

PyImageSearch Gurus Course

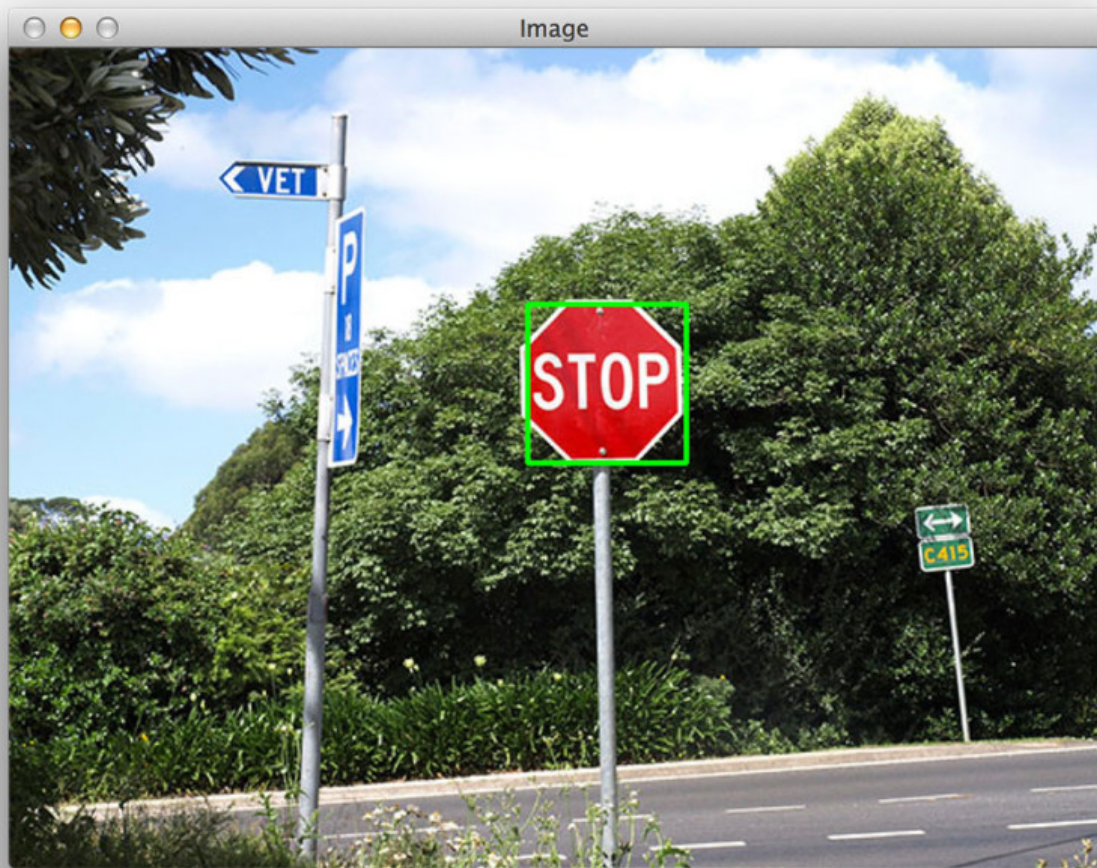
[🏠 \(HTTPS://GURUS.PYIMAGESEARCH.COM\)](https://gurus.pyimagesearch.com/) >

2.2.2: Object detection made easy

Topic Progress: (<https://gurus.pyimagesearch.com/topic/how-to-install-dlib/>)

(<https://gurus.pyimagesearch.com/topic/object-detection-made-easy/>)

← [Back to Lesson \(https://gurus.pyimagesearch.com/lessons/object-detection-the-easy-way/\)](https://gurus.pyimagesearch.com/lessons/object-detection-the-easy-way/)



(<https://gurus.pyimagesearch.com/wp-content/uploads/2015/08/object-detection-easy-stop-sign.jpg>).

Our previous two lessons reviewed **what an object detector is** (<https://gurus.pyimagesearch.com/lessons/what-are-object-detectors/>), followed by **installing the dlib library** (<https://gurus.pyimagesearch.com/topic/how-to-install-dlib/>) on our system.

