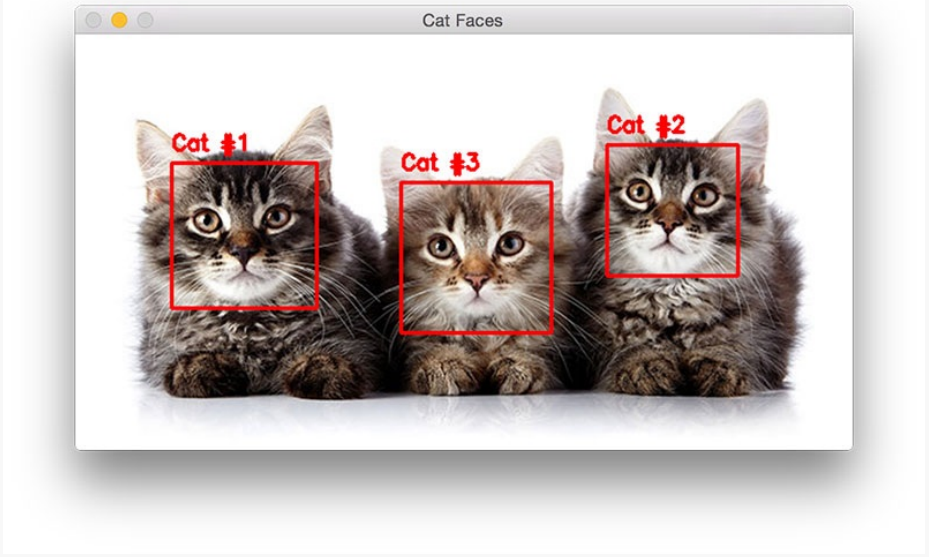
pyimagesearch
be awesome at building image search engines

Detecting cats in images with OpenCV

by Adrian Rosebrock on June 20, 2016 in Object Detection, Tutorials



(source)

Did you know that OpenCV can detect cat faces in images…*right out-of-the-box with* **no** *extras?*

I didn't either.

But after Kendrick Tan broke the story, I had to check it out for myself…and do a little investigative work to see how this cat detector seemed to sneak its way into the OpenCV repository without me noticing (much like a cat sliding into an empty cereal box, just waiting to be discovered).

In the remainder of this blog post, I'll demonstrate how to use OpenCV's cat detector to detect cat faces in images. This same technique can be applied to video streams as well.

Looking for the source code to this post?
Jump right to the downloads section.

Detecting cats in images with OpenCV

If you take a look at the OpenCV repository, specifically within the haarcascades directory (where OpenCV stores all its pre-trained Haar classifiers to detect various objects, body parts, etc.), you'll notice two files:

- haarcascade_frontalcatface.xml
- haarcascade_frontalcatface_extended.xml

Both of these Haar cascades can be used detecting "cat faces" in images. In fact, I used these very same cascades to generate the example image at the top of this blog post.

Doing a little investigative work, I found that the cascades were trained and contributed to the OpenCV repository by the legendary Joseph Howse who's authored a good many tutorials, books, and talks on computer vision.

In the remainder of this blog post, I'll show you how to utilize Howse's Haar cascades to detect cats in images.

Cat detection code

Let's get started detecting cats in images with OpenCV. Open up a new file, name it `cat_detector.py`, and insert the following code:

Detecting cats in images with OpenCV

```python
1  # import the necessary packages
2  import argparse
3  import cv2
4
5  # construct the argument parse and parse the arguments
6  ap = argparse.ArgumentParser()
7  ap.add_argument("-i", "--image", required=True,
8      help="path to the input image")
9  ap.add_argument("-c", "--cascade",
10     default="haarcascade_frontalcatface.xml",
11     help="path to cat detector haar cascade")
12 args = vars(ap.parse_args())
```

Lines 2 and 3 import our necessary Python packages while **Lines 6-12** parse our command line arguments. We only require a single argument here, the input `--image` that we want to detect cat faces in using OpenCV.

We can also (optionally) supply a path our Haar cascade via the `--cascade` switch. We'll default this path to `haarcascade_frontalcatface.xml` and assume you have the `haarcascade_frontalcatface.xml` file in the *same directory* as your `cat_detector.py` script.

