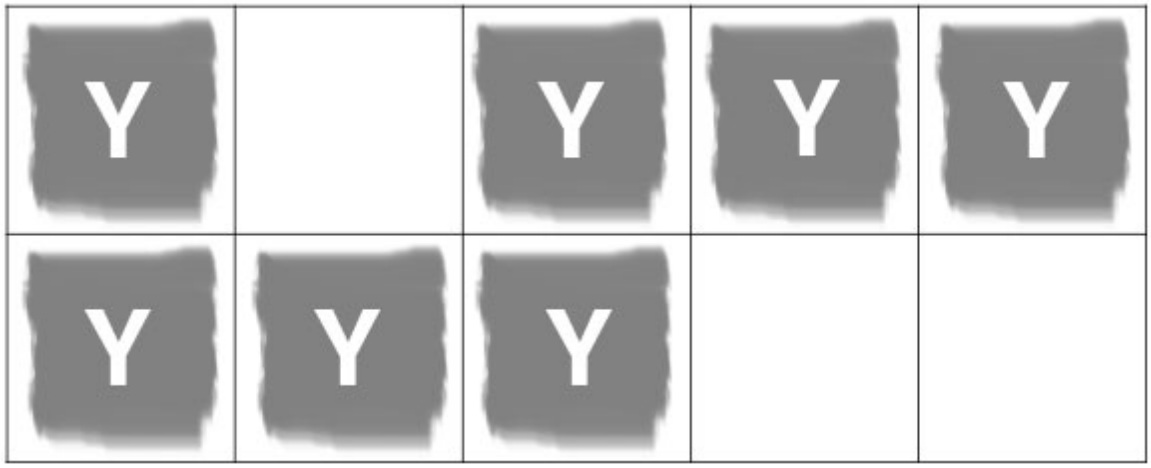
Continue to the next pixel and go repeat the process beginning with **Step 1**.

The second pass

The second pass of the connected-components labeling algorithm is much simpler than the first one. We start off by looping over the image once again, one pixel at a time.

For each pixel, we check if the label of the current pixel is a *root* (i.e. top of the tree) in the union-find data structure. If so, then we can proceed on to the next step — the label of the current pixel already has the smallest possible value based on how it is connected to its neighbors.

Otherwise, we follow the tree until we reach a root in the structure. Once we have reached a root, we assign the value at the root to the current pixel:



(https://gurus.pyimagesearch.com/wp-content/uploads/2015/08/connected_components_labels_complete.jpg)

FIGURE 5: AFTER APPLYING OUR UNION-FIND DATA STRUCTURE, THE BLOB HAS BEEN MERGED INTO A SINGLE CONNECTED COMPONENT.

By applying this second pass, we can connect blobs with different label values but that are actually part of the same blob. The key to efficiency is to use the union-find data structure for tree-traversal when examining label values.

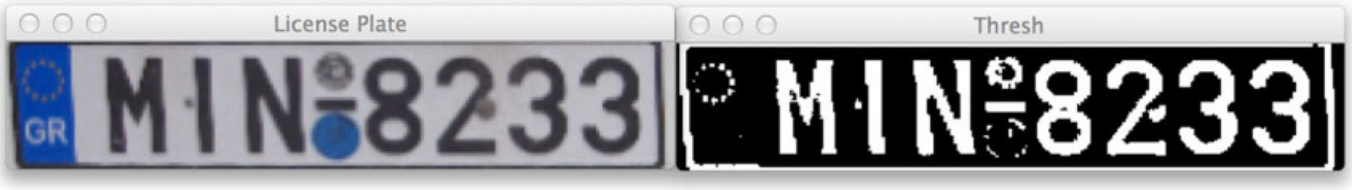
Applying connected-component analysis to license plate images

Oddly enough, being the *de facto* computer vision library, you would think that OpenCV has an easy way to perform connected-component analysis — *unfortunately it does not.*

Note: OpenCV 3.0 claims to have connected-component support, but I have not tried it out, and I'm unsure if the Python bindings are exposed.

Luckily, we have the scikit-image (http://scikit-image.org/) library which comes with a dead-simple method to perform connected component labeling. Even *if* OpenCV had a connected-component analysis function, I don't think it would be as straightforward and easy to use as the one provided by scikit-image.

