2.1.2: Template matching

Topic Progress: (https://gurus.pyimagesearch.com/topic/an-introduction-to-object-detection/)

(https://gurus.pyimagesearch.com/topic/template-matching/)

← Back to Lesson (https://gurus.pyimagesearch.com/lessons/what-are-object-detectors/)

So now that we have a better understanding of what object detectors actually are, let's take a look at the most basic form of object detection: *template matching*.

Template matching is both *extremely simple* and *practically effortless* but has a major drawback of only working under *very specific* (or highly controlled) conditions.

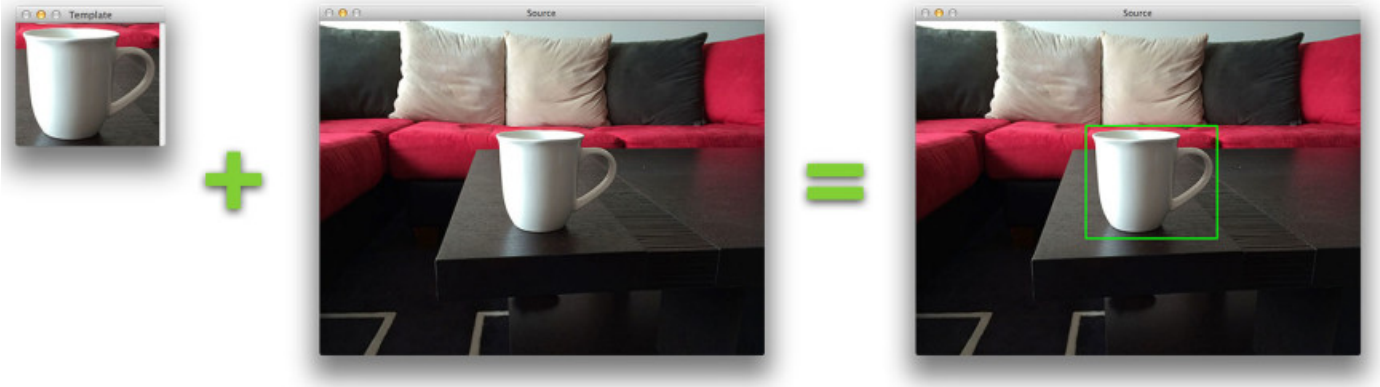
In the rest of this lesson, we'll explore template matching, apply it to detect objects in images, and then check out a few examples that show why it's not the best solution for object detection in most cases.

Objectives:

In this lesson, we will:

- Review the template matching algorithm.
- Apply template matching for object detection using OpenCV.
- Explore the limitations of template matching.

