# 2.12: Tips on training your own object detectors

Throughout this module, we have learned how to train our own custom object detectors.

We started by using **dlib (http://dlib.net/)**, a computer vision, image processing, and machine learning library that (amongst other functionality), includes mechanisms to quickly train object detectors.

We also created a **Python framework (https://gurus.pyimagesearch.com/lessons/getting-started-with-dalal-and-triggs-object-detectors/)** that we can leverage to rapidly construct an object detector using only a single JSON configuration file.

Using these two approaches, you have the tools to build and deploy your own object detectors — but before you go off in the real-world, let's review some common tips and tricks you should keep in mind when training your own detectors.

## Objectives:

In this lesson, we will:

- Learn various tips and tricks you should consider applying when training your own custom object detectors.

## Tips on training your own object detectors

Feedback

Below, I have included some of my favorite (and most useful) tips and tricks that I have used when training my own custom object detectors. My hope is that, by creating this list, you find it beneficial when training your own detectors.

Some of these tips are specific to dlib, some are related to only our custom object detection framework, while other suggestions are applicable to both.

Keep these suggestions in mind when working on your own custom object detectors, and you'll avoid common snags and pitfalls — and hopefully obtain a much more accurate object detector with less effort!

## Take special care labeling your data

Remember, your object detector is only as good as the data you put into it As the old machine learning mantra goes: ***"garbage in, garbage out."*** This means that you need to take extra special care when labeling and annotating your image data.

You don't need an *exact* pixel bounding box cropping of objects in an image; however, you should at least be careful enough to ensure parts of an object are not missing in a bounding box, or your bounding box being so large that it contains other objects.

While this isn't as big of a deal with our Python object detection framework, it is a *huge* deal with dlib, since dlib uses *non-object regions* as hard-negative data.

## Leverage parallel processing

As we noticed, it can take up to a couple seconds to process even small images using our Python framework. To speed up the process, you first need to consider if you are either:

1. Batch processing a large dataset of images for objects offline.
2. Performing real-time processing, where the goal is to perform object detection in near real-time.

While we didn't cover how to use multiple processor cores to speed up the detection process in this module (since multiprocessing would have made our code *substantially* more complicated), we can leverage multiprocessing in both of these situations.

In the case of batch processing, **distribute an image to each available core.**

Feedback

And in the case of real-time processing, **start by taking a single image, generate the image pyramid, and then distribute each layer of the pyramid to each available core on the processor.** This will allow each core to process a single layer of the pyramid, where the sliding window, HOG feature extraction, and SVM detection take place.

That said, it's important to keep in mind that your object detection pipeline will only be as fast as the time it takes to process the largest layer of the pyramid; therefore, one of the easiest ways to speed up your detection process is to resize your image to be as small as possible without sacrificing accuracy.

## Use dlib as a starting point

Whenever you work in machine learning, obtaining a baseline accuracy is extremely important.

Since it's very easy (and not to mention fast) to train an object detector with dlib, be sure to use the dlib framework as a starting point. Not only will this give you a baseline that you can use for comparison in future experiments, but also there is always the chance that the baseline will work well enough for your particular problem.

## Keep in mind the image pyramid and sliding window tradeoff

As mentioned in our **image pyramids (https://gurus.pyimagesearch.com/topic/image-pyramids/)** and **sliding windows (https://gurus.pyimagesearch.com/topic/sliding-windows/)** lessons, there is a tradeoff between the scale of the image pyramid and the step size of a sliding window.

Feedback

The smaller the pyramid scale, the *more* pyramid layers need to be evaluated. Similarly, as the sliding window step size *decreases*, our sliding window count *increases* as well. Combined together, we can exponentially increase the number of windows that need to be evaluated for a given image.

In general, you'll want to keep both your pyramid scale and sliding window step size as large as possible, *but without sacrificing accuracy*, to ensure the fastest detection speed.

## Tune detector hyperparameters

There are a number of hyperparameters that need to be tuned when building custom object detectors.

The dlib library can help *automatically* tune HOG and sliding window size, but if you're using the Python framework from this module, you'll need to tune these parameters yourself.

Whenever you construct a new object detector, you'll want to refer to the **preparing your experiment and training data (https://gurus.pyimagesearch.com/lessons/preparing-your-training-data/)** and **constructing your HOG descriptor (https://gurus.pyimagesearch.com/lessons/constructing-your-hog-descriptor/)** lessons to help tune these values.

In general, it's best to run `explore_dims.py` to determine the average width, height, and aspect ratio of the objects in your dataset. From there, find the closest power of two that matches these dimensions — and then divide the spatial dimensions by two. Using these dimensions, you can start tuning your HOG parameters.

Finally, consider if horizontal (or even vertical flips) of your bounding box ROIs can be used as additional training data.

## Run experiments and log your results

Training a custom object detector is not like programming a piece of software to play tic-tac-toe (https://en.wikipedia.org/wiki/Tic-tac-toe), where you can easily determine if the results are correct or not. Instead, you should let the accuracy on your validation and test sets drive future experiments.

Be sure to log *every* experiment you perform in a notebook or journal (I prefer using Evernote (https://evernote.com/)), so you can refer back to the results at a later date.

Feedback

If a set of parameters did not work well for a particular experiment, be sure to list them in your notebook and try to make an educated guess as to why they didn't perform well. Similarly, if a set of parameters *did* work well, make note of this — and continue down this path to see if accuracy can improve further.

Again, it's important to keep a formal notebook of results, so you can refer to them at a later date. Furthermore, you should approach parameter selection in a *methodical* way and not follow what your "gut" tells you will work (at least initially). Until you have developed enough experience through trial and error, a methodical approach is always the best.

*Eventually* you will get to the point where you can look at a problem and quickly ascertain the best starting point — but getting there is a lot of hard work.

# Summary

In this lesson, we reviewed some tips and tricks I have learned along the way when training custom object detectors. These tips include:

- Taking special care to label your training data
- Leveraging parallel processing
- Using dlib as a starting point
- Keeping in mind the image pyramid and sliding window tradeoff
- Tuning hyperparameters
- Running experiments and logging your results

Out of all the tips and tricks in your lesson, I cannot stress enough the importance of ***running experiments and logging your results***. Every time you try a new set of parameters, change your training data, and evaluate the detector, these experiment notes should be journaled in your notebook. After performing a set of experiments, it becomes easier to figure out what worked well (and what didn't), allowing you to create new experiments that can be used to increase detection accuracy.

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

Feedback

← Previous Lesson (https://gurus.pyimagesearch.com/lessons/training-your-custom-object-detector/) Next Lesson → (https://gurus.pyimagesearch.com/lessons/what-is-content-based-image-retrieval/)

## 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