2.11: Training your custom object detector

So far in this module, we have learned how to train a HOG + Linear SVM object detector on datasets that *already* have labeled bounding boxes.

But what if we wanted to train an object detector on our own datasets that *do not* provide bounding boxes?

How do we go about labeling our images and obtaining these bounding boxes?

And once we have these image annotations, how do we train our object detector?

In the remainder of this lesson, we'll be addressing each of these questions, starting by examining dlib's `imglab` tool, which we can use to annotate our images by drawing bounding boxes surrounding objects in our dataset.

From there, we'll explore how to use dlib to train a custom object detector on these manually labeled bounding boxes.

Objectives:

In this lesson, we will:

- Learn how to use the dlib `imglab` tool to label and annotate your own image datasets.

- Train a HOG + Linear SVM object detector on the annotations from `imglab`.
- Apply your object detector to images not part of the training set.

Training your own custom object detector

Before we dive into the details of this lesson, the first step is to compile dlib's `imglab` tool.

Compiling and using the imglab tool

Luckily, compiling and installing `imglab` is a fairly easy task.

If you do not already have the source code to dlib on your system from our **object detection made easy (https://gurus.pyimagesearch.com/topic/object-detection-made-easy/)** lesson (perhaps you installed via PIP rather than compiling from source), be sure to download the source from the dlib website (http://dlib.net/) using `wget` as shown below.

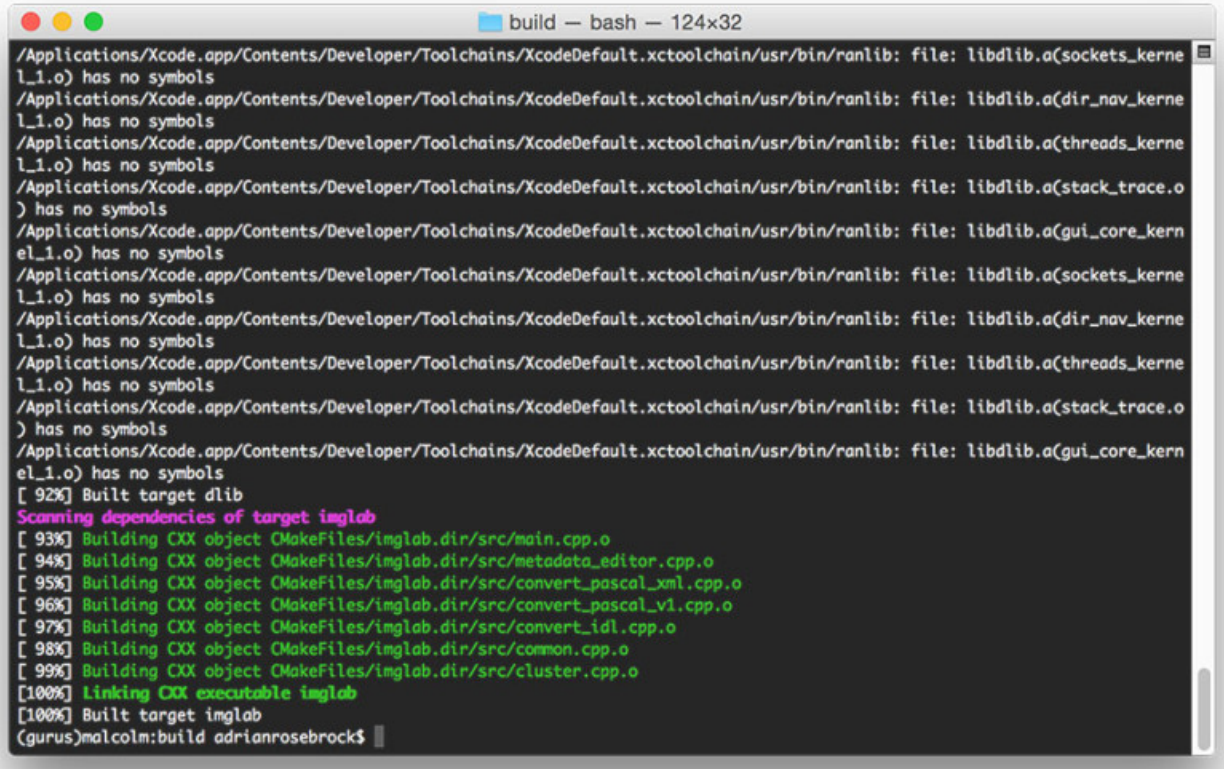
Once you have unpacked the archive, all you need to do is leverage the `cmake` command to take care of the build for you:

Compiling imglab — Shell

```
$ wget http://dlib.net/files/dlib-19.16.tar.bz2
$ tar xvjf dlib-19.16.tar.bz2
$ cd dlib-19.16/tools/imglab
$ mkdir build
$ cd build
$ cmake ..
$ cmake --build . --config Release
$ sudo make install
```

Note: *Gurus VM users have dlib installed in the `gurus` environment via PIP. You also have the `imglab` tool installed and you can call the binary from anywhere on the system by using the command, `imglab`.*

Assuming `imglab` compiled without error, your output should look similar to mine below:



(https://gurus.pyimagesearch.com/wp-content/uploads/2015/03/custom_object_detector_compiling_imglab.jpg)

FIGURE 1: COMPILING IMGLAB.

Feedback

To run `imglab`, you need to supply **two command line arguments** over **two separate commands**:

- The first is your ***output annotations file*** which will contain the bounding boxes you will manually draw on each of the images in your dataset.
- The second argument is the ***dataset path*** which contains the list of images in your dataset.

For this lesson, we'll be using a subset of the **MIT + CMU Frontal Images (http://vasc.ri.cmu.edu/idb/html/face/frontal_images/)** dataset as our *training data*, followed by a subset of the **CALTECH Web Faces (http://www.vision.caltech.edu/Image_Datasets/Caltech_10K_WebFaces/)** dataset for *testing*.

Note: The MIT + CMU Frontal Images server seems to have been taken down. For your convenience, I have included the training and testing datasets in the *Downloads* section at the bottom of this page.

First, let's initialize our annotations file with a list of images in the dataset path:

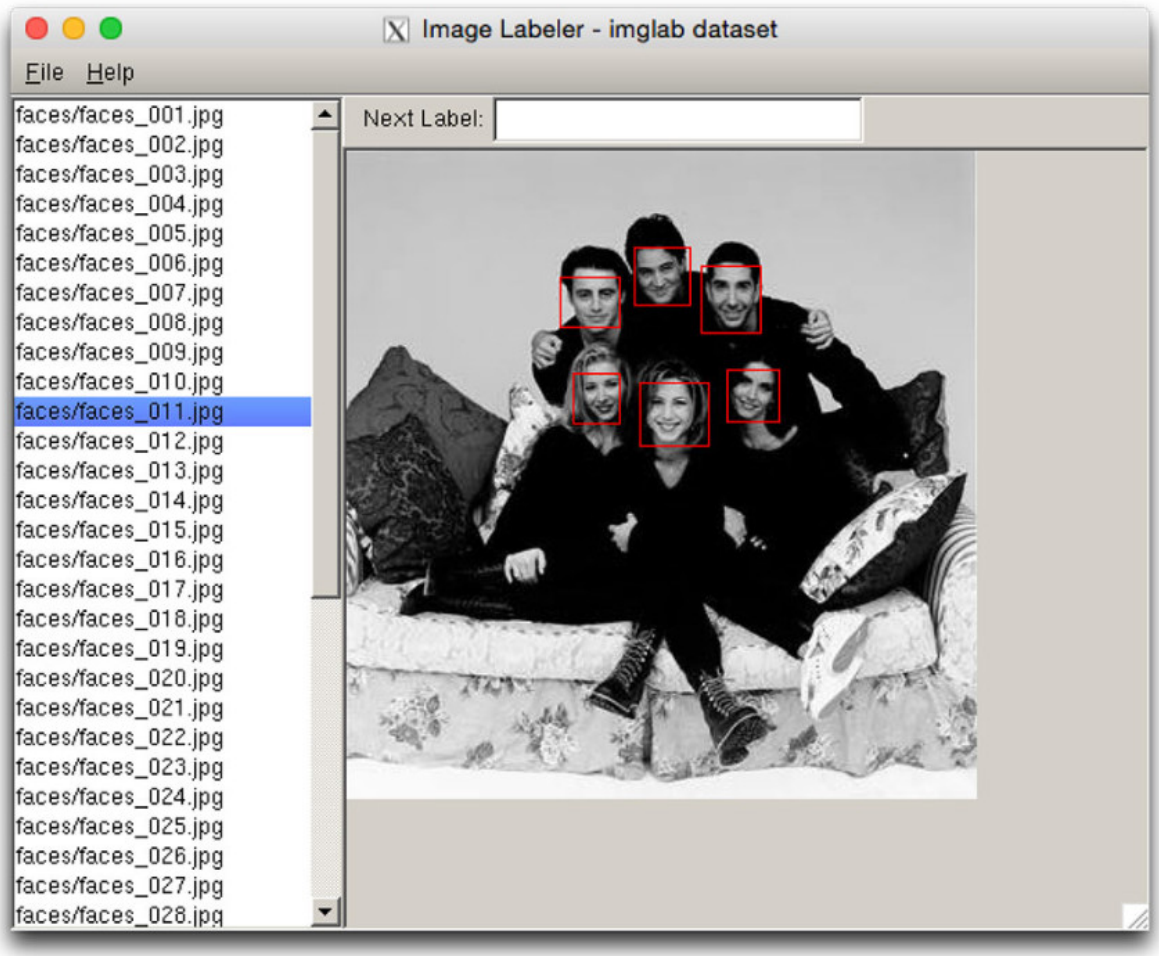
```
1 $ cd ~/Desktop/dlib_custom
2 $ imglab -c face_detector/faces_annotations.xml face_detector/faces
```