A first try at object detection: template matching



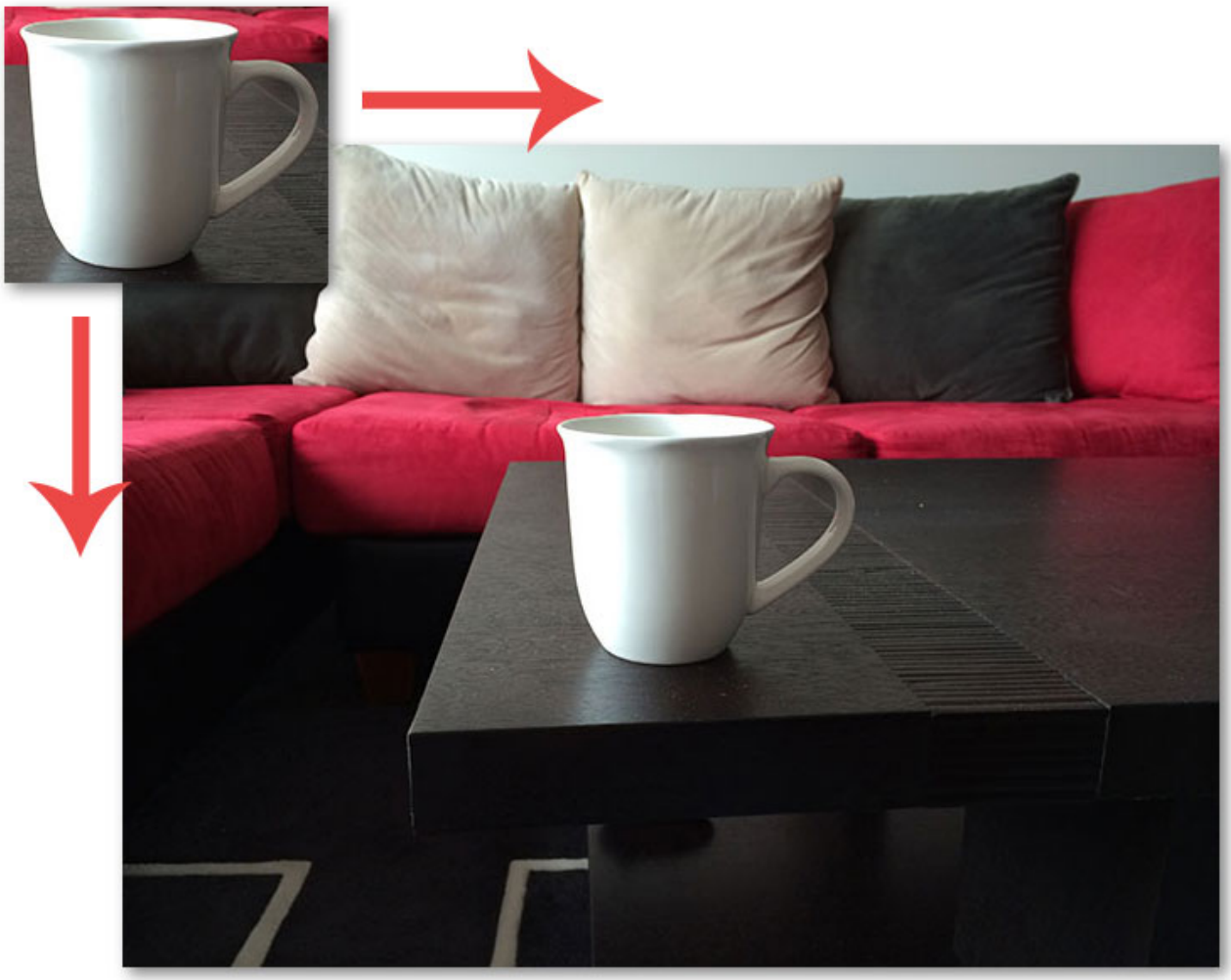
(https://gurus.pyimagesearch.com/wp-content/uploads/2015/08/template_matching_pipeline.jpg)

FIGURE 1: AN EXAMPLE OF THE TEMPLATE MATCHING PIPELINE WHERE WE TAKE THE OBJECT/TEMPLATE WE WANT TO DETECT *(LEFT)*, THE SOURCE IMAGE *(MIDDLE)*, AND THEN FINDING THE TEMPLATE IN THE SOURCE IMAGE *(RIGHT)*.

Before we can even apply template matching, we need to gather two images:

1. **Source image**: This is the image we expect to find a match to our template in.
2. **Template image:** The "object patch" we are searching for in the *source image*.

To find the template in the source image, we slide the template from left-to-right and top-to-bottom across the source:



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

FIGURE 1: APPLYING TEMPLATE MATCHING IS AS SIMPLE AS SLIDING THE TEMPLATE FROM LEFT-TO-RIGHT AND TOP-TO-BOTTOM ACROSS THE SOURCE AND COMPUTING A METRIC TO INDICATE HOW GOOD OR BAD THE MATCH IS AT EACH LOCATION.

