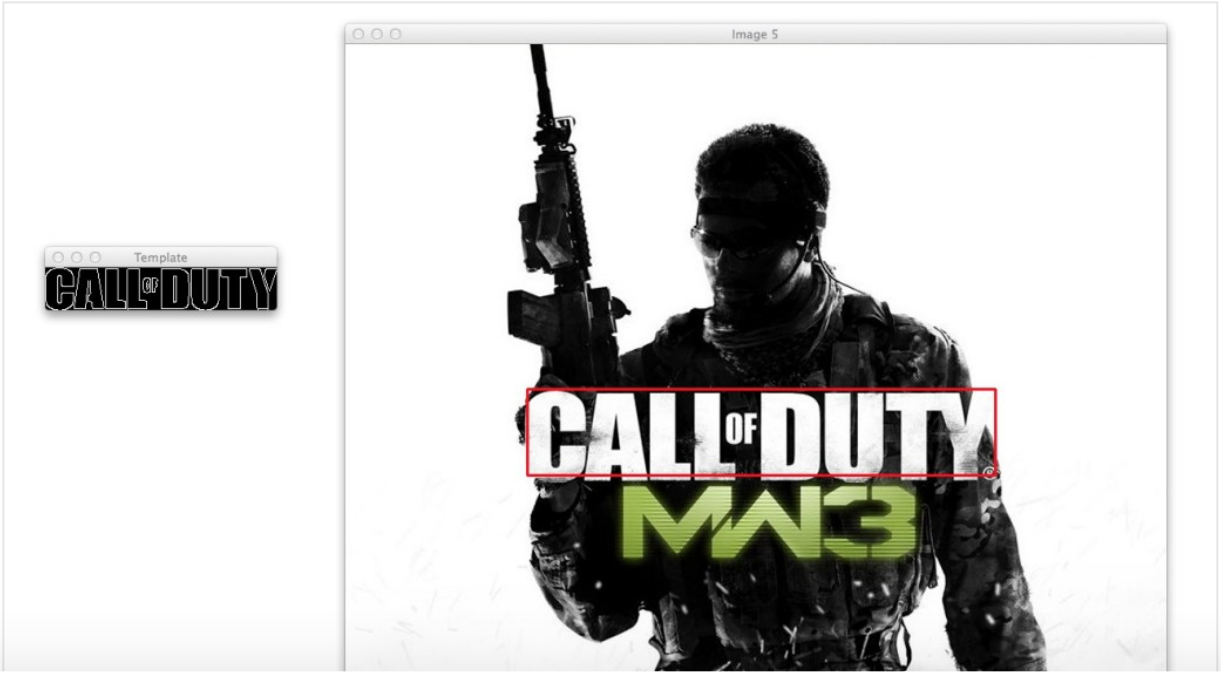
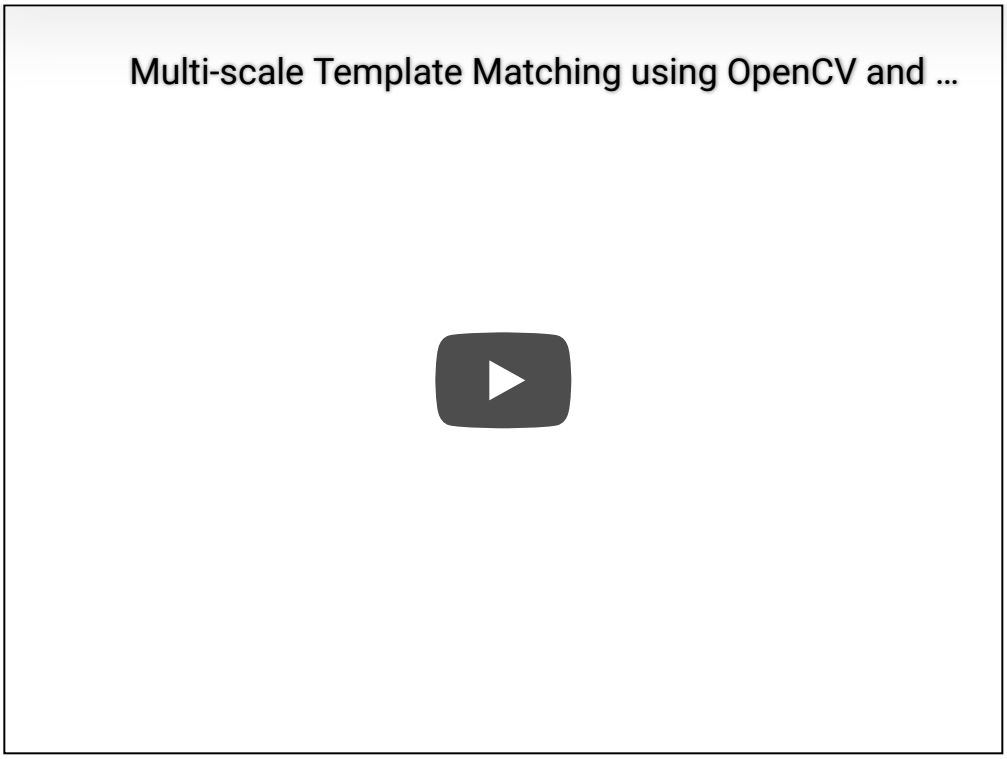
Navigation

pyimagesearch
be awesome at building image search engines

Multi-scale Template Matching using Python and OpenCV

by **Adrian Rosebrock** on January 26, 2015 in **Image Processing**, **Tutorials**





Seriously, back in college I was a Call of Duty fanatic — I even had Call of Duty posters hanging on the walls. And I had played all the games: the original Call of Duty games set during World War II; the Modern Warfare series (my favorite); even the Black Ops games. And while I was too sick to get myself off the couch this weekend, I could no-scope my through a game of Domination without a problem.

But by the end of Sunday afternoon I was starting to feel a little burnt out on my gaming session. Apparently, there is a only a finite amount of gaming I can do in a single sitting now that I'm not in college anymore.

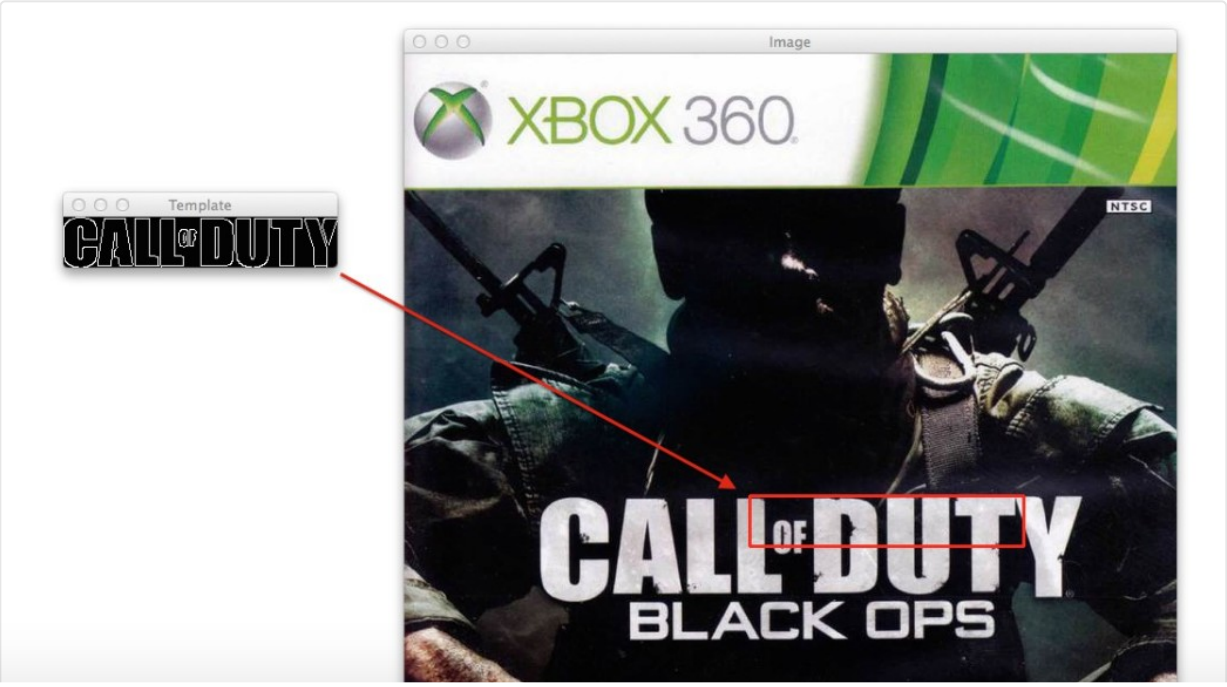
Anyway, I reached over to my laptop and started surfing the web. After a few minutes of browsing Facebook, I came across a template matching tutorial I did over at Machine Learning Mastery. In this