Note: *I've provided* `faces_annotations.xml` *pre-annotated in the download. You can name your annotation file differently so as not to overwrite mine.*

From there, we can start the annotation process by using the following command:

```
Starting the annotation process                                    Shell
1 $ imglab face_detector/faces_annotations.xml
```

As you can see, the `imglab` GUI is displayed to my screen, along with the images in my dataset of faces. To draw a bounding box surrounding each object in my dataset, I simply select an image, hold the shift key on my keyboard, and drag-and-draw the bounding rectangle, then release my mouse:



(https://gurus.pyimagesearch.com/wp-content/uploads/2015/03/custom_object_detector_annotating.jpg)

FIGURE 2: DRAWING BOUNDING BOXES SURROUNDING FACES IN AN IMAGE.

Note: It's important to label *all* examples of objects in an image; otherwise, dlib will implicitly assume that regions not labeled are regions that should *not* be detected (i.e., hard-negative mining applied during extraction time).

Finally, if there is an ROI that you are unsure about and want to be ignored entirely during the training process, simply double-click the bounding box and press the `i` key. This will cross out the bounding box and mark it as "ignored".

While annotating a dataset of images is a time consuming and tedious task, you should nonetheless take your time and take special care to ensure the images are properly labeled with their respective bounding boxes.

Remember, machine learning algorithms are only as good as their input data — *if you put garbage in, you'll only get garbage out*. But if you take the time to properly label your images, you'll get much better results.

Training our custom object detector

After annotating and labeling our image dataset, we are now ready to train our object detector:

train_detector.py Python

```
# import the necessary packages
from __future__ import print_function
import argparse
import dlib

# construct the argument parser and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-x", "--xml", required=True, help="path to input XML file")
ap.add_argument("-d", "--detector", required=True, help="path to output director")
args = vars(ap.parse_args())
```

Our `train_detector.py` script requires two command line arguments: the `--xml` path to where our face annotations live, followed by the `--detector` , the path to where we will store our trained classifier.

From there, we can train our actual detector:

train_detector.py Python

```
12 # grab the default training options for the HOG + Linear SVM detector, then
13 # train the detector -- in practice, the `C` parameter should be cross-validated
14 print("[INFO] training detector...")
15 options = dlib.simple_object_detector_training_options()
16 options.C = 1.0
17 options.num_threads = 4
18 options.be_verbose = True
19 dlib.train_simple_object_detector(args["xml"], args["detector"], options)
20
21 # show the training accuracy
22 print("[INFO] training accuracy: {}".format(
23     dlib.test_simple_object_detector(args["xml"], args["detector"])))
24
25 # load the detector and visualize the HOG filter
26 detector = dlib.simple_object_detector(args["detector"])
27 win = dlib.image_window()
28 win.set_image(detector)
29 dlib.hit_enter_to_continue()
```

The most important code block is found in **Lines 15-19** where we define the `options` to our dlib detector. The most important argument to set here is `C`, the "strictness" of our SVM. You are likely familiar with this parameter from our lesson on **training a custom object detector using our framework (https://gurus.pyimagesearch.com/lessons/the-initial-training-phase/)**. In practice, this value needs to be cross-validated and grid-searched to obtain optimal accuracy.

Line 19 then trains the dlib object detector.

Finally, **Lines 22 and 23** displays the output accuracy of the detector to our screen.