Feedback

At each *(x, y)* location, a metric is calculated to represent how "good" or "bad" the match is. Normally, we use the correlation coefficient to determine how "similar" the pixel intensities of the two patches are:

$$R(x,y) = \sum_{x',y'} (T'(x',y') \cdot I'(x+x',y+y'))$$

(https://gurus.pyimagesearch.com/wp-content/uploads/2015/08/template_matching_ccoeff_part1.png)

Where:

$$T'(x',y') = T(x',y') - 1/(w \cdot h) \cdot \sum_{x'',y''} T(x'',y'')$$
$$I'(x+x',y+y') = I(x+x',y+y') - 1/(w \cdot h) \cdot \sum_{x'',y''} I(x+x'',y+y'')$$

(https://gurus.pyimagesearch.com/wp-content/uploads/2015/08/template_matching_ccoeff_part2.png)

For the full derivation of the correlation coefficient, please see the OpenCV documentation (http://docs.opencv.org/modules/imgproc/doc/object_detection.html).

For each location *T* over *I*, the computed result metric is stored in our result matrix *R*. Each *(x, y)*-coordinate in the source image (that also has a valid width and height for the template image) contains an entry in the result matrix *R*:

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

FIGURE 3: AN EXAMPLE OF LAYERING THE RESULT MATRIX R OVER THE SOURCE IMAGE. NOTICE HOW THE BRIGHTEST REGION OF THE RESULT MATRIX OCCURS AT THE UPPER-LEFT CORNER OF THE COFFEE MUG, THUS INDICATING CORRELATION IS AT ITS MAXIMUM.

Here, we can see a visualization of our result matrix *R* overlaid on top of the original image. Notice how *R* is not the same size as the original template. This is because the *entire* template must fit inside the source image for the correlation to be computed. If the template exceeds the boundaries of the source, we do not compute the similarity metric.

Bright locations of the result matrix *R* indicate the best matches, where dark regions indicate there is very little correlation between the source and template images. Notice how the brightest region of the result matrix appears at the upper-left corner of the coffee mug.

While template matching is extremely simple to apply, it's only useful in a very limited set of circumstances. In nearly all cases, you'll want to ensure that the template you are detecting is ***nearly identical*** to the object you want to detect in the source. Even *small*, *minor deviations* in appearance can dramatically affect the results from template matching and render it effectively useless.

Template matching in action

To demonstrate template matching, let's code up a quick example to detect the coffee mug in the following image:

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

FIGURE 4: OUR GOAL IS TO APPLY TEMPLATE MATCHING TO DETECT THE COFFEE MUG IN THIS IMAGE.

Open up a new file, name it `template_matching.py` , and let's get coding:

template_matching.py Python

```
1  # import the necessary packages
2  import argparse
3  import cv2
4
5  # construct the argument parser and parse the arguments
6  ap = argparse.ArgumentParser()
7  ap.add_argument("-s", "--source", required=True, help="Path to the source image")
8  ap.add_argument("-t", "--template", required=True, help="Path to the template image")
9  args = vars(ap.parse_args())
10
11 # load the source and template image
12 source = cv2.imread(args["source"])
13 template = cv2.imread(args["template"])
14 (tempH, tempW) = template.shape[:2]
15
16 # find the template in the source image
17 result = cv2.matchTemplate(source, template, cv2.TM_CCOEFF)
18 (minVal, maxVal, minLoc, (x, y)) = cv2.minMaxLoc(result)
```

This first code segment is pretty simple. We start off by importing our packages and parsing command line arguments. We need two switches here: `--source`, which is the path to our source image (i.e. the image containing the coffee mug), and `--template`, the path to our `template` image (i.e. the cropped region of the coffee mug itself) we want to detect in the `source`.

We load both the `source` and `template` from disk on **Lines 12 and 13**, followed by performing the actual template matching on **Line 17** using the `cv2.matchTemplate` function.

The `cv2.matchTemplate` method requires three parameters: the `source` image, the `template`, and the template matching method. We'll pass in `cv2.TM_CCOEFF` to indicate we want to use the correlation coefficient method. A full list of matching functions can be found in the official OpenCV documentation (http://docs.opencv.org/modules/imgproc/doc/object_detection.html).