OpenCV and Python versions:
This example will run on **Python 2.7/Python 3.4+** and **OpenCV 2.4.X**.

Multi-scale Template Matching using Python and OpenCV

To start this tutorial off, let's first understand why the standard approach to template matching using `cv2.matchTemplate` is not very robust.

Take a look at the example image below:



Note: *Both the template and input images were matched on the edge map representations. The image on the right is simply the output of the operation after attempting to find the template using the edge map of both images.*

However, when we try to apply template matching using the `cv2.matchTemplate` function, we are left with a false match — this is because the size of the logo image on the *left* is **substantially smaller** than the Call of Duty logo on the game cover on the *right*.

Given that the dimensions of the Call of Duty template does not match the dimensions of the Call of Duty logo on the game cover, we are left with a false detection.

So what do we do now?

Give up? Start detecting keypoints? Extracting local invariant descriptors? And applying keypoint matching?

Not so fast.

While detecting keypoints, extracting local invariant descriptors, and matching keypoints ***would certainly work***, it's ***absolutely overkill for this problem.***

In fact, we can get away with a much easier solution — and with substantially less code.

The cv2.matchTemplate Trick

So as I hinted at in the beginning of this post, just because the dimensions of your template do not match the dimensions of the region in the image you want to match, ***does not*** mean that you cannot apply template matching.

Anyway, enough with the talking. Let's jump into some code. Open up your favorite editor, create a new file, name it `match.py` , and let's get started:

Multi-scale Template Matching using Python and OpenCV — Python

```python
# import the necessary packages
import numpy as np
import argparse
import imutils
import glob
import cv2

# construct the argument parser and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-t", "--template", required=True, help="Path to template image")
ap.add_argument("-i", "--images", required=True,
    help="Path to images where template will be matched")
ap.add_argument("-v", "--visualize",
    help="Flag indicating whether or not to visualize each iteration")
args = vars(ap.parse_args())

# load the image image, convert it to grayscale, and detect edges
template = cv2.imread(args["template"])
template = cv2.cvtColor(template, cv2.COLOR_BGR2GRAY)
template = cv2.Canny(template, 50, 200)
(tH, tW) = template.shape[:2]
cv2.imshow("Template", template)
```

The first thing we'll do is import the packages we'll need. We'll use NumPy for numerical processing, `argparse` for parsing command line arguments, `imutils` for some image processing convenience functions (included with the .zip of the code for this post), `glob` for grabbing the paths to our input images, and `cv2` for our OpenCV bindings.

We then parse our arguments on **Lines 8-15**. We'll need three switches: `--template` , which is the path to the template we want to match in our image (i.e. the Call of Duty logo), `--images` , the path to the directory including the images that contain the Call of Duty logo that we want to find, and an optional `--visualize` argument which lets us visualize the template matching search across multiple scales.

Figure 2: Extracting edges from the template image.

Now, let's work on the multi-scale trick:

Multi-scale Template Matching using Python and OpenCV — Python

```
24 # loop over the images to find the template in
25 for imagePath in glob.glob(args["images"] + "/*.jpg"):
26     # load the image, convert it to grayscale, and initialize the
27     # bookkeeping variable to keep track of the matched region
28     image = cv2.imread(imagePath)
29     gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
30     found = None
31
32     # loop over the scales of the image
33     for scale in np.linspace(0.2, 1.0, 20)[::-1]:
34         # resize the image according to the scale, and keep track
35         # of the ratio of the resizing
36         resized = imutils.resize(gray, width = int(gray.shape[1] * scale))
37         r = gray.shape[1] / float(resized.shape[1])
38
39         # if the resized image is smaller than the template, then break
40         # from the loop
41         if resized.shape[0] < tH or resized.shape[1] < tW:
42             break
```

We start looping over our input images on **Line 25**. We then load the image off disk, convert it to grayscale, and initialize a bookkeeping variable `found` to keep track of the region and scale of the

```
44          # detect edges in the resized, grayscale image and apply template
45          # matching to find the template in the image
46          edged = cv2.Canny(resized, 50, 200)
47          result = cv2.matchTemplate(edged, template, cv2.TM_CCOEFF)
48          (_, maxVal, _, maxLoc) = cv2.minMaxLoc(result)
49
50          # check to see if the iteration should be visualized
51          if args.get("visualize", False):
52              # draw a bounding box around the detected region
53              clone = np.dstack([edged, edged, edged])
54              cv2.rectangle(clone, (maxLoc[0], maxLoc[1]),
55                  (maxLoc[0] + tW, maxLoc[1] + tH), (0, 0, 255), 2)
56              cv2.imshow("Visualize", clone)
57              cv2.waitKey(0)
58
59          # if we have found a new maximum correlation value, then update
60          # the bookkeeping variable
61          if found is None or maxVal > found[0]:
62              found = (maxVal, maxLoc, r)
63
64      # unpack the bookkeeping variable and compute the (x, y) coordinates
65      # of the bounding box based on the resized ratio
66      (_, maxLoc, r) = found
67      (startX, startY) = (int(maxLoc[0] * r), int(maxLoc[1] * r))
68      (endX, endY) = (int((maxLoc[0] + tW) * r), int((maxLoc[1] + tH) * r))
69
70      # draw a bounding box around the detected result and display the image
71      cv2.rectangle(image, (startX, startY), (endX, endY), (0, 0, 255), 2)
72      cv2.imshow("Image", image)
73      cv2.waitKey(0)
```

On **Line 46** we compute the Canny edge representation of the image, using the exact same parameters as in the template image.

We then apply template matching using `cv2.matchTemplate` on **Line 47**. The `cv2.matchTemplate` function takes three arguments: the input image, the template we want to find in the input image, and the template matching method. In this case, we supply the `cv2.TM_CCOEFF` flag, indicating we are using the correlation coefficient to match templates.

care is taken to multiply the coordinates of the bounding box by the ratio on **Line 37** to ensure that the coordinates match the original dimensions of the input image.

Finally, we draw our bounding box and display it to our screen on **Lines 71-73**.

Multi-scale Template Matching Results

Don't take my word for it that this method works! Let's look at some examples.

Open up your terminal and execute the following command:

Multi-scale Template Matching using Python and OpenCV — Shell

```
$ python match.py --template cod_logo.png --images images
```

Your results should look like this:

Figure 3: Successfully applying multi-scale template match to find the template in the image.

As you can see, our method successfully found the Call of Duty logo, unlike the the basic template matching in **Figure 1** which failed to find the logo.

Figure 5: Once again, multi-scale template matching is able to find the logo *(left)* in the input image *(right)*.

Once again, our method was able to find the logo in the input image!