Note: *I've conveniently included the code, cat detector Haar cascade, and example images used in this tutorial in the "Downloads" section of this blog post. If you're new to working with Python + OpenCV (or Haar cascades), I would suggest downloading the provided .zip file to make it easier to follow along.*

Next, let's detect the cats in our input image:

Detecting cats in images with OpenCV

```python
14 # load the input image and convert it to grayscale
15 image = cv2.imread(args["image"])
16 gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
17
18 # load the cat detector Haar cascade, then detect cat faces
19 # in the input image
20 detector = cv2.CascadeClassifier(args["cascade"])
21 rects = detector.detectMultiScale(gray, scaleFactor=1.3,
22     minNeighbors=10, minSize=(75, 75))
```

On **Lines 15 and 16** we load our input image from disk and convert it to grayscale (a normal pre-processing step before passing the image to a Haar cascade classifier, although not strictly required).

Line 20 loads our Haar cascade from disk (in this case, the cat detector) and instantiates the `cv2.CascadeClassifier` object.

Detecting cat faces in images with OpenCV is accomplished on **Lines 21 and 22** by calling the `detectMultiScale` method of the `detector` object. We pass four parameters to the `detectMultiScale` method, including:

1. Our image, `gray`, that we want to detect cat faces in.
2. A `scaleFactor` of our image pyramid used when detecting cat faces. A larger scale factor will increase the speed of the detector, but could harm our true-positive detection accuracy. Conversely, a smaller scale will slow down the detection process, but increase true-positive detections. However, this smaller scale can also increase the false-positive detection rate as well. See the ***"A note on Haar cascades"*** section of this blog post for more information.
3. The `minNeighbors` parameter controls the minimum number of detected bounding boxes in a given area for the region to be considered a "cat face". This parameter is very helpful in pruning false-positive detections.
4. Finally, the `minSize` parameter is pretty self-explanatory. This value ensures that each detected bounding box is at least *width x height* pixels (in this case, *75 x 75*).

The `detectMultiScale` function returns `rects`, a list of 4-tuples. These tuples contain the *(x, y)*-coordinates and *width* and *height* of each detected cat face.

Finally, let's draw a rectangle surround each cat face in the image:

Detecting cats in images with OpenCV

```python
24 # loop over the cat faces and draw a rectangle surrounding each
25 for (i, (x, y, w, h)) in enumerate(rects):
26     cv2.rectangle(image, (x, y), (x + w, y + h), (0, 0, 255), 2)
27     cv2.putText(image, "Cat #{}".format(i + 1), (x, y - 10),
28         cv2.FONT_HERSHEY_SIMPLEX, 0.55, (0, 0, 255), 2)
29
30 # show the detected cat faces
31 cv2.imshow("Cat Faces", image)
32 cv2.waitKey(0)
```

Given our bounding boxes (i.e., `rects`), we loop over each of them individually on **Line 25**.

We then draw a rectangle surrounding each cat face on **Line 26**, while **Lines 27 and 28** displays an integer, counting the number of cats in the image.

Finally, **Lines 31 and 32** display the output image to our screen.

Cat detection results

To test our OpenCV cat detector, be sure to download the source code to this tutorial using the ***"Downloads"*** section at the bottom of this post.

Then, after you have unzipped the archive, you should have the following three files/directories:

1. `cat_detector.py` : Our Python + OpenCV script used to detect cats in images.
2. `haarcascade_frontalcatface.xml` : The cat detector Haar cascade.
3. `images` : A directory of testing images that we're going to apply the cat detector cascade to.