In this lesson, we'll write some code to perform actual object recognition, specifically *recognizing stop signs in images*.

This lesson will be very practical, hands-on, and focus heavily on the code required to train an object detector. We won't be worrying much about the underlying theory or process required to train the actual detector — that's what the rest of the lessons in this module are for!

Objectives:

Our primary objectives for this lesson include:

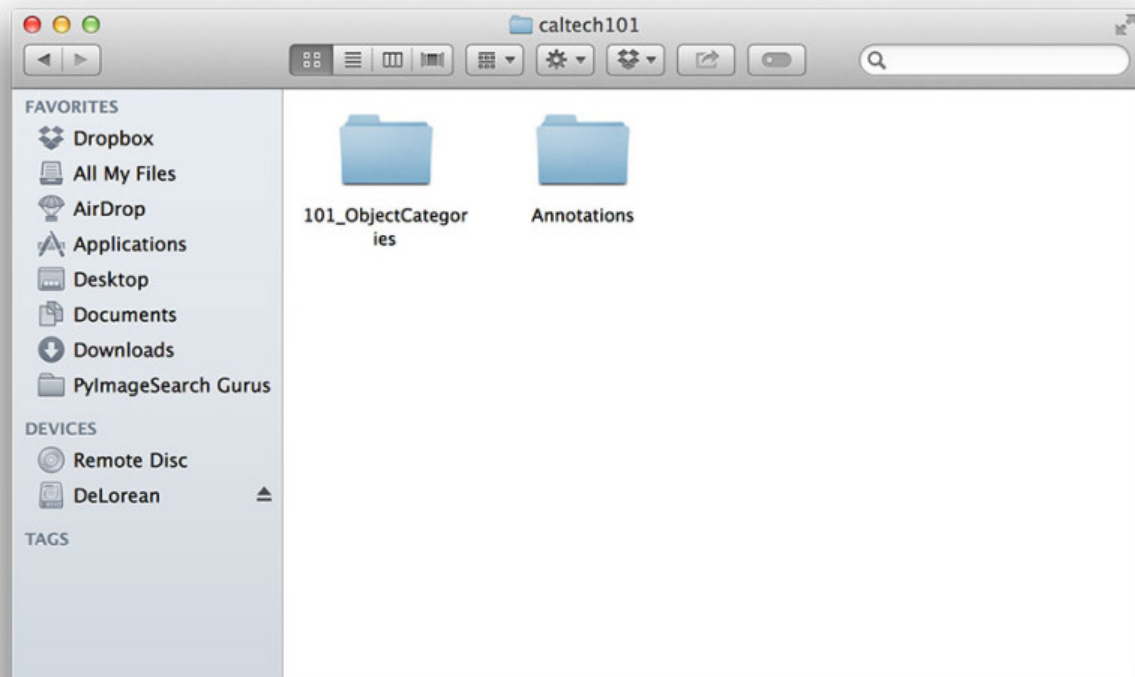
- Introducing the CALTECH-101 dataset for object recognition.
- Exploring the training data and how to structure the training detector in order to build an object detector.
- Leveraging the dlib library to train a classifier in detecting the presence of stop signs in images.

The CALTECH-101 dataset

Introduced by Fei-Fei et al. in 2004, the [CALTECH-101](http://www.vision.caltech.edu/Image_Datasets/Caltech101/) (http://www.vision.caltech.edu/Image_Datasets/Caltech101/), dataset is a very popular benchmark dataset for object detection and has been used by many researchers, academics, and computer vision developers to evaluate their object detection algorithms.

The dataset includes 101 categories, spanning a diverse range of objects including elephants, bicycles, soccer balls, and even human brains, just to name a few.

When you download the CALTECH-101 dataset, you'll notice that it includes both an `images` and an `annotations` directory:



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

FIGURE 1: THE CALTECH-101 DATASET STRUCTURE.