The same is true for **Figure 6** below:

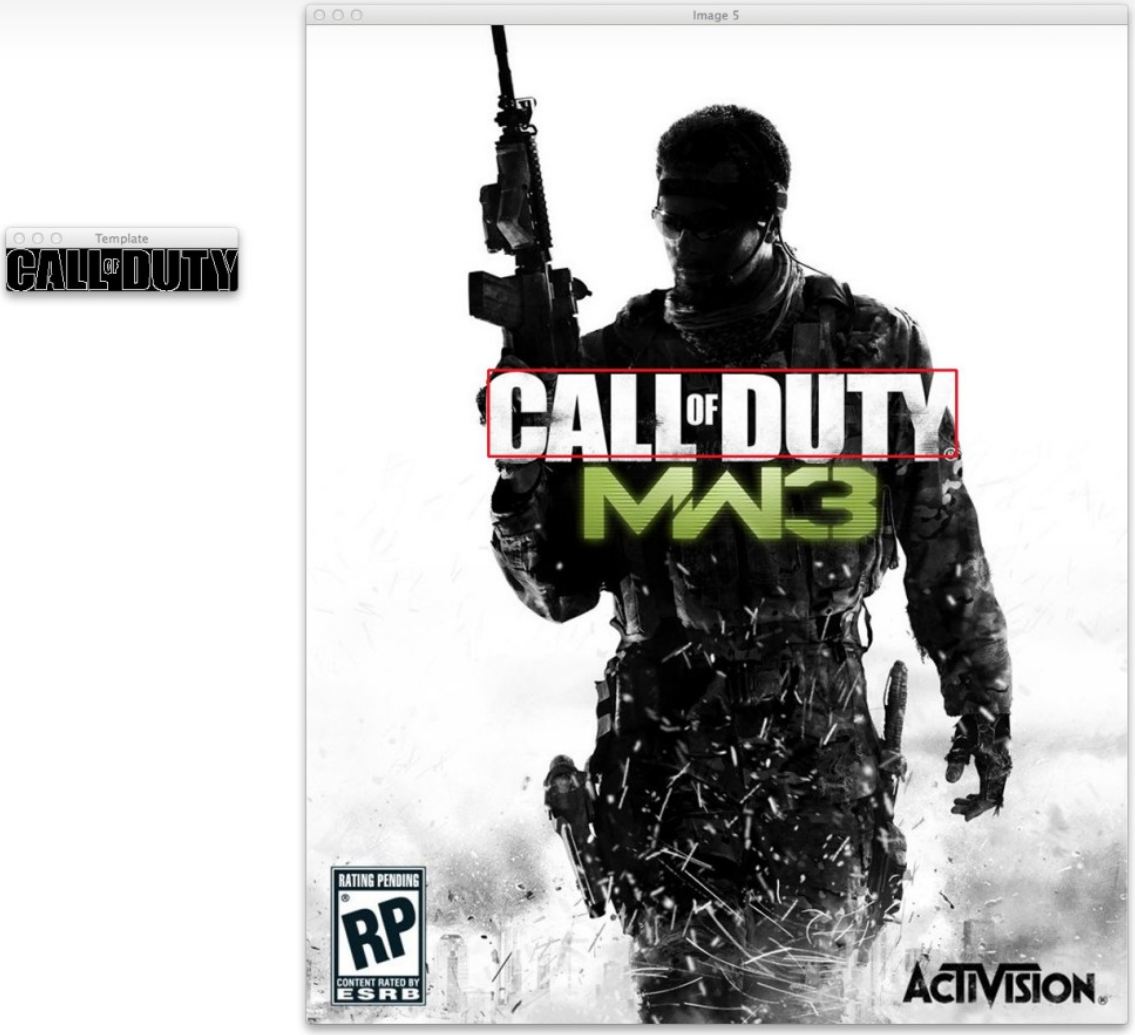
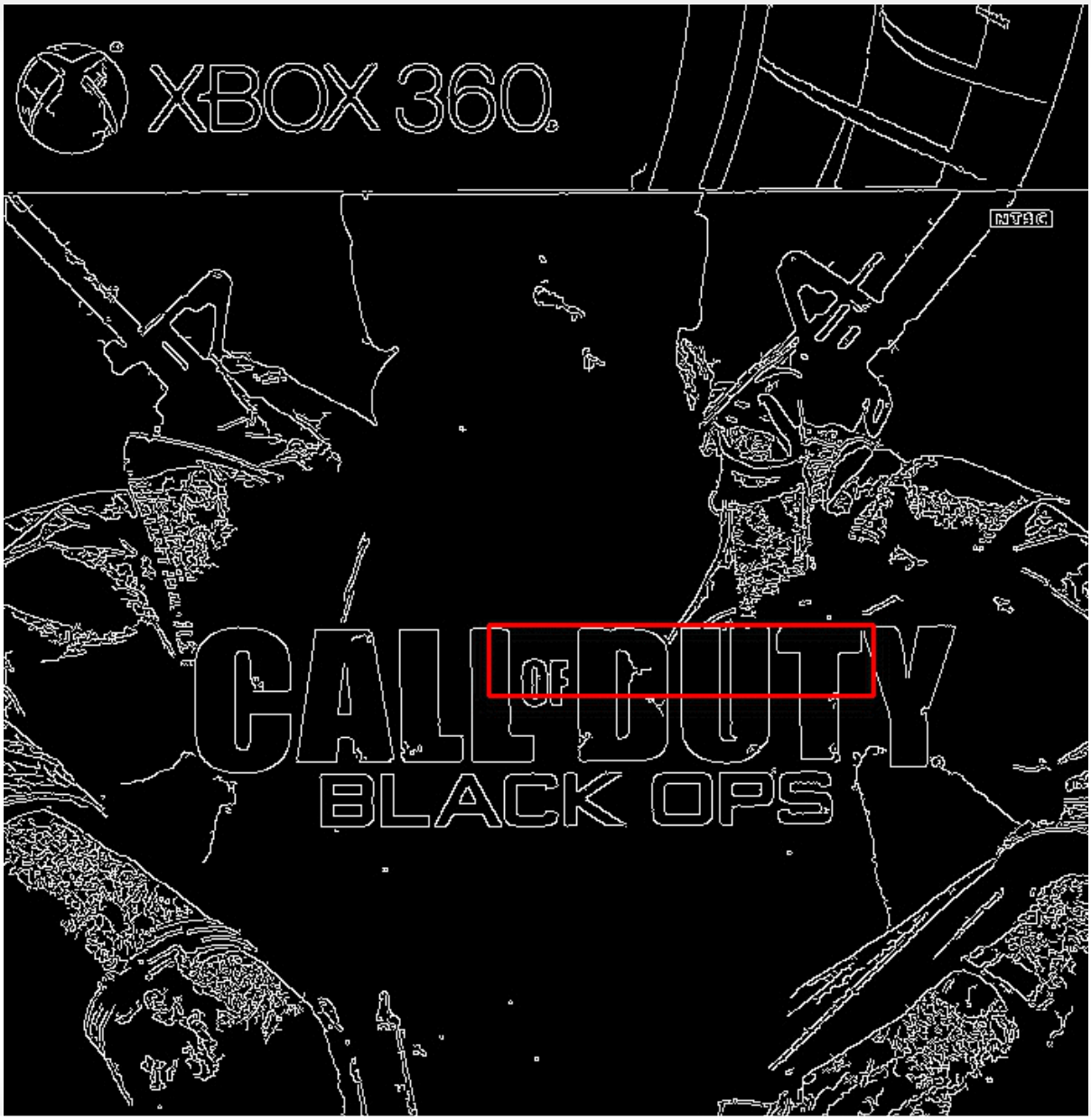


Figure 7: Multi-scale template matching using cv2.matchTemplate.