From there, execute the following command:

Detecting cats in images with OpenCV

```shell
$ python cat_detector.py --image images/cat_01.jpg
```

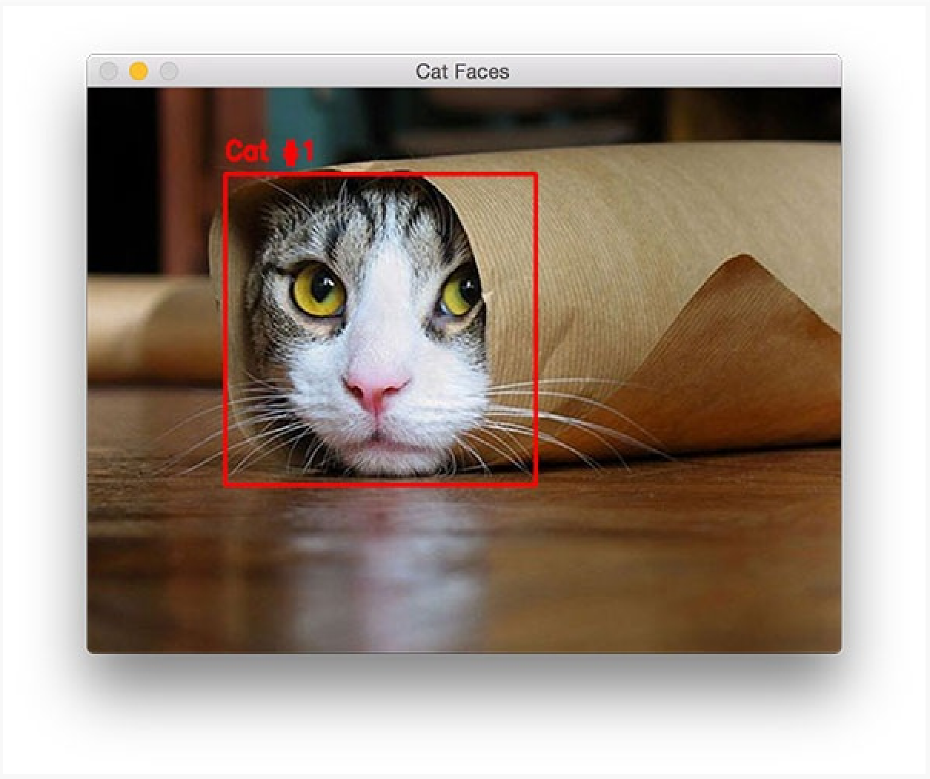


Figure 1: Detecting a cat face in an image, even with parts of the cat occluded (source).

Notice that we have been able to detect the cat face in the image, even though the rest of its body is obscured.

Let's try another image:

Detecting cats in images with OpenCV

```python
python cat_detector.py --image images/cat_02.jpg
```

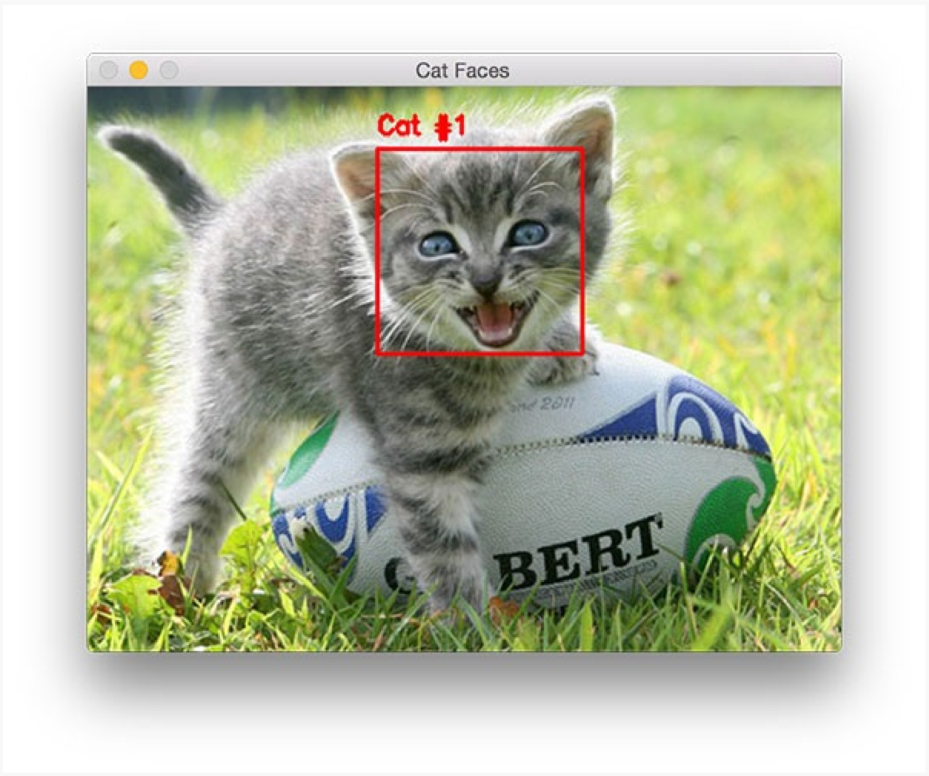


Figure 2: A second example of detecting a cat in an image with OpenCV, this time the cat face is slightly different (source).