To train our object detector, just execute the following command:

train_detector.py — Shell

```
1 $ python train_detector.py --xml face_detector/faces_annotations.xml \
2     --detector face_detector/detector.svm
```

Testing our custom object detector

Next up, let's define the `test_detector.py` script used to detect objects (in this case, faces) in images:

test_detector.py — Python

```python
# import the necessary packages
from imutils import paths
import argparse
import dlib
import cv2

# construct the argument parser and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-d", "--detector", required=True, help="Path to trained object detector")
ap.add_argument("-t", "--testing", required=True, help="Path to directory of testing images")
args = vars(ap.parse_args())

# load the detector
detector = dlib.simple_object_detector(args["detector"])

# loop over the testing images
for testingPath in paths.list_images(args["testing"]):
    # load the image and make predictions
    image = cv2.imread(testingPath)
    boxes = detector(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))

    # loop over the bounding boxes and draw them
    for b in boxes:
        (x, y, w, h) = (b.left(), b.top(), b.right(), b.bottom())
        cv2.rectangle(image, (x, y), (w, h), (0, 255, 0), 2)

    # show the image
    cv2.imshow("Image", image)
    cv2.waitKey(0)
```

If this code looks familiar, it's because we actually reviewed it once before in the **object detection made easy (https://gurus.pyimagesearch.com/topic/object-detection-made-easy/)** lesson.

Lines 2-5 import our necessary Python packages while **Lines 8-11** handle parsing command line arguments. We need two *required* command line arguments here: the path to our custom object `--detector`, followed by the path to our `--testing` directory.

From there, **Line 14** loads our object detector from disk.

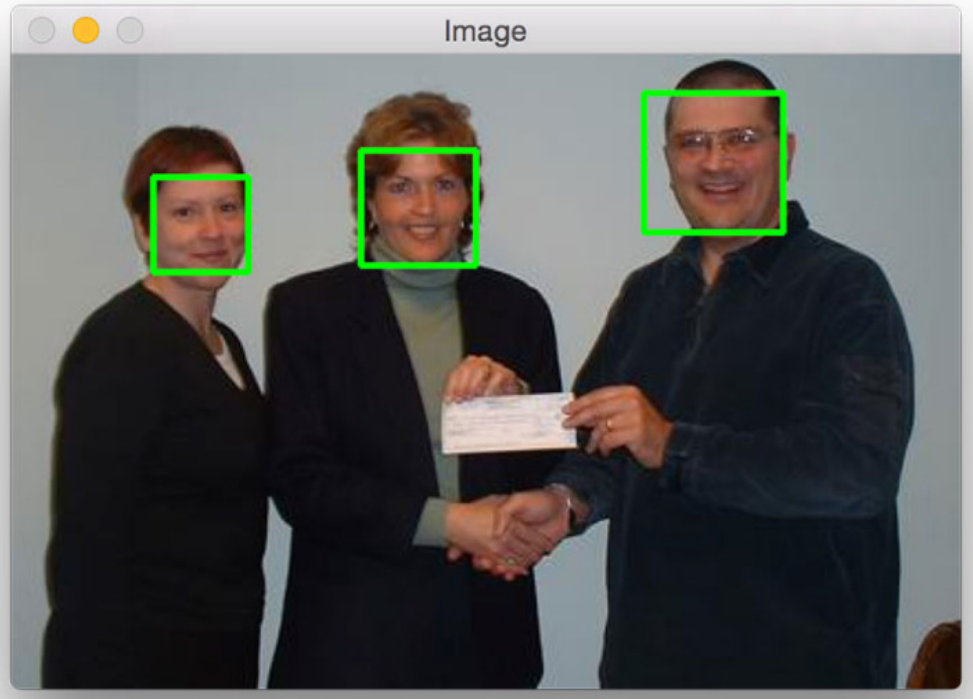
We then loop over the testing images (**Line 17**), load the image and detect objects (i.e., faces) in it (**Lines 19 and 20**), and finally draw the bounding boxes around the objects in the image (**Lines 23-25**).

Lines 28 and 29 display our output image.

To give our face detector a try, just execute the following command:

test_detector.py — Shell

```
$ python test_detector.py --detector face_detector/detector.svm \
    --testing face_detector/testing
```



(https://gurus.pyimagesearch.com/wp-content/uploads/2015/03/custom_object_detector_result_01.jpg)