For each image in the dataset, an associated bounding box (i.e. (x, y)-coordinates of the object) is provided. Our goal is to take both the *images* and the *bounding boxes* (i.e. the annotations) and train a classifier to detect the presence of a given object in an image:



[12, 96, 179, 256]



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

FIGURE 2: GIVEN AN IMAGE CONTAINING A STOP SIGN (LEFT) AND THE (X, Y) ANNOTATIONS OF THE STOP SIGN IN THE IMAGE (CENTER), OUR GOAL IS TO TRAIN A CLASSIFIER TO DETECT THE STOP SIGN IN THE IMAGE (RIGHT).

For the sake of this lesson, we'll be training only a single classifier using the CALTECH-101 dataset — a *stop sign detector*.

Training your own object detector

Training an object detector is very similar to training a classifier to (globally) classify the contents of an image, just like we did in the [Logistic Regression](https://gurus.pyimagesearch.com/topic/logistic-regression/) (<https://gurus.pyimagesearch.com/topic/logistic-regression/>), [Support Vector Machines](https://gurus.pyimagesearch.com/topic/support-vector-machines/) (<https://gurus.pyimagesearch.com/topic/support-vector-machines/>), [Decision Trees](https://gurus.pyimagesearch.com/topic/decision-trees/) (<https://gurus.pyimagesearch.com/topic/decision-trees/>), and [Random Forest](https://gurus.pyimagesearch.com/topic/random-forests/) (<https://gurus.pyimagesearch.com/topic/random-forests/>) lessons. However, this time we are presented with not only the *labels* of the images, but also annotations corresponding to the *bounding box* surrounding each object. We'll take these bounding boxes, extract features from them, and then use these features to build our object detector.

Like I said, this lesson is meant to be very hands-on and practical, so let's go ahead and get started — we'll dive into the theory and underlying concepts in the rest of this module.

train_detector.py

Python

```

1 # import the necessary packages
2 from __future__ import print_function
3 from imutils import paths
4 from scipy.io import loadmat
5 from skimage import io
6 import argparse
7 import dlib
8 import sys
9
10 # handle Python 3 compatibility
11 if sys.version_info > (3,):
12     long = int
13
14 # construct the argument parse and parse the arguments
15 ap = argparse.ArgumentParser()
16 ap.add_argument("-c", "--class", required=True,
17     help="Path to the CALTECH-101 class images")
18 ap.add_argument("-a", "--annotations", required=True,
19     help="Path to the CALTECH-101 class annotations")
20 ap.add_argument("-o", "--output", required=True,
21     help="Path to the output detector")
22 args = vars(ap.parse_args())

```

Take a second and examine our import section. You'll probably notice a few packages that you're unfamiliar with. The first is the `loadmat` function from `scipy`. The annotations/bounding boxes for the CALTECH-101 dataset are actually `.mat` files which are Matlab files, similar to `.pickle` files for Python and NumPy — they are simply the serialized bounding boxes for each image.

We'll also be using `scikit-image` instead of OpenCV for our `train_detector.py` script, mainly because we'll also be using `dlib` which is built to play nice with `scikit-image`.

Looking at our argument parsing section, you'll notice that we need three switches:

1. `--class` : This is the path to our *specific* CALTECH-101 class that we want to train an object detector for. For this example, we'll be using the *stop sign* class.
2. `--annotations` : For each image in the dataset, we also have the corresponding bounding boxes for each object in the image — the `--annotations` switch specifies the path to our bounding boxes directly for the specific class we are training on.
3. `--output` : After our model has been trained, we would like to dump it to file — this is the path to our output classifier.

Now, let's gather the images and bounding boxes to train our classifier:

train_detector.py	Python

```

24 # grab the default training options for our HOG + Linear SVM detector, then initialize the
25 # list of images and bounding boxes used to train the classifier
26 print("[INFO] gathering images and bounding boxes...")
27 options = dlib.simple_object_detector_training_options()
28 images = []
29 boxes = []
30
31 # loop over the image paths
32 for imagePath in paths.list_images(args["class"]):
33     # extract the image ID from the image path and load the annotations file
34     imageID = imagePath[imagePath.rfind("/") + 1:].split("_")[1]
35     imageID = imageID.replace(".jpg", "")
36     p = "{}/{}/annotation_{}.mat".format(args["annotations"], imageID)
37     annotations = loadmat(p)["box_coord"]
38
39     # loop over the annotations and add each annotation to the list of bounding
40     # boxes
41     bb = [dlib.rectangle(left=long(x), top=long(y), right=long(w), bottom=long(h))
42           for (y, h, x, w) in annotations]
43     boxes.append(bb)
44
45     # add the image to the list of images
46     images.append(io.imread(imagePath))