This cat's face is clearly different from the other one, as it's in the middle of a "meow". In either case, the cat detector cascade is able to correctly find the cat face in the image.

The same is true for this image as well:

Detecting cats in images with OpenCV

```shell
$ python cat_detector.py --image images/cat_03.jpg
```

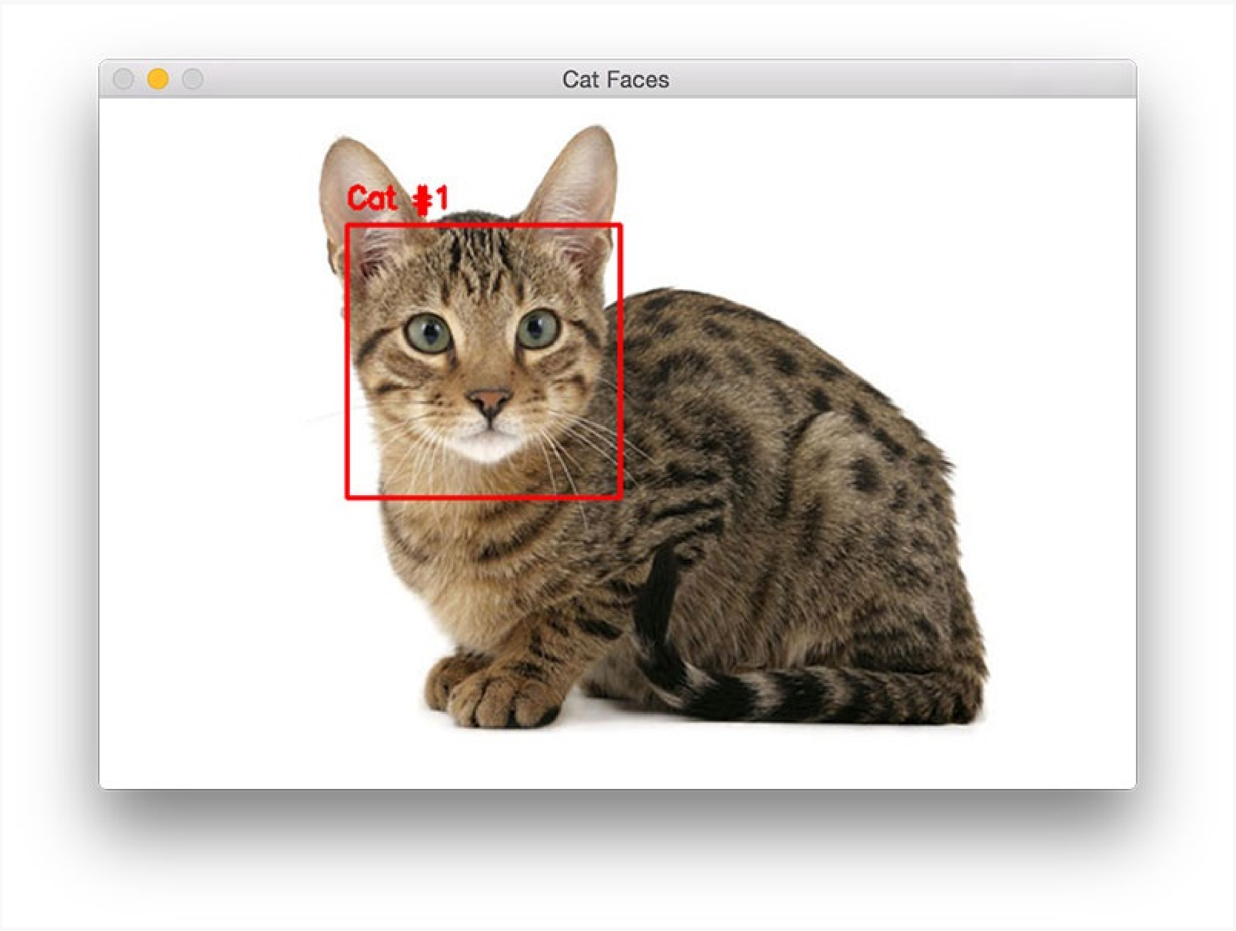


Figure 3: Cat detection with OpenCV and Python (source).

Our final example demonstrates detecting *multiple* cats in an image using OpenCV and Python:

```
$ python cat_detector.py --image images/cat_04.jpg
```

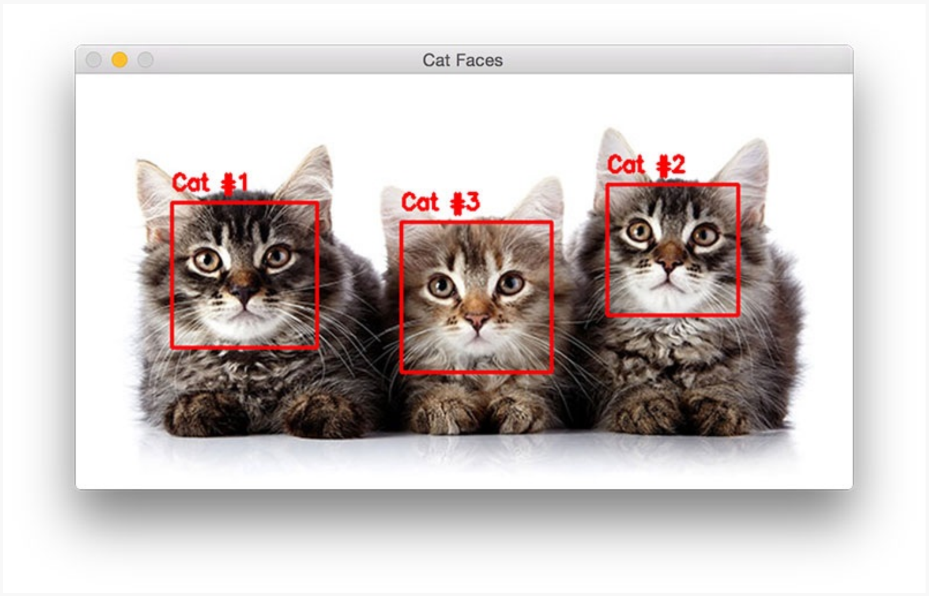


Figure 4: Detecting multiple cats in the same image with OpenCV (source).

Note that the Haar cascade can return bounding boxes in an order that you may not like. In this case, the middle cat is actually labeled as the third cat. You can resolve this "issue" by sorting the bounding boxes according to their *(x, y)*-coordinates for a consistent ordering.

A quick note on accuracy

It's important to note that in the comments section of the `.xml` files, Joseph Howe details that the cat detector Haar cascades can report *cat faces* where there are actually *human faces*.

In this case, he recommends performing *both* face detection *and* cat detection, then discarding any cat bounding boxes that *overlap* with the face bounding boxes.

A note on Haar cascades

First published in 2001 by Paul Viola and Michael Jones, *Rapid Object Detection using a Boosted Cascade of Simple Features*, this original work has become one of the most cited papers in computer vision.

This algorithm is capable of detecting objects in images, regardless of their location and scale. And perhaps most intriguing, the detector can run in real-time on modern hardware.

In their paper, Viola and Jones focused on training a *face detector*; however, the framework can also be used to train detectors for arbitrary "objects", such as cars, bananas, road signs, etc.

The problem?

The biggest problem with Haar cascades is getting the `detectMultiScale` parameters right, specifically `scaleFactor` and `minNeighbors`. You can easily run into situations where you need to tune *both* of these parameters on an image-by-image basis, which is far from ideal when utilizing an object detector.