Let's start by taking a look at the problem we are going to be solving using connected-component labeling:



(https://gurus.pyimagesearch.com/wp-content/uploads/2015/08/connected_components_plate_and_thresh.jpg)

FIGURE 6: LICENSE PLATE *(LEFT)* AND CORRESPONDING THRESHOLDED IMAGE *(RIGHT)*.

On the *left*, you can see an image of a license plate (which we will learn how to obtain in our **ANPR (https://gurus.pyimagesearch.com/lessons/what-is-anpr/)** module), and on the *right*, we can see the **thresholded (https://gurus.pyimagesearch.com/lessons/thresholding/)** binary image of the license plate.

Our goal is to use connected-component analysis to label each of the white "blobs" in the license plate and then analyze each of these blobs to determine *which regions are license plate characters* and *which ones can be discarded*. Again, we'll be able to do all of this using basic connected-component analysis.

With that said, let's jump into some code and see just how easy it is to perform connected-component labeling using scikit-image:

Feedback

connected_components_labeling.py — Python

```
1  # import the necessary packages
2  from skimage.filters import threshold_local
3  from skimage import measure
4  import numpy as np
5  import cv2
6
7  # load the license plate image from disk
8  plate = cv2.imread("license_plate.png")
9
10 # extract the Value component from the HSV color space and apply adaptive thresholding
11 # to reveal the characters on the license plate
12 V = cv2.split(cv2.cvtColor(plate, cv2.COLOR_BGR2HSV))[2]
13 T = threshold_local(V, 29, offset=15, method="gaussian")
14 thresh = (V < T).astype("uint8") * 255
15
16 # show the images
17 cv2.imshow("License Plate", plate)
18 cv2.imshow("Thresh", thresh)
```

We start off importing our necessary packages on **Lines 2-5**. All of these imports should look fairly familiar, with the exception of the `measure` sub-package from scikit-image. As we'll see in the next codeblock, the `measure` module contains our connected-component analysis method.

From there, we load our license plate image from disk, grab the Value channel from the HSV color space (I explain why we use the Value channel rather than a simple conversion to grayscale in our **license plate localization lesson (https://gurus.pyimagesearch.com/lessons/segmenting-characters-from-the-license-plate/)**), and perform adaptive thresholding to obtain our binary image displayed in **Figure 6** above.

Now that we have our thresholded image, we can apply connected-component analysis on it:

connected_components_labeling.py Python

```
20 # perform connected components analysis on the thresholded images and initialize the
21 # mask to hold only the "large" components we are interested in
22 labels = measure.label(thresh, neighbors=8, background=0)
23 mask = np.zeros(thresh.shape, dtype="uint8")
24 print("[INFO] found {} blobs".format(len(np.unique(labels))))
```

On **Line 22**, we make a call to the `label` method of `measure`, which performs our actual connected-component labeling. The `label` method requires a single argument, which is our binary `thresh` image that we want to extract connected-components from. We'll also supply `neighbors=8` to indicate we want to perform connected-component analysis with *8-connectivity*. Finally, the optional `background` parameter indicates that all pixels with a value of *0* should be considered *background* and ignored by the `label` method.

The `label` method returns `labels`, a NumPy array with the same dimension as our `thresh` image. Each *(x, y)*-coordinate inside `labels` is either *0* (indicating that the pixel is background and can be ignored) or a value *> 0*, which indicates that it is part of a connected-component. Each unique connected-component in the image has a unique label inside `labels`.