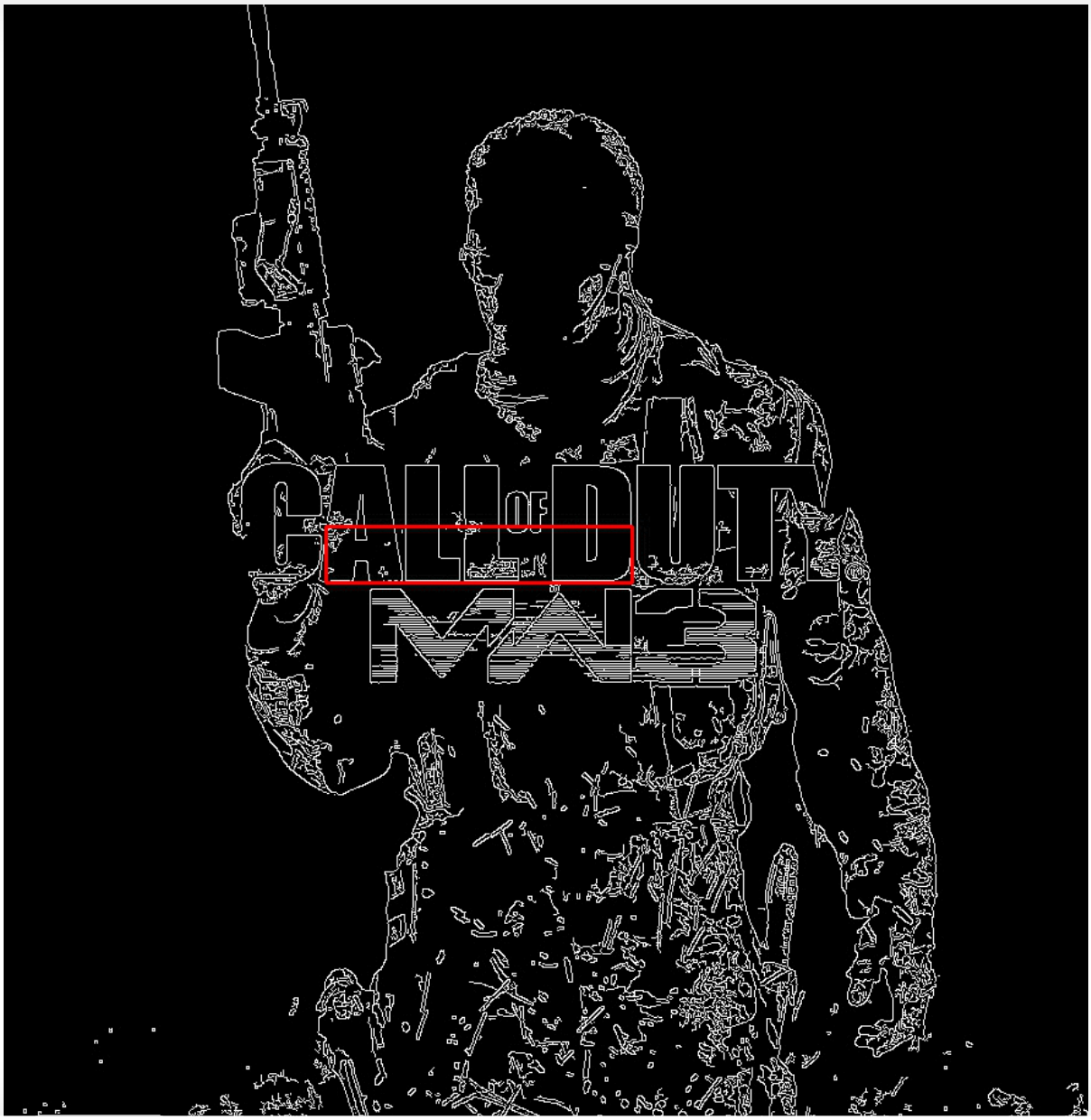
Once again, our multi-scale approach was able to successfully find the template in the input image!



We then apply template matching and find the *(x, y)*-coordinates of the image with the largest correlation coefficient.

Lastly, we store these values in a bookkeeping variable.

At the end of the algorithm we find the *(x, y)*-coordinates of the region with the largest correlation coefficient response ***across all scales*** and then draw our bounding box, as seen below:



If we are concerned about rotation on non-affine transformations we are better off taking the time to detect keypoints, extract local invariant descriptors, and apply keypoint matching.

But in the case where our templates are (1) fairly rigid and well-defined via an edge map and (2) we are only concerned with translation and scaling, then multi-scale template matching can provide us with very good results with little effort.

Lastly, it's important to keep in mind that template matching does not do a good job of telling us if an object *does not* appear in an image. Sure, we could set thresholds on the correlation coefficient, but in practice this is not reliable and robust. If you are looking for a more robust approach, you'll have to explore keypoint matching.

Summary

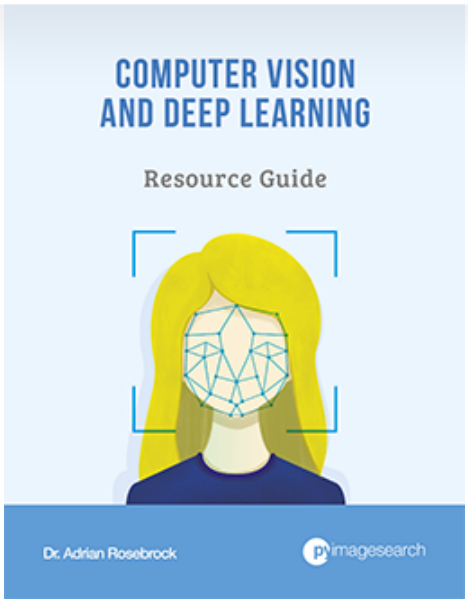
In this blog post we discovered how to make standard template matching more robust by extending it to work with *multiple scales*.

We also discovered that in cases where our template image is rigid and well-formed, that utilizing an edge map rather than the RGB or grayscale representation can yield better results when applying template matching.

Our method to multi-scale template matching works well if we are only concerned with translation and scaling; however, this method will not be as robust in the presence of rotation and non-affine transformations. If our template or input image exhibits these types of transformations we are better off applying keypoint detection, local invariant descriptors, and keypoint matching.

Downloads:

If you would like to download the code and images used in this post, please enter



Enter your email address below to get my **free 17-page Computer Vision, OpenCV, and Deep Learning Resource Guide PDF**. Inside you'll find my hand-picked tutorials, books, courses, and Python libraries to help you master computer vision and deep learning!

Your email address

DOWNLOAD THE GUIDE!

algorithms, **examples**, **image processing**, **multi-scale**, **template matching**

‹ Find distance from camera to object/marker using Python and OpenCV

I just open sourced my personal imutils package: A series of OpenCV convenience functions. ›

159 Responses to *Multi-scale Template Matching using Python and OpenCV*

Deven September 1, 2015 at 8:32 am #

REPLY

I understand this is very simple and straightforward method. My only concern for doing a multi-

but it is possible to make templates in the directory too and we can put many image templates for better result

Adrian Rosebrock September 12, 2015 at 6:46 am #

REPLY

Sure, you can use multiple templates. You would just need to loop over the templates individually and call `cv2.matchTemplate` for each of the, and keep track of the template that gave you the best result.

Brian December 2, 2015 at 7:36 pm #

REPLY

I noticed that you used a PNG file as the template and JPG files as the images you are searching. Is there any advantage to using PNG, JPG, or any other image format? Will certain formats process faster than others? Will some perform better or worse when template matching? Thanks!

Adrian Rosebrock December 3, 2015 at 6:18 am #

REPLY

No, there is no difference between image formats when performing template matching. Once the image is loaded from disk, the image is always represented as a NumPy array internally by OpenCV. The only decision when using various image file formats should be (1) image size and (2) lossy or lossless image compression.