The `scaleFactor` variable controls your image pyramid used to detect objects at various scales of an image. If your `scaleFactor` is too large, then you'll only evaluate a few layers of the image pyramid, potentially leading to you missing objects at scales that fall *in between* the pyramid layers.

On the other hand, if you set `scaleFactor` too low, then you evaluate *many* pyramid layers. This will help you detect more objects in your image, but it (1) makes the detection process slower and (2) ***substantially*** increases the false-positive detection rate, something that Haar cascades are known for.

To remember this, we often apply **Histogram of Oriented Gradients + Linear SVM detection** instead.

The HOG + Linear SVM framework parameters are normally much easier to tune — and best of all, HOG + Linear SVM enjoys a *much smaller false-positive detection rate*. The only downside is that it's harder to get HOG + Linear SVM to run in real-time.

Interested in learning more about object detection?

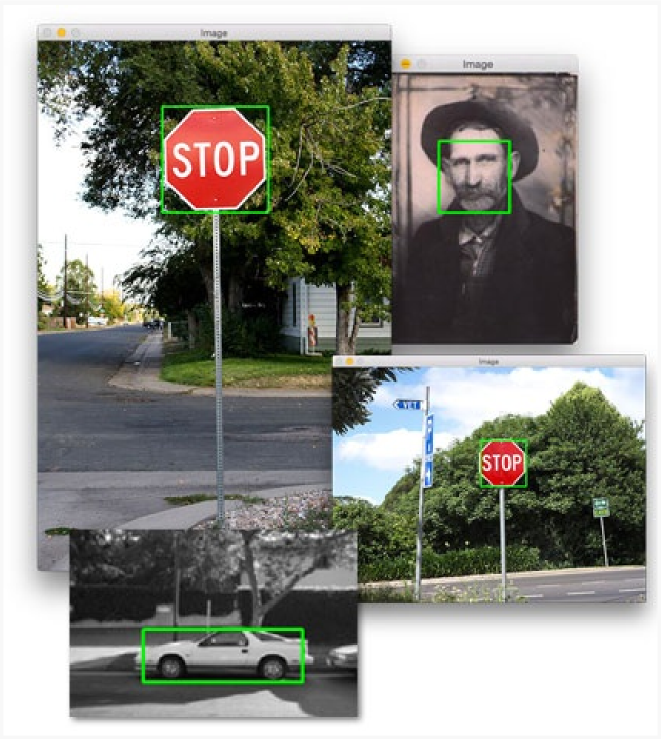


Figure 5: Learn how to build custom object detectors inside the PyImageSearch Gurus course.

If you're interested in **learning how to train your own custom object detectors**, be sure to take a look at the PyImageSearch Gurus course.

Inside the course, I have **15 lessons** covering **168 pages of tutorials** dedicated to teaching you how to build custom object detectors from scratch. You'll discover how to detect *road signs*, *faces*, *cars* (and nearly any other object) in images by applying the HOG + Linear SVM framework for object detection.

To learn more about the PyImageSearch Gurus course (and grab 10 FREE sample lessons), just click the button below:

Click here to learn more about PyImageSearch Gurus!

Summary

In this blog post, we learned how to detect cats in images using the default Haar cascades shipped with OpenCV. These Haar cascades were trained and contributed to the OpenCV project by Joseph Howse, and were originally brought to my attention in this post by Kendrick Tan.

While Haar cascades are quite useful, we often use HOG + Linear SVM instead, as it's a bit easier to tune the detector parameters, and more importantly, we can enjoy a *much* lower false-positive detection rate.

I detail how to build custom HOG + Linear SVM object detectors to recognize various objects in images, including cars, road signs, and much more *inside the PyImageSearch Gurus course.*

Anyway, I hope you enjoyed this blog post!

Before you go, be sure to signup for the PyImageSearch Newsletter using the form below to be notified when new blog posts are published.

Downloads:

If you would like to download the code and images used in this post, please enter your email address in the form below. Not only will you get a .zip of the code, I'll also send you a **FREE 11-page Resource Guide** on Computer Vision and Image Search Engines, including **exclusive techniques** that I don't post on this blog! Sound good? If so, enter your email address and I'll send you the code immediately!