```

We start off on **Line 27** by grabbing the default `options` for our dlib object detector. If you were to print these options to your console and examine them, you would see they mostly control HOG parameters such as the detection window size and whether horizontal flips of the image should be used as additional training data; along with the Support Vector Machine the dlib library is using under the hood:

dlib.simple_object_detector_training_options	Shell
1 \$ python	
2 >>> import dlib	
3 >>> options = dlib.simple_object_detector_training_options()	
4 >>> for opt in dir(options):	
5 ... print(opt)	
6 ...	
7 C	
8 ...	
9 add_left_right_image_flips	
10 be_verbose	
11 detection_window_size	
12 epsilon	
13 num_threads	

We'll also initialize the `images` list to store the images we are using to train our classifier as well as initialize the `boxes` list to store the bounding boxes for each of the images.

From there, we start looping over each of our input images on **Line 32**.

Lines 34-37 extract the image ID from the path and then use the image ID to load the corresponding `annotations` (i.e. bounding boxes) from disk. You can see the example `annotations` for image `image_0007.jpg` below:

Example annotations for image_0007.jpg	Shell
1 [[39 161 83 209]]	

Now that we have the (x, y)-coordinates of the stop sign in the image, we now need to construct the `rectangle` objects to represent the bounding box (**Lines 41 and 42**).

Once we have the bounding boxes, all we need to do is update the list of bounding boxes for the current image (**Line 43**) and add the `image` to the list of `images` on **Line 46** — the dlib library will need both `images` and `boxes` to train the classifier.

Believe it or not, the hard part is actually done! The dlib library will perform the rest of the heavy-lifting for us:

train_detector.py	Python
<pre> 48 # train the object detector 49 print("[INFO] training detector...") 50 detector = dlib.train_simple_object_detector(images, boxes, options) 51 52 # dump the classifier to file 53 print("[INFO] dumping classifier to file...") 54 detector.save(args["output"]) 55 56 # visualize the results of the detector 57 win = dlib.image_window() 58 win.set_image(detector) 59 dlib.hit_enter_to_continue() </pre>	

Feedback

Training the object detector is performed on **Line 50** by making a call to the `train_simple_object_detector` dlib function, then passing in our list of `images`, bounding boxes for each image, and our dlib `options`.

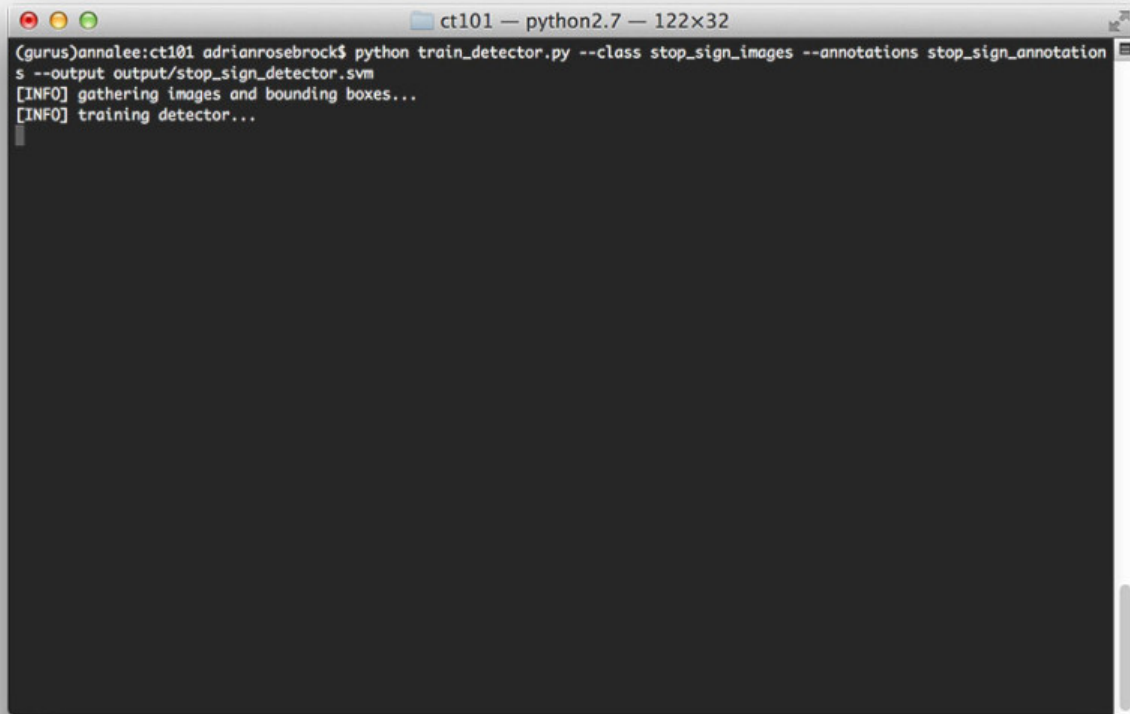
Line 54 takes our trained detector and dumps it to file.