After calling `cv2.matchTemplate`, we are left with the `result` matrix, where at each *(x, y)* location, a metric is calculated to represent how "good" or "bad" the match is. Brighter regions of the `result` correspond to a higher correlation between the `source` and `template`, and darker regions indicate very little match between the two regions.

Now that we have computed the `result` matrix, we can apply the `cv2.minMaxLoc` function to find the *(x, y)*-coordinates of the best match. We pass in our `result`, and in return we receive a 4-tuple consisting of the minimum value in the `result`, the maximum value in `result`, the *(x, y)*-coordinates of the minimum value, and the *(x, y)*-coordinates of the maximum value, respectively (**Line 18**).

Finally, we take the coordinates of the best match and then draw a bounding box on the `source` image:

template_matching.py Python

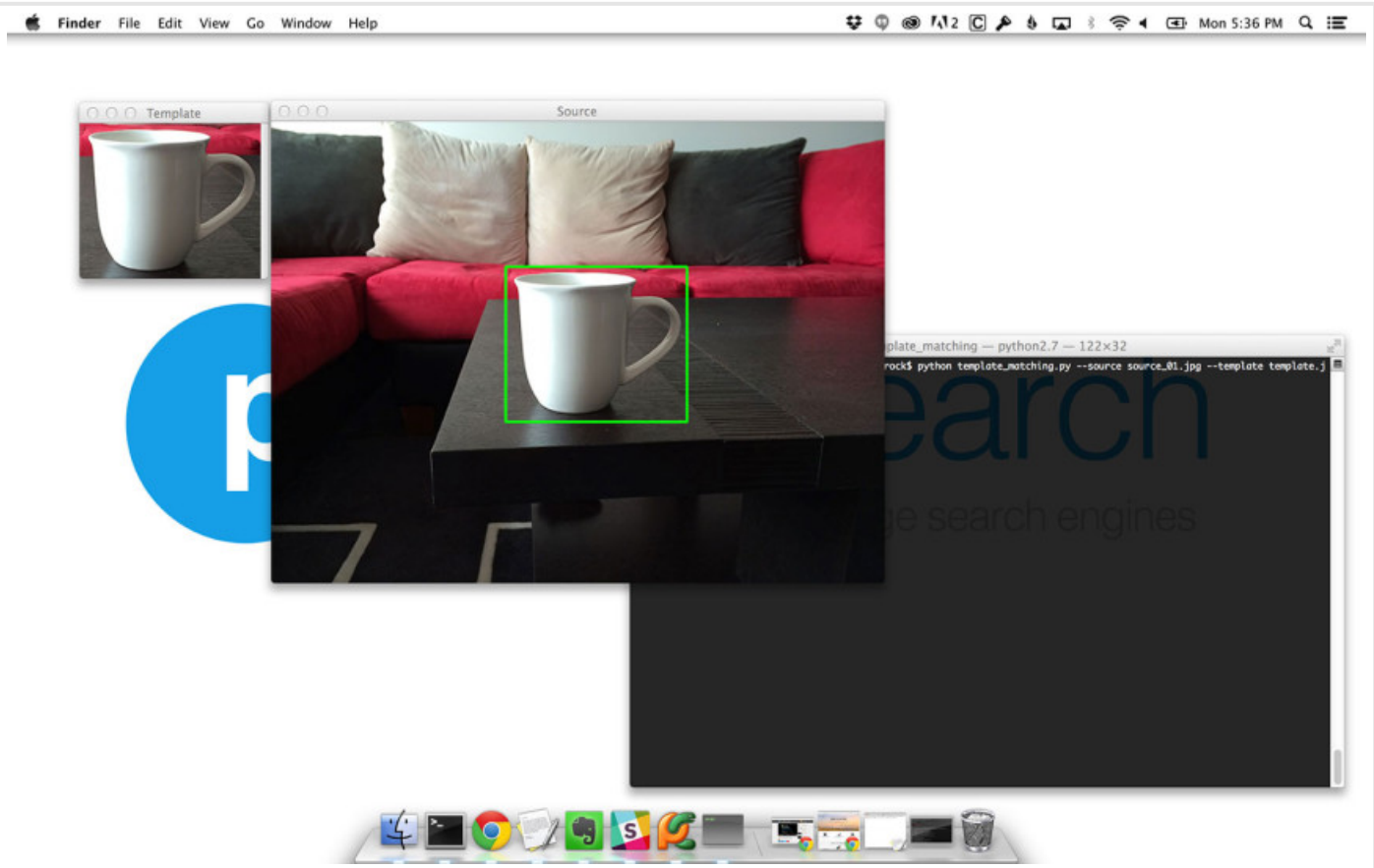
```python
# draw the bounding box on the source image
cv2.rectangle(source, (x, y), (x + tempW, y + tempH), (0, 255, 0), 2)

# show the images
cv2.imshow("Source", source)
cv2.imshow("Template", template)
cv2.waitKey(0)
```