Email address:

Your email address

DOWNLOAD THE CODE!

classification, **haar cascades**, **object detection**

‹ Considerations when setting up deep learning hardware

My Top 9 Favorite Python Deep Learning Libraries ›

16 Responses to *Detecting cats in images with OpenCV*

Joseph Howse June 20, 2016 at 6:36 pm #

REPLY

Thanks so much for featuring the cat face cascades! I am delighted to see them in action here alongside your other great tutorials!

Fun facts:

The haarcascade_frontalcatface_extended.xml version uses the "extended" Haar feature set to make it sensitive to diagonal features such as ears and whiskers. >^-^<

Besides the Haar versions, you will find an LBP version, lbpcascade_frontalcatface.xml, in OpenCV's lbpcascades folder. The LBP version is less accurate but faster; you might like it for Raspberry Pi or other platforms with limited resources.

Details about the training of the cat face cascades can be found in my book, OpenCV for Secret Agents, and in free presentations on my website's OpenCV landing page (http://www.nummist.com/opencv/).

Nine lives,

Joe

Adrian Rosebrock June 23, 2016 at 1:34 pm #

REPLY

You're the one we should be thanking Joe — you trained the actual cascades! 🙂

Kendrick Tan June 20, 2016 at 11:46 pm #

REPLY

Hi Adrian, I'm a bit stoked when I saw my name come up on the newsletter.

Anyway, I just came here to say thank you. Your tutorials really gave me a firm understanding of the basics of computer vision, and machine learning. Really appreciate your time and effort.

Adrian Rosebrock June 23, 2016 at 1:33 pm #

REPLY

Hey Kendrick! Thanks for doing the initial awesome post — without your post, the cat detection cascade would have totally passed me by. I should be the one thanking *you* 🙂

Linus June 21, 2016 at 7:39 am #

REPLY

Wow.
Thanks Adrian, this is so awesome! I've thought about the possibitity of detecting animals some time ago, but I've never found a

solution 😛
What about other animals (dogs etc.)? Are there also xml files available? And is using this in a live camera feed also possibe, with a moderate execution speed?

Adrian Rosebrock June 23, 2016 at 1:29 pm # REPLY

As far as I know, there is only the Haar cascade for cats — there isn't one for dogs. Although you could certainly train one. I would suggest using HOG + Linear SVM instead since it has a lower false-positive rate.

And yes, this method can be used to run in real-time. Take a look at Practical Python and OpenCV for more details on running Haar cascades in real-time.

Ranjeet Singh June 21, 2016 at 12:06 pm # REPLY

But now I want to know what that xml file is ? How it has been written ?

Adrian Rosebrock June 23, 2016 at 1:27 pm # REPLY

The XML file is a serialized Haar cascade. It is generated by training a machine learning algorithm to recognize various objects in images. To recognize your own objects, you'll need to train your own custom object detector.

swight November 10, 2016 at 7:23 pm # REPLY

How would I write the final image to file instead of displaying to screen?

Adrian Rosebrock November 14, 2016 at 12:16 pm # REPLY

You can just use the `cv2.imwrite` function:

```
cv2.imwrite("/path/to/my/image.jpg", image)
```

If you need help learning the basics of computer vision and image processing I would suggest you work through Practical Python and OpenCV.

Victor December 17, 2016 at 9:34 pm # REPLY

We trained an LBP cascade with better time and accuracy performance. If anyone wants this, we can push it to our 10imaging/opencv repo on GitHub.

Ridge December 19, 2016 at 8:01 pm # REPLY

@Victor – Sure, I'd like to see it.

Gismo Scha March 15, 2017 at 8:28 pm # REPLY

Thanks Adrian, it works!
How can i extract / copy the rectangled area and write to file?

Adrian Rosebrock March 17, 2017 at 9:31 am # REPLY

You would use NumPy array slicing to extract the bounding box and then use `cv2.imwrite` to write the image to disk. I discuss these basic operations inside Practical Python and OpenCV. I would suggest you start there if you are new to OpenCV.