Finally, **Lines 57-59** visualize the **Histogram of Oriented Gradients filter** (<https://gurus.pyimagesearch.com/lessons/histogram-of-oriented-gradients/>), so we can debug our system and ensure that the filter looks “something” like what we want to detect.

To train our classifier, just issue the following command:

train_detector.py	Shell
<pre> 1 \$ python train_detector.py --class stop_sign_images --annotations stop_sign_annotations \ 2 --output output/stop_sign_detector.svm </pre>	

You’ll notice from your terminal output that the object detector will start to train:

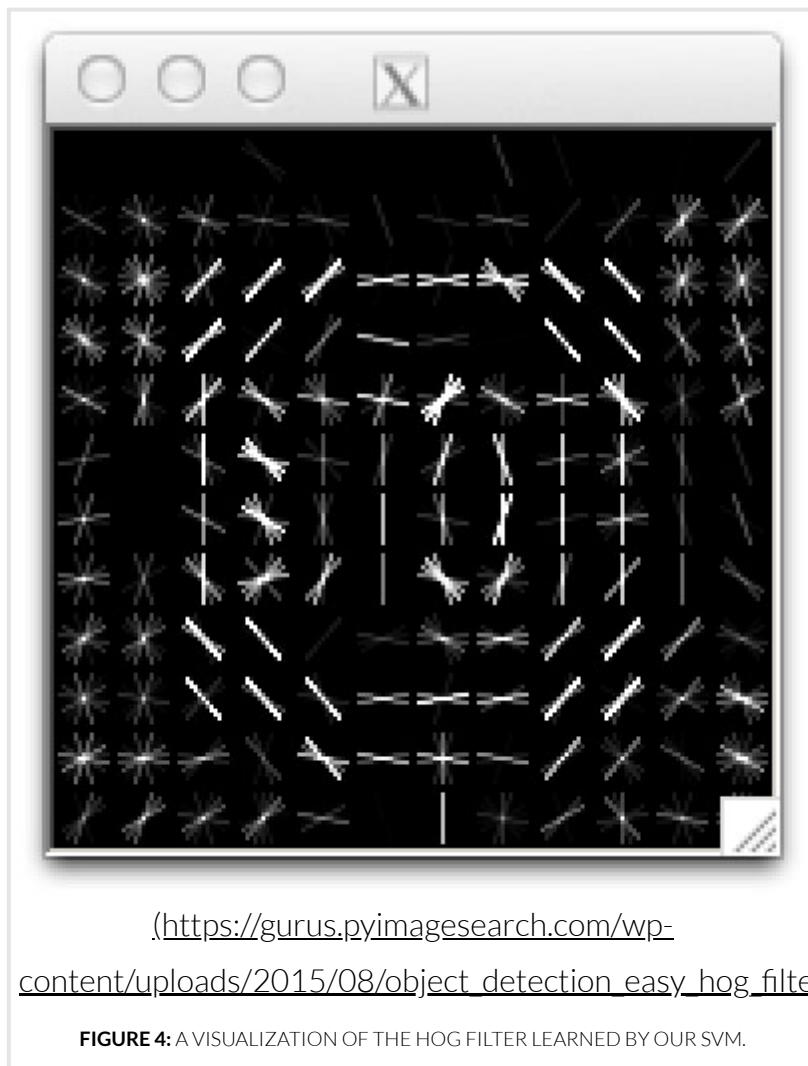
A terminal window titled 'ct101 — python2.7 — 122x32' is shown. The prompt is '(gurus)annalee:ct101 adrianrosebrock\$'. The command entered is 'python train_detector.py --class stop_sign_images --annotations stop_sign_annotation.s --output output/stop_sign_detector.svm'. The output shows two lines of information: '[INFO] gathering images and bounding boxes...' and '[INFO] training detector...'.

```
ct101 — python2.7 — 122x32
(gurus)annalee:ct101 adrianrosebrock$ python train_detector.py --class stop_sign_images --annotations stop_sign_annotation.s --output output/stop_sign_detector.svm
[INFO] gathering images and bounding boxes...
[INFO] training detector...
```