Brian December 3, 2015 at 9:07 pm #

REPLY

and automatically close so fast that you wouldn't be able to visualize them. The `cv2.waitKey` function is important to use when visualizing your results.

Finally, it sounds like you're just getting started learning OpenCV so. I would suggest going through the free crash courses on the blog and also grabbing a copy of Practical Python and OpenCV. This book will really help clear up any doubts or questions you have regarding the basics.

Also, I just tested the code out on my system using Python 2.7 and both OpenCV 2.4 and OpenCV 3.0. In both cases the code ran without error.

Brian December 5, 2015 at 3:42 pm #

REPLY

Thank you for the clarification! I saw the waitKey function and incorrectly assumed it was a delay. Which… gave me an idea.

A really cool update to your code is to change both of the waitKey functions into a 1 second delay, like this:
cv2.waitKey(1000)

That way, when the code runs using the "–visualize" argument it looks almost exactly like your animated gif. =)

Thank you so much for this example!

Brian December 5, 2015 at 5:16 pm #

REPLY

When I try to use an image captured by a camera (I'm using the Raspberry Pi camera module attached to RPI 2) to compare to a template (PNG file) I get an error which contains the following:
depth == CV-8U || depth == CV_32F

Cheers,
-Brian

Adrian Rosebrock December 6, 2015 at 7:13 am #

REPLY

Thanks for sharing Brian — and congrats on resolving the issue! (I consolidated both your comments into a single one, that way if other readers have a similar problem, they can see the answer as well).

Brian December 5, 2015 at 9:56 pm #

REPLY

I noticed using this method that the program will always draw a box around something in the image, even if the “target” isn’t present and even when it’s clearly not a match. Is there any way to create a “threshold” parameter so that the template match must be above a certain confidence level before the box is permitted to be drawn to help prevent false positives?

Adrian Rosebrock December 6, 2015 at 7:16 am #

REPLY

Technically, yes, you can define a threshold parameter — but it will have to be manually defined. If you take a look at the `maxVal` value returned from `cv2.minMaxLoc`, you can manually ensure that the correlation is above a given threshold, like this:

Python

```
1 if maxVal > myThreshold:
2     # draw the bounding box
```

Adrian Rosebrock December 31, 2015 at 6:57 am #

REPLY

Are you running your code on the same set of images? If so, then the `maxValue` would certainly not change when you re-run your script after printing them out to your terminal. There is likely a logic error elsewhere in your code. I would suggest adding some more print statements to your code and try to debug where the error is.

Brian May 21, 2016 at 10:57 am #

Posting my solution for benefit of others.
I was using the wrong maxVal variable! I should have been using the one that was packed in the “found” bookkeeping variable.

To keep things simple, I created a new variable called maxVal2 and used that to trigger an IF statement, like this:

(maxVal2, maxLoc, r) = found #unpack the bookkeeping variable
if maxVal2 > 5000000:
(draw the bounding box using Loc and r)

Working great now!

Saurav Mondal December 18, 2015 at 1:47 am #

REPLY

Hi Adrian, this is a awesome technique and can be used in various use cases.i have a small

In this line of code:
if found is None or maxVal > found[0]:
…what is the purpose of “is None” ?

Thanks!

Adrian Rosebrock December 31, 2015 at 7:05 am #

REPLY

Since we initialized `found = None` on `Line 30`, we need to make sure it is not `None` before we access an array in the list (which would cause an error to be thrown).

Soumen Naayak January 3, 2019 at 4:15 am #

REPLY

Okay…. how we can create threshold or detect that out template doesn’t match with input picture?

Adrian Rosebrock January 5, 2019 at 8:52 am #

REPLY

You will need to manually specify that threshold. You determine it via trial and error by examining the correlation values for matches vs. non-matches.

sai February 11, 2016 at 2:33 pm #

REPLY

Hi Adrian,

Thanks a lot for sharing!

I was searching for a solution to verify a video in which a single icon is displayed in random sizes at random positions (like a screensaver).

Your logic worked like a charm 🙂

REPLY

Adrian Rosebrock February 12, 2016 at 3:19 pm #

Fantastic! I'm glad it worked for you Rateesh! 🙂

REPLY

Mus Sli April 11, 2016 at 8:59 am #

Hi Rateesh ,

how you did it ?
can you help me or Adrian to find the solution for this Problem ?
or can you send your code ?

Thank you in advance 🙂

REPLY

Erik March 3, 2016 at 8:01 am #

Hi Adrian,

Thanks for a great post

The goal here is to determine the “inlier” keypoints (true matches) versus the “outliers” (keypoints that could not be matched between the two images). Based on the number of matched inliers, we can determine if an object contains a specific object (or not).

If you’re interested in learning more about this topic, take a look at Practical Python and OpenCV where I detail how to use keypoint detection, local invariant descriptors, and keypoint matching to identify the covers of books. The same approach can be used for other object matching as well.

Petr March 13, 2016 at 11:33 am #

REPLY

Hi Adrian, huge thanks for this great article!
I got two questions:

1. What is the reason for looping over multiple scales of the input image, instead of trying multiple scales of the template?
And do you agree that scaling the template could be faster, processing-wise, since the template is generally much smaller than the input image?

2. For what reasons do you discourage using template matching when there’s changes in rotation?
Is it simply because of the processing time it would take to loop over multiple rotation angles, especially if combined with looping over multiple scales?

Adrian Rosebrock March 13, 2016 at 12:35 pm #

REPLY

In general, the template is substantially smaller than the source image. Thus, in order to detect the template at various scales of the image, we apply an image pyramid. While it may be faster to resize the actual template, it won’t help us much if the region we want to detect in the image is larger than the template.

What I meant was looping over multiple scaled-up versions of the template, as opposed to looping over multiple scaled-down versions of the input image.

My thinking was that this would be faster, when it comes to resizing that we're doing inside the loop – because we'd be dealing with a smaller amount of input pixels to resize.

However, I completely forgot about the computational cost of matching the template.

And now I realized that when keeping the input image untouched, the cost of matchTemplate() would stay the same, or actually grow as we're going through the loop and the template gets larger and larger.

With your approach in this example, I'm assuming that matchTemplate() becomes faster and faster, as the input image gets smaller and smaller. Which probably makes the entire process more efficent than what I was trying to suggest.

Cheers!

Adrian Rosebrock March 14, 2016 at 3:28 pm #

REPLY

Correct, as the input image becomes smaller, there is less data to process, thus `cv2.matchTemplate` will run faster.

Krzysztof Skowronek August 21, 2018 at 7:58 am #

REPLY

I know that this is old post, but if you change the scale, you then change the values of coeficient calculations, so you could not compare results from different scales.

Hi, in mutiscale template matching , how can i get the co-ordinates of the image that matched with the template and also i need to click on that co-ordinates. please hepl me out in this . Thanks in advance

Adrian Rosebrock April 1, 2016 at 3:16 pm #

REPLY

The coordinates of the matched template are returned by `cv2.minMaxLoc` — see **Line 48**. As for clicking the coordinates, take a look at this blog post on mouse events, it should help point you in the right direction.

Himanshu July 15, 2017 at 12:44 pm #

REPLY

Hi, scikit-image url is not working ? Thanks

Adrian Rosebrock July 18, 2017 at 10:04 am #

REPLY

The scikit-image website is working just fine. Perhaps check our internet connection? Otherwise, I'm not sure what your question is.

gaspar teleki March 22, 2016 at 8:40 am #

REPLY

Dear Adrian!

gaspar teleki March 22, 2016 at 7:28 pm #

REPLY

Dear Adrian!

Thank you for the reply.
I guessed i could make templates in a “starsign database” folder and using a wide angle lens take one photo of the sky and match it with database.
The photo is 5mp so it will really need a fast solution.
Template matching may work i think:)
I will start to test as soon as i can get opencv working on my pi.