Now that we have the `labels`, we can loop over them individually and analyze each one to determine if it is a license plate character or not:

connected_components_labeling.py Python

```
26 # loop over the unique components
27 for (i, label) in enumerate(np.unique(labels)):
28     # if this is the background label, ignore it
29     if label == 0:
30         print("[INFO] label: 0 (background)")
31         continue
32 
33     # otherwise, construct the label mask to display only connected components for
34     # the current label
35     print("[INFO] label: {} (foreground)".format(i))
36     labelMask = np.zeros(thresh.shape, dtype="uint8")
37     labelMask[labels == label] = 255
38     numPixels = cv2.countNonZero(labelMask)
39 
40     # if the number of pixels in the component is sufficiently large, add it to our
41     # mask of "large" blobs
42     if numPixels > 300 and numPixels < 1500:
43         mask = cv2.add(mask, labelMask)
44 
45     # show the label mask
46     cv2.imshow("Label", labelMask)
47     cv2.waitKey(0)
48 
49 # show the large components in the image
50 cv2.imshow("Large Blobs", mask)
51 cv2.waitKey(0)
```

On **Line 27**, we start looping over each unique `label` inside `labels`.

If the current label is *0*, then we know we are examining the background, and we can safely ignore it (**Line 29-31**).

Note: In versions of `scikit-image` <= 0.11.X, the background label was originally *-1*. However, in newer versions of `scikit-image` (such as >= 0.12.X), the background label is *0*. Make sure you check which version of `scikit-image` you are using and update the code to use the correct background label as this *can* affect the output of the script.

However, in the case we are examining a foreground label, we construct a `labelMask` with the same dimensions as our `thresh` image. We then set all *(x, y)*-coordinates in `labelMask` that belong to the current `label` in `labels` to *white* (**Line 37**) — here, we are simply drawing the current blob on the `labelMask` image.

Last, all we need to do is determine if the current blob is a license plate character or not. For this particular problem, this filtering is actually quite simple — all we need to do is use the `cv2.countNonZero` to count the number of non-zero pixels in the `labelMask` (**Line 38**) and then make a check to see if

`numPixels` falls inside an acceptable range (**Lines 42 and 43**) to ensure that the blob is neither too small nor too big. Provided that `numPixels` passes this test, we accept the blob as being a license plate character.

Lines 46 and 47 display the *current* connected-component to our screen, and **Lines 50 and 51** display the final output image, showing only the license plate characters and none of the other blobs.

To execute our script, just open up a terminal and issue the following command:

```
connected_components_labeling.py                                    Shell
1 $ python connected_components_labeling.py
```

First, you'll notice our license plate image *(top)*, the thresholded image *(middle)*, and the current-connected component *(bottom)*. In this case, the current blob is the outer license plate tray:

(https://gurus.pyimagesearch.com/wp-content/uploads/2015/08/connected_components_initial.jpg)

FIGURE 7: PERFORMING A CONNECTED-COMPONENT ANALYSIS ON OUR LICENSE PLATE IMAGE.

After looping over all of the unique connected-components in the image, we'll finally arrive at the following screenshot, indicating that we have successfully filtered the license plate character blobs from the non-license plate character blobs:

(https://gurus.pyimagesearch.com/wp-content/uploads/2015/08/connected_components_final.jpg)

FIGURE 8: BY PERFORMING CONNECTED-COMPONENT LABELING, WE HAVE BEEN ABLE TO FILTER THE LICENSE PLATE CHARACTER BLOBS FROM THE REST OF THE REGIONS IN THE IMAGE.

Feedback

Below, you can see an animation of looping over *each* of the unique connected-components in the image:

(https://gurus.pyimagesearch.com/wp-content/uploads/2015/08/connected_components_animated.gif)

FIGURE 9: EACH OF THE 34 UNIQUE BLOBS IN OUR LICENSE PLATE IMAGE.

Summary

In this lesson, we learned about the classical Rosenfeld and Pfatlz algorithm for connected-component analysis. This algorithm consists of two passes:

- The first pass labels pixels in a binary input image based on either *4* or *8-connectivity*.
- The second pass connects regions of the image that are part of the same blob but have different labels.

Finally, we applied connected-component analysis to detect characters in a thresholded license plate image.

Downloads:

Download the Code (https://gurus.pyimagesearch.com/protected/code/computer_vision_basics/conn

	Quizzes	Status
1	Connected-component Labeling Quiz (https://gurus.pyimagesearch.com/quizzes/connected-component-labeling-quiz/)	

← Previous Lesson (https://gurus.pyimagesearch.com/lessons/histograms/) Next Lesson → (https://gurus.pyimagesearch.com/lessons/what-are-object-detectors/)

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)

Search