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

FIGURE 3: KICKING OFF THE STOP SIGN TRAINING PROCESS.

Followed by being dumped to file and the visualized HOG filter displayed to your screen:



As you can see, the HOG filter looks very similar to a stop sign!

Now that our object detector has been trained, we can use it to detect the presence of stop signs in new testing images.

Testing your object detector

In order to test our object detector, I have gathered 11 images of stop signs from Google that our classifier has not been trained on:



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

FIGURE 5: THE SET OF 11 IMAGES GATHERED FROM GOOGLE (AND NOT PART OF THE TRAINING SET) WE WILL BE USING TO EVALUATE OUR STOP SIGN DETECTOR.

We'll be using these images to evaluate our object detector and see if it can detect the presence of the stop sign in unseen images.

Let's go ahead and get started writing the testing code:

test_detector.py	Python
<pre> 1 # import the necessary packages 2 from imutils import paths 3 import argparse 4 import dlib 5 import cv2 6 7 # construct the argument parse and parse the arguments 8 ap = argparse.ArgumentParser() 9 ap.add_argument("-d", "--detector", required=True, help="Path to trained object detector") 10 ap.add_argument("-t", "--testing", required=True, help="Path to directory of testing images") 11 args = vars(ap.parse_args()) 12 13 # load the detector 14 detector = dlib.simple_object_detector(args["detector"]) </pre>	

In order to run our `test_detector.py` script, we'll need two switches. The first is the `--detector`, which is the path to our trained stop sign detector we created in the previous step. The second switch, `--testing`, is the path to the directory containing our stop sign images for testing.

Line 14 then loads our object detector from disk.

Now, for the fun part:

<pre>test_detector.py 16 # loop over the testing images 17 for testingPath in paths.list_images(args["testing"]): 18 # load the image and make predictions 19 image = cv2.imread(testingPath) 20 boxes = detector(cv2.cvtColor(image, cv2.COLOR_BGR2RGB)) 21 22 # loop over the bounding boxes and draw them 23 for b in boxes: 24 (x, y, w, h) = (b.left(), b.top(), b.right(), b.bottom()) 25 cv2.rectangle(image, (x, y), (w, h), (0, 255, 0), 2) 26 27 # show the image 28 cv2.imshow("Image", image) 29 cv2.waitKey(0)</pre>	Python
---	--------

We start looping over our testing images on **Line 17**. For each of these testing images, we load it from disk and then use our classifier to detect the presence of stop signs on **Line 20**. Our `detector` returns us a list of `boxes`, corresponding to the (x, y)-coordinates of the detected stop signs.

It's important to note that we take special care to convert our image from the BGR color space to the RGB color space before calling our `detector`. Remember, in our `train_detector.py` script, we used scikit-image which represents images in RGB order. But now that we are using OpenCV (which represents images in BGR order), we need to convert from BGR to RGB before calling the dlib object detector.