Jacob March 29, 2016 at 3:21 am #

REPLY

Hi Adrian

How to put text or labelling on that red retangle matched text

simillar text like:

https://www.pyimagesearch.com/2016/02/15/determining-object-color-with-opencv/

*sorry ugly english

Adrian Rosebrock March 29, 2016 at 7:02 am #

REPLY

You can draw a rectangle using `cv2.rectangle`. And drawing text can be done using `cv2.putText`. The blog post you linked to in your comment demonstrates how to use the

Hey Antonio — if you're worried about having a larger object that what appears in the frame, then you should upsample (i.e., increase the size of) the frame prior to processing. You could double the size of the input frame and then apply the same method detailed in this post.

hoangpx June 24, 2016 at 5:09 pm #

REPLY

why don't you scale the template image instead of original image. It'll save time.

acvguy March 28, 2018 at 4:48 am #

REPLY

How exactly? Also, you need to scale the template down, otherwise you would infer information, which is generally bad (unless complex AI is involved)

Hassan Nadeem July 23, 2016 at 3:29 am #

REPLY

Hi guys,

I had rewritten the python code in C++ for some work that I was doing and I thought to share it so that everyone could benefit.

Code is mostly the same except that my code has the image names hard-coded in the source code. If anyone needs it, the C++ code can be downloaded from here:
http://hassannadeem.com/assets/code/multi_scale_template_matching_cpp.zip

Have you ever had this kind of problem before ? If yes, did you find a solution ? Or did you have any idea of what creates this memory leak ?

For information, I run this program on a Raspberry Pi 3 model B.

Thanks !

Adrian Rosebrock September 5, 2016 at 12:45 pm #

REPLY

How big (in terms of width and height) are your images? Normally, we don't compare images that are larger than 600-800 pixels along their largest dimension. It could be the case that you are comparing two gigantic images and then running out of memory.

Cyril September 6, 2016 at 2:57 am #

REPLY

My Template does not exceed 400 pixels for the largest dimension, but I try to find this template in an image of dimensions 1280 x 720 pixels. So yes I try to find it in a gigantic images, but it's really strange that it runs out of memory after 45 minutes (The function is called maybe around 50 times).

Do you suggest that I have to reduce the size of the image in which I search the template ? In that case it worries me a little bit, because I don't know where is the template in the image, so I can't crop the image in a blind way, and also I have to know exactly the number of pixels between the center of the image and the template found in the image, to correct after that the position of the robot I command.

Is any way to change the scale of the image, find the template and then do the inverse to have the exact number of pixels like I found it in the native image ?

The thing works well if I detect something like a white dot painted/attached on my shell. Numbers are selected and tweeted correctly. Take a look at this tweet:

https://twitter.com/balbino_live/status/704621280037900288

But now I want to track my moves using a template of my body (or shell), but of course, I'm in a irregular shape, and I can be at the sun, or underwater, or going upwards or downwards.

Some suggestion to a better approximation? May I try with many rotated templates of my body? May I try finding a shape like an oval (I was using circles for the white dot)?

Thanks again, Adrian!

Adrian Rosebrock September 16, 2016 at 8:20 am #

REPLY

I took a look at your Twitter link and I do indeed see the lotto functionality. However, I'm not sure what you are referring to in the second half of your question? Can you elaborate more and perhaps explain what I'm looking at?

Balbino September 16, 2016 at 8:57 am #

REPLY

Thanks for your answer, Adrian.

First try:

The Pi camera, at random time intervals, detects if the turtle moved from the previous one, takes a snapshot, and translate its actual position to a lotto digit using a numbered net, until the whole draw is completed.

Problem for me:

This sounds like a neat project, thanks for the added information. My first thought would be to recommend thresholding, both normal thresholding and adaptive thresholding, to help you segment the foreground (the turtle) from the background (the rest of the image). Thresholding will work regardless of positioning.

The problem then is that color could vary dramatically. This isn't an issue as long as their is enough contrast between the background and the foreground, but if there isn't thresholding will not work.

I would also suggest applying template matching where you have a "template" image of each turtle position and then try to match the template to the input image.

Again, I don't think I know enough about this project to give super great advice off the top of my head, but this is where I would start.

Nicky Fandino October 12, 2016 at 12:49 am #

REPLY

Hey Adrian, I was wondering. Can I multi-scale the template, instead of the image ? I asked this because I have more than 1 objects in my image that [supposed to] match the template.

Adrian Rosebrock October 13, 2016 at 9:21 am #

REPLY

You certainly can.

Joe Z December 5, 2016 at 11:44 pm #

REPLY

template, thinking that theoretically, the contribution of those pixels to the CCOEFF calculation would be then be zero (due to the intensity scaling part of the calculation). Result: no bueno. False positives galore..

Any advice on how best to proceed?

Best,

Joe

Adrian Rosebrock December 7, 2016 at 9:49 am #

REPLY

Ouch, matching rotated rectangular templates is a real pain. Instead of using template matching I would instead use keypoints, local invariant descriptors, and keypoint matching. These methods are by their very definition invariant to rotation. I demonstrate how to build a system to recognize the covers of books inside Practical Python and OpenCV that would help you accomplish this. If you want to share more details about your project and the types of ROIs you are trying to match I can also give you a better idea if the book will help.

ruhi January 18, 2017 at 8:53 am #

REPLY

Hi adrian,

I am very thankful for posting this kind of solution. It helped me a lot. I am having query regarding different dpi images. i have single image with different dpi like 100dpi, 200dpi,…500dpi. So what dpi of template i should take for matching?Actually, i dont have idea about relation of dpi with image. So i am not able to detect Region of interest. My goal is to find text “PNR” which will be taken as a template for matching to destination image and after getting it as reference point, i have to detect outer rectangle where text is

Hello,

How would I extend the tutorial to be able to find multiple occurrences of the template in the target image. For example: If “CALL of DUTY” was present x times in the image, how would I draw x boxes around each instance.

Thank you for your time.
Mukesh.

Adrian Rosebrock February 15, 2017 at 9:02 am #

REPLY

To detect *multiple objects* via template matching you would:

1. Take the `result` variable outputted from `cv2.matchTemplate`
2. Threshold it to find (x, y)-coordinates that have a confidence greater than N (you would have to define N manually).
3. Treat each of these (x, y)-coordinates that have a high correlation score as a detected object.

Again, the trick is setting the correlation threshold right.

Travis February 15, 2017 at 12:23 pm #

REPLY

Hello,

Wanted to ask what was your reason for using CCOEFF method for this example instead of the other 5 methods?

Thanks

Adrian Rosebrock March 17, 2017 at 9:30 am #

REPLY

It's hard to say what the issue is without seeing your exact error message, but yes, make sure you are in the source code directory after unzipping the package *before* you execute the Python script.

Jan April 10, 2017 at 10:46 am #

REPLY

Hello Adrian,

I am so thrilled by your blog.

Nevertheless, running your code I get the following:

" (_, maxLoc, r) = found
TypeError: 'NoneType' object is not iterable."

I checked on stackoverflow and so on, but I do not get how that happened and how to solve it. Do you have any clue or advice?

Thanks in advance.

Adrian Rosebrock April 12, 2017 at 1:15 pm #

REPLY

If you are getting this error it's because no match was found (Lines 61 and 62).

vinod May 25, 2017 at 7:08 am #

REPLY

how to detect this using real time, i would like to detect using web camera. what are the changes that i can go??

Adrian Rosebrock May 28, 2017 at 1:26 am #

REPLY

You would need to update the code so that it accesses a video stream and applies template matching to each frame. I discuss how to access your webcam in this blog post as well as in Practical Python and OpenCV.

Sohit June 19, 2017 at 1:47 pm #

REPLY

Hello Adrian! Stumbled upon this post and this has helped me immensely to integrate opencv with pyautogui in a project I am working on!! Thx!