To see our script in action, just execute the following command:

template_matching.py Shell

```
$ python template_matching.py --source source_01.jpg --template template.jpg
```



(https://gurus.pyimagesearch.com/wp-content/uploads/2015/08/template_matching_source_01.jpg)

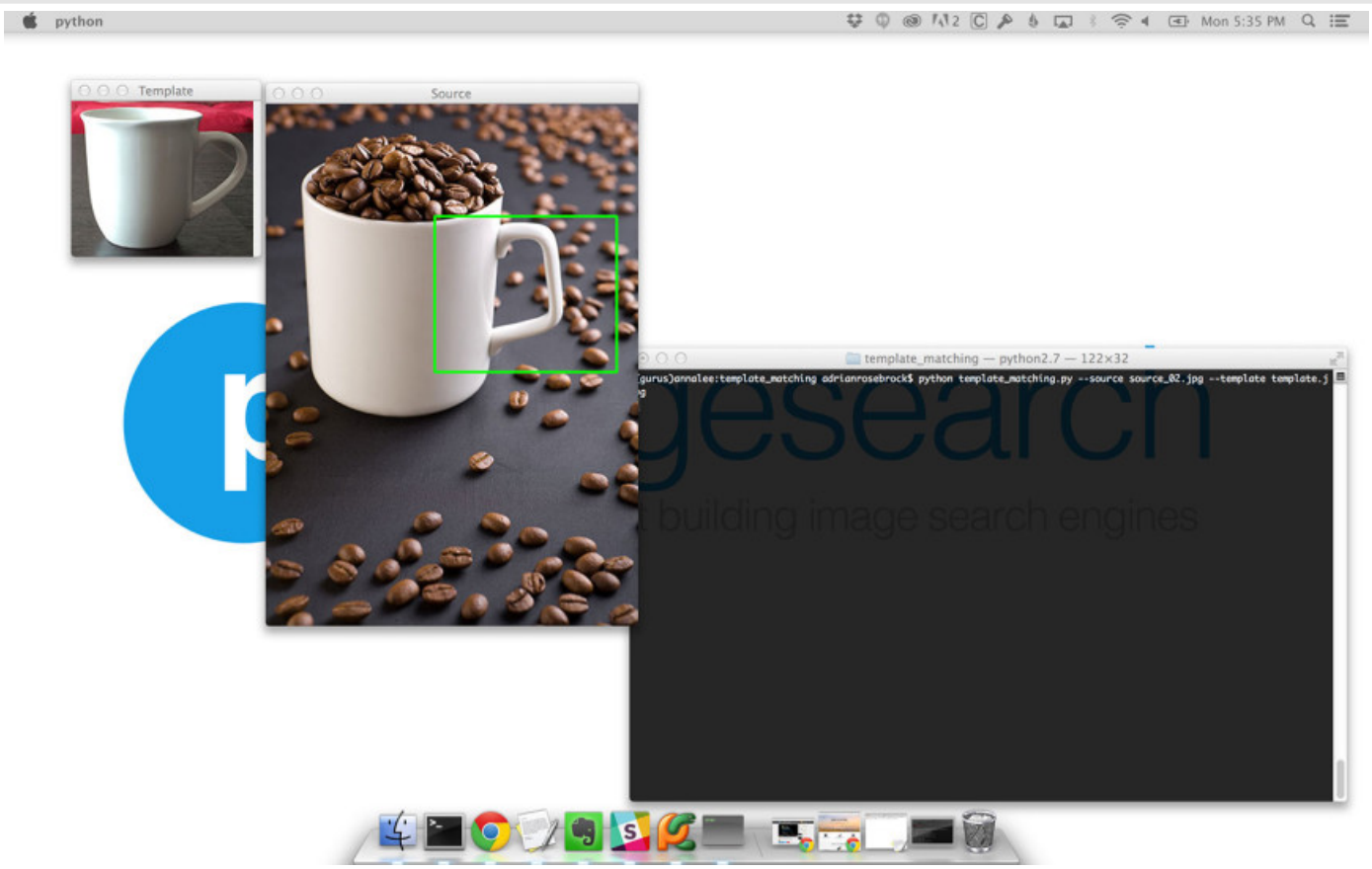
FIGURE 5: AN EXAMPLE OF APPLYING TEMPLATE MATCHING TO DETECT A COFFEE MUG.