Finally, **Lines 22-29** loop over the bounding boxes returned by the `detector` and draws them on our `image`.

Stop sign detection results

After all this hard work, let's see our stop sign detector in action. Just execute the following command:

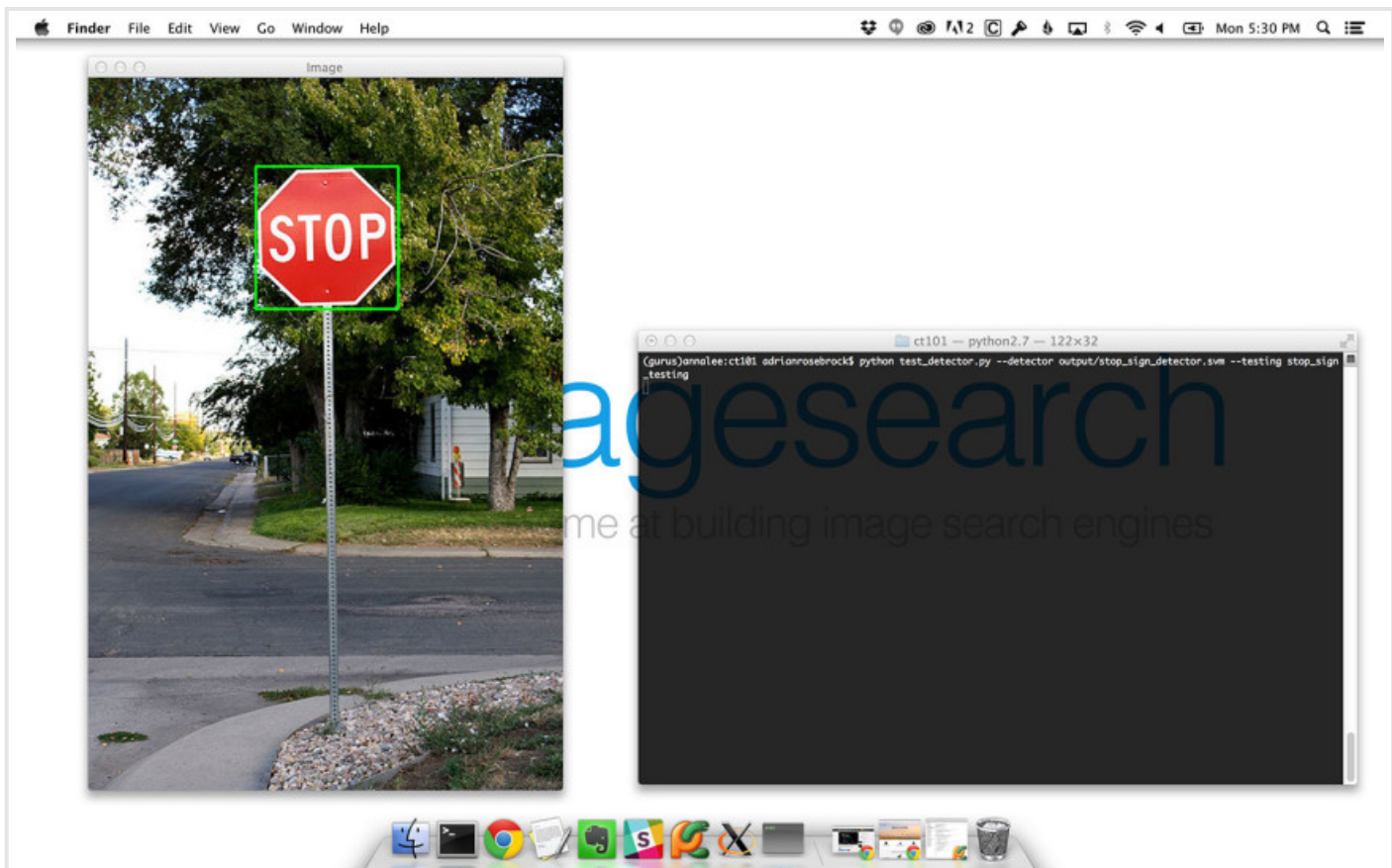
<pre>test_detector.py 1 \$ python test_detector.py --detector output/stop_sign_detector.svm --testing stop_sign_testing</pre>	Shell
---	-------

Below, I have included a sample of results from our testing set. Notice that in each case our object detector is able to correctly detect the location of the stop sign:



(https://gurus.pyimagesearch.com/wp-content/uploads/2015/08/object_detection_easy_stop_sign_01.jpg).