Adrian Rosebrock June 20, 2017 at 10:54 am #

REPLY

Thanks Sohit — I'm happy to hear the PyImageSearch blog helped you! Have a great day 🙂

If your template is larger than the image you are trying to find the template it, you can either (1) resize the template or (2) increase the size of your input image. Depending on the complexity of the objects you are trying to recognize, it might also be worth to train a custom HOG + Linear SVM detector.

Jason July 11, 2017 at 10:57 pm #

REPLY

What is the maximum value that maxval can take?

Subhodeep July 17, 2017 at 9:57 am #

REPLY

Hey Adrian, whats the reason for using ‘resize’ from ‘imutils’ rather than from ‘cv2’, if any ?

Adrian Rosebrock July 17, 2017 at 11:26 am #

REPLY

The `imutils.resize` function automatically takes care of ensuring the aspect ratio is the same while `cv2.resize` does not.

Nate August 1, 2017 at 11:47 am #

REPLY

Super basic question – where do I save the source code to be able to run it from the terminal? Thanks.

usage: match.py [-h] -t TEMPLATE -i IMAGES [-v VISUALIZE]
match.py: error: argument -i/–images: expected one argument

Does anyone know what would cause this?

Adrian Rosebrock August 10, 2017 at 8:47 am #

REPLY

Please take the time to read up on command line arguments before continuing.

Pavel August 21, 2017 at 7:47 am #

REPLY

Great as always, Adrian!

Do you know what the best to match template of binary dots field with image?

Adrian Rosebrock August 21, 2017 at 3:38 pm #

REPLY

Hi Pavel — I would need to see example images of what you are trying to detect.

Pavel August 22, 2017 at 1:20 pm #

REPLY

Good example of this task is constellation detection: for example to find on the star night photo https://goo.gl/images/4gp99F the constellation of the great bear https://goo.gl/images/JHHi8Y.

I just have a huge doubt regarding correlation coefficient and I hope i'll get better answer.

I have a list of template images and only one target image. Now I'll loop through the template images and match it with the target image and will get the max Val (Correlation coefficient value). So now I have the max Val (Correlation coefficient value) of one image which has the highest correlation coefficient value, so I'll show the image which is highly correlated, to the user. But the problem with this approach is, consider a scenario where the target image is no way related to the template images but still, i'll get a correlation coefficient value (r). So now what is happening is my code is showing me the image with the highest coeff value but the image is no way related to the template images. So I'm planning to set a threshold value or is there any other way to detect false positives?

I have no idea on how to set the threshold value. Please guide me.

Adrian Rosebrock September 5, 2017 at 9:37 am #

REPLY

You are correct, in this instance you would set a threshold to detect false positives. You would need to *manually* determine the threshold yourself by testing against a bunch of example images.

If you're concerned about false-positives a better approach would be to use keypoint detection + local invariant descriptors + keypoint matching. I demonstrate how to use this method (and filter out false-positives) inside Practical Python and OpenCV where we build a system to recognize the covers of books. Be sure to take a look, I think it will help you out a lot in this project.

Anuj September 7, 2017 at 7:01 am #

REPLY

What if the size of of call of duty on my test image is smaller than size of the template. How can we handle that scenario. Shall we then reduce the size of templates too over various scale?

Adrian Rosebrock October 19, 2017 at 4:52 pm #

REPLY

Take the resulting output from `cv2.matchTemplate` and then threshold it to determine all (x, y)-coordinates that have a confidence greater than `T` — these will be your detections. Please note that you will have to set "T" yourself. I would suggest experimentally tuning the value and seeing what works best for you.

Rohan Kathan November 13, 2019 at 6:03 am #

REPLY

I am new to using OpenCV and Python. Can you add this above specified line of program to match multiple occurences in a single image?

Nikita October 24, 2017 at 2:35 am #

REPLY

can i use this code for real time object detection. I want to detect the object from live streaming. Please tell me how can i do this? Thank you.

Adrian Rosebrock October 24, 2017 at 7:10 am #

REPLY

For simple objects, yes, absolutely. An example of accessing a video stream can be found here and inside my book, Practical Python and OpenCV.

hey Govind, have you figured out how to apply Adrian's advice of identifying multiple occurences into practice? If so, please do provide those lines of code here as I am new to OpenCV and want to learn about it or else explain the concept in depth for me understand. Thanks in advance.

Chris T December 3, 2017 at 3:36 pm #

REPLY

Thanks for all the help! Is there any way to return the pixel coordinates of the center of the image being matched? I'm using a circle for my template image.

Adrian Rosebrock December 5, 2017 at 7:40 am #

REPLY

The bounding box coordinates are returned on Lines 67 and 68. Add the x coordinates and divide by two. Do the same for the y coordinates. This will give you the center coordinate of the bounding box.

Bharath Ballamudi December 11, 2017 at 7:37 am #

REPLY

Thanks Adrain,

It works fine. However, this algorithm doesn't lend itself to the problem of detection of multiple instances of an object with varying colors. Say you want to detect the stars on a US map colored differently from that of in the template image. How would you tweak your code to suit the problem at hand?

Appreciate you time.

Adrian Rosebrock March 7, 2018 at 9:33 am #

REPLY

Hey CJ — what does “display the most accurate result” mean in this context? What exactly are you trying to display?

jardembek April 3, 2018 at 12:58 am #

REPLY

how can i run this example on python27 IDLE

Jaan April 13, 2018 at 3:29 pm #

REPLY

I mostly see OpenCV articles as it relates to comparing images. Can you use Template Matching to categorize scanned forms/documents? I have to enter data by sifting through tons of government forms. There can be 9 different versions of the same form (e.g., so the handwritten info could be 9 different places) and everything is named DOCUMENT(1xxx).TIF. It’s government, what do you expect?

I saw you had a tutorial on recognizing handwriting so i was thinking of put 2 & 2 together 🙂

Step 1: recognize the kind of document (Form RW-2107.rev53, badnames-2018-01.blah)
Step 2: if it’s a document we want, search the area on the document for handwriting there
Step 3: If there’s handwriting, read it!
Step 4: Go get coffee while the machine does my job

Adrian Rosebrock April 16, 2018 at 2:36 pm #

REPLY

If you want to use multiple templates you would need to loop over each of them individually, perform the matching process, and then keep the one with the largest correlation score.

Bhargav May 4, 2018 at 3:27 am #

REPLY

Hi Adrian,

Thanks a lot for such a Nice blog. Could you please let me know how should we match a template of size greater than Input image(Input image consists of a white space and template of reduced size).

Adrian Rosebrock May 9, 2018 at 10:37 am #

REPLY

Perhaps I'm misunderstanding the question but is there a reason you cannot resize your template so that it's smaller than your input image and then apply template matching?

Percy May 11, 2018 at 5:54 am #

REPLY

I cannot find the code, do we have to copy paste and use it, what abt the required packages? Can someone please help…I really loved this work!

Adrian Rosebrock May 14, 2018 at 12:06 pm #

REPLY

Scroll down to the "Downloads" section of this blog post and then you can download the

Thank you for this wonderful work!!! 🙂 Kudos to you!

Adrian Rosebrock May 17, 2018 at 6:36 am #

REPLY

There are a few ways to approach this problem but if you want a more robust approach I would actually recommend either:

1. Applying keypoint matching via keypoints local invariant descriptors
2. Training your own custom object detector for each logo

How many example images do you have per logo? Or do you only have one (i.e., the template)?

Percy May 17, 2018 at 7:16 am #

REPLY

First of all, a big thanks for your help!