Harman Singh March 18, 2017 at 5:06 pm #

REPLY

Thank Adrian, your tutorials are great and fun and also detects fairly good but while deploying this script on multiple images with different size, the output is not to accurate.

for example in 1st pic (cat rolled in paper) i got 2 rectangle one covering the face and other on the nose..(to get 1 rect, i adjusted scaleFactor to 1.715)

even after converting all the image to size still getting mixed result like mentioned above and sometimes none.

do i have to adjust scalefactor everytime? or is it me having this problem?

Thank You.

Adrian Rosebrock March 21, 2017 at 7:35 am #

REPLY

This is a common problem with Haar cascades (they are very prone to false-positives). You can read more about the issue in this blog post on Histogram of Oriented Gradients.

Leave a Reply

Name (required)

Email (will not be published) (required)

Website

SUBMIT COMMENT

Resource Guide (it’s totally free).

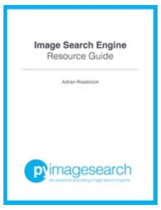


Image Search Engine
Resource Guide
Adrian Rosebrock
pyimagesearch

Click the button below to get my **free 11-page Image Search Engine Resource Guide PDF**. Uncover **exclusive techniques** that I don't publish on this blog and start building image search engines of your own.

Download for Free!

Deep Learning for Computer Vision with Python Book

You're interested in deep learning and computer vision, *but you don't know how to get started.* Let me help. **My new book will teach you all you need to know about deep learning.**

CLICK HERE TO PRE-ORDER MY NEW BOOK

You can detect faces in images & video.

Are you interested in **detecting faces in images & video?** But **tired of Googling for tutorials** that *never work?* Then let me help! I guarantee that my new book will turn you into a **face detection ninja** by the end of this weekend. Click here to give it a shot yourself.

CLICK HERE TO MASTER FACE DETECTION

PyImageSearch Gurus: NOW ENROLLING!

The PyImageSearch Gurus course is *now enrolling!* Inside the course you'll learn how to perform:

- Automatic License Plate Recognition (ANPR)
- Deep Learning
- Face Recognition
- *and much more!*

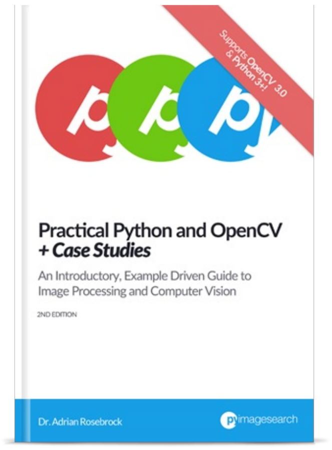
Click the button below to learn more about the course, take a tour, and get 10 (FREE) sample lessons.

TAKE A TOUR & GET 10 (FREE) LESSONS

Hello! I'm Adrian Rosebrock.

I'm an entrepreneur and Ph.D who has launched two successful image search engines, ID My Pill and Chic Engine. I'm here to share my tips, tricks, and hacks I've learned along the way.

Learn computer vision in a single weekend.



Supports OpenCV 3.0 & Python 3+!
Practical Python and OpenCV + Case Studies
An Introductory, Example Driven Guide to Image Processing and Computer Vision
2ND EDITION
Dr. Adrian Rosebrock
pyimagesearch

Want to learn computer vision & OpenCV? I can teach you in a **single weekend**. I know. It sounds crazy, but it's no joke. My new book is your **guaranteed, quick-start guide** to becoming an OpenCV Ninja. So why not give it a try? Click here to become a computer vision ninja.

CLICK HERE TO BECOME AN OPENCV NINJA

Subscribe via RSS

Never miss a post! Subscribe to the PyImageSearch RSS Feed and keep up to date with my image search engine tutorials, tips, and tricks

POPULAR

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

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

How to install OpenCV 3 on Raspbian Jessie
OCTOBER 26, 2015

Install guide: Raspberry Pi 3 + Raspbian Jessie + OpenCV 3
APRIL 18, 2016

Basic motion detection and tracking with Python and OpenCV
MAY 25, 2015

Install OpenCV 3.0 and Python 2.7+ on Ubuntu
JUNE 22, 2015

Accessing the Raspberry Pi Camera with OpenCV and Python
MARCH 30, 2015

Search

Search...

Find me on **Twitter**, **Facebook**, **Google+**, and **LinkedIn**.