FIGURE 6: USING OUR OBJECT DETECTOR TO DETECT THE PRESENCE OF A STOP SIGN IN AN IMAGE.



(https://gurus.pyimagesearch.com/wp-content/uploads/2015/08/object_detection_easy_stop_sign_01.jpg).

[content/uploads/2015/08/object_detection_easy_stop_sign_02.jpg](#)

FIGURE 7: A SECOND STOP SIGN DETECTION EXAMPLE.





Again, this is just a sample of results from our testing set. Be sure to download the code at the bottom of this lesson to see the results on all testing images.

The main takeaway from these results is that our HOG object detection algorithm is *substantially more robust* than using simple **template matching** (<https://gurus.pyimagesearch.com/topic/template-matching/>). However, there is a lot going on under the hood when using a HOG detector that we have not discussed yet. In order to fully appreciate and understand the HOG + Linear SVM algorithm, we'll be peeling back the curtains and taking a deep dive into the inner workings of the algorithm in the remainder of this module.

Summary

We started this lesson by familiarizing ourselves with the CALTECH-101 dataset for object detection.

We then trained our very first object detector, a stop sign detector, using the dlib library and the stop sign images from the CALTECH-101 dataset for training data.

Finally, we evaluated our stop sign detector on images downloaded from Google Image Search — our object detector had *never before see these images* and was still able to detect the presence of the stop signs in *all* evaluation images, demonstrating that our object detector is working quite well, especially given the minimal effort we put into it.

In the next few lessons of this module, we'll dive deeper into the underlying theory of object detection. We'll also take a look at the process and algorithms being applied behind the scenes.

Downloads:

[Download the Code](#)

https://gurus.pyimagesearch.com/protected/code/object_detector/object_detector.py

Quizzes		Status
1	Object Detection Made Easy Quiz (https://gurus.pyimagesearch.com/quizzes/object-detection-made-easy-quiz/)	

[← Previous Topic](#) (<https://gurus.pyimagesearch.com/topic/how-to-install-dlib/>).

Feedback

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! \(/pyimagesearch-gurus-course/\)](/pyimagesearch-gurus-course/)

Resources & Links

- [PyImageSearch Gurus Community](https://community.pyimagesearch.com/) (<https://community.pyimagesearch.com/>).
- [PyImageSearch Virtual Machine](https://gurus.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/) (<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/) (<https://gurus.pyimagesearch.com/course-syllabus-content-release-schedule/>).

- [Member Perks & Discounts \(https://gurus.pyimagesearch.com/pyimagesearch-gurus-discounts-perks/\)](https://gurus.pyimagesearch.com/pyimagesearch-gurus-discounts-perks/).
- [Your Achievements \(https://gurus.pyimagesearch.com/achievements/\)](https://gurus.pyimagesearch.com/achievements/).
- [Official OpenCV documentation \(http://docs.opencv.org/index.html\)](http://docs.opencv.org/index.html).

Your Account

- [Account Info \(https://gurus.pyimagesearch.com/account/\)](https://gurus.pyimagesearch.com/account/).
- [Support \(https://gurus.pyimagesearch.com/contact/\)](https://gurus.pyimagesearch.com/contact/).
- [Logout \(https://gurus.pyimagesearch.com/wp-login.php?action=logout&redirect_to=https%3A%2F%2Fgurus.pyimagesearch.com%2F&_wpnonce=5736b21cae\)](https://gurus.pyimagesearch.com/wp-login.php?action=logout&redirect_to=https%3A%2F%2Fgurus.pyimagesearch.com%2F&_wpnonce=5736b21cae).

 Search

© 2018 PyImageSearch. All Rights Reserved.