Yes I have one logo, say for example the logo of “Google”, and I want to search this Google logo in all the documents we have, so the size may vary with each document, but since it is a logo, the design wont change, however there maybe changes in number of pixels, scale, DPI, and color variations, since the documents will embed these logos in their own way…

1) I have seen the Keypoint descriptors but they draw lines or just mark the points on the image…What if I wish to create a box around it like u did and also would it work for varying scaled images…

2) I am unaware of how to do custom object dectector training for each logo…can u point me to some URL where I can learn this pls?

Thank you for ur time!

Percy May 18, 2018 at 4:27 am #

REPLY

Thank you very much Adrian, I would look into your suggestions…Awesome help!

Todd N. July 2, 2018 at 2:40 am #

REPLY

Hi Adrian,

Thanks for this wonderful article… I copied your code and ran it for fun. It has so much potential.

Also read through the comments and it amazing that you answered almost every question asked. There must have been repeated questions to the point where I got annoyed reading through it… I don't know how you do it man… kudos for having the right attitude and sharing not only your knowledge, but also your wisdom.

Adrian Rosebrock July 3, 2018 at 8:25 am #

REPLY

Thank you Todd, I really appreciate that 🙂 It can be tedious at times but I do my best to remind myself that we are all here to learn.

marx August 30, 2018 at 6:39 pm #

REPLY

Can you suggest an approach of rotation & scale invariant approach that does not have license limitation ?

Thanks Toby!

Sohrab January 21, 2019 at 6:35 pm #

REPLY

Hi Adrian,

Thanks for the wonderful article!
Just why scaling the main image and not the template? The template is smaller than the main image and scaling that instead could potentially save some time!

Adrian Rosebrock January 22, 2019 at 9:10 am #

REPLY

Try it and see! I recommend learning by doing, it's the best way to learn.

Percy Jameson January 30, 2019 at 12:49 am #

REPLY

Hi Adrian,

Greetings!

Loved this post, gives truly deep insight in rescaling the images and finding matches. I went through the various questions posted above, and I too had one of the same question – "Can u pls provide an excerpt of code for your suggestion:

1. Take the result variable outputted from cv2.matchTemplate
2. Threshold it to find (x, y)-coordinates that have a confidence greater than N (you would have to define N manually)

Akshay February 11, 2019 at 6:32 am #

REPLY

Hi Adrian,

I'm trying to match template image in a given image. But my problem is that the template image has some regions which are to be ignored. I have a binary image same as the size of the template image stating regions to ignore while trying to match the images.
In short i have 3 images namely the main image, template image and the binary image containing regions to ignore. How can I solve this use case. Need your help with this.

Thanks

Adrian Rosebrock February 14, 2019 at 1:32 pm #

REPLY

I would need to see example images of what you're working with to provide suggestions.

VENKATRAMAN R March 25, 2019 at 1:24 am #

REPLY

How to get the co-ordinates of the bounding box.
I want to get [x1, y1, x2, y2] of the bounding box

Adrian Rosebrock March 27, 2019 at 8:58 am #

REPLY

I'm not sure what you mean by sub-scale steps. If you think a particular scale works well then you could manually define the set of scales you want to test and include more fine-grained scales for the "most likely" scale the object could be at.

Brenden Romanowski April 9, 2019 at 4:07 pm #

REPLY

Wonderful article and just what I was looking for. Cheers!

Adrian Rosebrock April 12, 2019 at 12:17 pm #

REPLY

Thanks Brenden, I'm glad you enjoyed it!

Manthika April 24, 2019 at 3:55 pm #

REPLY

what is the best way to implement same in real-time.

Adrian Rosebrock April 25, 2019 at 8:37 am #

REPLY

This method can run in real-time on a CPU. You just need to access your camera first.

Alex April 26, 2019 at 1:35 pm #

REPLY

Hello Adrian
How difficult to apply the same technique to solve jigsaw puzzle? And would be nice if can be done in real time by just put each piece in front of the camera, then it tell us where that piece goes. Of course the complete picture already uploaded to the computer.
Thank you very much

Adrian Rosebrock May 1, 2019 at 11:47 am #

REPLY

As far as I understand, It's essentially impossible to solve a jigsaw problem. Even if you could reliably segment the puzzle pieces there are too many possible rotations for each piece. If each piece is square it would be possible though.

Saman May 1, 2019 at 7:55 am #

REPLY

Hi, is this better than feature matching?

Adrian Rosebrock May 1, 2019 at 11:17 am #

REPLY

"Better" can only be defined in context of the problem. Feature matching is typically more robust but also computationally more expensive. Multi-scale template matching could be less accurate but could also work better for objects with very little texture.

George Dong May 18, 2019 at 10:51 am #

REPLY

Adrian Rosebrock September 5, 2019 at 10:34 am #

REPLY

Strange, I guess Jason decided to take down the tutorial. I would reach out to him directly and ask why.

David Killen October 13, 2019 at 2:56 pm #

REPLY

There is a copy on archive.org at https://web.archive.org/web/20170918035306/https://machinelearningmastery.com/using-opencv-python-and-template-matching-to-play-wheres-waldo/

Adrian Rosebrock October 17, 2019 at 7:57 am #

REPLY

Thanks for sharing David!

Fanny November 13, 2019 at 3:06 am #

REPLY

Adrian, first of all keep up the good work, IMHO all of your posts are of very valuable information.

Regarding on how to detect when the logo is not present insteadd of using a threshold I thought on using “keypoint detection + local invariant descriptors + keypoint matching” but only on the selected area from template matching.

images never change its perspective, can we use homography when the image and the template will always share the same perspective?.

As you can infere I'm just a beginner so, please, be patient with my questions.

Thanks in advance

Before you leave a comment...

Hey, Adrian here, author of the PyImageSearch blog. I'd love to hear from you, but before you submit a comment, **please follow these guidelines:**

- **If you have a question, *read the comments first.*** You should also search this page (i.e., *ctrl* + *f*) for keywords related to your question. It's likely that I have already addressed your question in the comments.
- **If you are copying and pasting code/terminal output, *please don't.*** Reviewing another programmers' code is a very time consuming and tedious task, and due to the volume of emails and contact requests I receive, I simply cannot do it.
- **Be respectful of the space.** I put *a lot* of my own personal time into creating these free weekly tutorials. On average, each tutorial takes me 15-20 hours to put together. I love offering these guides to you and I take pride in the content I create. Therefore, I will not approve comments that include large code blocks/terminal output as it destroys the formatting of the page. Kindly be respectful of this space.
- **Be patient.** I receive 200+ comments and emails per day. Due to spam, and my desire to personally answer as many questions as I can, I hand moderate all new comments (typically once per week). I try to answer as many questions as I can, but I'm only one person. Please don't be offended if I cannot get to your question

Name (required)

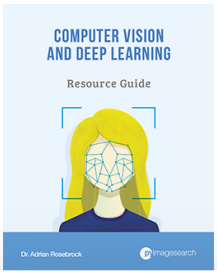
Email (will not be published) (required)

Website

SUBMIT COMMENT

Search...

Resource Guide (it's totally free).



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.

Free 17-day email crash course on Computer Vision, OpenCV, and Deep Learning

Interested in Computer Vision, Deep Learning, and OpenCV, but don't know where to start?

Let me help. I've created a *free* 17-day email crash course that is hand-tailored to **give you the best possible introduction to computer vision and deep learning.**

Sound good? Enter your email below to get started.

Email address

Start my email course!

Face recognition with OpenCV, Python, and deep learning
JUNE 18, 2018

Home surveillance and motion detection with the Raspberry Pi, Python, OpenCV, and Dropbox
JUNE 1, 2015

Install OpenCV and Python on your Raspberry Pi 2 and B+
FEBRUARY 23, 2015

Real-time object detection with deep learning and OpenCV
SEPTEMBER 18, 2017

Ubuntu 16.04: How to install OpenCV
OCTOBER 24, 2016

Find me on **Twitter**, **Facebook**, and **LinkedIn**.
Privacy Policy