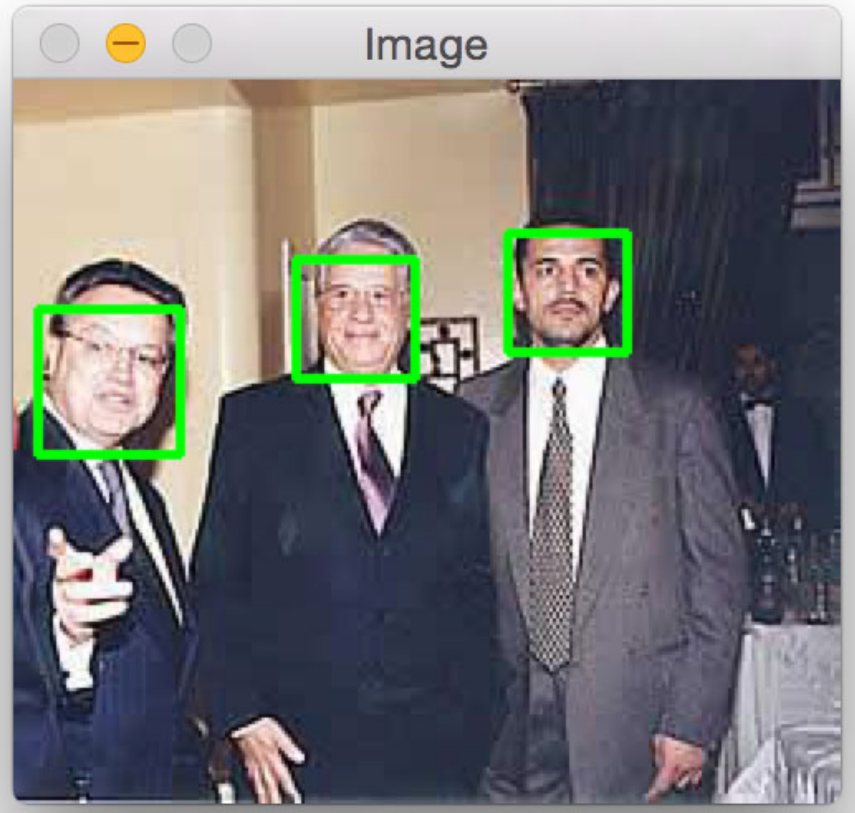
FIGURE 3: DETECTING FACES IN IMAGES USING OUR CUSTOM OBJECT DETECTOR.

Let's try another image:

(https://gurus.pyimagesearch.com/wp-content/uploads/2015/03/custom_object_detector_result_02.jpg)

FIGURE 4: ONCE AGAIN, OUR FACE DETECTOR IS ABLE TO DETECT THE PRESENCE OF FACES IN AN IMAGE.

We'll wrap this lesson up with another example:



(https://gurus.pyimagesearch.com/wp-content/uploads/2015/03/custom_object_detector_result_03.jpg)

FIGURE 5: A FINAL EXAMPLE OF DETECTING FACES IN IMAGES USING OUR CUSTOM OBJECT DETECTOR.

Summary

In this lesson, we learned how to use `imglab` to manually label and annotate a dataset of images. The `.xml` file of annotations produced by `imglab` can be directly fed into dlib for training.

The construction of this `.xml` file is a *huge convenience factor* when using the dlib library. But, of course, you could take the `.xml` file, write a simple parser to extract the *(x, y)*-coordinates for each bounding box, convert it to `.mat` format, and then use the `.mat` files with our **custom object detector framework**. (https://gurus.pyimagesearch.com/lessons/getting-started-with-dalal-and-triggs-object-detectors/)

Downloads:

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

	Quizzes	Status
1	Training Your Custom Object Detector Quiz (https://gurus.pyimagesearch.com/quizzes/training-your-custom-object-detector-quiz/)	

← Previous Lesson (https://gurus.pyimagesearch.com/lessons/re-training-and-running-your-classifier/) Next Lesson → (https://gurus.pyimagesearch.com/lessons/tips-on-training-your-own-object-detectors/)

Course Progress

Ready to continue the course?

Click the button below to **continue your journey to computer vision guru**.

I'm ready, let's go! (/pyimagesearch-gurus-course/)

Resources & Links

- PyImageSearch Gurus Community (https://community.pyimagesearch.com/)
- PyImageSearch Virtual Machine (https://gurus.pyimagesearch.com/pyimagesearch-virtual-machine/)
- Setting up your own Python + OpenCV environment (https://gurus.pyimagesearch.com/setting-up-your-python-opencv-development-environment/)
- Course Syllabus & Content Release Schedule (https://gurus.pyimagesearch.com/course-syllabus-content-release-schedule/)
- Member Perks & Discounts (https://gurus.pyimagesearch.com/pyimagesearch-gurus-discounts-perks/)
- Your Achievements (https://gurus.pyimagesearch.com/achievements/)
- Official OpenCV documentation (http://docs.opencv.org/index.html)

Your Account

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

Search

Feedback

/