On the left, we have our template image, and on the right, we have detected the template in the source. Pretty simple, right?

However, let's try another image where the `template` is not identical to the `source` :

template_matching.py Shell

```
$ python template_matching.py --source source_02.jpg --template template.jpg
```



(https://gurus.pyimagesearch.com/wp-content/uploads/2015/08/template_matching_source_02.jpg)

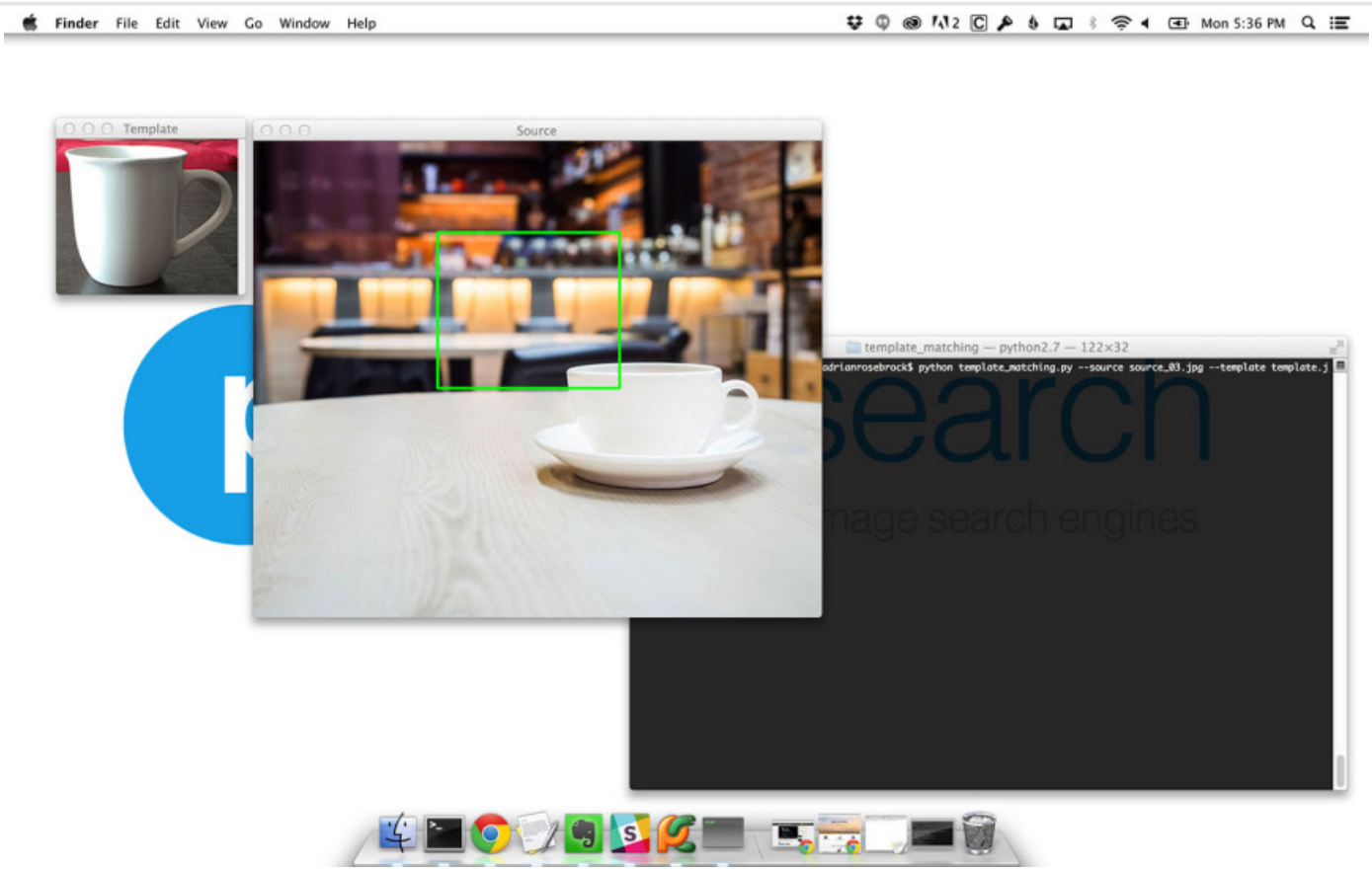
FIGURE 6: A SECOND EXAMPLE OF APPLYING TEMPLATE MATCHING, THIS TIME WHEN THE TEMPLATE IS NOT IDENTICAL TO THE SOURCE.

This time, we were not able to detect the full coffee mug, only the handle. The reason for this is two-fold. First, the mug in the `template` is not the same scale (i.e. width and height) as the mug in the `source` . Therefore, the bounding box we detect is only going to be as large as the `template` . Unless we extend our code to apply **multi-scale template matching (http://www.pyimagesearch.com/2015/01/26/multi-scale-template-matching-using-python-opencv/)**, we will never be able to place a full bounding box surrounding the larger coffee mug.

The second reason is that template matching is not robust enough to detect objects in source images that visually deviate from their template since we are computing similarity metrics over the *raw pixel intensities* instead of *feature vectors* or *abstract representations* of the image. This fact becomes even more apparent in the following example:

template_matching.py Shell

```
1 $ python template_matching.py --source source_03.jpg --template template.jpg
```



(https://gurus.pyimagesearch.com/wp-content/uploads/2015/08/template_matching_source_03.jpg)

FIGURE 7: AN EXAMPLE OF THE LIMITATIONS OF TEMPLATE MATCHING.

This time, our template matching code fails to detect the coffee mug altogether, even though both mugs are approximately the same scale in both the `template` and `source` images. In order to detect the coffee mug in this image, we'll need to apply more powerful object detection methods.

Summary

In this lesson, we reviewed the basics of template matching for object detection. As we saw, the template matching algorithm is very simple, easy to understand, and straightforward to implement.

However, template matching only works under very specific and/or controlled conditions. It is obviously not adaptable to dramatic changes in viewpoint, scale, occlusion, etc.

In order to build more powerful object detectors, we'll need to leverage more advanced computer vision and machine learning techniques, including *image pyramids*, *sliding windows*, and *non-maxima suppression*.

But before we dive too deep into the theory and inner-workings of state-of-the-art object detectors, let's keep things practical and have some fun — in our next lesson, I'll show you how to build a powerful object detector with little effort. You might be surprised at how easy it is given the right set of tools.

Downloads:

Download the Code (https://gurus.pyimagesearch.com/protected/code/object_detector/template_ma

Quizzes		Status
1	Template Matching Quiz (https://gurus.pyimagesearch.com/quizzes/template-matching-quiz/)	

← Previous Topic (https://gurus.pyimagesearch.com/topic/an-introduction-to-object-detection/)

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)